State soup: in-context skill learning, retrieval and mixing

author names withheld

Under Review for NGSM 2024

Abstract

A new breed of gated-linear recurrent neural networks has reached state-of-the-art performance on a range of sequence modeling problems. Such models naturally handle long sequences efficiently, as the cost of processing a new input is independent of sequence length. Here, we explore another advantage of these stateful sequence models, inspired by the success of model merging through parameter interpolation. Building on parallels between fine-tuning and in-context learning, we investigate whether we can treat internal states as task vectors that can be stored, retrieved, and then linearly combined, exploiting the linearity of recurrence. We study this form of fast model merging on Mamba-2.8b, a pretrained recurrent model, and present preliminary evidence that simple linear state interpolation methods suffice to improve next-token perplexity as well as downstream in-context learning task performance.

1. Introduction

Transformers [23] have become the standard neural network architecture for sequence modeling. Their parallelizable training and predictable scaling behavior [11] have led to unprecedented performance in a wide range of problem domains, with language modeling being the prime example. However, this architecture comes with the drawback that the memory and computational costs of inference scale quadratically with context length. This undesirable property has led to continued interest in recurrent models that propagate forward an internal state as a sequence is processed, for which inference costs instead scale linearly with context length.

Recently, great strides have been made in recurrent neural network (RNN) architecture research [see, e.g., 2, 6–8, 16, 26]. We now have a number of architectures that can be trained as efficiently as Transformers, some of which exhibit similar scaling laws in language modeling up to the billion-parameter range [2, 6, 7, 26]. At the core of these modern RNNs – and of particular importance to the present paper – is the use of gated-linear recurrences, which marry RNN gating techniques [5, 10] with classical linear state-space models, developed within control and linear filter theory. Linear recurrences enable computationally-efficient training (in particular for earlier ungated variants) and lead to good optimization properties when correctly parametrized [8, 16, 20].

Here, we further exploit the linearity of modern RNNs and introduce *state soups*. Inspired by model soups [18, 24, 25], which improve or change the training objective by linearly interpolating the parameters of several models, we propose instead to perform state-space interpolations. As we show below, linear interpolation is exact for a single gated-linear recurrent layer, and approximately correct in a number of experiments performed with Mamba 2.8B, a pretrained modern RNN comprising dozens of layers. Conceptually, state soups not only enable parallel information processing across independent models, but also the caching of preprocessed information that can be

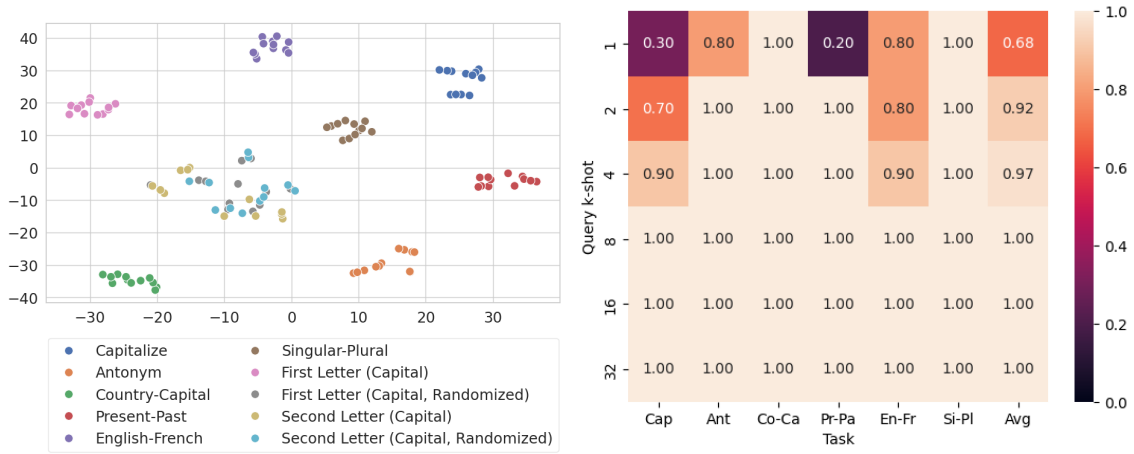

Figure 1: (Left) In our skill library, states from the same task are clustered under T-SNE projection. (Right) Using a query state obtained after processing k examples from a given task (y-axis), we check the probability that the closest state in the library is from the same task.

later retrieved to augment a novel query, reminiscent of retrieval-augmented generation [13] and complementary learning systems theory [12, 14]. We demonstrate the latter on a recently developed suite of in-context learning tasks [21], finding that query-based retrieval over a library of in-context learned skills is possible without any model fine-tuning. Finally, we find a few successful instances where a form of linear task arithmetic is possible, similarly to recent work on function vectors in Transformers [9, 21] and to the seminal discovery of word vector arithmetics [15].

2. State soup

We explore the idea of building a library of in-context learned (ICL) skills, represented by RNN states that can be used for retrieval and mixing. For this purpose, we employ a set of simple ICL tasks proposed by [21], where a single task is a sequence of k examples formatted as (`<question>` $\rightarrow$`<answer>\n`). We use a pretrained Mamba [7] model with 2.8B parameters for generating the skills for the library and subsequent testing. Each skill-representing state is obtained by processing 32 examples. For each task, we sample multiple disjoint sets of examples, so that we have multiple RNN states for each skill. For more technical details, see Appendix A.

Using this setup, we ask three questions. (1) **Task retrieval**: Given a short task example, can we retrieve the corresponding state from the skillset? (2) **State mixing**: Can we mix different states to boost results? (3) **State mixing with sequential data**: Can we apply mixing to sequential data?

2.1. Task retrieval

In Figure 1 (left), we take an intermediate layer[1] from every state in our skill library, and we project it to two dimensions using T-SNE. As a result, we obtain a proper clustering, where states corresponding to the same task are grouped together. Even more interestingly, tasks that Mamba cannot

1. We observed that intermediate layers encode the task most reliably, so we use the 32nd layer our of 64 in retrieval experiments.

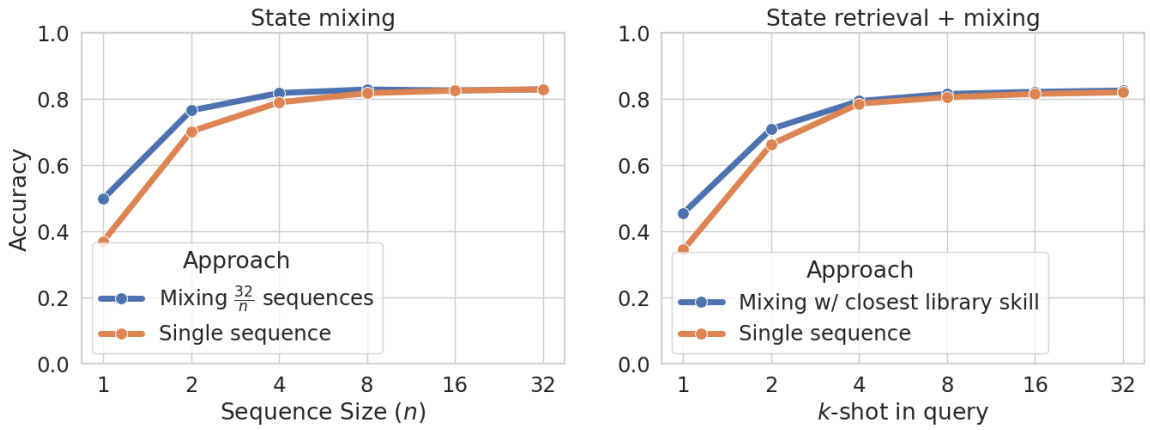

Figure 2: State retrieval and mixing improves few-shot learning performance. (Left) The x-axis represents the number of examples in the processed sequence. (Right) The x-axis represents the number of examples observed in the query state.

learn are also clustered together, suggesting that the task essence can be easily decoded from the state vector.

To quantitatively test the retrieval capabilities of the model, we verify if a query vector q_τ^k obtained by processing a k-shot example from a task τ from the skill library can be correctly identified as coming from τ. In particular, in Figure 1 (right), we check if the closest neighbor of q_τ^k is also a τ-state. We check different values of k, and we we observe that our approach is able to reliably identify the correct task even for k as low as 2-shots, while for some tasks, it is possible even with 1-shot.

2.2. State mixing

Having established that relevant-state retrieval can be easily accomplished, we turn to state mixing, with the goal of using a task library to enhance the few-shot performance of our model. To mix the states, we simply take a mean over them. In our first experiment, see Figure 2 (left), we compare the standard in-context learning with mixing. In the baseline setting (orange line) Mamba sees a single sequence of k examples, where k is depicted on the x axis. On the other hand, our simple mixing strategy (blue line) mixes $\frac{32}{k}$ states, each obtained by independently processing k examples. As such, mixing always uses 32 examples in total. We observe that although mixing 32 1-shot states does not lead to great results, mixtures of 4-shot states are already on par or even slightly better than processing the whole 32-shot at once.

With positive results from the first experiment, we can move to incorporating states retrieved from the state library. In this setup, each query state q_τ^k is obtained by processing k-shot demonstrations from task τ. We use q_τ^k to retrieve the most similar state from the library, which is then mixed with the query state. The mixed state is finally used as the initial state for processing the test sample. The results shown in Figure 2 (right) show that we can boost the performance, especially with small k

Additionally, we do preliminary tests of mixing states from distinct tasks. In particular, we find that mixing states resulting from the "counting" task (`one` $\rightarrow$ `two` $\rightarrow$ `...` $\rightarrow$ `fourteen` $\rightarrow$) and "English-French" translation task resulted in the model predicting the next number in French

(`quinze`). Note that obtaining this result required taking a weighted mean of the two states and is not yet backed by quantitative studies. However, we deem this an intriguing research direction.

2.3. Mixing with sequential data

In our previous experiments, the ordering of the mixed chunks was irrelevant. However, the data is inherently ordered in many practical scenarios, such as processing long sequences. Here, we propose a mixing strategy that takes the sequential nature of data into consideration.

A single discretized SSM layer that processes the sequence $x_1, \ldots, x_t$ can be depicted recursively as: $f(x_1, \ldots, x_t) = A_t f(x_1, \ldots, x_{t-1}) + B_t x_t$, see Appendix A for a detailed notation. Observe that due to the linearity of SSM, for each $k \in \{1, \ldots t-1\}$ we can write this equation as: $f(x_1, \ldots, x_t) = f(x_{k+1}, \ldots, x_t) + \left(\prod_{k'=1}^{t-k} A_{k'}\right) f(x_1, \ldots, x_k)$.

As such, in the linear setting, we can independently process sequences $x_1, \ldots, x_k$ and $x_{k+1}, \ldots, x_t$ and combine them exactly as long as we have the $\prod_k A_{k'}$ matrix. Fortunately, this matrix is computed for the parallel scan algorithm that enables efficient training of different SSM architectures [20]. We call this approach *A-decay mixing*.

For a single SSM layer, A-decay mixing gives us exactly the same solution as processing the whole sequence. However, in the full Mamba architecture, the inputs to the subsequent SSM layers depend on the outputs of the previous layer, and as such, A-decay mixing will only give us an approximation. To empirically test its quality, we take 10000 sequences from the realnewslike dataset [17], and we divide each of them into three chunks: $c_1, c_2, c_{\text{test}}$, each with 100 tokens. We check the prediction loss on c_{test} using different RNN states: (1) sequential processing of c_1 and c_2, (2) sequential processing of c_2, (3) mean-mixing of states after processing separately c_1 and c_2, (4) same as previous but with A-decay mixing. Figure 3 shows that the A-decay mixing outperforms both mean-mixing and starting from a state that only processed c_2. However, the performance is still worse than the model that saw both c_1 and c_2, suggesting that A-decay offers us only an approximate solution in cases when the RNN layers are stacked.

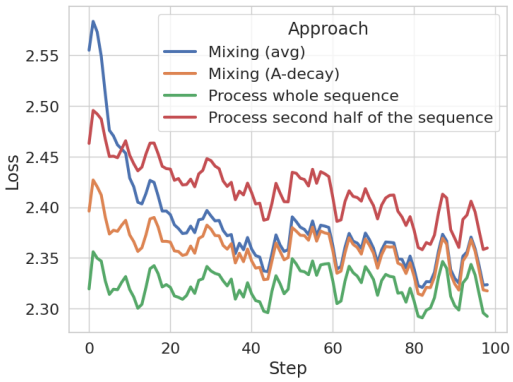

Figure 3: Different mixing approaches on the next token prediction problem.

3. Conclusion

Neural systems that combine some form of volatile fast learning with persistent memory stores have been previously studied in meta-learning [e.g., 19]. Our preliminary findings suggest that it is possible to leverage the in-context learning abilities of recurrent neural networks to generate task representations that can be committed to memory and then later retrieved and reused or even repurposed. As the capacity to learn and compress long ['many-shot'; 1] tasks in-context with RNNs increases, so will the relative advantage of our method against Transformer-based alternatives, which do not offer a way to reuse preprocessed states off-the-shelf at $\mathcal{O}(1)$ cost (but see [9, 21]). In future work, we wish to extend our setup to include broader and more realistic tasks, and to quantitatively analyze the ability to perform task arithmetics.

References

[1] Rishabh Agarwal, Avi Singh, Lei M Zhang, Bernd Bohnet, Stephanie Chan, Ankesh Anand, Zaheer Abbas, Azade Nova, John D Co-Reyes, Eric Chu, and others. Many-shot in-context learning. *arXiv preprint arXiv:2404.11018*, 2024.

[2] Maximilian Beck, Korbinian Pöppel, Markus Spanring, Andreas Auer, Oleksandra Prudnikova, Michael Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. xLSTM: extended long short-term memory. *arXiv preprint arXiv:2405.04517*, 2024.

[3] Sid Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, et al. Gpt-neox-20b: An open-source autoregressive language model. *arXiv preprint arXiv:2204.06745*, 2022.

[4] David M Chan, Roshan Rao, Forrest Huang, and John F Canny. Gpu accelerated t-distributed stochastic neighbor embedding. *Journal of Parallel and Distributed Computing*, 131:1–13, 2019.

[5] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[6] Soham De, Samuel L. Smith, Anushan Fernando, Aleksandar Botev, George Cristian-Muraru, Albert Gu, Ruba Haroun, Leonard Berrada, Yutian Chen, Srivatsan Srinivasan, and others. Griffin: mixing gated linear recurrences with local attention for efficient language models. *arXiv preprint arXiv:2402.19427*, 2024.

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

[8] Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. In *International Conference on Learning Representations*, 2022.

[9] Roee Hendel, Mor Geva, and Amir Globerson. In-context learning creates task vectors. *arXiv preprint arXiv:2310.15916*, 2023.

[10] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9 (8):1735–1780, 1997.

[11] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[12] Dharshan Kumaran, Demis Hassabis, and James L. McClelland. What learning systems do intelligent agents need? Complementary learning systems theory updated. *Trends in Cognitive Sciences*, 20(7):512–534, 2016.

[13] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

[14] James L. McClelland, Bruce L. McNaughton, and Randall C. O'Reilly. Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory. *Psychological Review*, 102(3): 419, 1995.

[15] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26, 2013.

[16] 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*, 2023.

[17] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

[18] Alexandre Rame, Guillaume Couairon, Corentin Dancette, Jean-Baptiste Gaya, Mustafa Shukor, Laure Soulier, and Matthieu Cord. Rewarded soups: towards pareto-optimal alignment by interpolating weights fine-tuned on diverse rewards. *Advances in Neural Information Processing Systems*, 36, 2024.

[19] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-learning with memory-augmented neural networks. In *International Conference on Machine Learning*, 2016.

[20] Jimmy T.H. Smith, Andrew Warrington, and Scott W. Linderman. Simplified state space layers for sequence modeling. In *International Conference on Learning Representations*, 2023.

[21] Eric Todd, Millicent L. Li, Arnab Sen Sharma, Aaron Mueller, Byron C. Wallace, and David Bau. Function vectors in large language models. In *International Conference on Learning Representations*, 2024.

[22] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008. URL http://jmlr.org/papers/v9/vandermaaten08a.html.

[23] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[24] Johannes Von Oswald, Seijin Kobayashi, Alexander Meulemans, Christian Henning, Benjamin F. Grewe, and João Sacramento. Neural networks with late-phase weights. In *International Conference on Learning Representations*, 2021.

[25] Mitchell Wortsman, Gabriel Ilharco, Samir Ya Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S. Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, and Ludwig Schmidt. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *International Conference on Machine Learning*, 2022.

[26] Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. Gated linear attention transformers with hardware-efficient training. *arXiv preprint arXiv:2312.06635*, 2023.

Appendix A. Technical details

In all of our experiments we use the Huggingface Mamba-2.8b implementation which utilizes the GPT-NeoX-20B tokenizer [3]. The basis of our experiments in sections 2.1 and 2.2 are the six main tasks introduced and investigated in [21], namely **Ant**onym (flawed: perfect, unrelated: related), **Cap**italize (lift: Lift, mirror: Mirror), **Co**untry-**Ca**pital (Egypt: Cairo, Poland: Warsaw), **En**glish-**Fr**ench (satisfy: satisfaire, here: ici), **Pr**esent-**Pa**st (assist: assisted, speak: spoke) and **Si**ngular-**Pl**ural (bat: bats, mouse: mice).

In the clustering experiments, we use two further tasks, First Letter, Capital (python: P, finch: F) and Second Letter, Capital (ecstatic: C, change: H) along with *Randomized* variations created by randomly shuffling the labels. This preserves the label distribution but removes any relationship between the input and the output.

A.1. Task Retrieval

In the clustering experiments, we create 12 16-shot demonstrations per task, while ensuring that no sample appears in multiple demonstrations. We then perform t-SNE dimensionality reduction [22] using the t-SNE-CUDA [4] library.

In the retrieval experiments, we create a library of 10 states per task, with each state resulting from processing 32-shot demonstration. For $k = 1, 2, 4, \ldots, 32$ we create 10 k-shot demonstrations, ensuring no overlap with the library. To measure the similarity of the queries to the states in the library, we use the cosine similarity of the SSM state at the 32nd layer.

A.2. State Mixing

In the state mixing experiments (Figure 2, left), we select 500 random test samples from the datasets, apart from the Country-Capital, Present-Past and Singular-Plural tasks, which only contain 197, 293 and 205 samples respectively - in those cases, we perform testing on all samples. We ensure that the test sample is not present in the few-shot samples or the samples used to obtain the mixed state.

In the "retrieval + mixing" experiments, we first randomly divide each dataset into two halves and only use samples from one of the halves to create the library, while testing the performance on the other half. Once again, we use 500 samples if possible, however, due to the halving, the Capitalize dataset is another dataset for which we use a smaller number (407) of test samples.

A.3. Mixing with sequential data

Here, we provide an explanation for the notation used in Section 2.3. The discretized SSM equation can be written as:

$$f(x_1, \ldots, x_t) = A_t f(x_1, \ldots, x_{t-1}) + B_t x_t, \tag{1}$$

where f represents the SSM function, $x_1, ..., x_t$ are the tokens to be processed and each token is a vector of dimensionality D. $A_t \in \mathbb{R}^{H \times H}$ and $B_t \in \mathbb{R}^{H \times D}$, are the parameters of the model, where H is the internal (hidden) dimensionality of the RNN. Both A_t and B_t might be time- and input-dependent, as in Mamba [7], or independent as in the original SSMs [8].

Appendix B. Additional results

B.1. State Retrieval

The clustering of states belonging to the same task (or lack thereof) after dimensionality reduction can be observed across the layers of the model. We can also vary whether we explore the model's SSM or the convolutional state. The dimensionality reduction method plays a crucial role, with t-SNE producing much more visible and interpretable patterns than PCA. The plots showing the 2-dimensional visualizations of the states are presented in Figures 4 and 5.

B.2. State Mixing

Due to limited space, the results presented in Figure 2 are aggregated across tasks. Per-task results can be found in Figures 6 and 7.

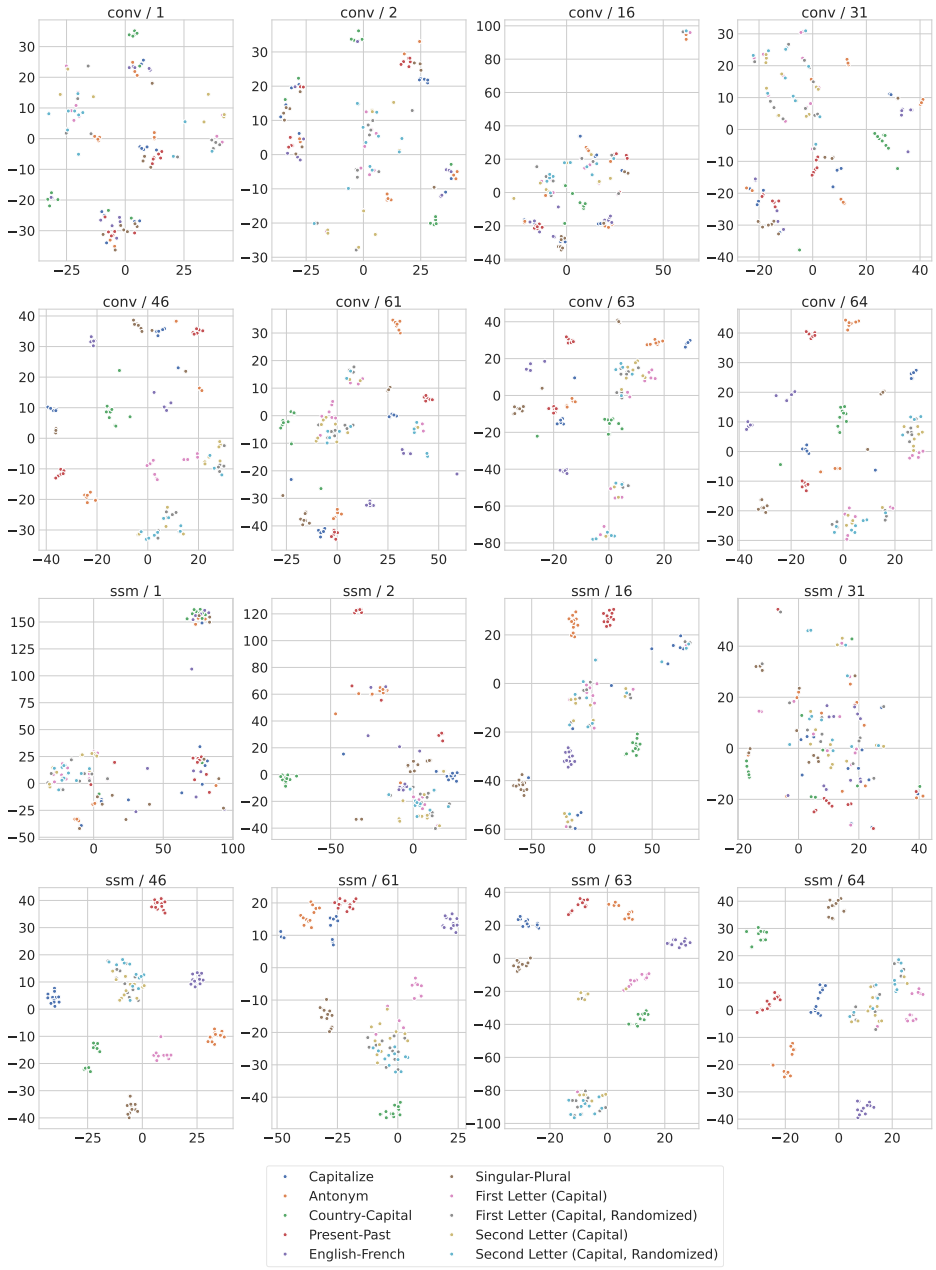

Figure 4: States clustering - dimensionality reduction performed with t-SNE. Both SSM and convolution states are investigated.

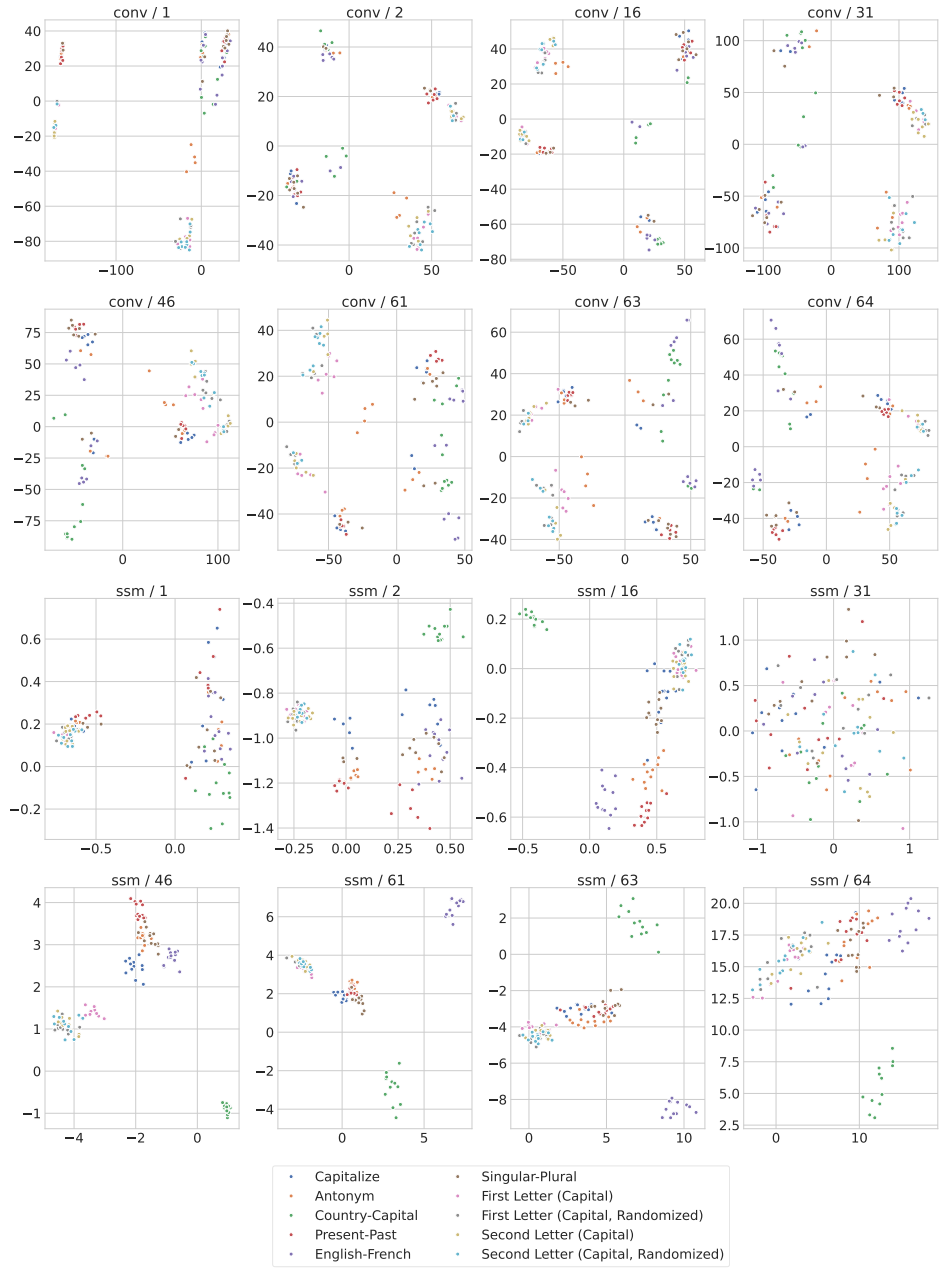

Figure 5: States clustering - dimensionality reduction performed with PCA. Both SSM and convolution states are investigated.

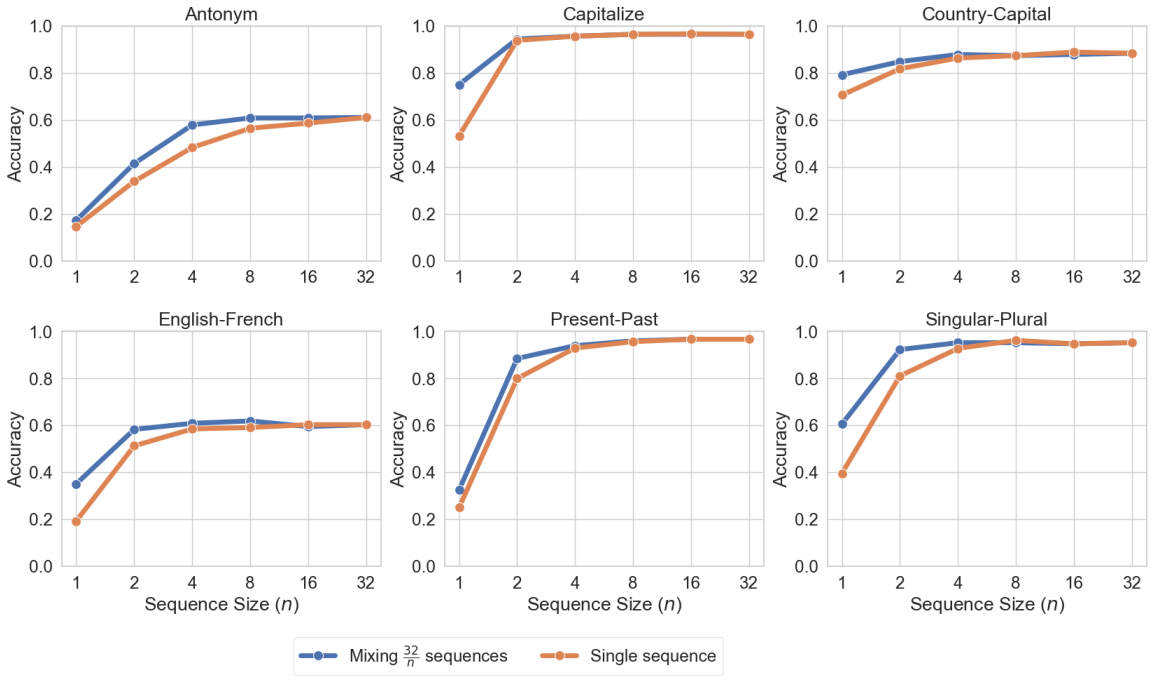

Figure 6: State mixing improves few-shot learning performance. The x-axis represents the number of examples in the processed sequence.

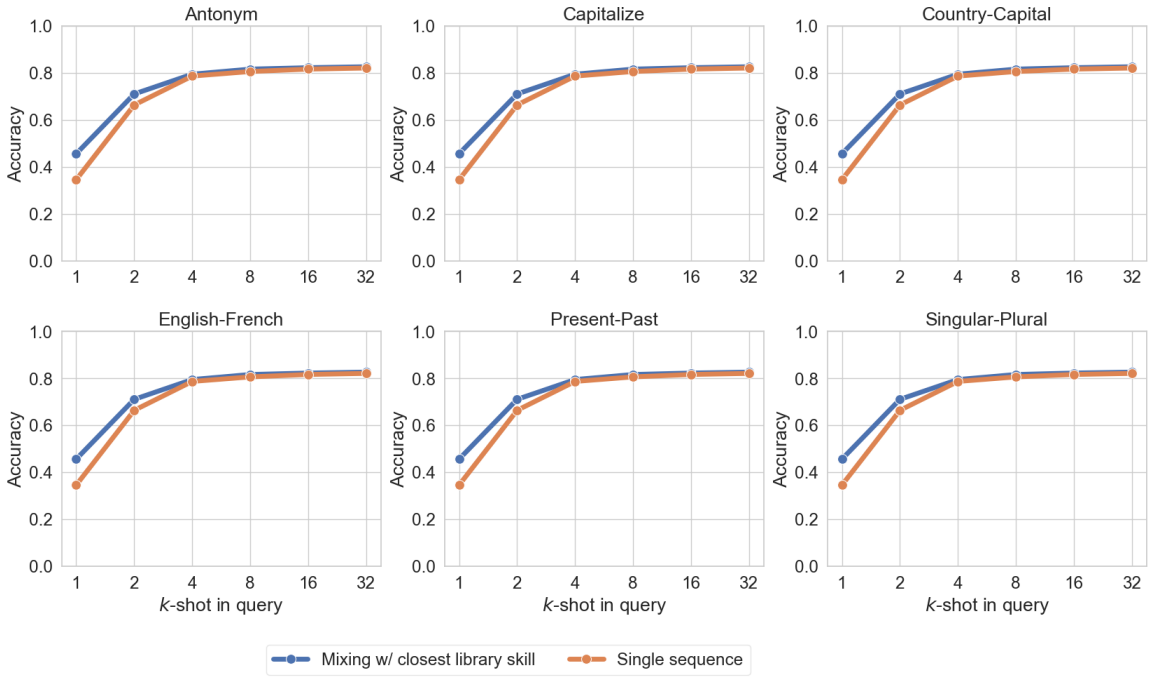

Figure 7: State retrieval and mixing improves few-shot learning performance. (Right) The x-axis represents the number of examples observed in the query state.