CONTEXT-SELECTIVE STATE SPACE MODELS: FEEDBACK IS ALL YOU NEED

Riccardo Zattra Giacomo Baggio Umberto Casti Augusto Ferrante Francesco Ticozzi
Department of Information Engineering
University of Padova
Padova, 35131, Italy
{riccardo.zattra, baggio, castiumber, augusto, ticozzi}@dei.unipd.it

ABSTRACT

Transformers, powered by the attention mechanism, are the backbone of most foundation models, yet they suffer from quadratic complexity and difficulties in dealing with long-range dependencies in the input sequence. Recent work has shown that *state space models* (SSMs) provide an efficient alternative, with the S6 module at the core of the Mamba architecture achieving state-of-the-art results on long-sequence benchmarks. In this paper, we introduce the **COFFEE** (COntext From FEEdback) model, a novel time-varying SSM that incorporates *state feedback* to enable context-dependent selectivity, while still allowing for parallel implementation. Whereas the selectivity mechanism of S6 only depends on the current input, COFFEE computes it from the internal state, which serves as a compact representation of the sequence history. This shift allows the model to regulate its dynamics based on accumulated context, improving its ability to capture long-range dependencies. In addition to state feedback, we employ an efficient model parametrization that removes redundancies present in S6 and leads to a more compact and trainable formulation. On the induction head task, COFFEE achieves near-perfect accuracy with two orders of magnitude fewer parameters and training sequences compared to S6. On MNIST, COFFEE largely outperforms S6 within the same architecture, reaching 97% accuracy with only 3585 parameters. These results showcase the role of state feedback as a key mechanism for building scalable and efficient sequence models.

1 Introduction

Recurrent Neural Networks and *Long Short-Term Memory Networks* have been the dominant architectures for sequence modeling up to the introduction of *Transformers* [1]. The latter are the backbone of modern *foundation models*, namely large-scale models that are pre-trained on massive amounts of data (text, audio, video, etc.) and subsequently fine-tuned, using task-specific data, for specific downstream tasks. By leveraging the celebrated attention mechanism, transformers achieved a groundbreaking improvement in generative Artificial Intelligence (AI), [2, 3, 4, 5, 6]. The attention mechanism, though revolutionary, suffers from some limitations. In particular, a quadratic time complexity with respect to the length of the input sequence and severe difficulties in capturing long-range dependencies call for further research and improvements. To this end, many variations of the attention mechanism have been proposed; see [7, 8, 9, 10, 11]. A completely different and extremely promising approach is the use of linear State-Space Models (SSMs): introduced in the 1960's, they quickly revolutionized control and automation engineering [12, 13], and their use spread to neighboring fields, from biology [14] to economics [15] and statistical modeling [16]. In fact, a significant stream of valuable contributions have recently appeared that replace the attention mechanism with SSMs [17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29]. Other interesting works, such as [30] and [31] employ nonlinear SSMs to address questions about asymptotic behavior of Transformers as the number of encoders layers increases.

The state of the art of SSM in sequence modeling is called **S6** and it lies at the core of the so-called **Mamba** architecture [20]. The latter exhibits great capabilities and is able to outperform the state-of-the-art transformer in challenging tasks, including the Long Range Arena (LRA), see [20] and [32]. The reason for this success lies in the fact that the *state* is, by definition, the mathematical object that encodes all the relevant information on the past evolution of a system needed

to predict its future behavior[1]. This *separation property* allows states to store elusive long-range dependencies. Of course, the main challenge is to derive a universal class of state-space models whose parameters can be learned for each specific task. The S6 module captures, or rather approximates, this universality by resorting to (linear) time-varying models.

In this paper, we propose a further dramatic improvement: a novel class of SSMs that we call COFFEE (COntext From FEEdback) which is endowed with feedback capabilities. Dynamical systems endowed with feedback loops are at the basis for the large majority of complex natural and artificial phenomena, as they provide a system with the possibility of changing its behavior on the basis of its current state, which, as previously recalled, contains all the relevant information about the past of the sequence under analysis. In this way, we are able to induce **context-selectivity** in the behavior of COFFEE: a property of paramount importance, which appears to give a momentous boost to the system's performance. With respect to S6, COFFEE has another advantage (which could also be implemented in the S6 model): by exploiting a change of basis in the state-space representation of the model, we are able to reduce appreciably the number of learnable parameters, offering an obvious edge in terms of both numerical implementation and mechanistic interpretability.
We tested COFFEE in several versions of the induction head benchmark and in the MNIST benchmark. In these tasks, our model appears to perform significantly better than the state-of-the-art S6 in more than one way. In fact, not only does COFFEE give much better results in terms of the accuracy of its output, but it also requires a smaller number of learnable parameters and a much smaller training set to converge. The closed-loop COFFEE model is obtained by a nonlinear state-feedback action on a time-varying linear model. In general, this kind of structure leads to computational drawbacks because it is not amenable to parallelization and hence cannot take advantage of the highly parallel architecture of modern GPU's. In COFFEE, however, the feedback structure is constrained in such a way that, on the one hand, the performance degradation (with respect to the case of a more general feedback structure) is negligible and, on the other, the Jacobian of the state transition function is guaranteed to be diagonal. As a consequence, we can leverage recent results in [33] to obtain scalable parallel training that is computationally highly efficient.

The remainder of this paper is organized as follows. In Section 2 we give a brief background on state-space models, with further details and useful proofs provided in A. Based on the review of state space models, in Section 3, we present our model along with its mathematical motivations. In Section 4, we prove the effectiveness of our model by comparing its performance with the state-of-the-art S6 model [20], in both the induction head and MNIST tasks. In Section 5, we provide some theoretical insights, based on an analysis made on a simplified version of the induction head task, whose details are presented in Appendix D. We summarize our findings in Section 6, where we also outline the next steps in the development and testing of the model.

2 Background on SSMs

2.1 Linear, discrete SSM

State space models are powerful tools to model, analyze and control dynamical systems [13]. These are models in which the relation between the input or control sequence $u(k) \in \mathbb{R}^m$ and the sequence $y(k) \in \mathbb{R}^p$ associated to the quantities of interest for the system at hand and called the *output*, is mediated by a signal $x(k) \in \mathbb{R}^n$, which is called the *state* of the system. For more details, see Appendix A. In this paper, we leverage the class of linear, time-varying, discrete-time systems with finite-dimensional state space. The models of this class are described by a couple of vector equations of the form

$$\begin{aligned} x(k) &= A(k)x(k-1) + B(k)u(k) \\ y(k) &= C(k)x(k) \end{aligned} \tag{1}$$

where the matrix $A(k) \in \mathbb{R}^{n\times n}$ is called *state transition matrix*, $B(k) \in \mathbb{R}^{n\times m}$ is called *input matrix*, $C(k) \in \mathbb{R}^{p\times n}$ is called *output matrix*. From (1) it follows that, given the state $x(k)$ at the present "time" k, the past of the input is irrelevant for the future of the output sequence. This key feature of the state is known as the *separation property*, and is the fundamental reason why the state functions as memory. The state can then be considered as a natural representative of the *context* set by the past data, and as such will be exploited in our setting in selecting the dynamics conveniently.

2.2 Relevance of SSMs to Sequence Modeling

In machine learning, especially in sequence modeling tasks (e.g. time series forecasting, language modeling), SSMs offer several advantages:
Efficiency: The convolutional formulation allows for fast evaluation via the Fast Fourier Transform (FFT), reducing the complexity from $\mathcal{O}(L^2)$ (as in the attention mechanism) to $\mathcal{O}(L \log L)$, L being the length of the input signal $u(k)$;

[1] See Section 2 and Appendix A for more details on this.

Long-range Dependencies: By construction, the state of the system contains a compressed representation of the system's past history. In the context of sequence modeling, the input $u(k)$ corresponds to the tokens provided to the model, meaning that the state is capable of "remembering" all previously seen tokens in a compact form;
Interpretability and Structure: The matrices $A(k), B(k), C(k)$ explicitly define the system dynamics, providing both structure and, ideally, interpretability.

Recent neural architectures such as S4 [19] and Mamba [20] leverage these ideas, combining them with modern training techniques and nonlinearities to build powerful sequence models. In particular, S4 uses a time-invariant model, namely a model of the form (1) where, however, the matrices A, B, C are independent of k, while the main novelty of S6, the core module of Mamba, is that it uses time-varying matrices $A(k), B(k), C(k)$.
In the context of sequence modeling, a state-space model as in Equation (1) is used as black box in foundation models. The exogenous input $u(k)$ represents the k-th input token fed into the model, the state $x(k)$ stores relevant information about the tokens up the k-th one, while $y(k)$ is the k-th output: for generative or prediction models $y(k)$ is used for the prediction of $u(k+1)$; more generally, $y(k)$ is the signal used to solve the task at hand.

In general, the matrices A, B, C are learned using a stochastic gradient descent. Computational efficiency and initialization are crucial in this context, therefore, the matrices A, B, C often have a specific structure, giving rise to the concept of structured state space models. For example, the matrix A is typically chosen to be diagonal.

3 COFFEE: Towards Context-Selective SSMs

In this section, we introduce a novel state space model which we call COFFEE (COntext From FEEdback). The state-of-the-art S6 SSM, proposed in [20], enables selectivity conditioned on the current token by adapting the system dynamics to the current input. In contrast, the main novelty of COFFEE is to replace this *token-based selectivity* with *context-based selectivity*, achieved through a feedback mechanism that modulates the dynamics based on the *state* of the model.

Both COFFEE and S6 are implemented within the core module architecture described in Appendix E, which maps sequences over a finite vocabulary V into embeddings in $\mathbb{R}^D$, and processes each embedding feature with a dedicated single-input single-output (SISO) state space model. To motivate our modifications, we now briefly review the structure of the S6 SISO component (a more detailed analysis is available in Appendix B).

3.1 Background: The S6 Model

As mentioned above, in S6 each feature $u(k)_i$ of the embedding $u(k) \in \mathbb{R}^D$ is handled independently by a dedicated SISO state-space model $\Sigma^{(i)}$. This design allows the architecture to scale across the embedding dimension by running D parallel SISO models.

The discrete-time update for each feature is:

$$\begin{cases} x(k)^{(i)} = e^{A^{(i)}\Delta_i(k)} x(k-1)^{(i)} + \big(A^{(i)}\big)^{-1}\big(e^{A^{(i)}\Delta_i(k)} - I\big) B(k)\, u(k)_i, \\ y(k)^{(i)} = C(k)\, x(k)^{(i)}, \end{cases} \tag{2}$$

where $x(k)^{(i)} \in \mathbb{R}^n$ is the hidden state of the i-th feature, $A^{(i)} = \mathrm{diag}(\lambda_0^{(i)}, \dots, \lambda_{n-1}^{(i)}) \in \mathbb{R}^{n\times n}$ is a diagonal transition matrix, and $u(k)_i$ denotes the i-th feature of the embedding vector $u(k)$.

The input-to-state and state-to-output mappings are defined as

$$B(k) = W_B u(k), \qquad C(k) = (W_C u(k))^\top,$$

with $W_B, W_C \in \mathbb{R}^{n\times D}$ shared across all features. Similarly, the step sizes $\Delta_i(k)$ are determined from the current input as the components of the vector

$$\Delta(k) = \zeta(W_D u(k)), \tag{3}$$

where $W_D \in \mathbb{R}^{D\times D}$ and $\zeta(\cdot)$ denotes the softplus function.

A distinctive aspect of S6 is its token-based selectivity: the parameters $B(k)$, $C(k)$, and $\Delta(k)$ depend directly on the current token $u(k)$. This enables the model to dynamically amplify or suppress contributions from different inputs, effectively filtering out irrelevant or noisy symbols while focusing on semantically meaningful ones. Such input-conditioned adaptation is particularly advantageous in sequential domains (e.g., speech, where disfluencies such as "uh" carry little meaning). This token-dependent, time-varying structure is widely regarded as the core innovation that underpins the strong empirical performance of Mamba and its descendants.

3.2 The COFFEE Model

The central novelty of COFFEE is the use of *state feedback* to achieve context-based selectivity. Unlike S6, where the update gates $\Delta_i(k)$ depend only on the current input token, COFFEE computes them as functions of the internal state, allowing the model to adapt its dynamics based on the accumulated context. It is well known and very intuitive that open-loop systems are fragile and sensitive to model uncertainties and disturbances, whereas closed-loop feedback ensures robustness; COFFEE exploits this principle to achieve context-based selectivity, improving robustness and coherence in generative modeling.

Beyond feedback, we further refine the S6 formulation through two steps: (i) linearizing the exponential dynamics to make the role of $\Delta(k)$ more interpretable, and (ii) eliminating parameter redundancy via changes of basis in the input and state spaces. A complete derivation of COFFEE is presented in Appendix C; below we summarize the key steps leading to the proposed model.

Step 1. Simplified Dynamics via Linearization. In S6, the step size $\Delta_i(k)$ controls the tradeoff between retaining past state and incorporating new input. However, its action is embedded within matrix exponentials, which complicates direct analysis. To make this dependence more explicit and obtain a more interpretable formulation, we approximate $e^{A^{(i)}\Delta_i(k)}$ with its first-order Taylor expansion around $\Delta_i(k) = 0$. This yields the dynamics:

$$\begin{cases} x(k)^{(i)} = \big(I + A^{(i)}\Delta_i(k)\big)\, x(k-1)^{(i)} + \Delta_i(k) B^{(i)} u(k)_i, \\ y(k)^{(i)} = C^{(i)} x(k)^{(i)}, \end{cases} \tag{4}$$

where $B^{(i)} \in \mathbb{R}^{n\times 1}$ and $C^{(i)} \in \mathbb{R}^{1\times n}$ are time-invariant, feature-specific parameters. The latter choice on the input and output matrices is motivated by the fact that, due to discretization, the input-to-state mapping in (2) is already feature-specific, while we additionally assign feature-specific $C^{(i)}$ to maintain symmetry and let selectivity be governed solely by the step size $\Delta_i(k)$.

By replacing exponentials with affine dependence on $\Delta_i(k)$, the simplified model in (4) preserves the essential gating mechanism of S6 while rendering the dynamics simpler and more amenable to mechanistic interpretation.

Step 2. Feedback for Context-Based Selectivity. A central strength of S6 lies in its token-based selectivity: the update gates $\Delta_i(k)$ are computed as functions of the current input token $u(k)$, so that the model can suppress irrelevant inputs and amplify informative ones. However, token-based selectivity is limited because identical tokens may play very different roles depending on context. For example, if we ask for an approximation of the number "111.11", where each digit "1" is represented by the same token, the model must treat different occurrences of "1" differently depending on their position.

To address this limitation, we observe that the hidden state $x(k-1)^{(i)}$ of each subsystem $\Sigma^{(i)}$ already encodes a compressed representation of the full history up to step $k-1$, and therefore provides a natural source of contextual information. We therefore replace the token-based gating (3) with a *state-dependent gating rule*:

$$\Delta_i(k) = \sigma\big(w_D^{(i)} \odot x(k-1)^{(i)}\big), \tag{5}$$

where $w_D^{(i)} \in \mathbb{R}^n$ is a learnable parameter vector, $\odot$ denotes the Hadamard product, and $\sigma(\cdot)$ is the sigmoid function applied componentwise to vectors.[2] By doing so, each subsystem adapts its update gate from its past history rather than the raw input token, enabling context-based selectivity: identical tokens can be filtered differently depending on their role in the evolving sequence. Concretely, the gating vector $\Delta_i(k) \in \mathbb{R}^n$ enters the dynamics in (4) as a diagonal state feedback matrix, which modulates the contribution of the past state through

$$(I + A^{(i)}\mathrm{diag}(\Delta_i(k)))x(k-1)^{(i)},$$

and the input through

$$\mathrm{diag}(\Delta_i(k))B^{(i)}u_i(k).$$

The resulting subsystem dynamics is therefore context-adaptive, taking the form

$$\Sigma^{(i)} = \begin{cases} x(k)^{(i)} = (I + A^{(i)}\mathrm{diag}(\Delta_i(k)))\, x(k-1)^{(i)} + \mathrm{diag}(\Delta_i(k))B^{(i)}\, u(k)_i, \\ y(k)^{(i)} = C^{(i)} x(k)^{(i)}. \end{cases} \tag{6}$$

[2]We use the sigmoid rather than the softplus, since its bounded range keeps $\Delta_i(k)$ limited, helping maintain the validity of the Taylor approximation in (4).

Remark 1 (Parameter efficiency). Compared to S6, the proposed feedback formulation is significantly more parameter-efficient: whereas S6 requires a dense $W_D \in \mathbb{R}^{D\times D}$, our version replaces it with D vectors $w_D^{(i)} \in \mathbb{R}^n$, i.e., Dn parameters in total. Since typically $n \ll D$ (e.g., $n = 16$, $D = 2048$ in Mamba[3]), this corresponds to a reduction from 4.2M to just 32K gating parameters.

Step 3. Eliminating Redundancy via Change of Basis. As a final step, we note that the parameterization of $(B^{(i)}, C^{(i)})$ in (6) exhibits parameter redundancy: through suitable changes of basis (corresponding to simple rescalings) the same input-output behavior can be preserved while reducing the number of parameters. This fact is formalized in the following proposition, whose proof is deferred to the Appendix (see Section C.3 where a few comments are also provided on the proposition assumptions).

Proposition 1 (Redundancy Elimination by Change of Basis)**.** *Assume that at least one embedding vector has all the entries that are non-zero and that all the entries of the $B^{(i)}$ are non-zero. For the family of subsystems $\{\Sigma^{(i)}\}_{i=0}^{D-1}$ defined in Eq. (6), there exist changes of basis in the input and state spaces such that the structure of Eq. (6) is preserved and:*

(i) all input-to-state matrices $B^{(i)}$ can be fixed to the constant vector $\mathbf{1} = [1, \dots, 1]^\top$, with their scaling absorbed into the corresponding $C^{(i)}$, and

(ii) one embedding vector can be normalized to $\mathbf{1}$.

The transformations in Proposition 1 preserve both the input-output map and the diagonal structure of $A^{(i)}$. Consequently, the input-to-state matrices $B^{(i)}$ are redundant, as their effect can always be absorbed into $C^{(i)}$, and one embedding vector can be fixed to $\mathbf{1}$. This reduces the parameter count by $nD + D$ without loss of expressivity, yielding a leaner architecture that is easier to train.

Final COFFEE Model. Combining Steps 1-3, the resulting COFFEE subsystem for feature i takes the form

$$\Sigma^{(i)} = \begin{cases} x(k)^{(i)} = (I + A^{(i)}\mathrm{diag}(\Delta_i(k)))\, x(k-1)^{(i)} + \Delta_i(k)\, u(k)_i, \\ y(k)^{(i)} = C^{(i)} x(k)^{(i)}, \end{cases} \tag{7}$$

where the state-dependent gates $\Delta_i(k)$ are given in Eq. (5). Here, the learnable parameters are the diagonal elements $\{\lambda_k^{(i)}\}_{k=0}^{n-1}$ of the matrix $A^{(i)}$, the output matrix $C^{(i)} \in \mathbb{R}^{1\times n}$, and the feedback vector $w_D^{(i)} \in \mathbb{R}^n$. To ensure numerical stability during training and inference, the choice of the parameters $\lambda_k^{(i)}$ is constrained to the interval $[-2, 0]$, which guarantees that the matrix $I + A^{(i)}$ is stable. A schematic representation of the resulting model is shown in Fig. 1.

Remark 2 (Scalable parallel training). By construction, $A^{(i)}$ is diagonal, and each component of $\Delta_i(k)$ depends only on the corresponding component of $x(k-1)^{(i)}$. As a result, the state dynamics in (7) decomposes into n independent scalar recursions. This in turn implies that the state-transition Jacobian of (7)

$$J(k)^{(i)} := \frac{\partial x(k)^{(i)}}{\partial x(k-1)^{(i)}}$$

is diagonal at every time step. The diagonal structure of the Jacobian drastically reduces both memory and computational requirements. In particular, as shown in [33], trajectory computation can be cast as an optimization problem, where diagonal Jacobians allow efficient parallel solvers with total work and memory scaling that are linear in the sequence length T and the state dimension n, i.e., $\mathcal{O}(Tn)$, while the parallel depth in T is only $\mathcal{O}(\log T)$. The overall run-time depends on the number of solver iterations required for convergence, which is typically small in practice. This enables efficient training on parallel hardware, in contrast with cubic and quadratic costs in the state dimension that arise with dense Jacobians.

Remark 3. (Model with output filtering) Notice that the model in (7), can be modified by adding other filtering capabilities on the output without compromising the possibility to parallelize the architecture. In particular, we can transform the output equation in:

$$y(k)^{(i)} = (C^{(i)}\gamma(k)^{(i)})x(k)^{(i)}$$

where $\gamma(k)^{(i)} \in \mathbb{R}$ is evaluated as:

$$\gamma(k)^{(i)} = \sigma((w_\gamma^{(i)})^\top x(k)^{(i)})$$

with $w_\gamma^{(i)} \in \mathbb{R}^n$ learnable parameters. As a consequence, the model would have other nD learnable parameters. More generally, the output equation, being static, does not represent a problem for parallelization. Thus, arbitrary non-linearities (if beneficial) can be added.

[3]The reported numbers are taken from the minimal PyTorch implementation of Mamba (`https://github.com/johnma2006/mamba-minimal`), numerically equivalent to the official release (`https://github.com/state-spaces/mamba/tree/main`).

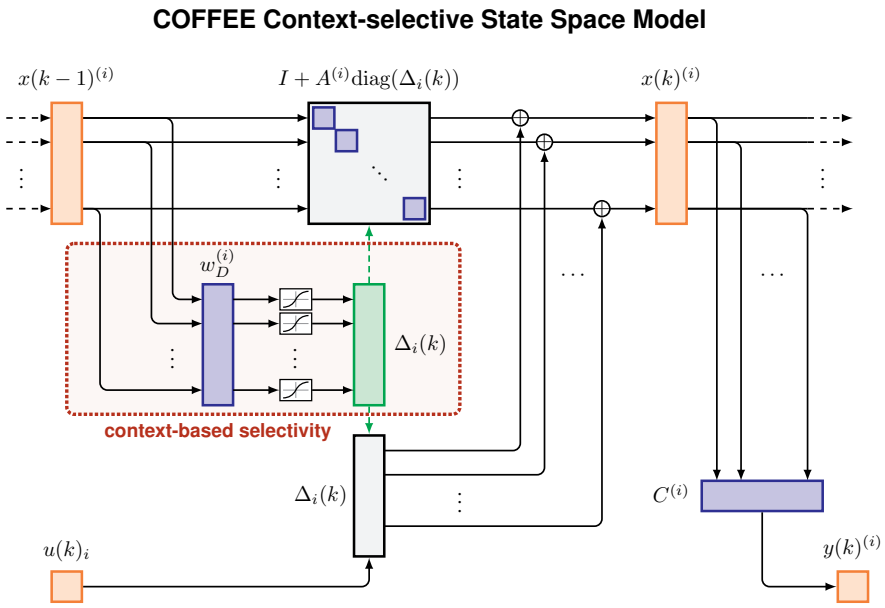


Figure 1: **Block diagram of the i-th feature subsytem of COFFEE.** Orange denotes the model signals, blue the learnable parameters, and green the selectivity or gating parameter, which is obtained from the past state through (5). By feeding this parameter back into the dynamics, the model forms a closed-loop mechanism that provides context-based selectivity, enabling identical tokens to be processed differently depending on their role in the sequence.

4 Experiments

In this section, we present numerical experiments on two primary tasks, the Induction Head (including variants) and MNIST. Implementation details are provided in Appendix G and Appendix E for the Induction Head, and in Appendix H for MNIST.

4.1 Results on the Induction Head Task

In the following, we present results from a series of experiments designed to solve the Induction Head Task, originally introduced in [34] and adopted as one of the motivating tasks in Mamba [20]. This task was proposed to evaluate in-context learning capabilities of modern deep learning architectures. An explanation of the induction head task is reported in Appendix D. In what follows, the adopted vocabulary is $V = \{1, 2, 3, 4, 5, 6, 7\}$.

Let us start by comparing the performance on the induction head task between the COFFEE model and S6. We test the two models on the IH task with $L_{\text{seq}} = 16$, $L_{\text{tri}} = 1$ and $L_{\text{tar}} = 1$, where L_{seq}, L_{tri} and L_{tar} are the lengths of the whole sequence, the trigger and the target, respectively. The results are reported in Table 1.

Table 1: Comparison between S6 and COFFEE on the IH task. Different learning rate were adopted for both S6 and COFFEE, the best ones are reported.

Model	n (state dimension)	η (learning rate)	D (embedding dimension)	Loss	Accuracy
S6	8	0.003	16	0.852	0.68
COFFEE	8	0.01	16	0.000	0.99+

At this point, the two models have exactly the same number of parameters (i.e. #parameters=784). Our model is able to reach $0.99+$ accuracy in just one epoch, namely with only 5'120'000 training sequences. The S6 performance is obtained by running the training algorithm for 100 epochs, namely 512'000'000 training sequences.

To test the limits of COFFEE we add an extra difficulty by reducing the state and embedding dimensions. Given the performance of S6 in the previous simplified scenario, we do not provide a comparison with COFFEE in more complicated versions of the IH task reported in Table 2.

Table 2: COFFEE model performance with reduced state and embedding dimensions

Model	n (state dimension)	η (learning rate)	D (embedding dimension)	Loss	Accuracy
COFFEE	1	0.01	9	0.000	0.99+

Our model has only 99 learnable parameters. The results are obtained within seven epochs, leading to 35'840'000 training sequences.

To further assess the capabilities of our model, we test it on a variation of the induction head task. The standard implementation of the induction head task, as explained in Appendix D, assumes a sequence with the following structure:

$$\text{noise} \parallel \text{trigger} \parallel \text{target} \parallel \text{noise} \parallel \text{trigger}$$

where $\parallel$ denotes sequence concatenation. We modify the above structure in:

$$\text{noise} \parallel \text{trigger} \parallel \text{noise} \parallel \text{target} \parallel \text{noise} \parallel \text{trigger}$$

We evaluate our model in this scenario with $L_{\text{seq}} = 16$, $L_{\text{tri}} = 1$ and $L_{\text{tar}} = 1$ and we obtain the results in Table 3.

Table 3: COFFEE model performance on this modified version of the IH task

Model	n (state dimension)	η (learning rate)	D (embedding dimension)	L_{noise} (noise length)	Loss	Accuracy
COFFEE	8	0.003	16	1	0.000	0.99+
COFFEE	8	0.003	16	2	0.002	0.99+

In the above table, "noise length" denotes the length of the noise subsequence between the trigger and target subsequences.

We now test our model on the induction head task with $L_{\text{seq}} = 16$, $L_{\text{tri}} = 1$ and $L_{\text{tar}} = 2$. Results are reported in Table 4.

Table 4: COFFEE model performance with $L_{\text{tri}} = 1$ and $L_{\text{tar}} = 2$

Model	n (state dimension)	η (learning rate)	D (embedding dimension)	Loss	Accuracy
COFFEE	8	0.01	16	0.004	0.99

Similarly, with $L_{\text{seq}} = 16$, $L_{\text{tri}} = 2$ and $L_{\text{tar}} = 1$ we obtain the results in Table 5.

Table 5: COFFEE model performance with $L_{\text{tri}} = 2$ and $L_{\text{tar}} = 1$

Model	n (state dimension)	η (learning rate)	D (embedding dimension)	Loss	Accuracy
COFFEE	8	0.003	16	0.660	0.78

To conclude, we test our model on the induction head task with $L_{\text{seq}} = 32, 64, 128, 256$, $L_{\text{tri}} = 1$ and $L_{\text{tar}} = 1$. The results and settings are reported in Table 6.

Table 6: COFFEE model performance with increased sequence length

Model	L_{seq} (sequence length)	n (state dimension)	η (learning rate)	D (embedding dimension)	Loss	Accuracy
COFFEE	32	8	0.01	16	0.000	0.99+
COFFEE	64	8	0.01	16	0.001	0.99+
COFFEE	128	8	0.01	16	0.000	0.99+
COFFEE	256	8	0.01	16	0.001	0.99+

Considering that S6 is the first SSM to introduce selectivity and is regarded as the state of the art, we conclude that our model deserves attention and further investigation. Based on the results above, COFFEE appears to be significantly more effective compared to the S6 model.

4.2 Results on the MNIST task

To further assess our model ability to capture dependencies between tokens, we tested our SSM on the MNIST dataset. The adopted architecture and the hyperparameters setting are fully explained in Appendix H. In Table 7 we report a summary of the results obtained: in the table, "COFFEE with output filtering" refers to the usual COFFEE model improved with filtering capabilities on the output (see Remark 3).

Table 7: Performance comparison between COFFEE and S6 using the MNIST dataset

Model	Epochs	n (state dimension)	# parameters	Loss	Accuracy
S6	100	2	5585	1.891	28.2%
S6	100	16	10085	1.876	28.6%
COFFEE	100	2	3385	0.107	96.6%
COFFEE with output filtering	100	2	3585	0.100	97.0%

This experiment demonstrates the practical applicability of COFFEE in a scenario that is very different from those considered in the previous section. As discussed in [20], selectivity is a fundamental property of powerful architectures for sequential modeling. In the previous section we established the relevance and efficiency of the selection mechanism introduced in COFFEE for tasks in the induction head family; with the results summarized in Table 7, we show how this embedded mechanism transfers to a completely different and more complex task, without incurring a prohibitive increase in model size.

In summary, we demonstrate that our model is flexible and able to outperform the current S6 model in significantly different tasks: for these reasons, we believe it deserves consideration as a building block in more complex architectures, such as Mamba [20].

5 Insights on the functioning principles of COFFEE

One of the major challenges in deep learning is related to the mechanistic interpretability of the learned models, the latter being essential towards the safe and responsible application of these tools in critical domains. In order to understand how these models encode and process information, the first crucial step is to gain awareness of the internal mechanisms and limitations of trained mathematical models.

Our COFFEE model has been designed with a few basic, interpretable functioning principles in mind:

- **P1:** The state represents the memory of the system, which is propagated to the next iteration by the state matrix.
- **P2:** The update gates $\Delta_i(k)$ select if and how much of the input $u(k)_i$ is transferred to the state dynamics and hence to memory.
- **P3:** Different areas of the state space allow the next encoded symbol to affect its future evolution differently (context selectivity).

Next, we investigate whether the proposed model is indeed capable of operating by exploiting the previous rules. An in-depth analysis, aimed at identifying the key principle at play, is possible only by considering small models with

limited degrees of freedom and simple tasks. Even the simple architecture that solved the inductive head task, with only *ninety-nine* parameters (see Section 4), turned out to be impractical to study due to the large dimensionality of the space and the lack of convenient visualization of the state and model parameter dynamics. A further recurring difficulty is presented by the fact that a model that is general enough to solve a wide spectrum of tasks can, in general, solve a given problem in many different ways, using inequivalent emerging "strategies".

We therefore decided to reduce the "complexity" of the induction head task as much as possible, so that we could geometrically represent the evolution of the state and more easily identify the choices of parameters that can implement the principles described above. In this way, we can verify that the learned solution is indeed behaving as expected. The simplified inductive head task (IH0, described in detail in Section F.1) considers sequences of four symbols chosen in $V = \{1, 2, 3\}$, where 1 represents the trigger. The trigger appears two times in each sequence, one of which as the last symbol. To solve this problem, we chose embeddings in $\mathbb{R}^2$, and a COFFEE model composed of two one-dimensional SSM, so that the evolution can be depicted in $\mathbb{R}^2$ as well.

In order to highlight the role of the functioning principles **P1-3** above, we introduce further model simplifications:

- Principle **P1** is implemented by choosing the state matrix to be the identity (i.e. all $\lambda^{(i)} = 0$ in Equation 7.), thus implementing a perfect memory. Furthermore, we also assume $C^{(i)} = 1$ for $i = 1, 2$, so the state values correspond to the outputs, and simplify interpretation.
- To implement **P2**, the update gates $\Delta_i(k)$ are chosen as functions of the corresponding state features: Given their definition in (5), the sigmoid functions play a key role. By choosing $w_D^{(i)} = 1$ in our reduced model, the state value remains the only relevant parameter. High positive values of the state components correspond to update gates close to one, and negative values bring the update gates close to zero.
- Regarding **P3**, in the simplified model the state belongs to $\mathbb{R}^2$, and its components directly activate the corresponding sigmoids. Loosely speaking, a state in the upper half-plane will cause the next input to "save" its second component (feature) in memory, and the right half-plane will memorize the first one. The first quadrant will save both, and the third quadrant neither.

With these simplifications in place, the only parameters left to learn are the embeddings. Taking into account the previous observations and the fact that the inductive head task consists of memorizing the symbol following the first trigger, we can intuitively guess the general form of a successful embedding. In particular, the **trigger symbol** embedding shall move the state in a region of activation of the sigmoid that is high and neutral with respect to the embedding of the target symbols. Natural candidates are vectors on the bisector of the first quadrant. The embeddings of the **target symbols** shall (i) be well distanced, in order to ensure distinguishability of the output; (ii) move the state from the triggered position to an area that is closer to their position than the others, so that their symbol is chosen as the final output; (iii) lower the activation of the sigmoid, and hence the update gates, so that further inputs will not cause the state to move too much towards the embedding of different symbols. Natural candidates for the simplified problems are large vectors in the third quadrant.

We would like to verify that the learned solutions for the IH task respect these principles. Once all the proposed model simplifications are implemented, the training phase does not always end successfully due to the extremely limited number of parameters to be optimized (6, corresponding to the embedding), and a suitable initialization must be considered. With this in mind, we trained the simplified model starting from a tentative embedding that satisfies the design principles sketched above, where:

$$\begin{aligned} 1 &\rightarrow [6, 6]^\top \\ 2 &\rightarrow [-10, -1]^\top \\ 3 &\rightarrow [-1, -10]^\top. \end{aligned}$$

From this initial condition, the learning phase converges extremely quickly, and the learned model solves the task with 100% accuracy. The learned solution corresponds to the embedding:

$$\begin{aligned} 1 &\rightarrow [5.394, 5.3433]^\top \\ 2 &\rightarrow [-10.2643, -1.5753]^\top \\ 3 &\rightarrow [-1.5395, -10.3404]^\top. \end{aligned}$$

The full trajectories of the state variables, as they react to input and context, are depicted in Figure 2 in the Appendix. Their behavior is as expected, in accordance with principles **P1-3**.

In light of this, the question "Is the automatically learned solution exploiting the same mechanism and degrees of freedom we describe in **P1-2-3**?" can be answered positively, at least in the case of the simplified model composed of

perfect memories (integrators). In the general case, the introduction of nontrivial state dynamics induces a drift field which further complicates the analysis, and feedback matrices w_D influence nontrivially the geometry of the activation areas, and yet a general agreement with the proposed functioning principles can be seen, albeit not as clearly.

6 Conclusion

In this work we propose a new context-selective state-space model (COFFEE) in which a state-feedback mechanism is designed to obtain a significant advantage in the model expressivity with a low number of trainable parameters while remaining suitable for parallel implementation. In the tests conducted, the COFFEE model largely outperforms S6 in both the induction head and the MNIST when the two are used within the same simple architecture. So far, COFFEE is still a proof of principle, since its performance in more complex scenarios (as, for example, the long-range arena) has not yet been tested. Of course, these scenarios require a Mamba-like architecture with several COFFEE modules. Nonetheless, the accuracy that is achieved with a single COFFEE module and a limited set of parameters is remarkable.

We are currently working on incorporating the COFFEE module in a suitable architecture, and compare it with Mamba and other, non state-space models on more challenging and diverse benchmarks. The numerical feasibility and efficiency of such implementation is granted by the chosen structure of COFFEE: since ensures a diagonal Jacobian, it remains amenable to parallelization (using the techniques recently proposed in [33]).

References

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186, 2019.

[3] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[4] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[5] Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

[6] Tao Ge, Si-Qing Chen, and Furu Wei. Edgeformer: a parameter-efficient transformer for on-device seq2seq generation. *arXiv preprint arXiv:2202.07959*, 2022.

[7] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

[8] Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, 33:17283–17297, 2020.

[9] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.

[10] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020.

[11] Jonathan Ho, Nal Kalchbrenner, Dirk Weissenborn, and Tim Salimans. Axial attention in multidimensional transformers. *arXiv preprint arXiv:1912.12180*, 2019.

[12] Rudolf E Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45, 1960.

[13] Thomas Kailath. *Linear Systems*. Prentice-Hall, 1980.

[14] Mihajlo D. Mesarović. Systems theory and biology—view of a theoretician. In M. D. Mesarović, editor, *Systems Theory and Biology*, pages 59–87, Berlin, Heidelberg, 1968. Springer Berlin Heidelberg.

[15] Geoffrey M. Hodgson. Economics and systems theory. *Journal of Economic Studies*, 14(4):65–86, 04 1987.

[16] Anders Lindquist and Giorgio Picci. *Linear Stochastic Systems: A Geometric Approach to Modeling, Estimation and Identification*. Springer Berlin, Heidelberg, 2015.

[17] Jimmy TH Smith, Andrew Warrington, and Scott W Linderman. Simplified state space layers for sequence modeling. *arXiv preprint arXiv:2208.04933*, 2022.

[18] Ankit Gupta, Albert Gu, and Jonathan Berant. Diagonal state spaces are as effective as structured state spaces. *Advances in neural information processing systems*, 35:22982–22994, 2022.

[19] Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*, 2021.

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

[21] 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*, pages 26670–26698. PMLR, 2023.

[22] Ramin Hasani, Mathias Lechner, Tsun-Hsuan Wang, Makram Chahine, Alexander Amini, and Daniela Rus. Liquid structural state-space models. In *International Conference on Learning Representations*, 2023.

[23] Tri Dao and Albert Gu. Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality. In *International Conference on Machine Learning*, 2024.

[24] Luca Zancato, Arjun Seshadri, Yonatan Dukler, Aditya Golatkar, Yantao Shen, Benjamin Bowman, Matthew Trager, Alessandro Achille, and Stefano Soatto. B'MOJO: Hybrid state space realizations of foundation models with eidetic and fading memory. In *Advances in Neural Information Processing Systems*, 2024.

[25] Jindong Jiang, Fei Deng, Gautam Singh, Minseung Lee, and Sungjin Ahn. Slot state space models. *Advances in Neural Information Processing Systems*, 37:11602–11633, 2024.

[26] Nicola Muca Cirone, Antonio Orvieto, Benjamin Walker, Cristopher Salvi, and Terry Lyons. Theoretical foundations of deep selective state-space models. *Advances in Neural Information Processing Systems*, 37:127226–127272, 2024.

[27] Rom Parnichkun, Stefano Massaroli, Alessandro Moro, Jimmy T.H. Smith, Ramin Hasani, Mathias Lechner, Qi An, Christopher Re, Hajime Asama, Stefano Ermon, Taiji Suzuki, Michael Poli, and Atsushi Yamashita. State-free inference of state-space models: The *Transfer function* approach. In *International Conference on Machine Learning*, 2024.

[28] T. Konstantin Rusch and Daniela Rus. Oscillatory state-space models. In *International Conference on Learning Representations*, 2025.

[29] Naoki Nishikawa and Taiji Suzuki. State space models are provably comparable to transformers in dynamic token selection. In *International Conference on Learning Representations*, 2025.

[30] Borjan Geshkovski, Cyril Letrouit, Yury Polyanskiy, and Philippe Rigollet. A mathematical perspective on transformers. *Bulletin of the American Mathematical Society*, 62(3):427–479, 2025.

[31] Álvaro Rodríguez Abella, João Pedro Silvestre, and Paulo Tabuada. The asymptotic behavior of attention in transformers. *arXiv preprint arXiv:2412.02682*, 2024.

[32] Carmen Amo Alonso, Jerome Sieber, and Melanie N Zeilinger. State space models as foundation models: A control theoretic overview. In *2025 American Control Conference (ACC)*, pages 146–153, 2025.

[33] Xavier Gonzalez, Andrew Warrington, Jimmy T Smith, and Scott W Linderman. Towards scalable and stable parallelization of nonlinear rnns. *Advances in Neural Information Processing Systems*, 37:5817–5849, 2024.

[34] Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, et al. In-context learning and induction heads. *arXiv preprint arXiv:2209.11895*, 2022.

[35] Diederik P Kingma and Jimmy Ba. A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 1412(6), 2014.

[36] Albert Gu, Tri Dao, Stefano Ermon, Atri Rudra, and Christopher Ré. Hippo: Recurrent memory with optimal polynomial projections. *Advances in neural information processing systems*, 33:1474–1487, 2020.

[37] Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. Mnist handwritten digit database. `http://yann.lecun.com/exdb/mnist/`, 1998. Accessed: 2025-08-12.

[38] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

[39] Faris Badawi, Anders Lindquist, and Michele Pavon. A stochastic realization approach to the smoothing problem. *IEEE Transactions on Automatic Control*, 24(6):878–888, 1979.

[40] Augusto Ferrante and Giorgio Picci. Minimal realization and dynamic properties of optimal smoothers. *IEEE Transactions on Automatic Control*, 45(11):2028–2046, 2000.

[41] Chai Wah Wu. Prodsumnet: reducing model parameters in deep neural networks via product-of-sums matrix decompositions. *arXiv preprint arXiv:1809.02209*, 2018.

A Useful Concepts of State Space Models

In the following, a brief introduction to **state space models** is provided. The specific outline is given below:

1. Modeling approaches for dynamical systems
2. Formal definitions and change of bases on linear time invariant state space models.
3. Discretization.

A.1 Modeling Approaches

Mathematical models for dynamical systems have been studied for centuries. Two common approaches to modeling the relation between input and output of such systems are:

1. **Input-output view**: In this approach, the system is directly characterized by its input-output relation. The latter is typically described by means of a (high order) differential equation.
2. **State-space view**: This approach introduces an auxiliary variable to model the system: the **state**. The latter provide a complete description of the configuration of the system at each time instant. As a consequence, the state at present time contains all the information about the past of the system that is relevant for its future evolution. The input-output view is still present, but now the state acts as a "mediator" between the input and the output.

State space models are widely used in numerous fields, including mathematics, engineering, economics, and biology, and have recently been employed in the deep learning field to maintain a compressed representation of the context provided by previously seen tokens, where the tokens take on the role of the inputs.

From now on, we focus only on state space models in which the state is defined over a finite-dimensional vector space.

A.2 Definitions and changes of basis

In this section, we aim to provide a brief introduction to linear state-space models. Both continuous- and discrete-time state-space models are defined along with a relation between the two. We also define the important property of algebraic equivalence.

Definition 1. A *linear, continuous-time state space model* is a representation of a system in which the input $u(t)$ and the output $y(t)$, $t \in \mathbb{R}$, are continuous-time signals and their relation is given by:

$$\begin{cases} \dot{x}(t) = A(t)x(t) + B(t)u(t) \\ y(t) = C(t)x(t) \end{cases} \tag{8}$$

where the continuous-time, vector-valued signal $x(t) \in \mathbb{R}^n$ is called *state* of the model. The matrix $A(t) \in \mathbb{R}^{n \times n}$ is called *state transition matrix*, $B(t) \in \mathbb{R}^{n \times m}$ is called *input* matrix (or input to state matrix) and $C(t) \in \mathbb{R}^{p \times n}$ is called *output* matrix (or state to output matrix).

A *linear, discrete-time state space model* is a representation of a system in which the input $u(k)$ and the output $y(k)$, $k \in \mathbb{Z}$, are discrete-time signals and their relation is given by:

$$\begin{cases} x(k) = A(k)x(k-1) + B(k)u(k) \\ y(k) = C(k)x(k) \end{cases} \tag{9}$$

Again the discrete-time, vector-valued signal $x(k) \in \mathbb{R}^n$ is called *state* of the model. Matrix dimensions and names are the same as those used for the model in (8).

We refer to n as the state space dimension, while m and p are referred to as the input and output space dimensions, respectively. The first equation in (8) is a first order differential equation (or a set of first order differential equations for $n > 1$) describing how the state evolves over time. Once the state evolution is determined, the second equation in (8) yields the evolution of the variables of interest modeled by $y(t)$. Linear state-space models are often represented as triples in the form $\Sigma = (A(k), B(k), C(k))$ ($\Sigma = (A(t), B(t), C(t))$ for continuous-time models).
The interpretation of the equations in (9) is with the difference that, given the discrete-time nature, the state evolution must obey a difference equation (or a set of difference equations).
If the matrices A, B, C are independent of t (or of k in discrete-time), the model is said to be *time-invariant*.

There is an inherent redundancy in the state space description because we can arbitrarily select the basis in the state space $\mathbb{R}^n$ and this choice does not affect the input-output relation of the model. As a consequence, there are infinitely many

state space models corresponding to the same input/output dynamics. Two linear models[4] $\Sigma_1 = (A_1(k), B_1(k), C_1(k))$ and $\Sigma_2 = (A_2(k), B_2(k), C_2(k))$ differing only for the choice of basis in the state space are said to be *algebraically equivalent*. A simple and direct computation shows that, in this case, there exists a non-singular matrix $T \in \mathbb{R}^{n\times n}$ such that $A_2(k) = T^{-1}A_1(k)T$, $B_2(k) = T^{-1}B_1(k)$ and $C_2(k) = C_1(k)T$. The matrix T induces the change of basis and is such that the states $x_1(t)$ of Σ_1 and $x_2(t)$ of Σ_2 are related by $x_1(t) = Tx_2(t)$.

We can also perform a change of basis in the input or in the output space. Of course, these changes affect the input-output relationship. However, in the setting of this work, this is not a problem. In fact, the input sequence is made up of tokens whose embedding in $\mathbb{R}^m$ is learned so that it can absorb the selection of the basis. The latter transforms the model by only changing $B(k)$ to $B(k)V$ where $V \in \mathbb{R}^{m\times m}$ is the (non-singular) matrix inducing the change of basis in the input space. A similar argument holds for the change of basis in the output space which transforms the model by only changing $C(k)$ to $L^{-1}C(k)$ where $L \in \mathbb{R}^{p\times p}$ is the (non-singular) matrix inducing the change of basis in the output space.

A.3 Discretization

Consider a continuous-time state space model $\Sigma_c = (A(t), B(t), C(t))$ whose input $u(t)$ is piece-wise constant on intervals of the same length T, so that for all $t_1, t_2 \in [kT, (k+1)T)$, with $k \in \mathbb{Z}$, $u(t_1) = u(t_2)$. In this case, we can concentrate the input information into a new discrete-time signal $u_d(k) := u(kT)$ and by integrating the first of Equations (8) for each interval $[kT, (k+1)T)$, we obtain a discrete-time model where the discrete state and output are obtained by sampling the original state and output at the instants kT: $x_d(k) := x(kT)$, $y_d(k) := y(kT)$.
In the case of a time-invariant continuous-time models, we can easily derive closed-form expressions for the discrete-time model $\Sigma_d = (A_d, B_d, C_d)$ which is time-invariant as well:

$$\begin{cases} A_d = e^{AT} \\ B_d = \displaystyle\int_0^T e^{A\tau}Bd\tau = A^{-1}\left(e^{AT} - I\right)B \\ C_d = C \end{cases} \tag{10}$$

where the last equality in the expression of B_d holds in the case of non-singular state matrix A.

B The S6 state space model

SSMs have recently been explored as efficient sequence models, motivated by their ability to capture long-range dependencies with favorable scaling, and as a structured alternative to attention [32, 19, 18, 17, 21]. Early work focused on linear time-invariant (LTI) models, which are computationally efficient but limited in expressivity: their fixed recurrence forces every token to influence the state, preventing the model from ignoring irrelevant inputs.

The S6 model [20] addresses this by introducing linear time-varying (LTV) dynamics. The state matrices $(A(k), B(k), C(k))$ depend on the input embedding, enabling token-dependent gating of information. This design, motivated by tasks such as *induction head* and *selective copy*, allows the model to update memory selectively. Unlike LTI SSMs, the time-varying recurrence cannot be expressed as a convolution, precluding fast FFT-based implementations. To remain efficient, [20] proposed a hardware-aware parallel algorithm that minimizes GPU memory traffic, enabling scalable training and inference.

To describe the model more concretely, consider its input representation. S6 operates on a multi-dimensional array $[B, L, D]$, with batch size B, sequence length L, and embedding dimension D. Each $L \times D$ matrix is an embedded sequence, where columns correspond to embedding features. The model processes these features independently by instantiating D parallel SISO SSMs: the i-th SSM receives the i-th feature across all tokens. Each feature thus maintains its own state, while the recurrences are coupled through shared token-dependent matrices $B(t), C(t)$ and adaptive step sizes $\Delta_i(t)$, enabling selective memory updates.

Formally, for each feature $i = 1, \ldots, D$, the model defines a SISO state space system:

$$\Sigma^{(i)} = \begin{cases} \dot{x}(t)^{(i)} = A^{(i)}x(t)^{(i)} + B(t)u(t)_i \\ y(t)^{(i)} = C(t)x(t)^{(i)} \end{cases}$$

where

[4] Here we refer to discrete-time models but everything can be repeated *verbatim* for the continuous-time case.

- $x(t)^{(i)} \in \mathbb{R}^n$ is the hidden state associated with feature i. The dimension n is shared across features and treated as a tunable hyperparameter.
- $u(t)_i$ denotes the i-th coordinate of the token embedding $u(t) \in \mathbb{R}^D$, which acts as the scalar input for the i-th SSM.
- $A^{(i)} \in \mathbb{R}^{n\times n}$ is the transition matrix, chosen diagonal with learnable entries, and independently parameterized for each feature.
- $B(t), C(t) \in \mathbb{R}^n$ are input- and output-mapping vectors, shared across features but dependent on the entire token embedding:
$$B(t) = W_B u(t)^\top, \qquad C(t) = (W_C u(t)^\top)^\top,$$
with $W_B, W_C \in \mathbb{R}^{n\times D}$ learnable. This design makes the recurrence token-dependent, allowing the model to gate information based on content.
- $y(t)^{(i)}$ represents the feature-level output at time t.

Note that this is a continuous-time formulation. In practice, the typical sequence inputs, such as text, are inherently discrete. A standard way to bridge this gap is to view $u(t)$ as a piecewise constant signal derived from the discrete embeddings, and then apply a discretization procedure. In S6, the chosen approach is zero-order hold (ZOH), which admits closed-form discrete-time matrices (see Sec. A.3).

In the S6 model, discretization is applied independently to each feature-specific system $\Sigma^{(i)}$. Importantly, the sampling interval is not fixed but **input-dependent**: for each token $u(t)$, the model defines a vector of adaptive step sizes

$$\Delta(t) = \zeta(W_D u(t)^\top) \in \mathbb{R}^D, \tag{11}$$

where $W_D \in \mathbb{R}^{D\times D}$ is learnable and the softplus $\zeta(\cdot)$ ensures positivity. Each $\Delta_i(t)$ controls the discretization of the i-th feature's SSM. Rather than being literally interpreted as different sampling times, these values can be understood as feature-specific gates that balance the influence of the past state and the current input when forming the update.

The resulting discrete-time system for feature i is

$$\Sigma^{(i)} = \begin{cases} x(k)^{(i)} = e^{A^{(i)}\Delta_i(k)} x(k-1)^{(i)} + \left(A^{(i)}\right)^{-1}\left(e^{A^{(i)}\Delta_i(k)} - I\right) B(k) u(k)_i \\ y(k)^{(i)} = C(k) x(k)^{(i)} \end{cases} \tag{12}$$

In S6, $A^{(i)}$ is chosen diagonal with entries $\lambda_j^{(i)}$ and the previous expression simplifies to

$$\Sigma^{(i)} = \begin{cases} x(k)^{(i)} = \begin{bmatrix} e^{\lambda_0^{(i)}\Delta_i(k)} & & \\ & \ddots & \\ & & e^{\lambda_{n-1}^{(i)}\Delta_i(k)} \end{bmatrix} x(k-1)^{(i)} + \begin{bmatrix} \frac{(e^{\lambda_0^{(i)}\Delta_i(k)}-1)B_0(k)}{\lambda_0^{(i)}} \\ \vdots \\ \frac{(e^{\lambda_{n-1}^{(i)}\Delta_i(k)}-1)B_{n-1}(k)}{\lambda_{n-1}^{(i)}} \end{bmatrix} u(k)_i \\ y(k)^{(i)} = [C_0(k) \quad \cdots \quad C_{n-1}(k)]\, x(k)^{(i)} \end{cases} \tag{13}$$

The S6 parameterization is designed to enable input-dependent selectivity. The input and output mappings are token-dependent,

$$B(k) = W_B u(k)^\top, \qquad C(k) = (W_C u(k)^\top)^\top,$$

which allows the model to decide at each step whether to admit information into the state or propagate it to the output [20].

Selectivity is further controlled by the adaptive step sizes :

$$\Delta(k) = \zeta(W_D u(k)^\top), \qquad k \in \mathbb{Z}.$$

In the discrete update (Eq. (13)) if the eigenvalues $\lambda_j^{(i)}$ are in the left half of the complex plane (that is, they correspond to stable modes in continuous time), $\Delta_i(k)$ interpolates between two extremes:

- $\Delta_i(k) \to \infty$: past context is forgotten, the state depends only on $u(k)_i$;
- $\Delta_i(k) \to 0$: the input is ignored, the state is preserved.

This mechanism provides fine-grained control over memory and update.

Finally, stability is ensured by constraining the eigenvalues of $A^{(i)}$ to be negative via

$$\lambda_j^{(i)} = -e^{\mu_j^{(i)}},$$

with $\mu_j^{(i)}$ unconstrained.

C Extended Derivations for COFFEE

Our formulation builds on the S6 model (Appendix B). The derivation proceeds in three steps:

1. Linearize the exponential in Equation (12) via a first-order Taylor expansion to clarify the role of $\Delta(t)$ and to enhance the mechanistic interpretability.
2. Incorporate state feedback to enable context-dependent selectivity.
3. Remove parameter redundancy through changes of basis in the state and input spaces.

C.1 Simplified Dynamics via Linearization

Expanding the exponential in Equation (12) around $\Delta_i(k) = 0$ gives

$$\Sigma^{(i)} = \begin{cases} x(k)^{(i)} = \big(I + A^{(i)}\Delta_i(k)\big)x(k-1)^{(i)} + \big(A^{(i)}\big)^{-1}\big(I + A^{(i)}\Delta_i(k) - I\big)\, B(k)u(k)_i, \\ y(k)^{(i)} = C(k)x(k)^{(i)}, \end{cases}$$

which simplifies to

$$\Sigma^{(i)} = \begin{cases} x(k)^{(i)} = \big(I + A^{(i)}\Delta_i(k)\big)x(k-1)^{(i)} + \Delta_i(k)B(k)u(k)_i, \\ y(k)^{(i)} = C(k)x(k)^{(i)}. \end{cases} \tag{14}$$

In S6, the matrices $B(k), C(k)$ are shared across all subsystems and depend on the input embedding. However, after discretization, the effective input-to-state mapping:

$$\big(A^{(i)}\big)^{-1}\big(e^{A^{(i)}\Delta_i(k)} - I\big)B(k),$$

involves $A^{(i)}$, making it inherently feature-specific. This motivates assigning each subsystem its own time-invariant matrices $B^{(i)}$ and $C^{(i)}$. We therefore equip each subsystem with feature-specific $B^{(i)}$ and $C^{(i)}$, both time-invariant.[5] This yields:

$$\Sigma^{(i)} = \begin{cases} x(k)^{(i)} = \big(I + A^{(i)}\Delta_i(k)\big)x(k-1)^{(i)} + \Delta_i(k)B^{(i)}u(k)_i \\ y(k)^{(i)} = C^{(i)}x(k)^{(i)}, \end{cases} \tag{15}$$

where $B^{(i)} \in \mathbb{R}^n$ is a column vector, $C^{(i)} \in \mathbb{R}^n$ a row vector, and $A^{(i)} \in \mathbb{R}^{n\times n}$ diagonal. Since (15) is derived via Taylor expansion, the model is accurate if the $\Delta_i(k)$ remain small. Following S6, we set

$$\Delta(k) = \sigma(W_D u(k)^\top), \quad W_D \in \mathbb{R}^{D\times D}.$$

Parameter count. The resulting model has $3nD + D^2$ parameters:

- nD from $\{B^{(i)}\}$,
- nD from $\{C^{(i)}\}$,
- nD from the diagonals of $\{A^{(i)}\}$,
- D^2 from W_D.

This matches S6, which has nD parameters in each of W_B and W_C, nD for $\{A^{(i)}\}$, and D^2 for W_D, again totaling $3nD + D^2$. We remark that these counts refer only to the SSM block. The full architecture (presented in Appendix E) includes an additional $|M|D$ parameters from the embedding matrix, for a total of $3nD + D^2 + |M|D$, where M is the vocabulary size.

C.2 Feedback for Context-Based Selectivity

Our model introduces context-based selectivity by letting the update gates depend on the hidden state, rather than the raw input token. In S6, gating is computed as:

$$\Delta(k) = \zeta(W_D u(k)^\top), \qquad W_D \in \mathbb{R}^{D\times D}.$$

[5] Time variation arises solely through $\Delta_i(k)$.

where we recall that $\zeta(\cdot)$ is the softplus function. In contrast, we replace this token-based rule with a state-feedback law:

$$\Delta_i(k) = \sigma\big(w_D^{(i)} \odot x(k-1)^{(i)}\big), \qquad w_D^{(i)} \in \mathbb{R}^n, \tag{16}$$

where gating is conditioned on the compressed context encoded in the hidden state of each subsystem $\Sigma^{(i)}$. In addition to introducing context-dependent selectivity, this change reduces the gating parameters from D^2 in S6 to Dn. Since typically $n \ll D$, this corresponds to a considerable reduction in the number of gating parameters. The state-dependent gating $\Delta_i(k)$ is incorporated into the subsystem dynamics (15) as

$$\Sigma^{(i)} = \begin{cases} x(k)^{(i)} = (I + A^{(i)}\mathrm{diag}(\Delta_i(k)))\, x(k-1)^{(i)} + \mathrm{diag}(\Delta_i(k))B^{(i)}\, u(k)_i, \\ y(k)^{(i)} = C^{(i)} x(k)^{(i)}. \end{cases} \tag{17}$$

C.3 Eliminating Redundancy via Change of Basis

The number of parameters in (17) can be reduced through a suitable change of basis. This is formalized in Proposition 1 in the main text; here we provide the proof.

Proof of Proposition 1. Let j be the index of the embedding vector $v^{(j)}$ whose entries $v_i^{(j)}$ are all nonzero, and consider the family of systems in Equation (17). We rewrite this family as

$$\begin{cases} x(k)^{(i)} = \big(I + A^{(i)}\mathrm{diag}(\Delta_i(k)))\big)\, x(k-1)^{(i)} + \mathrm{diag}(\Delta_i(k)))[B^{(i)}v_i^{(j)}][u(k)_i / v_i^{(j)}], \\ y(k)^{(i)} = C^{(i)} x(k)^{(i)}. \end{cases} \tag{18}$$

It is therefore apparent that the j-th embedding vector can be normalized to $\mathbf{1}$ a the price of rescaling each $B^{(i)}$ by multiplying each of its entries by $v_i^{(j)}$. This is indeed the simplest example of change of basis in the input space which is simply $\mathbb{R}$.

Now we are ready for a change of basis on the state space which does not affect the input/output behavior of the state model. We select the change of basis induced by the matrix $T_i := \mathrm{diag}\big(B_0^{(i)}v_i^{(j)}, \ldots, B_{n-1}^{(i)}v_i^{(j)}\big)$, with $B_k^{(i)}$ being the k-th entry of the vector $B^{(i)}$. This transformation does not affect the state transition matrix but normalizes the input matrix to $\mathrm{diag}(\Delta_i(k)))\mathbf{1} = \Delta_i(k)$. Finally, after the change of basis, the output matrix becomes $(B^{(i)})^\top v_i^{(j)} \odot C^{(i)}$, where $\odot$ denotes the Hadamard product. □

Notice that assuming that each entry of $B^{(i)}$ is non-zero is very reasonable as if one entry, say the k-th, is zero it means that the k-th component of the state is not affected by the input and hence it can be eliminated without affecting the input/output relationship of the model. The fact that at least one of the embeddings has all entries which are non-zero is a very reasonable assumption, as the embedding parameters are learned.

In summary, the change-of-basis argument reveals two sources of redundancy:
(i) One embedding vector can be fixed arbitrarily (e.g., to $\mathbf{1}$) without altering the input-output behavior, which reduces the embedding parameters from $|M|D$ to $(|M|-1)D$;
(ii) $B^{(i)}$ and $C^{(i)}$ need not be learned independently: any rescaling of $B^{(i)}$ can be absorbed into $C^{(i)}$, so only their product matters. One set can therefore be fixed, saving nD parameters.

After applying all the steps described above, the resulting COFFEE dynamics is given in Equation (7) of the main text. The overall parameter count decreases from $3nD + D^2 + |M|D$ in S6 to $3nD + (|M|-1)D$ in our model.

D Induction Heads Task

The term "induction heads" is used by [34] to denote a type of attention heads in the Transformers architecture. These heads learn the ability to complete a sequence by memorizing the symbol (or symbols) following a certain subsequence called "trigger" and by producing such a symbol whenever the trigger appears again. In [34], strong evidence is presented that supports the assumption that this ability might be one of the fundamental mechanisms for "in-context learning". For this reason, such an ability is a benchmark to assess to what extent the models are capable of identifying patterns within sequences.

D.1 Task description

Consider a sequence of symbols extracted from a certain finite vocabulary V. In our setting, we select

$$V = \{1, 2, 3, 4, 5, 6, 7\}$$

To test the model's ability to capture patterns, the model is presented with a sequence of arbitrary length that follows the structure outlined below:

$$\text{noise 1} \parallel \text{trigger} \parallel \text{target} \parallel \text{noise 2} \parallel \text{trigger} \tag{19}$$

where $\parallel$ denotes sequence concatenation, and noise 1, noise 2, trigger, and target indicate subsequences of symbols. Each sequence is constructed in such a way that the same trigger subsequence appears exactly twice. The two noise subsequences may contain the target subsequence, but they must not include the trigger subsequence. The trigger subsequence serves to indicate the beginning of a pattern to be reproduced. Upon encountering the trigger subsequence, the model is expected to recognize that it must "pay attention" to the subsequent elements, called *target* subsequence, and retain this information. While the trigger subsequence is fixed, the target subsequence varies across sequences. We consider the task successfully completed when the model outputs the target subsequence after observing the final trigger subsequence in Equation (19). This outcome would demonstrate that the model has correctly learned the pattern "trigger $\parallel$ target".

When the sequence length is fixed, the following steps are necessary to build an appropriate sequence:

1. Choose the length of the trigger subsequence
2. Choose a fixed trigger subsequence with symbols from V
3. Choose and fix the length of the target subsequence

Remark 4. Fixed trigger subsequence means that once it is selected (e.g. randomly), it must be used consistently to generate all training, validation, and test sequences. In other words, in all training, validation and test sequences, the trigger subsequence is always the same in both of its occurrences in Equation (19).

Let us denote by L_{seq}, L_{tri}, and L_{tar} the lengths of the sequence, trigger, and target subsequences, respectively. These lengths should be chosen such that:

$$L_{\text{seq}} - 2L_{\text{tri}} - L_{\text{tar}} \geq 1$$

to ensure that there is at least one of the two "noise" subsequences, consisting of at least one element. In our experiments, we considered sequences constructed according to the following combinations:

1. **Trigger** and **target** subsequences both of length one
2. **Trigger** subsequence of length two and **target** of length one
3. **Target** subsequence of length two and **trigger** of length one

Once the trigger and the target length are fixed, to generate admissible sequences we randomly select the position of the initial symbol of the first "trigger" subsequence, the target subsequence and the two noise subsequences. We use suitable uniform discrete independent random variables for all these selections with the caution of discarding sequences in which at least one of the target and the noise subsequences contains the trigger.

Remark 5. Whenever the length of the target subsequence is greater than 1, extra caution is required. In fact, in this case, when the model encounters the last trigger subsequence, it can output only one symbol and cannot complete the target subsequence. To address this issue, we pad the input sequence by concatenating it with the padding sequence made of $L_{\text{tar}} - 1$ copies of the special symbol "0". For example, an admissible sequence with $L_{\text{tri}} = 3$ and $L_{\text{tar}} = 3$ could be:

$$\underbrace{3,2,6}_{\text{noise}},\underbrace{5,6,7}_{\text{trigger}}\underbrace{2,4,3}_{\text{target}},\underbrace{1,2,2,6}_{\text{noise}},\underbrace{5,6,7}_{\text{trigger}},\underbrace{0,0}_{\text{padding}}$$

In this way, the model can observe the entire trigger subsequence and produce the complete target subsequence. Upon encountering the last symbol, "7", the correct first output would be "2", and the subsequent zeros would allow the model to also output the rest of the target subsequence, i.e.,"4" and "3".

All the generated input sequences are transformed, by the usual embedding process, into sequences of vectors. The latter are used as inputs to a state-space model. Specifically, given a linear discrete-time state-space model

$$\begin{cases} x(k) = Ax(k-1) + Bu(k) \\ y(k) = Cx(k) + Du(k) \end{cases}$$

embedding vectors of symbols are the various $u(k)$ for $k = 0, ..., L_{\text{seq}} - 1$. For example, given the sequence:

$$\underbrace{3,2,6}_{\text{noise}},\underbrace{5,6,7}_{\text{trigger}}\underbrace{2,4,3}_{\text{target}},\underbrace{1,2,2,6}_{\text{noise}},\underbrace{5,6,7}_{\text{trigger}},\underbrace{0,0}_{\text{padding}}$$

$u(0)$ is the embedding of the symbol "3", $u(1)$ is the embedding of the symbol "2", and so on.

We finally emphasize that this task can be defined over any set V. In our case, we used $V = \{1, 2, 3, 4, 5, 6, 7\}$, but one could choose a larger set or a vocabulary consisting of different symbols, such as $V = \{$"cat", "dog", ...$\}$. The elements of the vocabulary are symbols, and the relevant thing is to highlight their mutual relations.

E Core Module Architecture

We now provide a description of the core module architecture employed to address the induction head task. A detailed account of all steps involved in data processing is presented. The operations performed by the core module in our setup are as follows:

1. The core module receives a sequence or a batch of sequences as input. Each sequence is built according to the rules presented in Appendix D. The input can be regarded as an array of dimension [B, L], where B denotes the batch size and L the length of each sequence within the batch. [6]
2. Each symbol is mapped to a real-valued vector, referred to as the *embedding vector*. The dimensionality of the embedding space is a user-defined hyperparameter and can be adjusted according to the specific requirements of the task. Formally, the embedding procedure consists in applying the following **learned** map:

   $$\text{emb}() : M \to \mathbb{R}^D$$

   where M is the model's vocabulary while D denotes the dimension of the embedding space and sometimes is also called *model dimension*.

   Remark 6. Notice that, the domain of the emb() map is M. Since we assume that $V \subseteq M$ (V is the vocabulary used to build sequences), this transformation is well-defined: it guarantees that every symbol in any input sequence has a corresponding embedding vector.

   After applying this mapping, we obtain a three-dimensional array with shape [B, L, D], where D denotes the embedding dimensionality.
3. The resulting sequence of embedding vectors is fed into a state space model, which contains additional learnable parameters.
4. For each input (i.e. the embedding vector of a symbol) the model outputs another vector of the same dimension, specifically a vector in $\mathbb{R}^D$. After processing all input sequences, the state space model produces a multidimensional array of shape [B, L, D]. For each batch index $b = 0, \ldots, B-1$, this corresponds to an $L \times D$ matrix containing all output vectors associated with the input sequence.
5. In general, the outputs produced by the state space model do not correspond exactly to the embedding vectors associated with symbols in M. However, the ultimate objective is for the model to generate symbols from the set M. To achieve this, the first step involves computing the distance between each output vector and every embedding vector corresponding to symbols in M. This operation results in a multidimensional array of shape [B, L, $|M|$].

   After having evaluated the distances we have:

   $$\text{Distances} = \begin{bmatrix} \begin{bmatrix} d^{(0)}_{0,0} & \cdots & d^{(0)}_{0,|M|-1} \\ \vdots & \ddots & \vdots \\ d^{(0)}_{L-1,0} & \cdots & d^{(0)}_{L-1,|M|-1} \end{bmatrix} \\ \vdots \\ \begin{bmatrix} d^{(B-1)}_{0,0} & \cdots & d^{(B-1)}_{0,|M|-1} \\ \vdots & \ddots & \vdots \\ d^{(B-1)}_{L-1,0} & \cdots & d^{(B-1)}_{L-1,|M|-1} \end{bmatrix} \end{bmatrix}$$

   where $d^{(b)}_{l,m}$ denotes the distance between the output vector at time l of batch b and the embedding vector of the symbol $m \in M$, for every $l = 0, \ldots, L-1$, $b = 0, \ldots, B-1$, and $m \in M$.

[6] More generally, we use the notation [d_1,..., d_l] to indicate an array having l dimensions, where the first dimension is d_1 and the l-th dimension is d_l

6. We now apply the softmin[7] function along the rows of the matrices within the multidimensional array $\mathrm{Distances}$. This operation produces another multidimensional array, denoted as $\mathrm{Softmin}$, which has the same dimensions as $\mathrm{Distances}$, namely $[B, L, |M|]$.
   By applying the softmin function to the rows of $\mathrm{Distances}$ array, we obtain a probability distribution where the smallest $d_{l,m}^{(b)}$ for every $l = 0, ..., L-1$, $b = 0, ..., B-1$ and $m \in M$ has the maximum probability.

   The resulting matrix after this step is:

$$\mathrm{Softmin} = \begin{bmatrix} \begin{bmatrix} s_{0,0}^{(0)} & \cdots & s_{0,|M|-1}^{(0)} \\ \vdots & \ddots & \vdots \\ s_{L-1,0}^{(0)} & \cdots & s_{L-1,|M|-1}^{(0)} \end{bmatrix} \\ \vdots \\ \begin{bmatrix} s_{0,0}^{(B-1)} & \cdots & s_{0,|M|-1}^{(B-1)} \\ \vdots & \ddots & \vdots \\ s_{L-1,0}^{(B-1)} & \cdots & s_{L-1,|M|-1}^{(B-1)} \end{bmatrix} \end{bmatrix}$$

7. After applying the softmin function, we then apply an element-wise transformation known as the logit[8] function. The logit function maps probabilities from the interval $(0, 1)$ to the entire real line $\mathbb{R}$.
   In this manner, we obtain another multidimensional array, which we denote as Logits. Each entry is given by $\mathrm{Logits}_{k,i,j} = \mathrm{logit}(\mathrm{Softmin}_{k,i,j})$. Naturally, the Logits array has the same dimensions as $\mathrm{Softmin}$, namely $[B, L, |M|]$. The resulting matrix after this step is:

$$\mathrm{Logits} = \begin{bmatrix} \begin{bmatrix} \mathrm{logit}_{0,0}^{(0)} & \cdots & \mathrm{logit}_{0,|M|-1}^{(0)} \\ \vdots & \ddots & \vdots \\ \mathrm{logit}_{L-1,0}^{(0)} & \cdots & \mathrm{logit}_{L-1,|M|-1}^{(0)} \end{bmatrix} \\ \vdots \\ \begin{bmatrix} \mathrm{logit}_{0,0}^{(B-1)} & \cdots & \mathrm{logit}_{0,|M|-1}^{(B-1)} \\ \vdots & \ddots & \vdots \\ \mathrm{logit}_{L-1,0}^{(B-1)} & \cdots & \mathrm{logit}_{L-1,|M|-1}^{(B-1)} \end{bmatrix} \end{bmatrix}$$

8. To conclude, the core module architecture produces as output two multidimensional arrays. The first is exactly Logits. The second is a multidimensional array of dimensions $[B, L]$, obtained by applying the $\arg\max$ along the last dimension of Logits. This means that we compute the $\arg\max$ over all rows of every matrix within the multidimensional array Logits. The resulting array of shape $[B, L]$ is denoted as $\mathrm{Predictions}$. Formally, this matrix is constructed as follows:

$$\mathrm{Predictions} = \begin{bmatrix} \arg\max_{m \in M} & \mathrm{logit}_{0,m}^{(0)} & \cdots & \arg\max_{m \in M} & \mathrm{logit}_{L-1,m}^{(0)} \\ & & \vdots & & \\ \arg\max_{m \in M} & \mathrm{logit}_{0,m}^{(B-1)} & \cdots & \arg\max_{m \in M} & \mathrm{logit}_{L-1,m}^{(B-1)} \end{bmatrix}$$

   This matrix, for each row (i.e. for each batch), contains the predicted symbols for all time steps $l = 0, \ldots, L-1$. Notice that, as a consequence of the previous operations, taking the $\arg\max$ corresponds to selecting, for

[7] Given a vector $x = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$, the softmin function produces another vector $\mathrm{softmin}(x) \in \mathbb{R}^n$ whose entries are defined by

$$\mathrm{softmin}(x)_i = \frac{e^{-x_i}}{\sum_{j=1}^{n} e^{-x_j}}, \quad i = 1, \ldots, n.$$

[8] The logit function is defined as the inverse of the sigmoid function. For a real number $p \in (0, 1)$, it is given by:

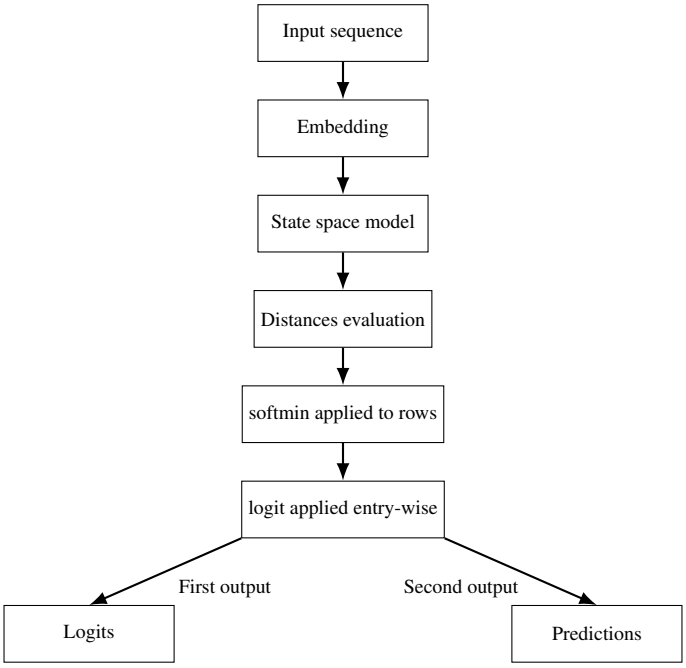
$$\mathrm{logit}(p) = \log\left(\frac{p}{1-p}\right)$$

every batch and every time step, the symbol whose embedding vector is closest (with respect to the Euclidean norm) to the output produced by the state space model.

Remark 7. The multidimensional array $\mathrm{Predictions}$ can be obtained in multiple ways. For each $l = 0, \dots, L-1$, the predicted symbol $m_{\text{pred}} \in M$ is the one closest to the output vector of the state-space model. Therefore, taking the $\arg\min$ over the rows of the matrices in the $\mathrm{Distances}$ array, or the $\arg\max$ over the rows of the matrices in the $\mathrm{Softmin}$ or Logits arrays, will yield the same $\mathrm{Predictions}$ matrix.

We use cross-entropy loss as the loss function, which requires the logits as input. Therefore, the Logits array is used for computing the loss, while the $\mathrm{Predictions}$ array is used to evaluate the accuracy.

The core module architecture is summarized in the following block diagram:



F Towards a mechanistic interpretation of COFFEE: auxiliary material

F.1 Simplified Version of the Induction Head Task: IH0

To take a first step toward a mechanistic interpretability we simplify the induction head task. The solution of the problem by hand presents several challenges related to the embedding and state dimension, the sequence length, and the number of symbols in the vocabulary.

For this reason, we focus on addressing a toy version of the induction head task. We set:

1. Embedding dimension $D = 2$ and state dimension $n = 1$
2. Sequence length $L_{\text{seq}} = 4$
3. Vocabulary made of three symbols $V = \{1, 2, 3\}$

An embedding in $\mathbb{R}^2$ allows us to employ only two one-dimensional state-space models. It is hence possible to visualize the trajectories of both states on a Cartesian plane. The choice of setting $L_{\text{seq}} = 4$ and using a vocabulary consisting of only three symbols, significantly reduced the number of possible sequences that could be generated according to the rules described in Appendix D.

By specilizing COFFEE as in Equation (7) for $D = 2$ and $n = 1$, we obtain the following setting:

$$\Sigma^{(i)} = \begin{cases} x(k)^{(i)} = (1 + \lambda^{(i)}\Delta_i(k))x(k-1)^{(i)} + \Delta_i(k)u(k)_i \\ y(k)^{(i)} = C^{(i)}x(k)^{(i)} \end{cases} \tag{20}$$

with:

$$\Delta_i(k) = \sigma(W_D^{(i)} x(k-1)^{(i)})$$

with $i = 0, 1$; $x(k)^{(i)} \in \mathbb{R}$ and $W_D^{(i)} \in \mathbb{R}$. The possible sequences are now of the form:

$$\text{trigger} \parallel \text{target} \parallel \text{noise} \parallel \text{trigger} \quad \text{or} \quad \text{noise} \parallel \text{trigger} \parallel \text{target} \parallel \text{trigger}$$

where, the trigger, target and noise subsequences are all of length one, resulting in a total sequence length of $L_{\text{seq}} = 4$. With these choices, the number of possible input sequences is reduced to just 8, allowing us to check the validity of a proposed model by inspection.

F.2 COFFEE simplified model and state evolution for IH0

We illustrate how the reduced COFFEE model, introduced to illustrate the functioning principles of the method, is able to solve IH0 by depicting the learned encoded symbols and the state evolution corresponding to a given input sequences. By setting $\lambda^{(i)} = 0$ and $C^{(i)} = 1$ for all $i = 0, 1$, each **state** $x(k)^{(i)}$ becomes equal to the corresponding output $y(k)^{(i)}$, and the system behaves as a perfect **integrator**. This is desirable to our aim since it implments a perfect dynamical memory in absence of inputs. The model thus becomes:

$$\Sigma^{(i)} = \begin{cases} x(k)^{(i)} = x(k-1)^{(i)} + \Delta_i(k) u(k)_i \\ y(k)^{(i)} = x(k)^{(i)} \end{cases} \tag{21}$$

with:

$$\Delta_i(k) = \sigma(W_D^{(i)} x(k-1)^{(i)})$$

where $x(k)^{(i)} \in \mathbb{R}$ and $W_D^{(i)} \in \mathbb{R}$ for $i = 0, 1$. We set $W_D^{(i)} = 1$ for $i = 0, 1$, in order to allow for more direct interpretability of the effect of state feedback in this setting.

The only parameters to be learned at this point are embedding vectors for the symbols in the vocabulary $V = \{1, 2, 3\}$. According to the principles **P1-P3** described in Section 5, we start with the embedding parameters initialization:

$$\begin{aligned} 1 &\rightarrow [6, 6]^T \\ 2 &\rightarrow [-10, -1]^T \\ 3 &\rightarrow [-1, -10]^T. \end{aligned}$$

Recall that, without loss of generality, symbol "1" is assumed to be the trigger. The final, learned solution is:

$$\begin{aligned} 1 &\rightarrow [5.394, 5.343]^T \\ 2 &\rightarrow [-10.264, -1.575]^T \\ 3 &\rightarrow [-1.539, -10.340]^T. \end{aligned}$$

Such solution can provably solve correctly the problem with 100% accuracy. This can be verified by inspection, since the possible input sequences are just 8.

In fact, with these parameter choices and the introduced simplifications, it is manageable to visualize how the possible input sequences are processed by the simplified model. All vectors are represented in R^2, and we use z_1, z_2 to denote the abscissa and ordinate, respectively (we avoid using x, y to avoid confusion with the state $x(k)$ and the output $y(k)$). Specifically, the outputs of $\Sigma^{(0)}$ generate the z_1-coordinates of the output vectors, while the outputs of $\Sigma^{(1)}$ generate the z_2-coordinates.

A picture illustrating how the input sequence "$1\|2\|3\|1$" is processed is provided next, which also illustrates the intuitive motivation for the particular choice of parameters. It is instructive to compare it with the trajectory associated to the sequence "$3\|1\|2\|1$", in order to see how the different positioning of the trigger with respect to the noise influences the dynamics.

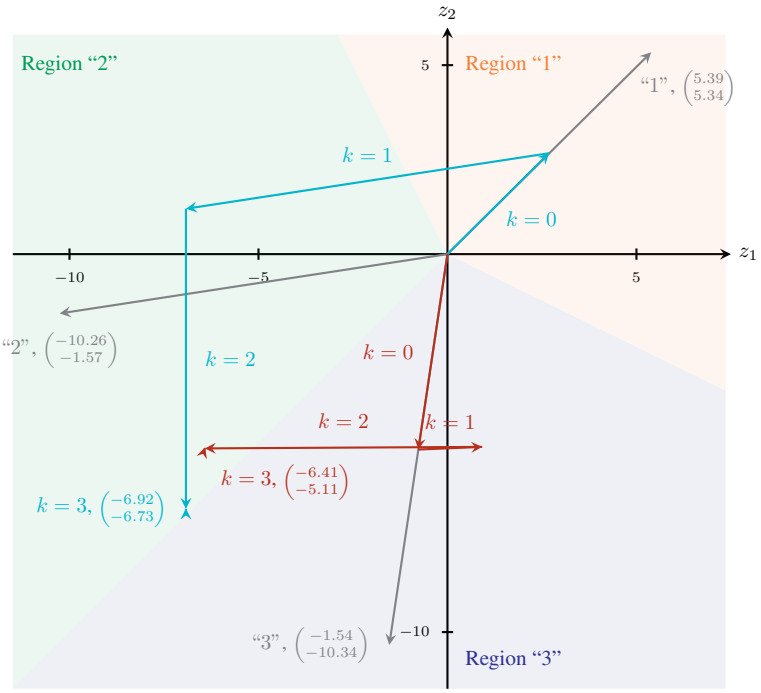


Figure 2: Embeddings (in grey) and trajectories of the state for the reduced model solving the IH0 problem, with input $1\|2\|3\|1$ and $3\|1\|2\|1$ (in blue and red, respectively). The last trigger does not move significantly the state since the sigmoids in the input gates are not activated. The colored regions indicate the sections of the plane corresponding to each possible choice of the final output symbol (e.g final states in Region 2 lead to output "2").

In the above diagram the **arrow heads** of the light blue vectors point to the output $y(k) \in \mathbb{R}^2$ (and also the state $x(k) \in \mathbb{R}^2$, because $y(k)^{(i)} = x(k)^{(i)}, i = 0, 1$) of the simplified SSM model at the time steps $k = 0, 1, 2, 3$), whose coordinates $\begin{pmatrix} z_1 \\ z_2 \end{pmatrix} \in \mathbb{R}^2$ are also provided. Vectors in gray are the embedding vectors of the symbols in V with their coordinates.

The qualitative behavior for the "$1\|2\|3\|1$" sequence is as expected: the trigger moves the state in an activation area; the target symbol moves it towards its own representation, where the gate activation is also lower; the subsequent symbols do not move the state outside of the correct decision area. For the other sequence, after the first noise symbol moves the state in an area of low activation for the second state variable, the dynamics plays as expected but mostly along the first variable (horizontally). All other sequences have similar behavior to the ones depicted.

G Results on the Induction Head Task (IH)

In the subsequent paragraphs, additional details on the training procedure for the induction head task are given. Moreover, initialization details are provided for both the COFFEE model and S6.

G.1 Experiment settings

We do not rely on a predefined dataset with a fixed set of sequences. For instance, by setting the sequence length to $L_{\text{seq}} = 16$ and defining the vocabulary as $V = \{1, 2, 3, 4, 5, 6, 7\}$, the total number of possible sequences that can be generated, according to the rules described in Appendix D, exceeds several hundred billion.

To address this, we developed a software tool capable of generating such sequences randomly on demand. We created two independent instances of this tool to generate training and validation sequences.

Although it is theoretically possible for the same sequence to appear in both the training and validation sets, this is extremely unlikely given that the total number of training sequences used is capped at 512 million.

As the loss function, we employed **cross-entropy**, and for optimization, we used the **Adam** optimizer [35]. The learning rate is specified for each experiment individually.

In each training iteration, we randomly generate and extract 512 sequences. This process is repeated 10'000 times, which we define as one **epoch**. The maximum number of epochs is set to 100. However, if good performance is achieved before reaching this limit, the training process is stopped early.

At the end of each epoch, we evaluate the model's generalization ability on a randomly generated set of 10'000 sequences, computing the **loss** and/or **accuracy**. This evaluation also allows us to save the model that performs best during training.

G.2 Models Initialization

In this section, we provide details about the initialization of our model and the S6 model used in our experiments.

Our model

The parameters of our model are initialized as follows:

1. Transition matrices $A^{(i)} \in \mathbb{R}^{n \times n}$. These matrices are diagonal, with entries initialized to 0;
2. State-to-output row vectors $C^{(i)} \in \mathbb{R}^n$, whose entries are initialized according to the standard normal distribution $\mathcal{N}(0, 1)$;
3. The vectors $W_D^{(i)} \in \mathbb{R}^n$, whose entries are also initialized according to the standard normal distribution $\mathcal{N}(0, 1)$.

Regarding the embedding matrix, we initialize it as follows:

1. We generate a random matrix $L \in \mathbb{R}^{D \times |M|}$ with entries sampled from a uniform distribution in the interval $[0, 1)$;
2. The matrix L is then decomposed using QR decomposition as $L = QR$, where $Q \in \mathbb{R}^{D \times |M|}$ has all the columns **orthogonal** to each other and of **norm** one;
3. Finally, we take $Q^T \in \mathbb{R}^{|M| \times D}$ and set it as the initial embedding matrix. This procedure yields an orthonormal initialization of the embedding vectors.

S6 model

The parameters in S6 are initialized as follows:

1. Transition matrices $A^{(i)} \in \mathbb{R}^{n \times n}$. These matrices are diagonal, with entries initialized according to HiPPo theory [36]. In particular, the i-th diagonal entry is initialized as $-(i + 1), i = 0, ..., n - 1$. For more details see [20], sec. 3.6 pag. 9;
2. The weight matrix $W_B \in \mathbb{R}^{n \times D}$ is initialized according to the standard normal distribution $\mathcal{N}(0, 1)$;
3. The weight matrix $W_C \in \mathbb{R}^{n \times D}$ is initialized according to the standard normal distribution $\mathcal{N}(0, 1)$;
4. The weight matrix $W_D \in \mathbb{R}^{D \times D}$ is initialized according to the standard normal distribution $\mathcal{N}(0, 1)$;
5. The embedding matrix in $\mathbb{R}^{|M| \times D}$, is initialized according to the standard normal distribution $\mathcal{N}(0, 1)$.

H MNIST Architecture and Results

In this section, we describe the experimental setup used to obtain the results in Section 4.2. Specifically, we (i) describe the MNIST dataset, (ii) a limitation of the sMNIST task in evaluating SSM performance, and (iii) present the employed architecture to address this classification task.

H.1 MNIST Dataset

The MNIST dataset [37] is a widely adopted benchmark in machine learning, particularly for image classification tasks. It comprises 70'000 grayscale images of handwritten digits from 0 to 9, each with a resolution of 28×28 pixels.

The dataset is split into a training set of 60'000 images and a test set of 10'000 images. Thanks to its simplicity and standardized format, MNIST serves as a common starting point for evaluating new models and algorithms.

In our experiments, we construct a validation set by randomly sampling 10'000 images from the original MNIST training set. Consequently, the dataset is partitioned into 50'000 training images, 10'000 validation images, and 10'000 test images. Moreover, we did not use the full 28×28 pixel images. Instead, we opted to crop them to 25×25 pixels.

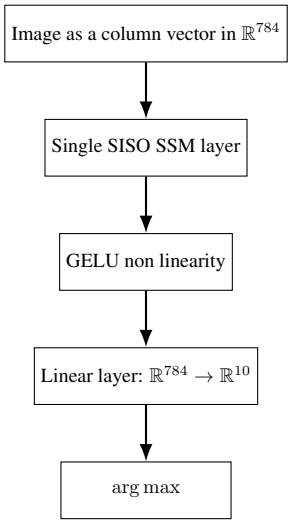
H.2 Architecture Description

In this section, we first discuss a limitation in assessing the true performance of an SSM layer in solving the **sMNIST** task, a variant commonly used to evaluate state-space models performance. We then present our architecture and its subsequent developments. Our approach allows for a more accurate evaluation of the effectiveness of state-space models.

H.2.1 Limitations of the sMNIST Task

A review of the literature shows that, in general, to assess the ability of state space models to capture long-range dependencies, they are tested on a modified version of the MNIST task called **sMNIST**, which stands for **sequential MNIST**. In this task, each 28×28 pixel image is vectorized. Specifically, the image is converted into a column vector in $\mathbb{R}^{784}$ by vertically stacking all columns of size $\mathbb{R}^{28}$, a method known as column-major order. Alternatively, the same result can be achieved by horizontally stacking all rows of the image to form a vector in $\mathbb{R}^{784}$, known as row-major order.

Drawing inspiration from the literature, we initially attempted to solve the **sMNIST** task using the following architecture:



With this architecture, we take as input the vectorized image in $\mathbb{R}^{784}$. The image, now represented as a sequence of pixels, is processed by an SSM layer parameterized with our model. A GELU [38] activation function is applied to each element of the resulting output sequence.[9] After the GELU, we still have a vector in $\mathbb{R}^{784}$, which is mapped to $\mathbb{R}^{10}$ by a linear layer. This linear layer produces the logits for each of the ten classes. The predicted class is the one with the highest logit, obtained by taking the $\arg\max$ over the entries of the resulting vector in $\mathbb{R}^{10}$. Specifically,

$$\hat{i} = \underset{i \in \{0,\dots,9\}}{\arg\max} \quad v_i$$

where with $\hat{i}$ we denote the predicted class and with v_i we denote the i-th entry of the vector $v \in \mathbb{R}^{10}$.

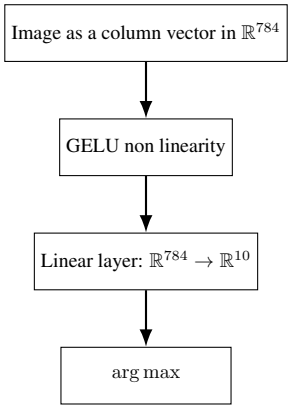
Each symbol in the input sequence represents, by construction, a pixel value. Consequently, each symbol is embedded as a scalar, resulting in a model dimension of $D = 1$. By training the above architecture with a single state space model (since $D = 1$) with:

1. State dimension $n = 8$;

[9] The Gaussian Error Linear Unit (GELU) activation function is defined as $\mathrm{GELU}(x) = x \cdot \Phi(x)$, where $\Phi(x)$ is the cumulative distribution function (CDF) of the standard normal distribution. For more details, see [38].

2. 100 epochs;
3. Batch size 512;
4. Using an initial learning rate of $\eta = 0.01$, which was reduced to $\eta = 0.005$ once the training loss fell below 0.150;
5. Data augmentation, using a **roto-translation** with a maximum rotation of 5 degrees and a maximum translation of 0.01 along both the x and y axes [10],

we obtained a **loss** of 0.226 and a **accuracy** of 93.6%. The employed loss function is the cross-entropy loss. To evaluate the contributions of the SSM layer and the linear layer to the observed performance, we trained the same architecture without the SSM layer, as illustrated in the following diagram:



we trained the architecture, again, with:

1. 100 epochs;
2. Batch size 512;
3. An initial learning rate of $\eta = 0.01$, which was reduced to $\eta = 0.005$ once the training loss fell below 0.150;
4. Data augmentation implemented with a **roto-translation** with a maximum rotation of 5 degrees and maximum translation of 0.01 on both x and y axes;

as a result, we obtained a **loss** of 0.284 and a **accuracy** of 92.0%.

These results suggest that in this configuration the contribution of the SSM layer to the performance is negligible compared to that of the linear layer.

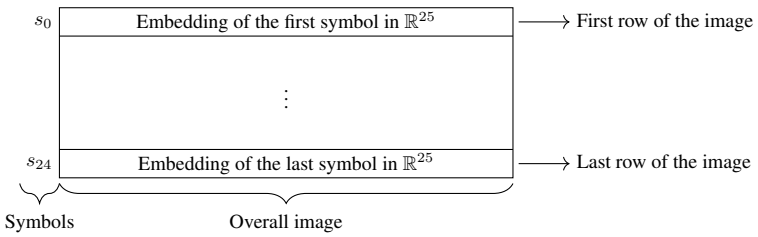
In light of this experiment, it becomes evident that, in order to properly assess the SSM's ability to solve the task, the use of linear layers should be minimized. Their strong effectiveness can obscure the contributions of state space models.

H.2.2 Our Architecture

Our model is designed as a general tool for sequence processing. In general, when dealing with sequences, we take a sequence of symbols from some vocabulary, associate an embedding vector to each symbol, and then process the sequence of embedding vectors with the model. For instance, this is done for the induction head task presented in Appendix D.

With the MNIST dataset, we want to adopt the same approach. As previously mentioned, we use cropped images of size 25×25 pixels. We treat an image as a sequence of **twenty-five** symbols (i.e. one for each row). For each symbol s_i, $i = 0, \dots, 24$, its embedding vector in $\mathbb{R}^{25}$ is given by the i-th row of the image. Thus, processing an image is equivalent to processing a sequence of twenty-five symbols embedded in $\mathbb{R}^{25}$. To clarify this idea, consider the following diagram.

[10] For instance, if we set translate=(a, b) (i.e. x, y translations respectively), then horizontal shift is randomly sampled in the range $[-\text{img}_{width} \cdot a, \text{img}_{width} \cdot a]$ and vertical shift is randomly sampled in the range $[-\text{img}_{height} \cdot b, \text{img}_{height} \cdot b]$.



Drawing inspiration from the celebrated two filter formula used for the smoothing problem, see [39] and [40], we can consider the image as a sequence of symbols $s_0 \| \dots \| s_{24}$, with each $s_i \in \mathbb{R}^{25}$, and build two sequences: one using the rows as embedding vectors and the other using the columns. Let $s_0 \| \dots \| s_{24}$ denote the sequence created from the rows and $s_0^{'} \| \dots \| s_{24}^{'}$ the one from the columns. The index i, $i = 0, \dots, 24$, can now be interpreted as **time**. By thinking of a sequence as a discrete time signal of finite support, given that we have access to all the signals in advance, we process the same image in four ways:

1. By rows from $i = 0$ to $i = 24$;
2. By columns from $i = 0$ to $i = 24$;
3. By rows from $i = 24$ to $i = 0$;
4. By columns from $i = 24$ to $i = 0$.

The architecture now becomes:

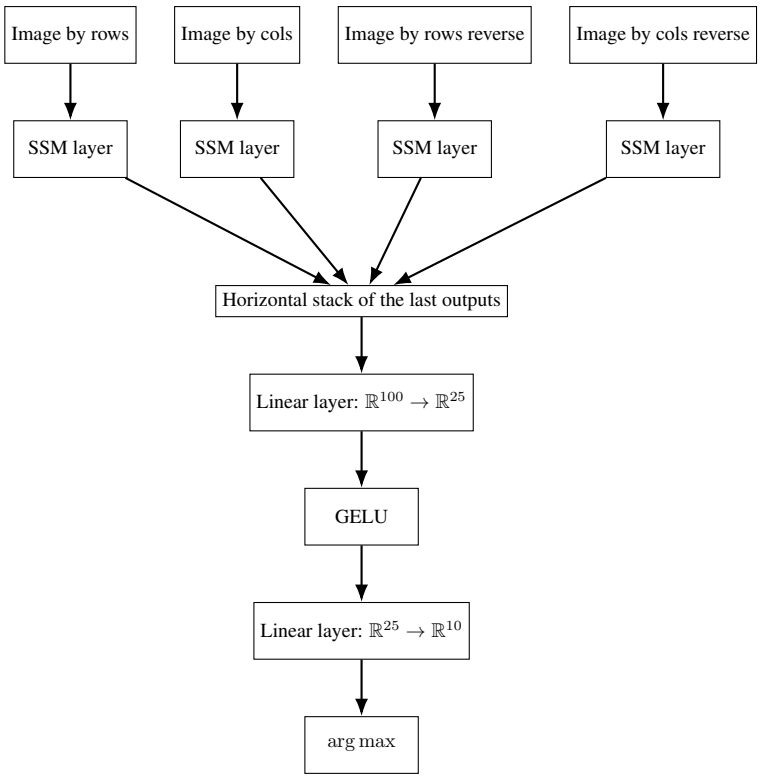


Figure 3: Proposed architecture.

In this setup, we process the images by rows, columns, and by rows and columns in reverse order. Each SSM layer produces an output matrix in $\mathbb{R}^{25\times25}$, from which we consider only the last row. We then take the last rows of all four matrices and stack them horizontally to obtain a vector in $\mathbb{R}^{100}$. This vector is mapped to the corresponding logits using two linear layers separated by a GELU activation function. The predicted class is again selected as:

$$\hat{i} = \underset{i\in\{0,\dots,9\}}{\arg\max} \quad v_i$$

where with $\hat{i}$ we denote the predicted class and with v_i we denote the i-th entry of the vector $v \in \mathbb{R}^{10}$.

Let us now analyze the number of learnable parameters in this architecture:

1. In Appendix C, we see that an SSM layer has $3nD$ parameters, with n being the state dimension and D the embedding dimension. In our case, we use $n = 2$ and $D = 25$, so a single SSM layer has 150 parameters. Since we have four of them, the total number of parameters for the SSM layers is 600;
2. The first linear layer, which maps vectors from $\mathbb{R}^{100}$ to $\mathbb{R}^{25}$, has $2500 + 25 = 2525$ parameters: 2500 for the linear transformation and 25 for the bias, resulting in an overall affine transformation;
3. The second linear layer, which maps vectors from $\mathbb{R}^{25}$ to $\mathbb{R}^{10}$, has $250 + 10 = 260$ parameters: 250 for the linear transformation and 10 for the bias, resulting in an overall affine transformation.

Taking all these contributions into account, the overall architecture has 3385 parameters.

We trained this model with:

1. State dimension $n = 2$;
2. 100 epochs;
3. Batch size 512;
4. An initial learning rate $\eta = 0.01$ that was reduced to $\eta = 0.005$ when the training loss fell below 0.450;
5. Data augmentation implemented with a roto-translation with a maximum angle of 5 degrees and maximum translation of 0.01 on both x and y axes.

With these parameters, we obtain a **loss** of 0.107 and a **accuracy** of 96.6%.

For comparison with the S6 model, we trained the same architecture using the S6 model within the SSM layers instead of our model. As mentioned in Appendix C, our model has fewer parameters than S6, so the total number of parameters in this case is 5885. Initially, we tried with the same settings, namely:

1. State dimension $n = 2$;
2. 100 epochs;
3. Batch size 512;
4. With an initial learning rate $\eta = 0.01$ that was reduced to $\eta = 0.005$ when the training loss fell below 0.450;
5. Data augmentation implemented with a roto-translation with a maximum angle of 5 degrees and maximum translation of 0.01 on both x and y axes,

and we obtained a **loss** of 1.891 with a **accuracy** of 28.2%.

We conducted a second experiment by increasing the state dimension for the S6 model. We trained the architecture with:

1. State dimension $n = 16$;
2. 100 epochs;
3. Batch size 512;
4. With an initial learning rate $\eta = 0.01$ that was reduced to $\eta = 0.005$ when the training loss fell below 0.450;
5. Data augmentation implemented with a roto-translation with a maximum angle of 5 degrees and maximum translation of 0.01 on both x and y axes.

In this case, the architecture has $10'085$ parameters, and we obtained a **loss** of 1.876 with a **accuracy** of 28.6%.

Based on these results, we can conclude that our model achieves far superior performance compared to the S6 model, and this performance is obtained with fewer parameters. Moreover, our model is also competitive with the highly optimized models presented in [41], even without specific optimization.