

Flash STU: Fast Spectral Transform Units

Y. Isabel Liu*

Princeton University

isabel.liu@princeton.edu

Windsor Nguyen*

Princeton University

mn4560@princeton.edu

Yagiz Devre

Princeton University

yagiz.devre@princeton.edu

Evan Dogariu

New York University

ed2719@nyu.edu

Anirudha Majumdar

Princeton University

ani.majumdar@princeton.edu

Elad Hazan

Princeton University

ehazan@princeton.edu

Abstract—Recent advances in state-space model architectures have shown great promise for efficient sequence modeling, but challenges remain in balancing computational efficiency with model expressiveness. We propose the Flash STU architecture, a hybrid model that interleaves spectral state space model layers with sliding window attention, enabling scalability to billions of parameters for language modeling while maintaining a near-linear time complexity. We evaluate the Flash STU and its variants on diverse sequence prediction tasks, including linear dynamical systems, robotics control, and language modeling. We find that, given a fixed parameter budget, the Flash STU architecture consistently outperforms the Transformer and other leading state-space models such as S4 and Mamba-2.

I. INTRODUCTION

Transformers have become the dominant architecture for sequence modeling due to their powerful self-attention mechanism, which allows them to capture complex dependencies across sequences [1]. This capability makes them particularly effective for tasks requiring long-range context and recall such as language modeling. However, Transformers suffer from quadratic computational complexity in sequence length, making them inefficient for processing extremely long sequences.

State Space Models (SSMs), in contrast, offer a more computationally efficient alternative by maintaining a fixed-size latent state that does not grow with sequence length [2]. This design enables SSMs to scale more efficiently for tasks requiring long-horizon memory while maintaining subquadratic complexity. Recent architectures, such as S4 [2] and Mamba [3], have demonstrated that SSMs can match or even outperform Transformers in certain structured sequence prediction tasks, particularly in domains like dynamical systems, robotics, audio modeling, and time-series forecasting. However, SSMs face limitations in capturing complex nonlinear dependencies and may struggle with tasks requiring exact retrieval of past inputs, such as copying mechanisms or highly structured symbolic reasoning [4].

Our work contributes to this discussion by building on the Spectral Transform Unit [5], henceforth referred to as the *STU*, a novel state-space-inspired model that integrates spectral filtering techniques [6] for robust and efficient sequence modeling. Unlike standard SSMs, STU employs fixed convolutional filters that do not need to be learned, derived from a robust theoretical foundation.

We introduce a new *hybrid* model architecture called **Flash STU**¹ which combines the STU and sliding window attention [7]. We observe that Flash STU achieves superior performance over both Transformers and other leading SSM variants on sequence prediction tasks across multiple modalities, including natural language, robotics, and synthetic dynamical systems. We also investigate the optimization landscape of STU in comparison to other models and find that STU models are easier to optimize and provide more stability during training.

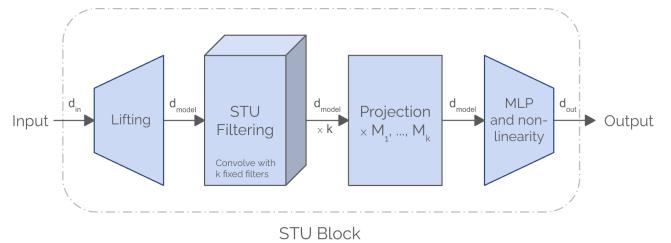


Fig. 1: Basic architecture of the Spectral Transformer Unit.

A. Learning in linear dynamical systems

Many modern neural architectures for sequence modeling can be understood through the lens of learning in *linear dynamical systems (LDS)*, a fundamental class of models in control theory, time-series forecasting, and machine learning. LDSs are defined by the state-space equations:

$$\begin{aligned} x_{t+1} &= Ax_t + Bu_t + w_t \\ y_t &= Cx_t + Du_t + \zeta_t \end{aligned}$$

where $x_t \in \mathbb{R}^{d_{\text{hidden}}}$ is the hidden state, $u_t \in \mathbb{R}^{d_{\text{in}}}$ the input, and $y_t \in \mathbb{R}^{d_{\text{out}}}$ the observed output. Matrices A, B, C, D parameterize the system, and w_t, ζ_t represent noise terms.

Expanding these recursive equations in the noiseless case (assuming $D = 0$), we obtain:

$$y_t = \sum_{i=1}^t CA^i Bu_{t-i}$$

which explicitly shows how the system's past inputs contribute to future outputs. The spectral radius $\rho(A)$, which measures

¹Our optimized LLM pretraining code is open sourced at <https://github.com/hazan-lab/flash-stu/>.

*Equal contribution. Order determined alphabetically by last name.

the largest absolute eigenvalue of A , determines how much past information is retained. If $\rho(A) \approx 1$, the system has long memory, but this also makes learning difficult due to slow decay in dependencies.

Challenges in learning LDS. The task of learning in LDS involves predicting future observations given past inputs. This is particularly challenging when the system is *marginally stable* (i.e., $\rho(A) \approx 1$), as long memory effects require learning methods that scale efficiently.

Several classical approaches have been proposed:

- **System Identification:** Estimates A, B, C, D directly but is non-convex and unreliable for marginally stable systems. A notable exception under certain generative and noise assumptions was given by [8].
- **Autoregressive Learning:** Models LDS behavior but requires many parameters for long-memory systems. The autoregressive coefficients can be learned over the past inputs (open loop) [9], or over the past observations (closed loop) [10], or both. For a thorough discussion of the advantages/disadvantages of these methods see [11].
- **Kalman Filtering:** Provides optimal state estimation under Gaussian noise assumptions but is sensitive to adversarial settings and instability [12], [13].

Each of these methods faces trade-offs between efficiency, accuracy, and robustness.

A more recent perspective on learning in LDS involves *spectral filtering* [6], [14], which focuses on spectral properties of the transition matrix A rather than direct estimation of system parameters. The core idea is to represent sequence dynamics using *fixed convolutional filters* derived from dominant spectral components of the system.

This approach has several advantages:

- 1) **Bypasses direct parameter estimation** – Unlike system identification, spectral filtering does not require recovering A, B, C, D , avoiding issues with non-convex optimization. Further, the theoretical properties of spectral filtering suggests that the number of parameters do not scale with the hidden dimension of the system, or the dimension of the transition matrix A .
- 2) **Handles marginal stability** – Spectral filtering is capable to learn marginal stable systems with symmetric transition matrices, namely: it remains effective even when $\rho(A) \approx 1$. This has the intuitive capability of learning in the presence of long context, a theoretical advantage that we verify experimentally henceforth.
- 3) **Computational efficiency** – Instead of maintaining a full latent state or learning a long Markov operator, spectral filtering converts the input sequence to a compressed form. As a convolutional model, it can be implemented using the fast Fourier transform algorithm for computing convolutions, enabling subquadratic complexity.

The *Spectral Transform Unit (STU)* extends spectral filtering techniques to sequence modeling. Instead of learning full transition matrices, STUs apply fixed spectral filters derived from the dominant eigenvectors of a Hankel matrix, ensuring

that past information is captured efficiently. This enables stable long-range sequence modeling with significantly lower computational cost compared to Transformers and traditional state-space models.

B. Description of the Spectral Transform Unit

The STU architecture is depicted in Figure 1. Given an input sequence, STU transforms it through the following steps: first, the input may be lifted to a higher-dimensional space via a learned transformation. Then, a convolution operation is applied using a set of **fixed** filters that do not require training, followed by a learned transformation that projects the filtered sequence to the output space. Finally, an optional nonlinearity is applied.

Mathematically, the output is given by:

$$\hat{y}_t = \sigma \left(\sum_{i=1}^k M_i \cdot \langle \Phi_i, u_{t:t-L} \rangle \right), \quad (1)$$

where M_i are learned projections, σ is a nonlinearity, and $\Phi_{1:k}$ are fixed convolutional filters that can be precomputed. The filters $\Phi_{1:k}$ correspond to the eigenvectors of the largest eigenvalues of the Hankel matrix:

$$Z = \int_{\alpha=0}^1 \mu_\alpha \mu_\alpha^\top d\alpha, \quad \mu_\alpha = (1, \alpha, \dots, \alpha^L). \quad (2)$$

These filters can be computed **a priori** and stored before observing the actual sequence, making STU particularly efficient.

The mathematical properties that underline spectral filtering are derived from the following remarkable fact about Hankel matrices: the spectrum of any Hankel matrix over the real numbers decays exponentially [15]. This fact was used by [6] to design a filtering basis that is very sparse in terms of the number of filters needed. The residual error, due to this Hankel property, is exponentially small. More details about the spectral filters are described in Appendix A.

A key optimization technique is the tensordot approximation (**STU-T**) [5], which decomposes the projection tensors M_i into two smaller matrices:

$$M_i \approx M_i^1 \times M_i^2, \quad M_i^1 \in \mathbb{R}^{d_{\text{model}} \times k}, \quad M_i^2 \in \mathbb{R}^{d_{\text{model}} \times d_{\text{out}}} \quad (3)$$

This approximation reduces computational complexity by approximately a factor of k , saving memory and leading to significant improvements in efficiency. Although the expressivity of the model is slightly reduced under this regime, we show that STU-T retains competitive empirical performance, making it a practical alternative for scaling spectral state-space models efficiently.

II. EXPERIMENTS WITH SYNTHETIC DATA

We begin our investigation of STU’s properties with some simple yet representative synthetic tasks that have become commonplace in the sequence modeling literature. In particular, we aim to understand the behavior of STU in environments with long memory and nonlinearities, especially as we introduce feed-forward layers and deeper architectures. We compare against S4 [2] and Mamba-2 [4], standard SSM

architectures, as well as the vanilla transformer layer [1]. Note that Mamba-2 contains a nonlinear selection mechanism that can be beneficial in certain synthetic tasks at the cost of extra complexity and the loss of the convolutional inference mode. Nonlinear gating can also be added to STU layers for use in larger models, as done in Section III.

A. Linear dynamical systems

Linear dynamical systems were presented in the previous sections. The task of sequence prediction in this case is to predict \hat{y}_{t+1} given all the previous inputs and outputs $u_{1:t}, y_{1:t}$ and to minimize the loss vs. the ground truth output y_{t+1} . We evaluate the mean squared error of STU’s predictions vs. other methods where the sequence was generated by a linear dynamical system.

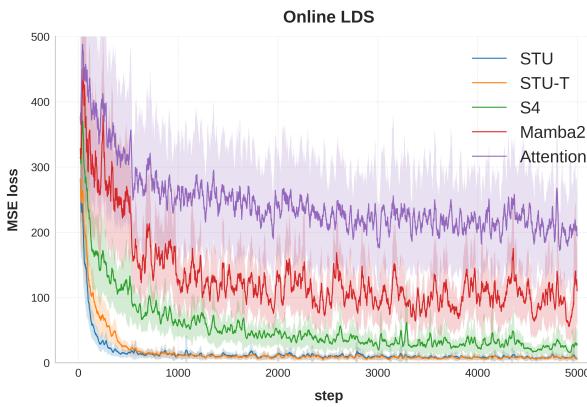


Fig. 2: Mean squared error $\|\hat{y}_{t+1} - y_{t+1}\|^2$ of the different layers on a single sequence from an LDS.

Experiment details and conclusions. We apply random inputs to linear dynamical systems with random system matrices with $d_{\text{input}} = d_{\text{output}} = 5$ and hidden state dimension $d_{\text{hidden}} = 256$. The transition matrix A is symmetrized and normalized so that $\rho(A) = 0.99$ to give the system an effective memory of ≈ 100 , which is what we set the context length to. Such a setting has the properties of (1) long memory and (2) a large hidden state, which we believe to be major sources of complexity in real-world applications.

We compare against a standard attention layer, with 8 heads and the ALiBi [16] attention score modification, a diagonal S4 layer [2] with hidden dimension equal to d_{hidden} , and a Mamba-2 layer (also with hidden dimension d_{hidden}) [4].

All layers have a width of 32 and are trained for 5,000 steps with the RMSProp optimizer. Results are plotted in Figure 2 with error bars over 16 trials.

We can see that the STU layer is a powerful and robust SSM primitive. Both the vanilla STU and its approximate version STU-T are able to reliably achieve small loss in this setting with quick and robust convergence, while the performance of the other methods vary across random seeds. Note that the width of the STU layer does not need to be as large as d_{hidden} to perfectly capture the dynamics, which is consistent with theory. Furthermore, STU-T approximation is

able to roughly match vanilla STU’s performance, even on these multi-input multi-output (MIMO) systems.

B. Optimization behavior

We saw in the linear dynamical system experiment that the STU layers seem to have a comparatively easier time optimizing. This is expected since an STU layer under MSE loss is a convex parameterization, whereas the losses of the other models are non-convex². Following [17], we choose two random directions in the high-dimensional space, move along these directions by varying amounts x_step and y_step , compute the loss of the model with perturbed parameters for each coordinate pair in these directions, and plot the loss values as heights on a 2D grid. By doing so, we are able to get a sense for the local geometry of a reduced-dimension loss landscape.

Figures 3-6 visualize these loss landscapes for STU, S4, Mamba-2, and attention layers, respectively, after 10 steps of training on the LDS. Figure 16 in Appendix B instead visualize the local geometry through curvature in terms of Hessian eigenvalue ratios. Flatter minima are preferable since it has been proposed that reducing sharpness helps with generalization [18]. One of the main strengths of STU is its clean optimization landscape. This benefit becomes more important in larger and more complex models, and its effects are felt in practice.

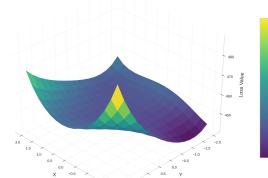


Fig. 3: Local loss landscape of the **STU** layer.

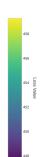


Fig. 4: Local loss landscape of the **S4** layer.

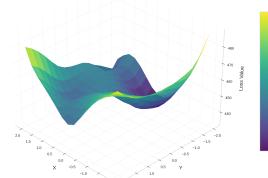


Fig. 5: Local loss landscape of the **Mamba-2** layer.

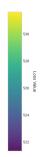


Fig. 6: Local loss landscape of the **attention** layer.

C. Other synthetic tasks

We also investigate the performance of the STU layer in synthetic tasks that are more well-known in the deep learning

²For S4, Mamba-2, and any other model that parameterizes an LDS through its system matrices, this non-convexity even grows with sequence length. One can often relax the problem to a convex one by directly learning a convolutional kernel, though at the cost of maintaining parameter count that grows with sequence length. STU is able to avoid both of these issues with a convex and efficient parameterization.

literature. We briefly introduce the induction heads [19] and associative recall [20] tasks, our experimental setups, and the corresponding results here. For completeness, we also include in Appendix B the results on the selective copy [21] task as well as extra experimental details.

In the induction heads task, the model is required to recall one token (sampled uniformly from a vocabulary) immediately after a special `flag` token; the rest of the context consists of the same special `blank` token, which the model should learn to ignore. The associative recall task is harder, as it first gives the model an (unordered) sequence of key-value pairs and then asks for the value associated with a query key after. The model must keep track of the entire context, not just a single token. Both of these tasks have nonlinear dynamics and require single-token precision to fully solve. Furthermore, since we apply deeper models, the optimization is non-convex for all the models considered.

Experiment details and conclusions. We train two-layer models, with MLP layers in between, using the cross-entropy loss, the Adam optimizer with tuned learning rate, and training examples in batches of size 64. For induction heads we set the context length to 128, vocabulary size to 10, and all the model widths to 32. Associative recall was ran with context length 32, vocabulary size of 5, and model widths of 8. Accuracies during training are plotted in Figures 7 and 8, respectively, and are averaged over 8 trials with error bars.

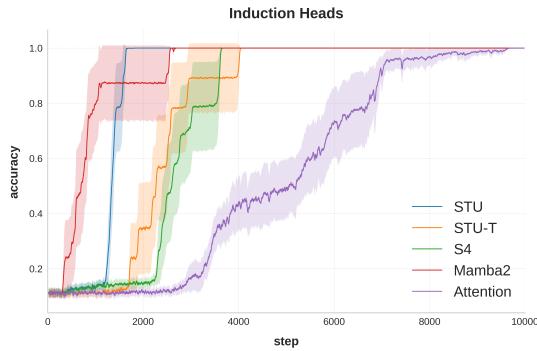


Fig. 7: Prediction accuracy for the token immediately following the special flag token during training.

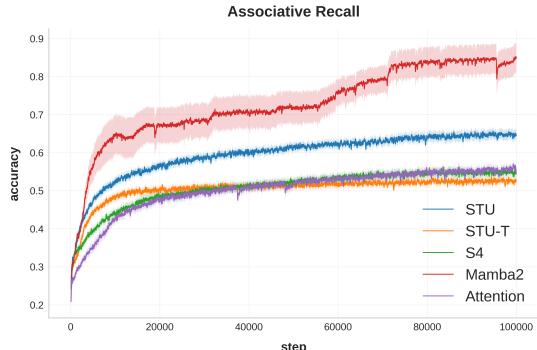


Fig. 8: Prediction accuracy for the value corresponding to the given query during training.

Naturally, Mamba-2 performs the best on these sorts of selec-

tion tasks; the selection mechanism in the Mamba architecture was specifically designed for this type of functionality. What is surprising, however, is that on both tasks STU is able to improve over S4 and attention without the use of any extra nonlinearity.

For the induction heads task, STU-T and the baselines appear to learn in stages, with the accuracy plateauing occasionally. Moreover, there is significant variation across seeds for which time step some models begin to solve the task. Both of these observations point to the difficulty of the underlying optimization problem: while all models considered are expressive enough to solve induction heads, STU appears to have an easy time finding a solution without the use of a gating or selection mechanism.

On the harder associative recall task, S4, STU, and attention models were unable to completely learn the task³. Once again, however, STU does noticeably better than S4 despite both methods having a time-invariant, convolutional structure.

Overall, the optimization behavior of STU on these synthetic tasks gives us confidence that it is a simple, easily optimizable, yet expressive layer for use in larger models, which will be the main focus for the rest of the paper.

III. EXPERIMENTS WITH ROBOTICS SEQUENCE PREDICTION

We now consider next-state prediction for actuator coordinates using the MuJoCo physics engine [22], a more challenging sequence prediction task. The goal is to learn the dynamics of a certain physical simulation agent, for example the Ant-v1 system,

$$x_{t+1} = f(x_t, u_t),$$

where $x_t \in \mathbb{R}^{29}$ and $u_t \in \mathbb{R}^8$ correspond to the state and action at time t , respectively.

More precisely, x_t corresponds to the positions, orientations, joint angles, and velocities of the various limbs of the Ant-v1 controller, whereas u_t represents the torques applied to the joints of the Ant-v1 controller and is generated by an agent pretrained using proximal policy optimization [23]. Unlike the synthetic linear dynamical system in Section II, the dynamics f for this particular MuJoCo prediction task are nonlinear and hybrid, i.e. non-smooth.

In the experiments below, we defined the loss function to be the squared Euclidean distance between the predicted state vector \hat{x}_{t+1} and the true state vector x_{t+1}

$$L(\theta) = \frac{1}{2} \|\hat{x}_{t+1}(\theta) - x_{t+1}\|_2^2 = \frac{1}{2} \sum_{i=1}^n (\hat{x}_{t+1,i}(\theta) - x_{t+1,i})^2,$$

where $\hat{x}_{t+1}(\theta)$ is the predicted state vector parameterized by θ , and x_{t+1} is the true state vector.

Model architectures. The Mixture-of-Experts (MoE) architecture has seen newfound resurgence in modern neural

³The fact that attention did not solve our associative recall task is surprising. Considering the nature of the problem, one might have expected the transformer’s attention mechanism to dominate here. We suspect this to have been caused by optimization issues, as the attention models were not very robust to optimizer hyperparameters.

networks thanks to its ability to increase model capacity without a proportional increase in computation [24]. We ran small ablation studies and found that a sparse mixture-of-experts (MoE) over gated MLPs after the model’s main sublayer performed the best. The gated MLP splits the inputs into two parts: a main component and a “gate” that modulates the main component using the SiLU [25] activation function. The gating MoE network then dynamically selects which top experts to use for each input based on a learned distribution, where each expert is a gated MLP, applies the top_k selected expert(s) to the input, and combines their outputs using a weighted summation, where the weights come from the gating mechanism. Figure 18 shows the model architectures in detail. Table I gives the specification for each tested architecture in these two sets of experiments.

We attempted to maintain the same parameter count of around 0.5M for each model given the various architectural differences. For STU, Transformer, and Mamba-2 models, we ensured that they have the same widths and depths and only adjusted the MLP intermediate hidden dimension size as the primary method to equalize the parameter count as commonly seen in the literature [26]. As for the STU-T model, its tensordot approximation allowed for extra parameter space that can be allocated towards other areas in the model. For example, using exactly the same configurations as STU, STU-T only has 0.05M parameters; to scale it up for fair model comparisons, here we choose to expand the “width” of STU-T, which is the model dimension d_{model} in Figure 1.

Experiment controllers. We tested our models on three MuJoCo controllers: Ant-v1, HalfCheetah-v1, and Walker2D-v1 [27]. For each task, we ran a pretrained PPO model over 1,000-step trajectories offline and saved the model’s state information at each step. For each task, this was repeated across 3,000 different seeds to generate 3,000 different trajectories with 1,000 steps each. The full results are in Section D of the appendix. Note that we only provide the results (Figure 9 and Table II) for Ant-v1 here for brevity. We also note that the results presented hereby are partial, after hyperparameter tuning. The full results of these ablation studies can be found in Section D.1.

Additional experiments. To further validate the results, we conduct next-step prediction and autoregressive next-step prediction using the trained models (Figure 10 and 11, with losses averaged over 500 predictions for each model). Next-step prediction uses ground truth states and actions from a fixed input window to forecast the next state, mirroring the setting of the evaluation stage during training. Autoregressive next-step prediction, on the other hand, incorporates the model’s own past predictions as inputs, allowing errors to accumulate over time.

TABLE I: Model specifications for the Ant-v1 Task .

Model	Parameters	Width/Depth	MLP Scale	Filters	MoE	Time / Train Step
STU	0.55M	64/4	1	16^4	False	11.4ms
STU-T ⁵	0.47M	128/4	8	16	False	11.4ms
Transformer	0.51M	64/4	6	-	True	21.3ms
Mamba-2	0.51M	64/4	4	-	True	40.5ms

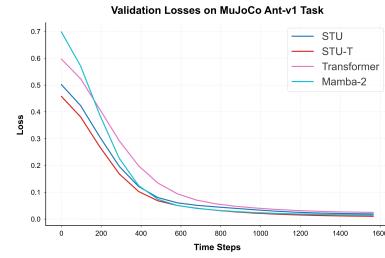


Fig. 9: Ant-v1 comparative training results.

TABLE II: Ant-v1 comparative validation loss results.

Model	Validation Loss
STU	0.0181
STU-T	0.0092
Transformer	0.0237
Mamba-2	0.0139

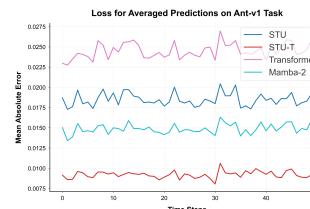


Fig. 10: Ant-v1 comparative next-step prediction results.

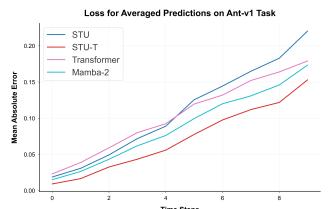


Fig. 11: Ant-v1 comparative autoregressive next-step prediction results.

A. Takeaways from robotics experiments

Based on our experiment results, we can see that STU-T outperforms all other models on a parameter-adjusted basis. We summarize our main findings below.

Attention vs. state space models. It has been observed that Transformers struggle with tasks requiring the recall of extremely long-range dependencies [28], whereas state space models (SSMs) excel in this domain. The Transformer’s quadratic complexity in sequence length [1] contrasts with SSMs’ subquadratic complexity and ability to theoretically capture infinitely long dependencies [2].

Mamba [3], a recent SSM variant, introduces selective state spaces that dynamically adapt to input sequences, achieving state-of-the-art performance across various tasks. In our study, Mamba-2 significantly outperforms Transformer in MuJoCo robotics sequence prediction tasks. More excitingly, the STU-T model outperforms Mamba-2 while being faster. Another notable observation on the comparison of the models’ training losses is that the starting losses of STU and STU-T are relatively lower than those of Transformer and Mamba-2.

Tensordot approximation effect. As noted in the introduction, the tensordot approximation of STU-T models is a significant optimization in both computational and memory efficiency. We found that STU-T models are much smaller yet have performance on par, and at most times exceeding, that of full STU models. Moreover, the parameter savings due to the tensordot approximation can be reallocated towards other areas of the model. This makes STU-T models our best

⁴Ablation studies [5] on the performance of STU using different numbers of filters (K) show that loss stops decaying and plateaus at around $K = 15$.

⁵One advantage of the tensordot approximation is that it allows us to allocate parameters elsewhere in STU-T while maintaining equal parameter count. For an overview of STU-T’s performance with similar model hyperparameters as the other models in this experiment, see Table IX.

performing variant given equalized parameter counts, beating state-of-the-art Mamba-2 model in all three MuJoCo tasks.

Mixture-of-STUs. We found that formulating our MLP layer in the SwiGLU configuration [29] improves performance in STU models; however, formulating the MLP layer as a sparsely-gated mixture-of-SwiGLUs hurts performance. We envision there is a ripe field of research in architectural design for the STU waiting to be explored, e.g. adaptive online methods and variants of MoE [11], which we leave to future work.

Width vs. Depth. The STU model benefits more from increasing model depth, whereas the Transformer model is affected more by increases in model width. When model width is held constant, we observe that the Transformer stops improving with increased depth—particularly after just two layers⁶. See Tables VIII-X for these ablations.

IV. EXPERIMENTS WITH LANGUAGE MODELING

In this section, we explore the STU’s ability for sequence prediction in the context of language modeling. We model our architecture in the style of the LLaMA family of models [30], and we open source a simple, fully distributed large language model (LLM) pretraining pipeline for the community to enjoy and build upon.

A. Experiment setup

Data. We pretrain on roughly 10B high-quality tokens from FineWeb-Edu [31], a large-scale, open source dataset for language modeling. We tokenized the dataset into 95 training shards and 1 validation shard, each containing about 100M tokens, using the `o200k_base` tokenizer from the OpenAI `tiktoken`⁷ library.

General design choices. For each model, we used RMSNorm [32] to pre-normalize the inputs before each attention layer, Flash STU layer, and MLP layer. We followed the standard pre-norm residual connection pattern around each of the sublayers, i.e. the output of each sublayer is $x + \text{Sublayer}(\text{RMSNorm}(x))$. To further stabilize training, we capped logits [33], [34], [35] in each attention layer at 50.0. Following NanoGPT, we rounded up the vocabulary size of each model to the nearest multiple of 64 in order to use more efficient CUDA kernel pathways. We tied embeddings, and we did not use dropout. See Table XI for the full architectural configurations.

Transformer architecture. We followed the GPT-2-styled Transformer from NanoGPT [36]. We added small optimizations such as FlashAttention-2 [37]. For the 2B training run, we used the ALiBi [16] modification to the attention scores. We used position interpolation [38] to help the model “length generalize” beyond its context window size from

⁶We suspect this could be related to the “induction heads” phenomenon observed in attention-based architectures [19]. Induction heads are specialized to recognize and replicate patterns in the input data, facilitating in-context learning. These heads typically emerge within the first two layers of a Transformer, providing many of its learning capabilities. As a result, additional layers contribute diminishing returns, since the core functionalities are already established early on.

⁷<https://github.com/openai/tiktoken/>

training. It has been shown that position interpolation works even with ALiBi [39], so we used an interpolation factor of 0.25, extending the model’s effective context window size at inference time by a factor of 4. For the 500M runs, we used Rotary Positional Embeddings (RoPE) [40]. In terms of performance, we find that there is no significant performance difference between the two positional embedding schemes. We used highly optimized Triton ⁸ kernels [41] for the implementation of our MLP (SwiGLU), RMSNorm, and loss function.

Flash STU architecture. We augmented the STU-T model with Flash FFT [42] for efficient convolutions. We found that the tensordot approximation was necessary to scale up STU-based models, as it was difficult to scale beyond 1B parameters without experiencing frequent out-of-memory (OOM) errors. Following the work from the state space literature [4], [26] relating to the role of attention layers, we opted to use a simple *hybrid* model architecture with alternating layers consisting of STU-T and local attention, both followed by an MLP layer. To maintain efficiency, we used the same optimized Triton kernels from the Transformer, and we replaced global attention with sliding window attention [7] with window size being one-eighth of the sequence length.

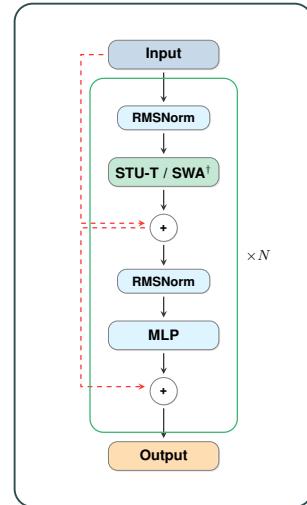


Fig. 12: Flash STU model architecture, alternating between STU-T and (sliding window) attention[†].

Training. We used fused AdamW [51] with default PyTorch hyperparameters, and we set the maximum learning rate and minimum learning rate to 3.0×10^{-4} and 3.0×10^{-5} , respectively, following [52], [53]. We followed a linear decay with warmup learning rate schedule [54] and allocated 10% of the total number of training steps towards warming up the learning rate to the maximum learning rate before linearly decaying back down to the minimum learning for the remainder of the training run. For better memory efficiency,

⁸<https://openai.com/index/triton/>

TABLE III: Performance comparison of Flash STU and Transformer models across various language modeling evaluations, such as MMLU [43], HellaSwag [44], PIQA [45], BoolQ [46], WinoGrande [47], CommonsenseQA [48], OpenBookQA [49], and ARC [50].

Model	MMLU (acc ↑)	Hella. (acc_n ↑)	PIQA (acc_n ↑)	BoolQ (acc ↑)	Wino. (acc ↑)	CSQA (acc ↑)	OBQA (acc_n ↑)	ARC-e (acc ↑)	ARC-c (acc_n ↑)	Average (↑)
Flash STU 550M	26.31	28.64	60.94	52.23	50.67	19.66	26.00	45.16	23.63	37.03
Mamba-2 Hybrid 546M	25.82	28.51	57.62	51.87	49.09	18.67	27.60	44.36	22.53	36.23
Mamba-2 561M	24.15	26.83	57.62	46.54	50.12	20.23	26.60	44.95	23.38	35.60
Transformer 564M	25.16	26.85	56.53	51.41	50.51	19.82	25.00	38.64	21.25	35.02

we accumulated gradients and performed gradient updates only after every 8 training steps. Each model was trained for 16 hours across $8 \times \text{H100}$ 80GB HBM3 GPUs using the Fully Sharded Data Parallel (FSDP) framework from PyTorch [55].

B. Results

First, we investigate whether Flash STU can scale to billion-parameter models without memory or stability issues. This crucial scaling property [56] is the determining factor of model architecture adoption in the modern LLM era. After verifying successful training at the 2B scale, we focus our benchmark evaluations on compute-optimal model sizes for 10B training tokens, which is approximately 500M parameters [52]. See Table XII for model configurations. Our main result is that the Flash STU architecture **outperforms** the Transformer, Mamba-2, and even a Mamba-2 hybrid⁹. Our results are presented in Figures 13, 14 and Tables III, IV.

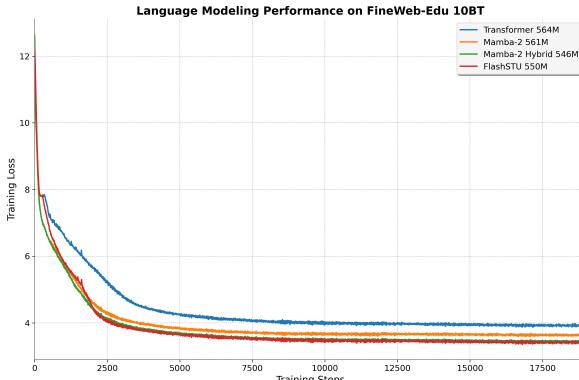


Fig. 13: Training curves for Flash STU and baseline models at different scales.

TABLE IV: Performance of 500M models.

Model	Validation Loss
Flash STU	3.40
Mamba-2 Hybrid	3.44
Mamba-2	3.63
Transformer	3.92

⁹For simplicity, we alternate the Mamba and attention layers as commonly seen in the literature [57], [58]. See [26] for a more fine-grained approach to the hybridization of the Mamba architecture.

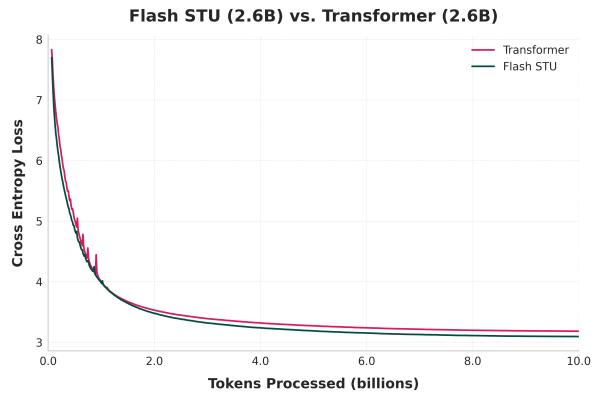


Fig. 14: Comparison of Flash STU and Transformer models on validation set.

TABLE V: Performance of 2B models.

Model	Validation Loss	Perplexity	Mean Time/Step
Flash STU	3.09	22.08	$\approx 3.0\text{s}$
Transformer	3.18	24.11	

Qualitatively, we found that the Transformer was more sensitive to hyperparameter tuning, e.g. learning rate. Additionally, the Transformer was more prone to spikes in its loss compared to Flash STU despite our best attempts to stabilize training.

We also note that although our Flash STU implementation is fairly optimized, it is not yet at the level at which the community has optimized the Transformer model in recent years. Similarly, while Flash STU has the edge in terms of asymptotic time complexity, the Transformer is heavily optimized for matrix multiplication units on modern accelerator hardware, which generally translates to superior wall-clock performance in practice. However, we notably found that the average time per step was approximately around the same for both models. Thus, we believe that the Flash STU model demonstrates great promise for future improvements given that it outperformed the Transformer, while having an asymptotically better time complexity, as well as Mamba and its attention hybrid variant.

V. CONCLUSION AND FUTURE DIRECTIONS

This paper presented a methodological extension and experimental evaluation of the theoretically-founded Spectral

Transformers. We evaluate our hybrid Flash STU architecture on three modalities: synthetic data, robotics control, and language modeling. We notably found that the STU is a viable and competitive alternative to other popular sequence models such as the Transformer and various state space models.

VI. ACKNOWLEDGEMENTS

We thank Naman Agarwal, Xinyi Chen, and Daniel Suo for helpful conversations. Y. Isabel Liu, Windsor Nguyen, Yagiz Devre, and Evan Dogariu were funded in part by Elad Hazan’s grants from the Office of Naval Research and the National Science Foundation. We would like to give special thanks to Princeton Research Computing¹⁰ and Princeton Language and Intelligence¹¹ (PLI) for supplying us with the compute required for making our experiments possible.

REFERENCES

- [1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention Is All You Need,” 2017.
- [2] A. Gu, K. Goel, and C. Ré, “Efficiently Modeling Long Sequences with Structured State Spaces,” 2022.
- [3] A. Gu and T. Dao, “Mamba: Linear-Time Sequence Modeling with Selective State Spaces,” 2024.
- [4] T. Dao and A. Gu, “Transformers are SSMs: Generalized Models and Efficient Algorithms Through Structured State Space Duality,” 2024.
- [5] N. Agarwal, D. Suo, X. Chen, and E. Hazan, “Spectral State Space Models,” *arXiv preprint arXiv:2312.06837*, 2023.
- [6] E. Hazan, K. Singh, and C. Zhang, “Learning Linear Dynamical Systems via Spectral Filtering,” *Advances in Neural Information Processing Systems*, vol. 30, 2017.
- [7] I. Beltagy, M. E. Peters, and A. Cohan, “Longformer: The Long-Document Transformer,” 2020.
- [8] M. Hardt, T. Ma, and B. Recht, “Gradient descent learns linear dynamical systems,” *Journal of Machine Learning Research*, vol. 19, no. 29, pp. 1–44, 2018.
- [9] S. Arora, E. Hazan, H. Lee, K. Singh, C. Zhang, and Y. Zhang, “Towards provable control for unknown linear dynamical systems,” 2018.
- [10] M. Kozdoba, J. Marecek, T. Tchrakian, and S. Mannor, “On-line learning of linear dynamical systems: Exponential forgetting in kalman filters,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 4098–4105, 2019.
- [11] E. Hazan and K. Singh, “Introduction to Online Nonstochastic Control,” *arXiv preprint arXiv:2211.09619*, 2022.
- [12] R. E. Kalman, “A new approach to linear filtering and prediction problems,” *Journal of Basic Engineering*, vol. 82.1, pp. 35–45, 1960.
- [13] B. Anderson and J. Moore, “Kalman filtering: Whence, what and whither?,” *Mathematical System Theory: The Influence of RE Kalman*, pp. 41–54, 1991.
- [14] E. Hazan, H. Lee, K. Singh, C. Zhang, and Y. Zhang, “Spectral filtering for general linear dynamical systems,” in *Advances in Neural Information Processing Systems*, pp. 4634–4643, 2018.
- [15] B. Beckermann and A. Townsend, “On the singular values of matrices with displacement structure,” *SIAM Journal on Matrix Analysis and Applications*, vol. 38, no. 4, pp. 1227–1248, 2017.
- [16] O. Press, N. A. Smith, and M. Lewis, “Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation,” 2022.
- [17] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein, “Visualizing the Loss Landscape of Neural Nets,” 2017.
- [18] P. Foret, A. Kleiner, H. Mobahi, and B. Neyshabur, “Sharpness-Aware Minimization for Efficiently Improving Generalization,” 2020.
- [19] C. Olsson, N. Elhage, N. Nanda, N. Joseph, N. DasSarma, T. Henighan, B. Mann, A. Askell, Y. Bai, A. Chen, *et al.*, “In-context learning and induction heads,” *arXiv preprint arXiv:2209.11895*, 2022.
- [20] S. Arora, S. Eyuboglu, A. Timalsina, I. Johnson, M. Poli, J. Zou, A. Rudra, and C. Ré, “Zoology: Measuring and improving recall in efficient language models,” *arXiv preprint arXiv:2312.04927*, 2023.
- [21] M. Arjovsky, A. Shah, and Y. Bengio, “Unitary Evolution Recurrent Neural Networks,” in *Proceedings of The 33rd International Conference on Machine Learning*, pp. 1120–1128, 2016.
- [22] E. Todorov, T. Erez, and Y. Tassa, “MuJoCo: A physics engine for model-based control,” in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, IEEE, 2012.
- [23] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal Policy Optimization Algorithms,” 2017.
- [24] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean, “Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer,” 2017.
- [25] P. Ramachandran, B. Zoph, and Q. V. Le, “Searching for Activation Functions,” 2017.
- [26] R. Waleffe, W. Byeon, D. Riach, B. Norick, V. Korthikanti, T. Dao, A. Gu, A. Hatamizadeh, S. Singh, D. Narayanan, *et al.*, “An empirical study of mamba-based language models,” *URL https://arxiv.org/abs/2406.07887*, 2024.
- [27] HumanCompatibleAI, “PPO Seals Ant-v1, HalfCheetah-v1, and Walker2D-v1.” Available at <https://huggingface.co/HumanCompatibleAI/ppo-seals/>, 2023.
- [28] Y. Tay, M. Dehghani, S. Abnar, Y. Shen, D. Bahri, P. Pham, J. Rao, L. Yang, S. Ruder, and D. Metzler, “Long range arena: A benchmark for efficient transformers,” *arXiv preprint arXiv:2011.04006*, 2020.
- [29] N. Shazeer, “GLU Variants Improve Transformer,” 2020.
- [30] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, “Llama: Open and efficient foundation language models,” 2023.
- [31] G. Penedo, H. Kydlíček, L. B. allal, A. Lozhkov, M. Mitchell, C. Raffel, L. V. Werra, and T. Wolf, “The FineWeb Datasets: Decanting the Web for the Finest Text Data at Scale,” 2024.
- [32] B. Zhang and R. Sennrich, “Root Mean Square Layer Normalization,” 2019.
- [33] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, “Neural Combinatorial Optimization with Reinforcement Learning,” 2017.
- [34] xAI, “Grok 1 open release.” <https://github.com/xai-org/grok-1/>, 2024.
- [35] G. Team, M. Riviere, S. Pathak, P. G. Sessa, C. Hardin, S. Bhupatiraju, L. Huszenot, T. Mesnard, B. Shahriari, A. Ramé, *et al.*, “Gemma 2: Improving open language models at a practical size,” *arXiv preprint arXiv:2408.00118*, 2024.
- [36] A. Karpathy, “build-nanopt.” <https://github.com/karpathy/build-nanopt>, 2024.
- [37] T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. Ré, “FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness,” 2022.
- [38] S. Chen, S. Wong, L. Chen, and Y. Tian, “Extending Context Window of Large Language Models via Positional Interpolation,” 2023.
- [39] F. Al-Khateeb, N. Dey, D. Soboleva, and J. Hestness, “Position Interpolation Improves ALiBi Extrapolation,” 2023.
- [40] J. Su, M. Ahmed, Y. Lu, S. Pan, W. Bo, and Y. Liu, “Roformer: Enhanced transformer with rotary position embedding,” *Neurocomputing*, vol. 568, p. 127063, 2024.
- [41] B. Hsu, “Liger-Kernel.” <https://github.com/linkedin/Liger-Kernel>, 2024.
- [42] D. Y. Fu, H. Kumbong, E. Nguyen, and C. Ré, “FlashFFTConv: Efficient Convolutions for Long Sequences with Tensor Cores,” 2023.
- [43] D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt, “Measuring massive multitask language understanding,” 2021.
- [44] R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi, “Hellaswag: Can a machine really finish your sentence?,” 2019.
- [45] Y. Bisk, R. Zellers, R. L. Bras, J. Gao, and Y. Choi, “Piqa: Reasoning about physical commonsense in natural language,” 2019.
- [46] C. Clark, K. Lee, M.-W. Chang, T. Kwiatkowski, M. Collins, and K. Toutanova, “Boolq: Exploring the surprising difficulty of natural yes/no questions,” 2019.
- [47] K. Sakaguchi, R. L. Bras, C. Bhagavatula, and Y. Choi, “Winogrande: An adversarial winograd schema challenge at scale,” 2019.
- [48] A. Talmor, J. Herzig, N. Lourie, and J. Berant, “Commonsenseqa: A question answering challenge targeting commonsense knowledge,” 2019.

¹⁰<https://researchcomputing.princeton.edu/>

¹¹<https://pli.princeton.edu/>

- [49] T. Mihaylov, P. Clark, T. Khot, and A. Sabharwal, “Can a suit of armor conduct electricity? a new dataset for open book question answering,” 2018.
- [50] P. Clark, I. Cowhey, O. Etzioni, T. Khot, A. Sabharwal, C. Schoenick, and O. Tafjord, “Think you have solved question answering? try arc, the ai2 reasoning challenge,” 2018.
- [51] I. Loshchilov and F. Hutter, “Decoupled Weight Decay Regularization,” 2019.
- [52] J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. d. L. Casas, L. A. Hendricks, J. Welbl, A. Clark, *et al.*, “Training compute-optimal large language models,” *arXiv preprint arXiv:2203.15556*, 2022.
- [53] J. W. Rae, S. Borgeaud, T. Cai, K. Millican, J. Hoffmann, F. Song, J. Aslanides, S. Henderson, R. Ring, S. Young, *et al.*, “Scaling language models: Methods, analysis & insights from training gopher,” *arXiv preprint arXiv:2112.11446*, 2021.
- [54] A. Defazio, A. Cutkosky, H. Mehta, and K. Mishchenko, “When, Why and How Much? Adaptive Learning Rate Scheduling by Refinement,” 2023.
- [55] Y. Zhao, A. Gu, R. Varma, L. Luo, C.-C. Huang, M. Xu, L. Wright, H. Shojanazeri, M. Ott, S. Shleifer, *et al.*, “Pytorch fsdp: experiences on scaling fully sharded data parallel,” *arXiv preprint arXiv:2304.11277*, 2023.
- [56] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, “Scaling laws for neural language models,” *arXiv preprint arXiv:2001.08361*, 2020.
- [57] O. Lieber, B. Lenz, H. Bata, G. Cohen, J. Osin, I. Dalmedigos, E. Safahi, S. Meirom, Y. Belinkov, S. Shalev-Shwartz, *et al.*, “Jamba: A hybrid transformer-mamba language model,” *arXiv preprint arXiv:2403.19887*, 2024.
- [58] L. Ren, Y. Liu, Y. Lu, Y. Shen, C. Liang, and W. Chen, “Samba: Simple hybrid state space models for efficient unlimited context language modeling,” *arXiv preprint arXiv:2406.07522*, 2024.
- [59] P. Zhou, J. Feng, C. Ma, C. Xiong, S. Hoi, and W. E, “Towards Theoretically Understanding Why SGD Generalizes Better Than ADAM in Deep Learning,” 2021.

APPENDIX

A. Spectral filters

Following [5], we construct spectral filters by extracting the top K eigenvectors from the Hankel matrix:

$$Z(i, j) = \frac{2}{(i+j)^3 - (i+j)}, \quad (4)$$

These top K eigenvectors are then scaled by their corresponding eigenvalues, each raised to the $\frac{1}{4}$ -th power. These scaled eigenvectors serve as our spectral filters and are used to convolve inputs into the spectral basis.

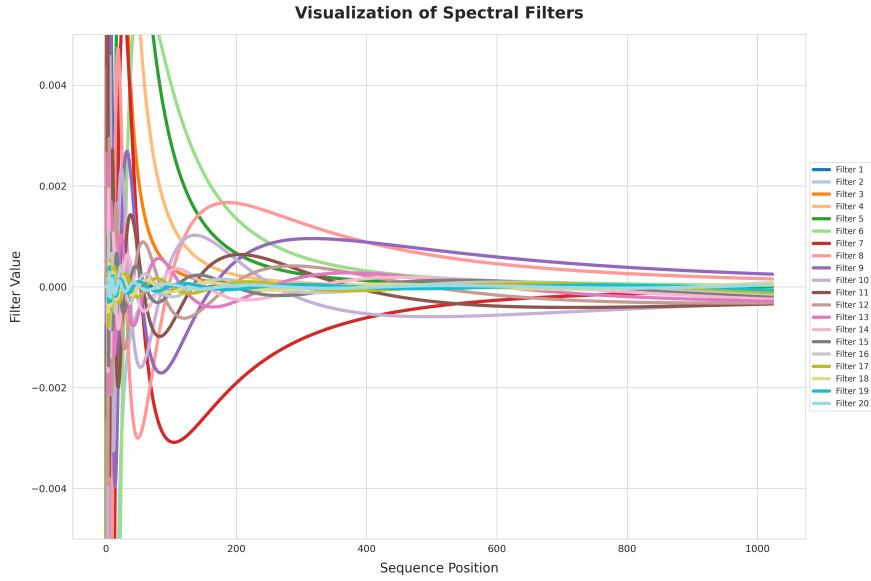


Fig. 15: Visualization of the first 20 spectral filters used in our STU-based models.

B. Additional synthetic experiments

Extra experimental details. The synthetic experiments were all run on a single A100 GPU, with each trial for each model taking around a minute to complete. The STU implementation followed Figure 1 exactly, with the tensordot approximation replacing the STU layer in the STU-T implementation. The softmax attention baseline is FlashAttention-2 [37], and for S4 we used the official implementation¹² [2]. Whenever we introduce nonlinearities in the synthetic experiments, it is always ReLU; we leave a close investigation of multiplicative gating nonlinearities like GLU to future work. We used a learning rate of 0.0024 for Adam, which was tuned to be best for the attention and Mamba-2 baselines (as they were the least robust to learning rate). Anecdotally, we note that STU seems stable with much larger learning rates, which we hope practitioners will be able to take advantage of.

Local curvature of the loss. In Figure 16, eigenvalue ratios $|\lambda_{\min}/\lambda_{\max}|$ of the loss Hessian are plotted after 10 steps of training on the LDS task from the main paper. A ratio near 1.0 means the curvature is similar in all directions, while a ratio near 0.0 means the curvature is stronger in one direction. The contour lines overlaid on the heat map show the loss landscape, allowing us to visualize how this local geometry relates to the overall loss surface. Red areas represent more spherical curvature, and blue areas represent more elongated curvature. Movement along the x and y axes corresponds to local movement in parameter space.

For STU, we see very smooth behavior and obvious directions of progress toward well-conditioned areas, whereas the loss landscape is more complicated for Mamba-2, attention, and especially S4.

¹²<https://github.com/state-spaces/s4/blob/main/models/s4/s4d.py/>

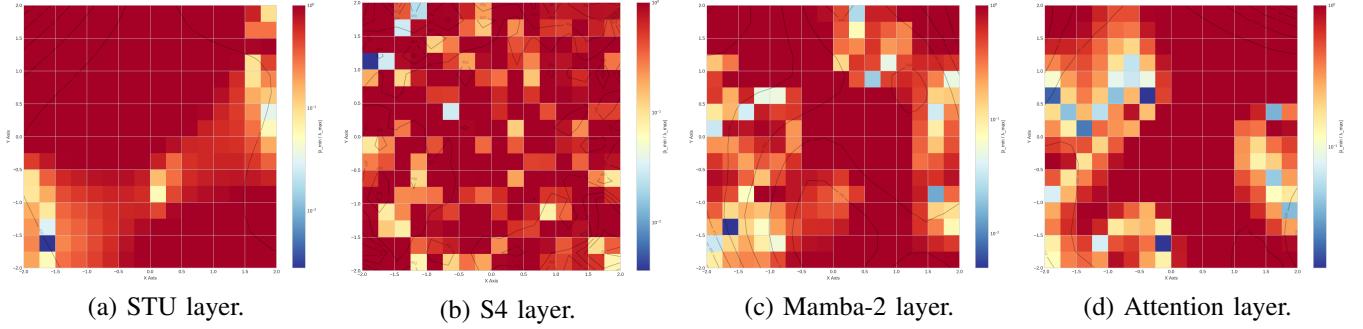


Fig. 16: Heat maps of the ratio $|\lambda_{\min}/\lambda_{\max}|$ of (an estimate of a dimensionality-reduced version of) the loss Hessian.

Selective copy. To better understand how the STU layer performs on different types of small sequence prediction problems, we also experiment on the selective copy task [21]. The selective copy task requires the model to recall (in order) a sequence of a fixed number of tokens, which are sampled uniformly from a vocabulary and distributed randomly throughout the context of otherwise blank tokens. Doing this successfully requires the ability to synthesize single-token resolution information across the whole sequence.

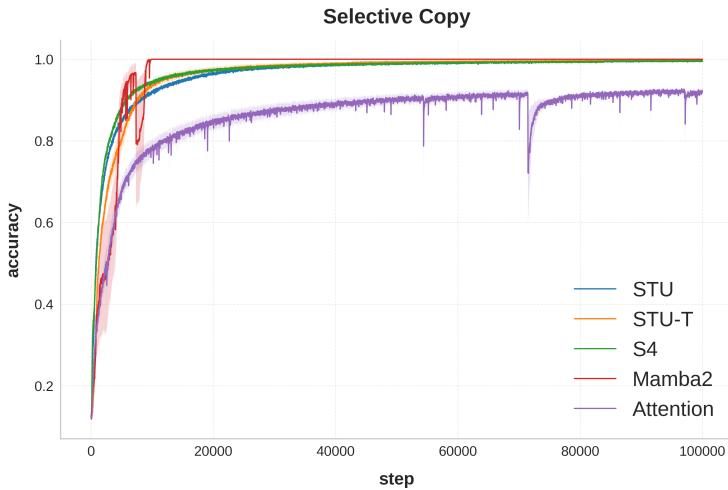


Fig. 17: Prediction accuracy for repeating the non-blank tokens in the context during training.

Experiment details and conclusions. As before, we train two-layer models with MLP layers and nonlinearities in between. The models have width 32 and are trained with a tuned Adam optimizer. The selective copy dataset is constructed with a sequence length of 128 tokens with a vocabulary of 10 tokens to copy each training example. This task, like those in the main paper, is in the regime where models are wide enough to memorize the vocabulary but not the whole context. We leave the study of phenomena under different width and depth scalings to future work.

As we see in Figure 17, Mamba-2 appears to converge to a solution with qualitatively different behavior than the other SSM methods. Put loosely, we suspect that Mamba finds a nonlinear, selective solution while STU and S4 find a linear one which also solves this instance of the task. It is up to architecture designers to choose whether to value a more expressive layer (like Mamba-2) or a simpler, more parameter-efficient, and easier to optimize layer (like STU-T) for use in large models. This experiment serves as a good reminder that this tradeoff looks different depending on the task and scale. Lastly, similarly to the associative recall task, we find that our attention model seems to have difficulty optimizing even though we tune the learning specifically for attention.

To see empirical results of STU and S4 on Long Range Arena benchmark, see Table 1 of [5].

C. Additional model architectures

Figure 18 shows the model architectures used for the robotics experiments in Section III. We ran small ablation studies and found that global skip connections often improved performance in our MuJoCo robotics experiment setting. In non-hybrid STU architectures, we found that performance decreases when using an MoE architecture, as discussed in III-A, so we used a simple gated MLP instead.

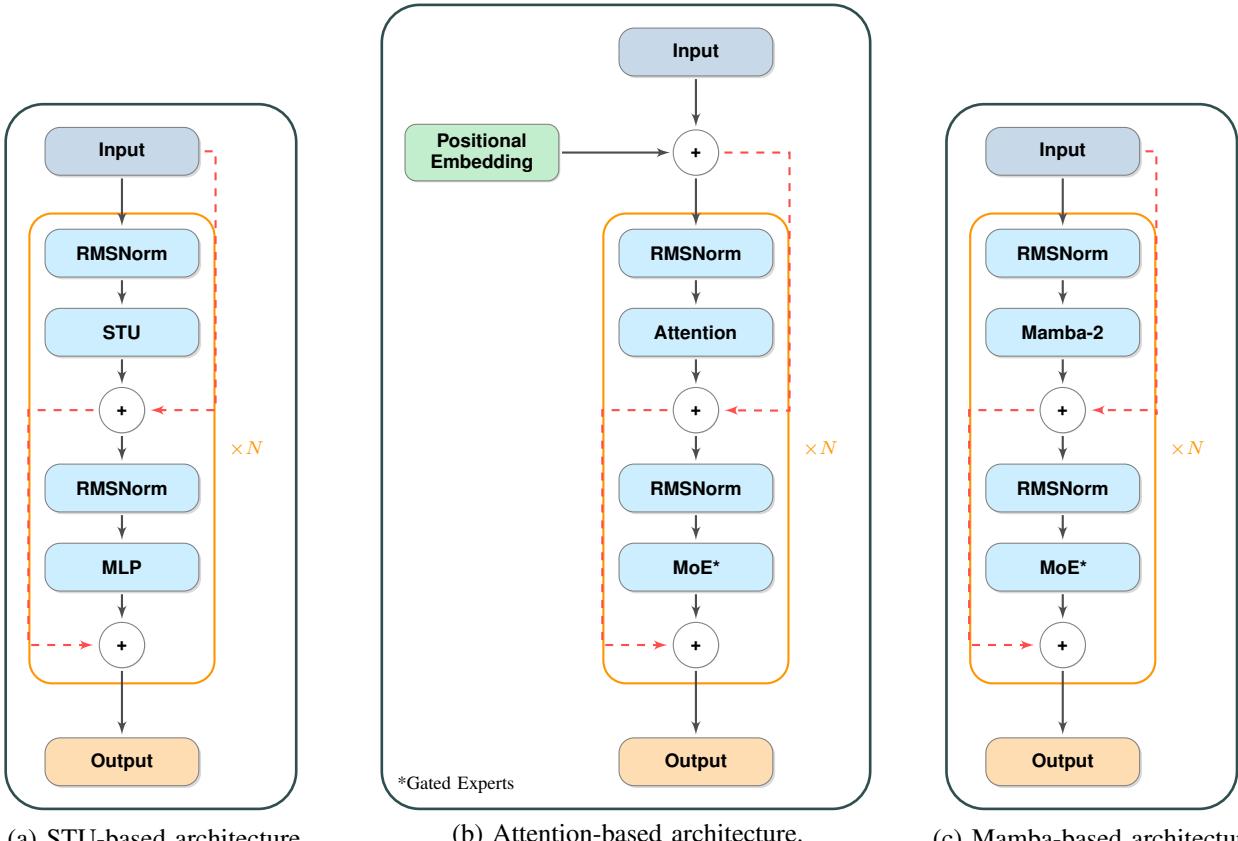


Fig. 18: Comparison of STU-based, Attention-based, and Mamba-based model architectures.

D. Additional experiments with robotics data

In this section we give the remaining experimental results over robotics data for the other two MuJoCo tasks: HalfCheetah-v1 and Walker2D-v1. Compared to Ant-v1 (shown in Section 2 in the main paper), these two tasks only involve motion in a 2D plane. As a result, the state representations for HalfCheetah-v1 and Walker2D-v1 are inherently less complicated, potentially affecting the relative performance of different models on these tasks in theory.

In this setting, the comparison of the models' performances still remains consistent, with Transformer showing the least competitive results, Mamba-2 demonstrating significant improvements over Transformer, and STU-T outperforming both Transformer and Mamba-2. HalfCheetah-v1 task's training results are shown in Figure 19 and Table VI, (auto-regressive) next-step predictions results in Figures 20 and 21. Walker2D-v1 task's training results are shown in Figure 22 and Table VII, (auto-regressive) next-step predictions results in Figures 23 and 24.

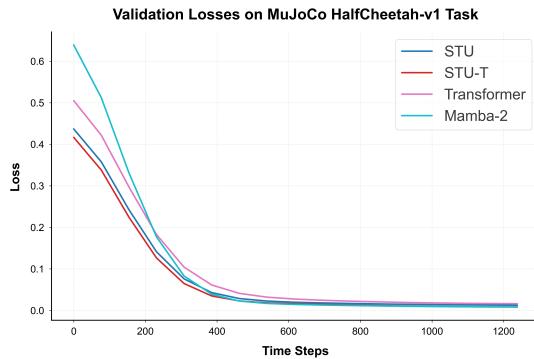


Fig. 19: HalfCheetah-v1 comparative training results.

TABLE VI: HalfCheetah-v1 comparative validation loss results.

Model	Validation Loss
STU	0.0119
STU-T	0.0076
Transformer	0.0157
Mamba-2	0.0081

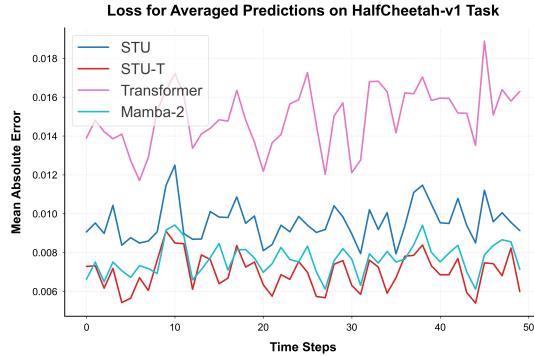


Fig. 20: HalfCheetah-v1 comparative next-step prediction results (with losses averaged over 500 predictions for each model).

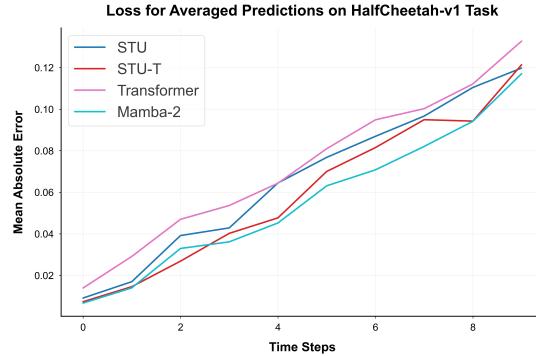


Fig. 21: HalfCheetah-v1 comparative auto-regressive next-step prediction results (with losses averaged over 500 predictions for each model).

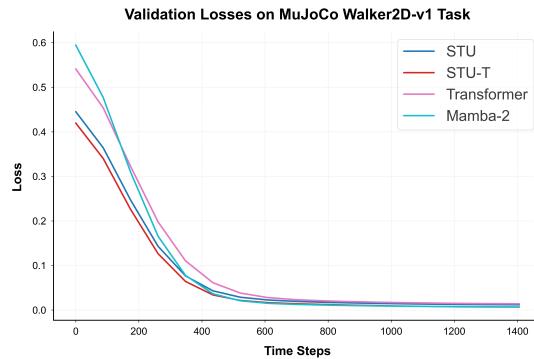


Fig. 22: Walker2D-v1 comparative training results.

TABLE VII: Walker2D-v1 comparative validation loss results.

Model	Validation Loss
STU	0.0112
STU-T	0.0062
Transformer	0.0134
Mamba-2	0.0066

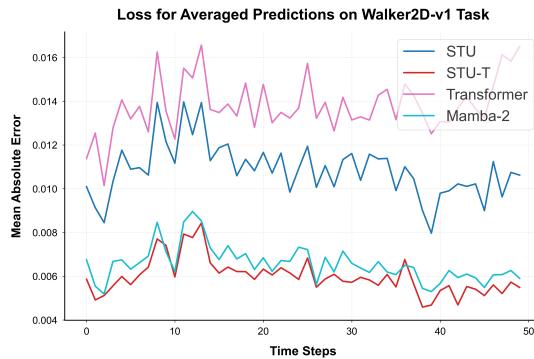


Fig. 23: Walker2D-v1 comparative next-step prediction results (with losses averaged over 500 predictions for each model).

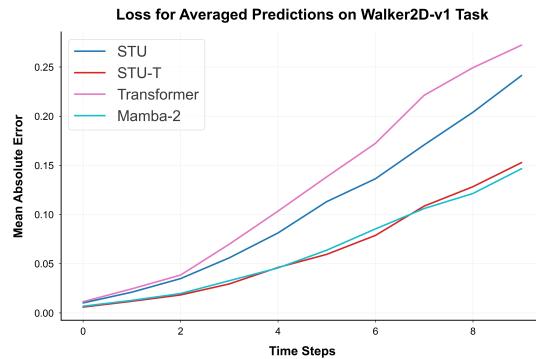


Fig. 24: Walker2D-v1 comparative auto-regressive next-step prediction results (with losses averaged over 500 predictions for each model).

For the (auto-regressive) next-step prediction figures presented above and in Section III of the paper, we mainly focus on showing the loss, i.e. the difference between the models' predictions and the ground truths, averaged over 500 predictions. To further demonstrate what these predictions actually look like, below we give two prediction trajectories plotted across 50 time steps specifically on two example features of Ant-v1: the angle between the two links on the back right (Figure 25) and the x-coordinate angular velocity of the torso (Figure 26).

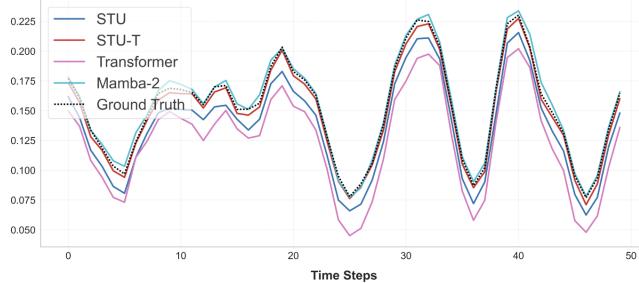


Fig. 25: Ant-v1 next-step prediction (averaged) trajectory of the angle between the ant's two links on the back right.

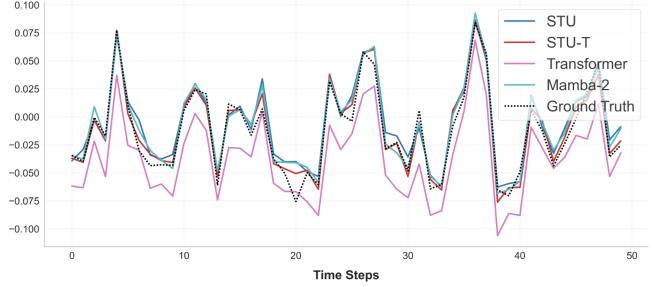


Fig. 26: Ant-v1 next-step prediction (averaged) trajectory of the x-coordinate angular velocity of the torso.

1) Hyperparameters for robotics experiments: We conducted comprehensive ablation studies to investigate the impact of various hyperparameters on the performance of STU, STU-T, and Transformer models in the context of robotics tasks. These studies explore the effects of model width, depth, and input noise on the final Mean Squared Error (MSE) loss. Tables VIII, IX, and X present the results of these experiments.

TABLE VIII: Ablation studies for STU models.

Model	Parameter Count	Width	Layers	Noise/Frequency	MSE Loss
STU	0.18M	32	4	0.0/0.0	0.0217
STU	0.63M	64	4	0.0/0.0	0.0139
STU	1.33M	96	4	0.0/0.0	0.0120
STU	2.30M	128	4	0.0/0.0	0.0108
STU	0.05M	32	1	0.0/0.0	0.0449
STU	0.09M	32	2	0.0/0.0	0.0306
STU	0.18M	32	4	0.0/0.0	0.0217
STU	0.27M	32	6	0.0/0.0	0.0203
STU	0.18M	32	4	0.0/0.0	0.0217
STU	0.18M	32	4	0.1/0.1	0.0357
STU	0.18M	32	4	0.5/0.1	0.0561

TABLE IX: Ablation studies for STU-T models.

Model	Parameter Count	Width	Layers	Noise/Frequency	MSE Loss
STU-T	0.06M	32	4	0.0/0.0	0.0239
STU-T	0.12M	64	4	0.0/0.0	0.0146
STU-T	0.20M	96	4	0.0/0.0	0.0116
STU-T	0.28M	128	4	0.0/0.0	0.0105
STU-T	0.02M	32	1	0.0/0.0	0.0464
STU-T	0.03M	32	2	0.0/0.0	0.0328
STU-T	0.06M	32	4	0.0/0.0	0.0239
STU-T	0.09M	32	6	0.0/0.0	0.0218
STU-T	0.06M	32	4	0.0/0.0	0.0239
STU-T	0.06M	32	4	0.1/0.1	0.0429
STU-T	0.06M	32	4	0.5/0.1	0.0688

TABLE X: Ablation studies for Transformer models.

Model	Parameter Count	Width	Layers	Noise/Frequency	MSE Loss
Transformer	0.07M	32	4	0.0/0.0	0.0472
Transformer	0.17M	64	4	0.0/0.0	0.0294
Transformer	0.30M	96	4	0.0/0.0	0.0214
Transformer	0.47M	128	4	0.0/0.0	0.0204
Transformer	0.02M	32	1	0.0/0.0	0.0545
Transformer	0.04M	32	2	0.0/0.0	0.0464
Transformer	0.07M	32	4	0.0/0.0	0.0472
Transformer	0.10M	32	6	0.0/0.0	0.0462
Transformer	0.07M	32	4	0.0/0.0	0.0472
Transformer	0.07M	32	4	0.1/0.1	0.0637
Transformer	0.07M	32	4	0.5/0.1	0.0961

Note that the noise level refers to the standard deviation of Gaussian noise added to the input data, while the frequency represents the probability of applying this noise to each data point. For example, 0.1/0.1 means that, on average, Gaussian noise with a standard deviation of 0.1 is applied to 10% of the data points.

2) *Hessian heat maps for robotics experiments:* The Hessian eigenvalue ratio heat maps in Figure 27 support our findings on model performance in continuous control tasks. In particular, the sharpness across the landscapes is a salient point for understanding generalization. Flatter minima are generally associated with better generalization [59], while sharper minima can signal sensitivity to parameter changes. All models show a central red region indicating flat loss landscapes near the optimum, suggesting good generalization in this relatively simple setting. However, the STU and STU-T models exhibit smoother transitions to higher curvature regions, implying better optimization stability. In contrast, the Transformer model displays asymmetry with larger high-curvature areas, which may explain its less favorable performance. The STU-T model's minimal high-curvature regions suggest a more globally smooth landscape, contributing to its robust performance across tasks and hyperparameter settings.

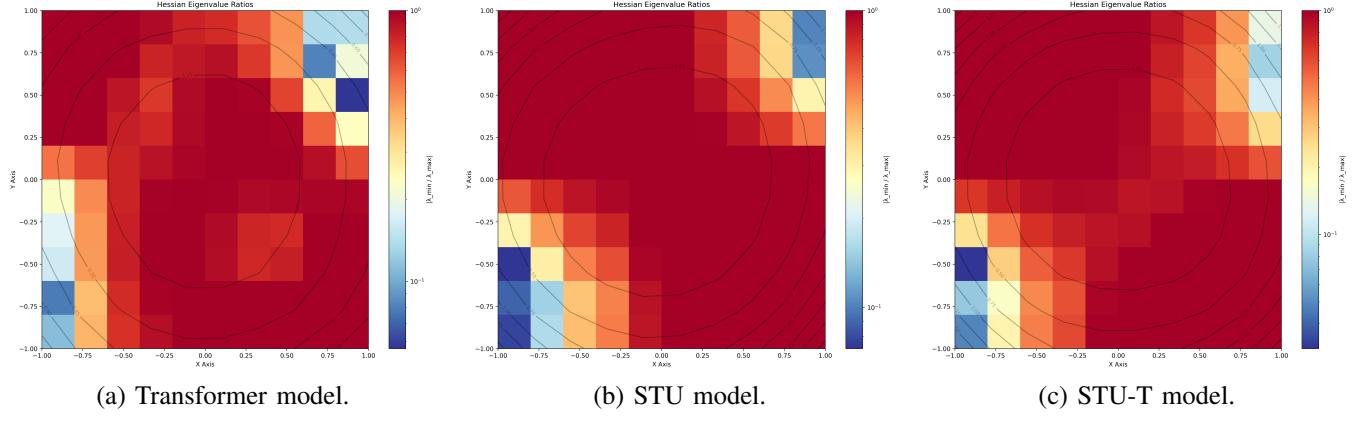


Fig. 27: Hessian heat maps for models with 4 layers and model width of 128.

E. LLM experiments

TABLE XI: Model and training configuration details for 2B LLM training run.

Model Architecture			
	Description	Flash STU	Transformer
Parameter Count	Total number of parameters	2,672M	2,667M
Embedding Dimension	Dimensionality of embedding space	1,536	1,536
Number of Heads	Attention heads (not multi-queried or multi-grouped)	8	8
Number of Layers	Transformer layers	26	25
ALiBi Attention	Attention scores modification using linear biases	Yes (interpolation factor: 0.25)	Yes (interpolation factor: 0.25)
Sliding Window Size	Sliding window attention context lookback size	1,024	8,192
Sequence Length (Training)	Input sequence length during training	8,192	8,192
Sequence Length (Inference)	Input sequence length during inference via position interpolation	32,768	32,768
Vocabulary Size	Size of the model's vocabulary	200,064	200,064
MLP Expansion Factor	Expansion factor in MLP layers	12	12
Bias	Use of bias terms in linear layers	No	No
Dropout	Dropout rate	0.0	0.0
Number of Filters	Number of filters (Flash STU only)	24	—
Use Hankel_L	Alternative Hankel matrix (Flash STU only)	No	—
Learnable Filters	Learnable filters (Flash STU only)	Yes	—
Training and Optimization			
Epochs	Number of training epochs	1	1
Global Batch Size	Number of tokens processed per step	524,288	524,288
Micro Batch Size	Batch size per GPU	1	1
Gradient Accumulation Steps	Number of steps before performing a gradient update	8	8
Warmup Steps	Number of warmup steps	1,907	1,907
Evaluation Period	Evaluation frequency (steps)	25	25
Max Grad Norm	Maximum gradient norm for clipping	1.0	1.0
Optimizer Configuration			
Optimizer	Optimizer type	AdamW	AdamW
Learning Rate Schedule	LR scheduling strategy	Linear decay with warmup	Linear decay with warmup
Max Learning Rate	Maximum learning rate	3.0×10^{-4}	3.0×10^{-4}
Min Learning Rate	Minimum learning rate	3.0×10^{-5}	3.0×10^{-5}
Betas	Optimizer betas	(0.9, 0.999)	(0.9, 0.999)
Epsilon	Optimizer epsilon	1.0×10^{-8}	1.0×10^{-8}
Weight Decay	Weight decay factor	1.0×10^{-2}	1.0×10^{-2}
AMSGrad	Use AMSGrad variant	No	No
Fused	Use fused optimizer	Yes	Yes
Optimization Techniques			
Activation Checkpointing	Enable activation checkpointing	Yes	Yes
Use Flash FFT	Enable Flash FFT (Flash STU only)	Yes	—
Use Tensordot Approx.	Enable tensordot approximation	Yes	—
Use Attention	Enable attention mechanism	Yes	Yes
Softcap	Softcap threshold	50.0	50.0
Torch Compile	Enable Torch compile optimization	No	No
Distributed Training Configuration			
FSDP	Fully Sharded Data Parallel	Yes	Yes
DDP	Distributed Data Parallel	No	No
Mixed Precision	Use mixed precision training	Yes	Yes
Torch Dtype	Data type for PyTorch tensors	bfloat16	bfloat16
CPU Offload	Offload computations to CPU	No	No
Sharding Strategy	Strategy for model sharding	Full Shard	Full Shard
FSDP Modules	Modules to apply FSDP	(STU, Attention, MLP)	(Attention, MLP)
State Dict Type	Type of state dictionary	Full	Full
Auto Wrap Policy	Policy for automatic wrapping	Partial	Partial
Backward Prefetch	Backward prefetch strategy	Backward Pre	Backward Pre
Forward Prefetch	Forward prefetch	No	No
Sync Module States	Synchronize module states across replicas	Yes	Yes
Use Original Params	Use original parameters during training	Yes	Yes

TABLE XII: Model and training configuration details for 500M LLM training runs.

	Flash STU	Transformer	Mamba-2	Mamba-2 Hybrid
Model Architecture				
Parameters	550M	564M	561M	546M
Hidden Dim	896	896	1024	896
Attention Heads	8	8	32	32
Layers	12	12	54	56
Sequence Length	8192	8192	8192	8192
Attention Context Window	1024	8192	—	1024
Tied Weights	True	True	True	True
SSM State Dim	—	—	128	128
MLP Scale	12	12	2	2
Softcap	50	50	—	50
RoPE Theta	10,000	10,000	—	—
ALiBi	False	False	—	True
Training				
Global Batch Size	524,288	524,288	524,288	524,288
Min/Max Learning Rate	3e-4/3e-5	3e-4/3e-5	3e-4/3e-5	3e-4/3e-5
LR Schedule	Linear decay w/ warmup			
Training Precision	bfloating16			

To further illustrate the capabilities of the Flash STU 2B model, we present a sample output generated using a temperature of 1.0 and `top_k` sampling with $k = 50$:

Starter prompt: *"In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English."*

Generated output: *"In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English. Researchers now believe that these animals could now be a kind of hominid species.*

This discovery was recently made by scientists from the University of London. Previously, scientists had thought that this species stood only 5 feet (1.2 meters) for 15 minutes. But this new discovery was the first time a unicorn's eyes were ever revealed.

It was a perfect choice to do a lot of research on the species. In the end, to this day, we still can't say why the unicorns lived in this remote world - they were just another big black monkey. But the discovery proved that these unicorns used words and were able to communicate with other beings using the same words.

The study, which involved researchers from both Washington State University and Cornell University, confirmed that the unicorn family has existed for more than 10,000 years. It was also the first the species was studied and studied in its entirety.

'While there are also other rare wild animals like unicorns and hippopotami, most of these findings were not reported, but confirmed by numerous studies. In order to find their way out of an endangered species and find them, many endangered species require conservation programmes,' said lead researcher, Dr Thomas W. Huxley.

One of the key findings of the study in this week's issue is that the unicorns were able to walk forward and backward across the entire landscape. Their front limbs were able to reach all the way across the same landmass and in the same direction as their front legs were able to. These unique back limbs helped the animals to survive and thrive.

Other research has suggested that these unicorns have also been observed in the Andes Mountains. This is where the researchers believe they may have been able to find their way to the Andes. The area they found is the Amazon Valley.

The scientists believe that they came across an area that is part of the Amazon basin. In order to find out the true nature of this area, the researchers used satellite imagery from NASA for their research.

The scientists also believe that the unicorns might now also be found in other parts of the world, including North

and South America. These discoveries might help scientists study an unknown area, such as the Amazonia and the Andes."