Efficiency

Presented by

Ali Zafar Sadiq (mzw2cu)
Zach Yu (rqx6rs)

Presentation Outline

Paper 1: Mamba: Linear-Time Sequence Modeling with Selective State Spaces

Paper 2: Efficient Streaming Language Models with Attention Sinks

Paper 3: Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity

Paper 1

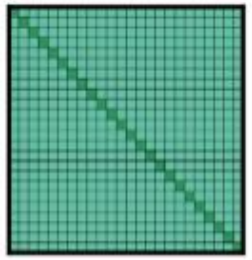
Mamba: Linear-Time Sequence Modeling with Selective State Spaces

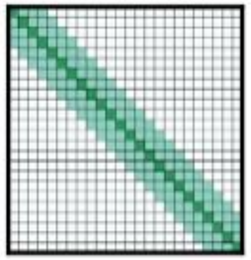
Issues with Transformers

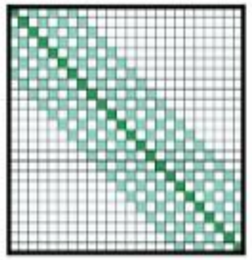
Transformers:

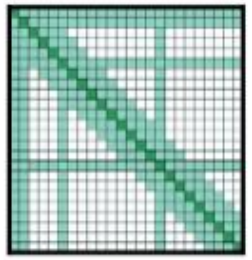
- Self-Attention and their variants
  - Limitations:
    - Context window
    - Computation scaling with sequence length

Alternative Approach is
State Space Model (SSM)


(a) Full n^2 attention

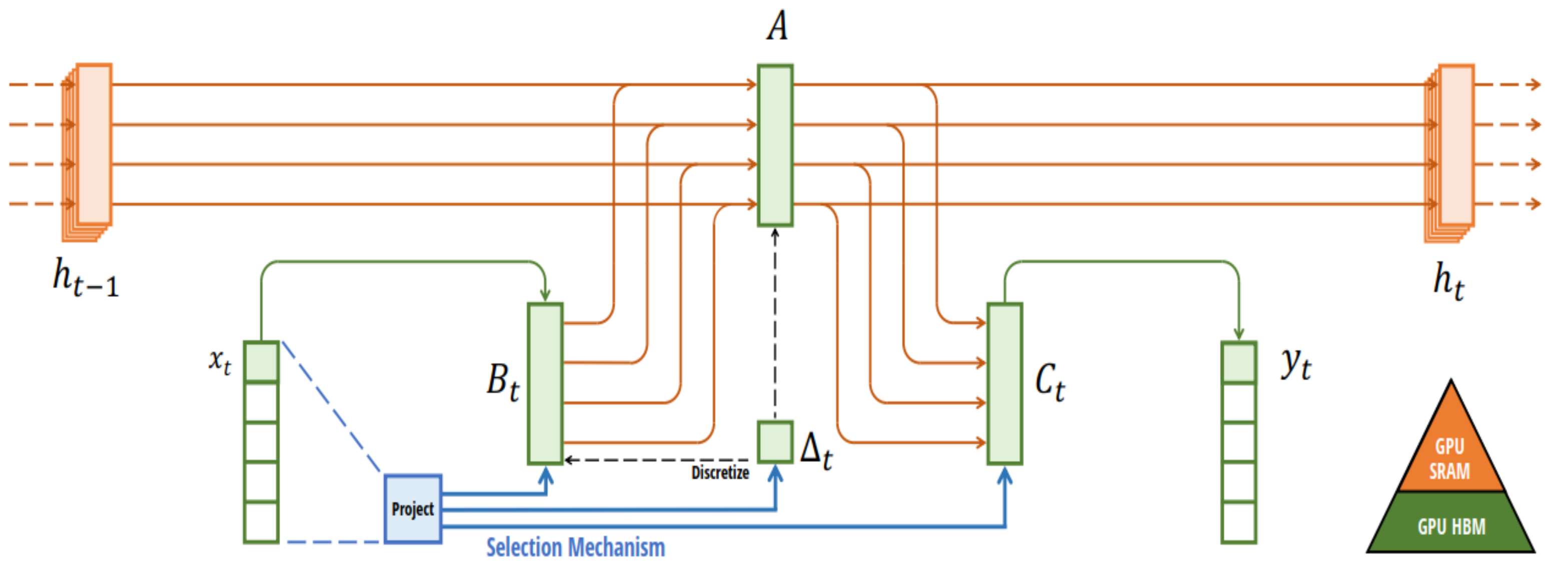

(b) Sliding window attention


(c) Dilated sliding window


(d) Global+sliding window

State Space Models

- In control engineering:
  - mathematical model used to represent physical systems through
    - input variables, output variables, and state variables
  - State space model h_t
  - Input X
  - Output Y
  - A state/latent matrix
  - B input/control matrix
  - C output matrix
  - Δ regulates how much to focus on or ignore the current input xt.



State Space Models

Continuous to Discrete Transformation:

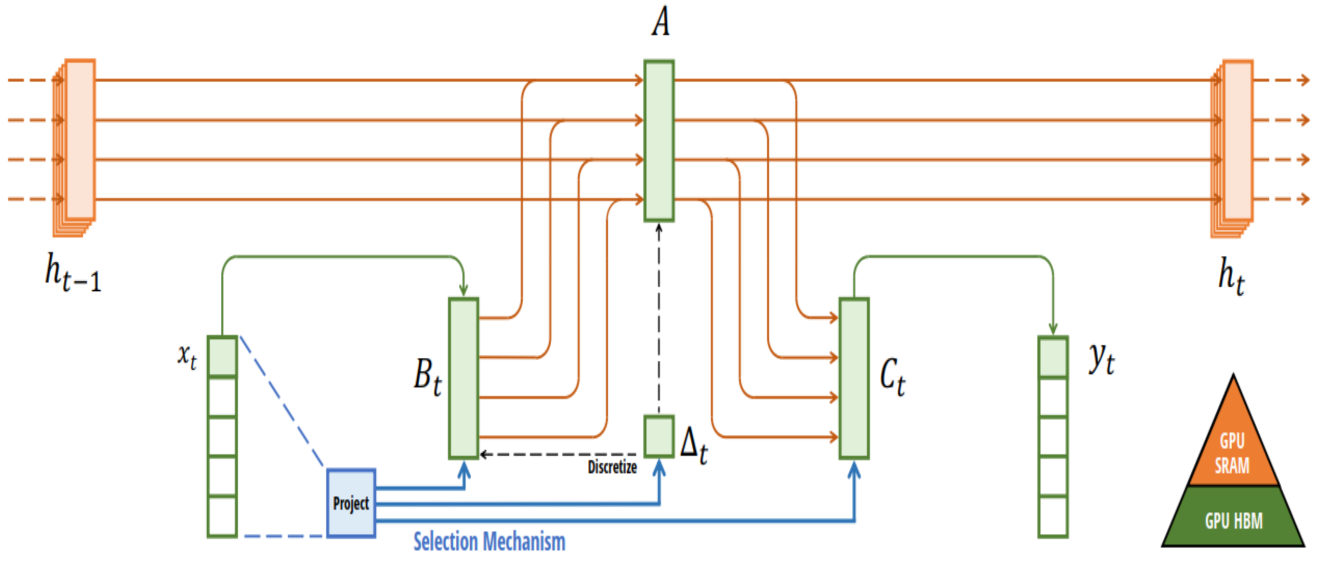
Inspired by a continuous system that maps a 1-dimensional sequence x(t) to y(t) via an implicit latent state h(t).

$$h'(t) = Ah(t) + Bx(t) \quad (1a)$$

$$y(t) = Ch(t) \quad (1b)$$

Parameterization:

- Defined using four key parameters: Δ, A, B, and C.



State Space Models

Discretization transforms continuous parameters into discrete ones using fixed formulas (e.g., zero-order hold).

$$\overline{A} = \exp(\Delta A) \qquad \overline{B} = (\Delta A)^{-1}(\exp(\Delta A) - I) \cdot \Delta B$$

After parameterization, the sequence transformation is computed: i) Linear recurrence (autoregressive)
ii)Global Convolution (parallelizable)

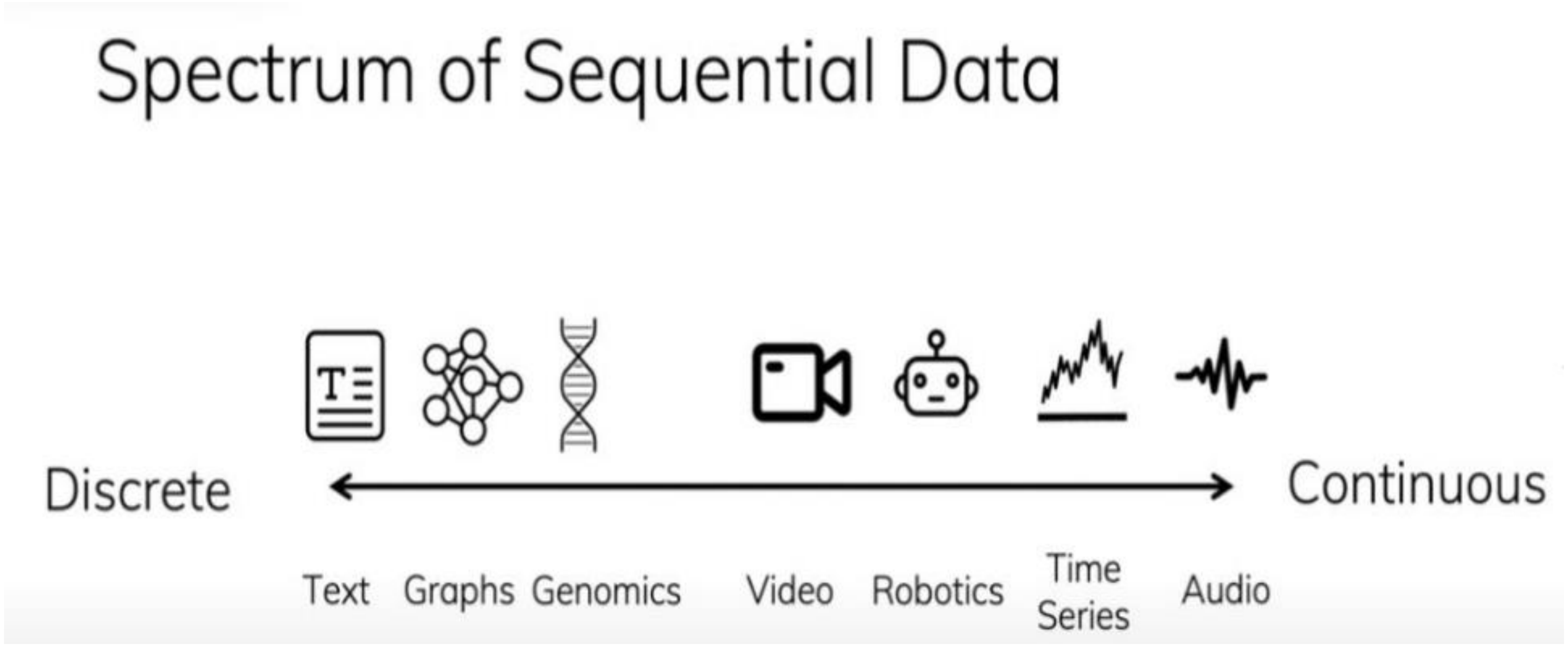
$$h_t = \overline{A}h_{t-1} + \overline{B}x_t \qquad (2a)$$
$$y_t = Ch_t \qquad (2b)$$

$$\overline{K} = (C\overline{B}, C\overline{AB}, \ldots, C\overline{A}^k\overline{B}, \ldots) \qquad (3a)$$
$$y = x * \overline{K} \qquad (3b)$$

Linear Time Invariance (LTI):

- $(\Delta, \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C})$, and consequently discretized$(\boldsymbol{A}, \boldsymbol{B})$ are fixed for all time-steps. This property is called linear time invariance (LTI),
- Here $\boldsymbol{A} \in \mathrm{R}^{N \times N}$, $\boldsymbol{B} \in \mathrm{R}^{N \times 1}$, $\boldsymbol{C} \in \mathrm{R}^{1 \times N}$ matrices can all be represented by N numbers

In Case of Sequential Data



Motivation: Selection as a Means of Compression

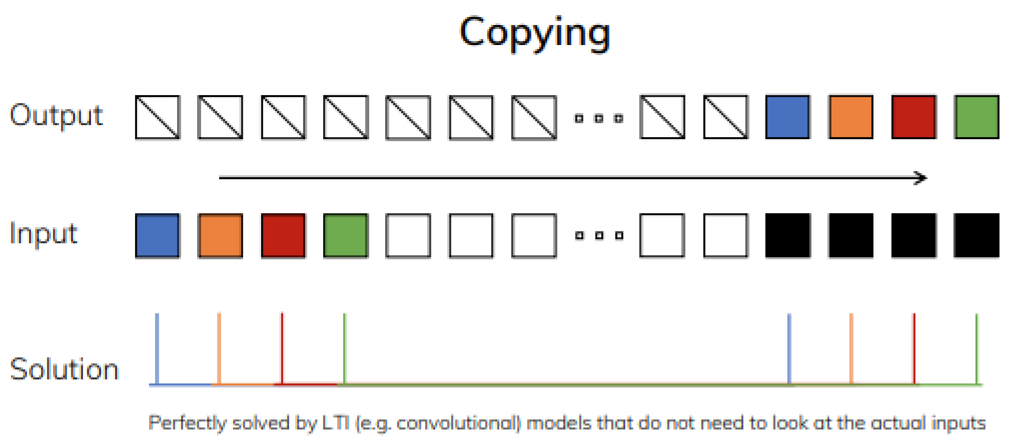
Sequence models must compress a potentially large context into a smaller, fixed-size state.

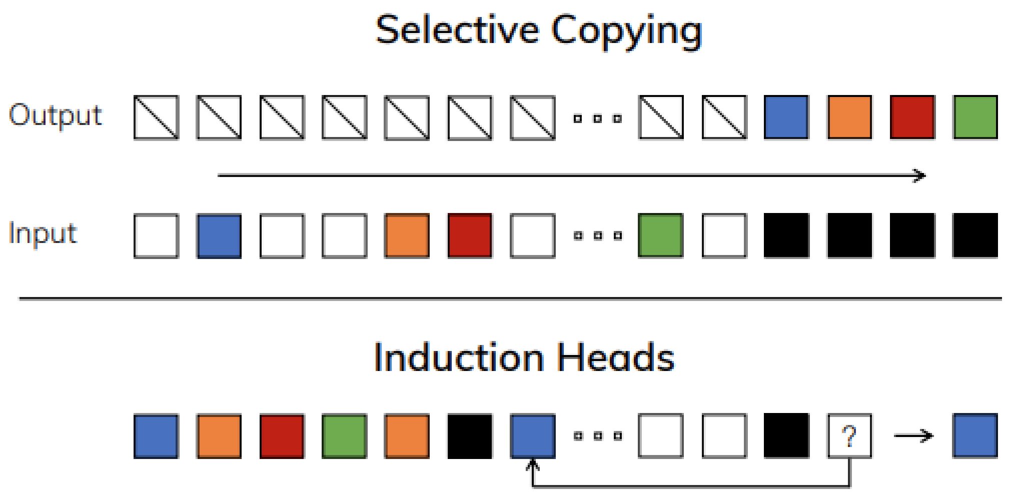
- **Attention-based Models (Transformers):**
  - Effectiveness*:* They use the full context explicitly (no compression).
  - Inefficiency*:* Autoregressive inference requires storing the entire context (KV cache), leading to slow linear-time inference and quadratic-time training.

- **Recurrent Models:**
  - Efficiency*:* They maintain a finite state, enabling constant-time inference and linear-time training.
  - Limitation*:* Their performance is constrained by how well this finite state compresses the context.

Motivation: Selection as a Means of Compression

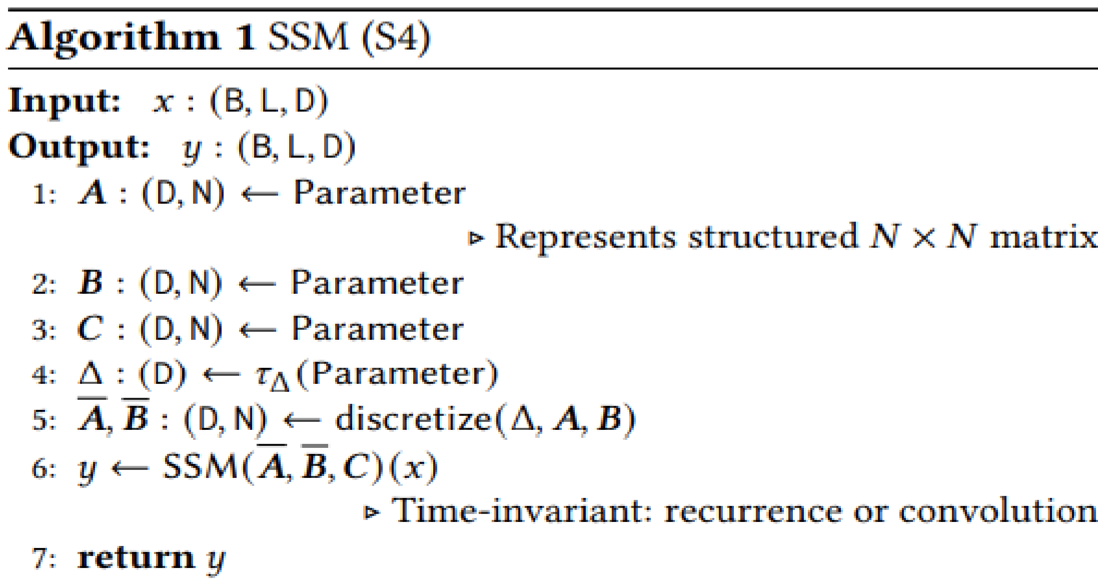
- For LTI models
  - in recurrent view, for constant dynamics (e.g. the $(\boldsymbol{A}, \boldsymbol{B})$), transitions in cannot let them select the correct information
  - In global convolutional view, they lack of content-awareness.


Copying
Output
Input
Solution
Perfectly solved by LTI (e.g. convolutional) models that do not need to look at the actual inputs



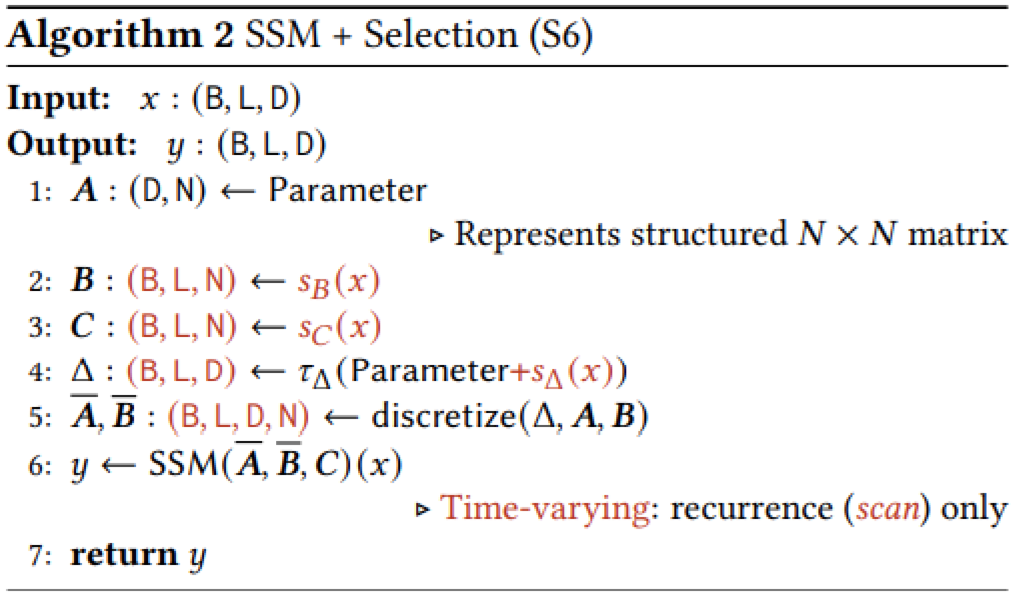
Improving SSM with Selection

- **Input-Dependent Parameterization:**
    - Instead of using fixed (time-invariant) parameters (such as the recurrent dynamics in RNNs or convolution kernels in CNNs), the model makes these parameters depend on the input.
    - The key change is to make parameters (Δ, B, C) functions of the input rather than constant.
    - This shifts the model from being time-invariant to time-varying.



Algorithm 1 SSM (S4)

```
Input: $x : (\mathsf{B}, \mathsf{L}, \mathsf{D})$
Output: $y : (\mathsf{B}, \mathsf{L}, \mathsf{D})$
1: $\boldsymbol{A} : (\mathsf{D}, \mathsf{N}) \leftarrow$ Parameter
                              ▹ Represents structured $N \times N$ matrix
2: $\boldsymbol{B} : (\mathsf{D}, \mathsf{N}) \leftarrow$ Parameter
3: $\boldsymbol{C} : (\mathsf{D}, \mathsf{N}) \leftarrow$ Parameter
4: $\Delta : (\mathsf{D}) \leftarrow \tau_\Delta(\text{Parameter})$
5: $\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}} : (\mathsf{D}, \mathsf{N}) \leftarrow \text{discretize}(\Delta, \boldsymbol{A}, \boldsymbol{B})$
6: $y \leftarrow \text{SSM}(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \boldsymbol{C})(x)$
                              ▹ Time-invariant: recurrence or convolution
7: return $y$
```



Algorithm 2 SSM + Selection (S6)

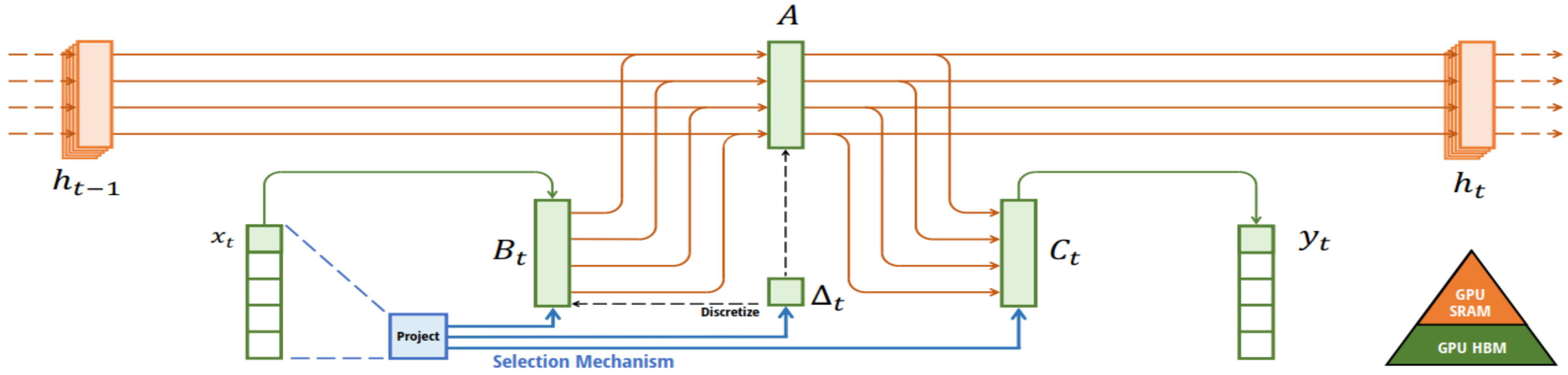
```
Input: $x : (\mathsf{B}, \mathsf{L}, \mathsf{D})$
Output: $y : (\mathsf{B}, \mathsf{L}, \mathsf{D})$
1: $\boldsymbol{A} : (\mathsf{D}, \mathsf{N}) \leftarrow$ Parameter
                              ▹ Represents structured $N \times N$ matrix
2: $\boldsymbol{B} : (\mathsf{B}, \mathsf{L}, \mathsf{N}) \leftarrow s_B(x)$
3: $\boldsymbol{C} : (\mathsf{B}, \mathsf{L}, \mathsf{N}) \leftarrow s_C(x)$
4: $\Delta : (\mathsf{B}, \mathsf{L}, \mathsf{D}) \leftarrow \tau_\Delta(\text{Parameter} + s_\Delta(x))$
5: $\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}} : (\mathsf{B}, \mathsf{L}, \mathsf{D}, \mathsf{N}) \leftarrow \text{discretize}(\Delta, \boldsymbol{A}, \boldsymbol{B})$
6: $y \leftarrow \text{SSM}(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \boldsymbol{C})(x)$
                              ▹ Time-varying: recurrence (scan) only
7: return $y$
```

Efficient Implementation of Selective SSMs

Selective Scan: Hardware-Aware State Expansion

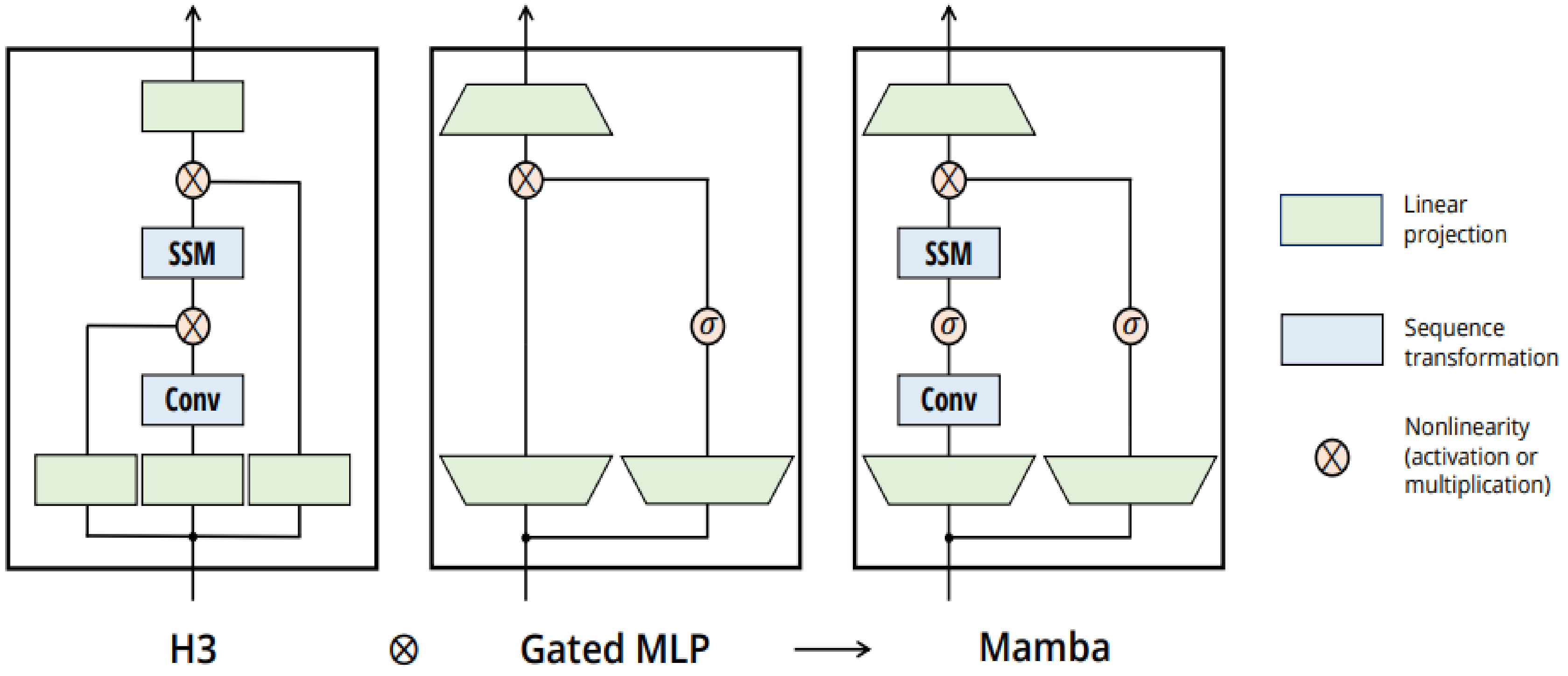
Techniques Used:

- **Kernel Fusion:** Combines multiple operations to reduce memory I/O.
- **Parallel Scan:** Parallelizes the recurrence despite its sequential nature using work-efficient algorithms.
- **Re-computation:** Avoids storing intermediate states during the forward pass and recomputes them during backpropagation to save memory.
- **Memory Hierarchy Optimization:**
  - Instead of materializing the full state h (shape (B,L,D,N)) in high-bandwidth GPU memory, the model loads parameters (Δ,A,B,C) from slower memory (HBM) into fast on-chip SRAM.
  - Discretization and recurrence are then performed in SRAM, and only the final outputs (shape (B,L,D)) are written back, reducing memory usage.



Simplified SSM Architecture

Simplified Block Design: Combines H3 block (basis of most SSM architectures) with MLP block. Instead of interleaving, Mamba blocks are stacked homogeneously.
Key Modifications:
•Compared to H3: Replaces the first multiplicative gate with an activation function.
•Compared to MLP: Adds an SSM to the main branch.
Activation Function: Uses SiLU/Swish activation.
Architecture Inspiration: Based on gated attention unit (GAU), similar to how GAU modified attention mechanisms.



Properties of Selection Mechanisms

Variable Spacing:

•Selectivity filters out irrelevant noise tokens between important inputs, such as language fillers like “um.”

•In gated RNNs, this is achieved when the gate gt approaches 0, effectively ignoring certain inputs.

Filtering Context:

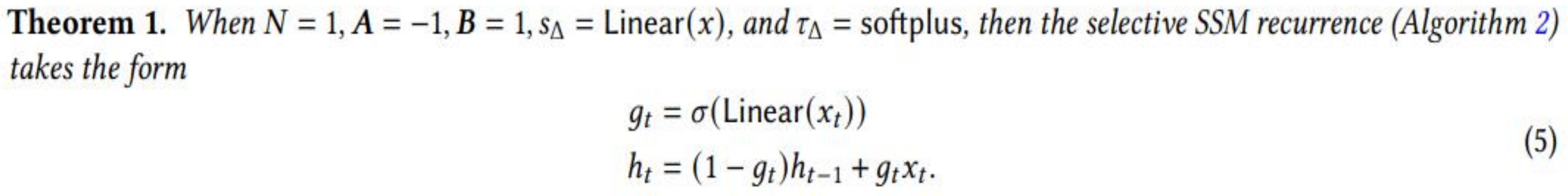
•Many sequence models don’t improve with longer context because they cannot ignore irrelevant information effectively (e.g., global convolutions or LTI models).

•Selective models can reset their state to remove extraneous history, enabling performance to improve monotonically with increased context length.

Boundary Resetting:

•When multiple independent sequences are concatenated (e.g., packing documents or episode boundaries in reinforcement learning), traditional models like Transformers use attention masks to separate them, whereas LTI models may mix information.

•Selective SSMs can reset their state at boundaries (e.g., when $\Delta t \to \infty$ or $gt \to 1$), maintaining sequence separation.



Theorem 1. *When* $N = 1$, $\boldsymbol{A} = -1$, $\boldsymbol{B} = 1$, $s_\Delta = \mathsf{Linear}(x)$, *and* $\tau_\Delta = \mathsf{softplus}$, *then the selective SSM recurrence (Algorithm 2) takes the form*

$$\begin{aligned} g_t &= \sigma(\mathsf{Linear}(x_t)) \\ h_t &= (1 - g_t)h_{t-1} + g_t x_t. \end{aligned} \tag{5}$$

Properties of Selection Mechanisms

Interpretation of Δ:

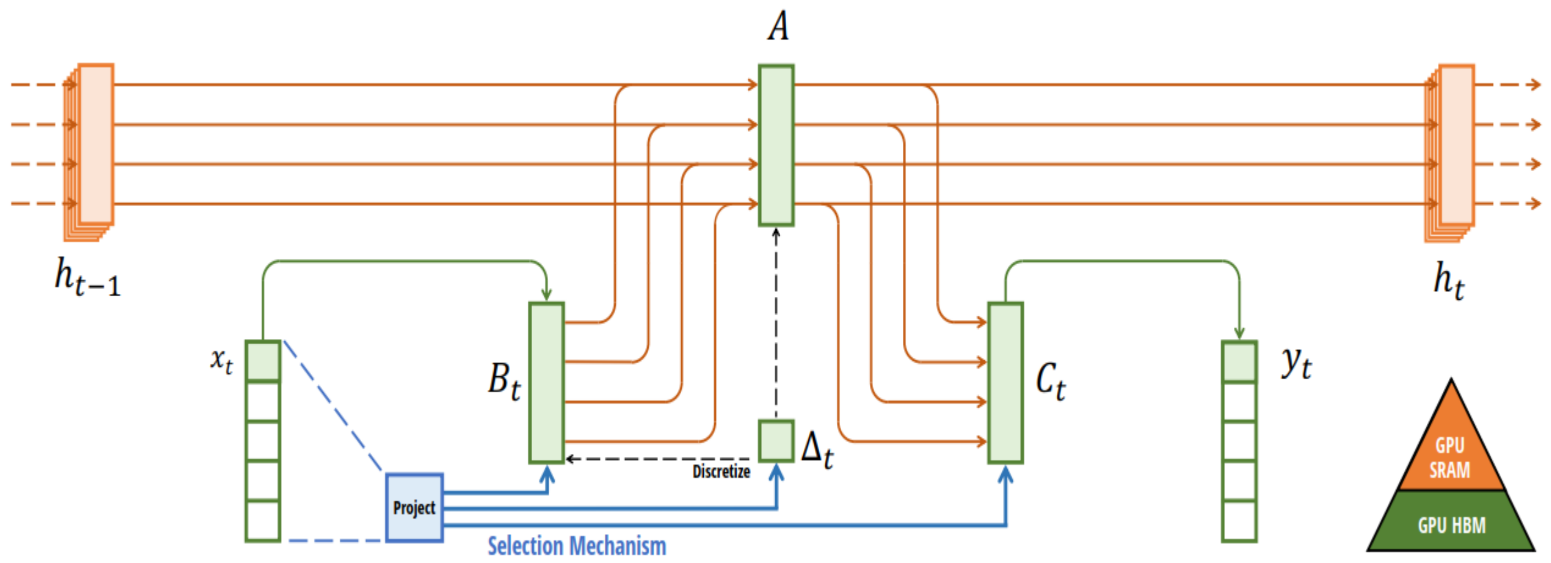
•A large Δ resets the state (focusing on the current input) and a small Δ maintains the previous state (ignoring the current input).

•This parameter generalizes the concept of RNN gating in a continuous system discretized by Δ.

Interpretation of $\boldsymbol{A}$:

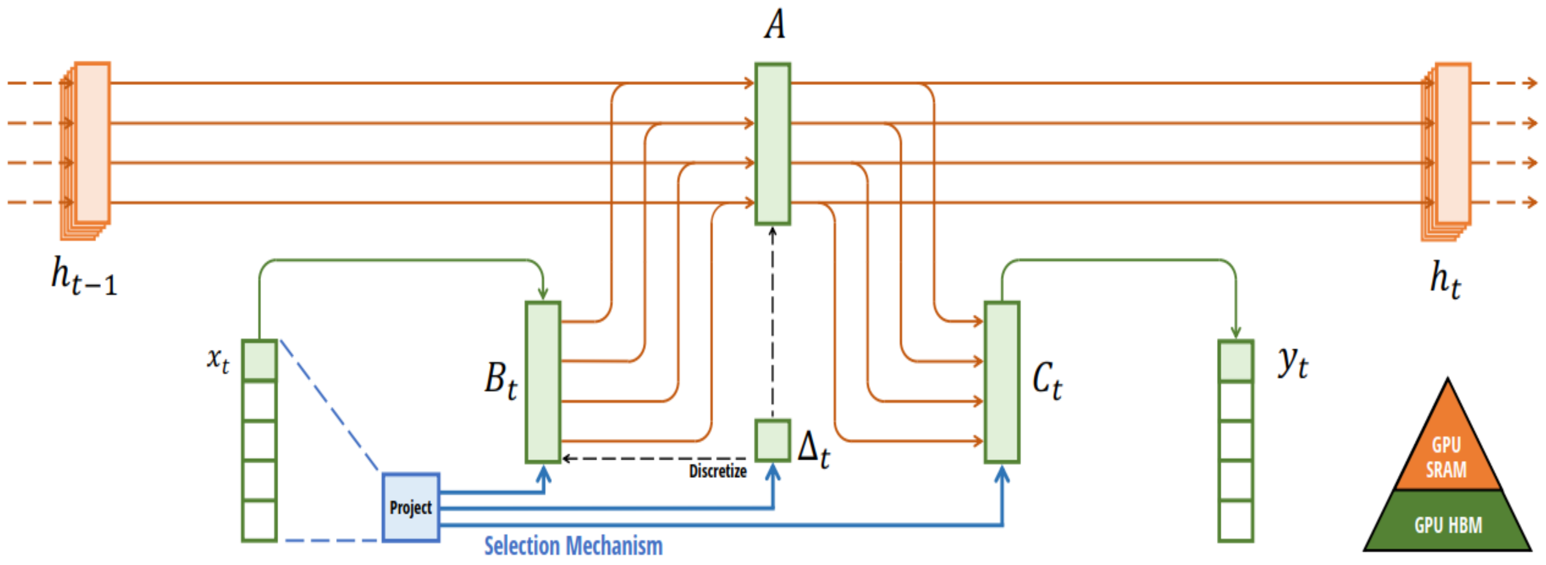
•Although $\boldsymbol{A}$ could be made selective, it primarily influences the model through its interaction with Δ (via $\boldsymbol{A}$ = exp(Δ$\boldsymbol{A}$)).

•Selectivity in Δ is sufficient, and making $\boldsymbol{A}$ selective might yield similar performance, but is omitted for simplicity



State Space Models

- **Interpretation of $\boldsymbol{B}$ and $\boldsymbol{C}$:**
- These parameters control the flow of information: $\boldsymbol{B}$ determines whether an input xt is incorporated into the state ht, and $\boldsymbol{C}$ governs how the state influences the output yt.
- Making $\boldsymbol{B}$ and $\boldsymbol{C}$ selective allows finer modulation of the recurrent dynamics based on both the current input and the context stored in the hidden state.



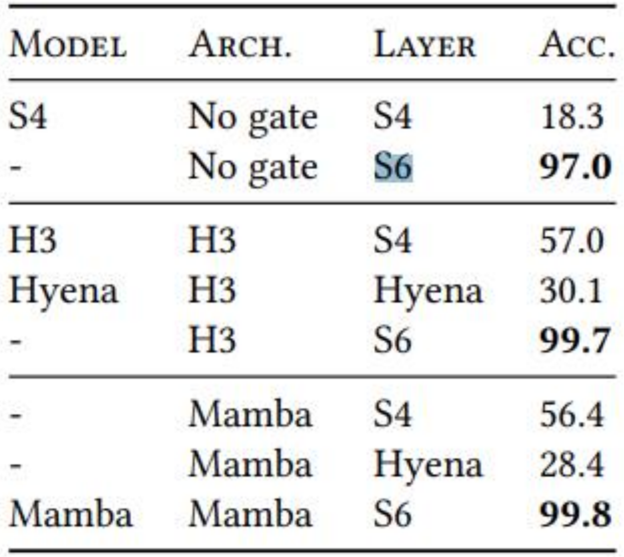
Evaluation

Selective Copying

- Copying evaluates a model's ability to memorize and reproduce sequences.
    - Linear Time-Invariant (LTI) State Space Models (SSMs), which use linear recurrences and global convolutions, can solve this task without true memorization—they can exploit fixed-length convolutions that just count time.

- The Selective Copying task introduces randomized spacing between tokens, making simple convolutional tricks ineffective.

•**Previous Claim:** Some researchers argue that gating mechanisms (like those in gated recurrent units or attention mechanisms) enable data-dependent modeling and improve performance.
•**Counterargument in This Paper:** Gating does not interact along the sequence axis, and cannot affect the spacing between token



MODEL	ARCH.	LAYER	ACC.
S4	No gate	S4	18.3
-	No gate	S6	**97.0**
H3	H3	S4	57.0
Hyena	H3	Hyena	30.1
-	H3	S6	**99.7**
-	Mamba	S4	56.4
-	Mamba	Hyena	28.4
Mamba	Mamba	S6	**99.8**

Table 1: (**Selective Copying**.)
Accuracy for combinations of architectures and inner sequence layers.

UNIVERSITY of VIRGINIA

Evaluation

Induction Heads

- To perform associative recall and copy.
  - if the model has seen a bigram such as “Harry Potter” in the sequence--> When “Harry” appears model should be able to predict “Potter”

- Dataset:
  - Models trained on the induction heads task at sequence length 256,
  - with a vocab size of 16
  - Evaluated sequence lengths from 2^6 to 2^20 at test time.

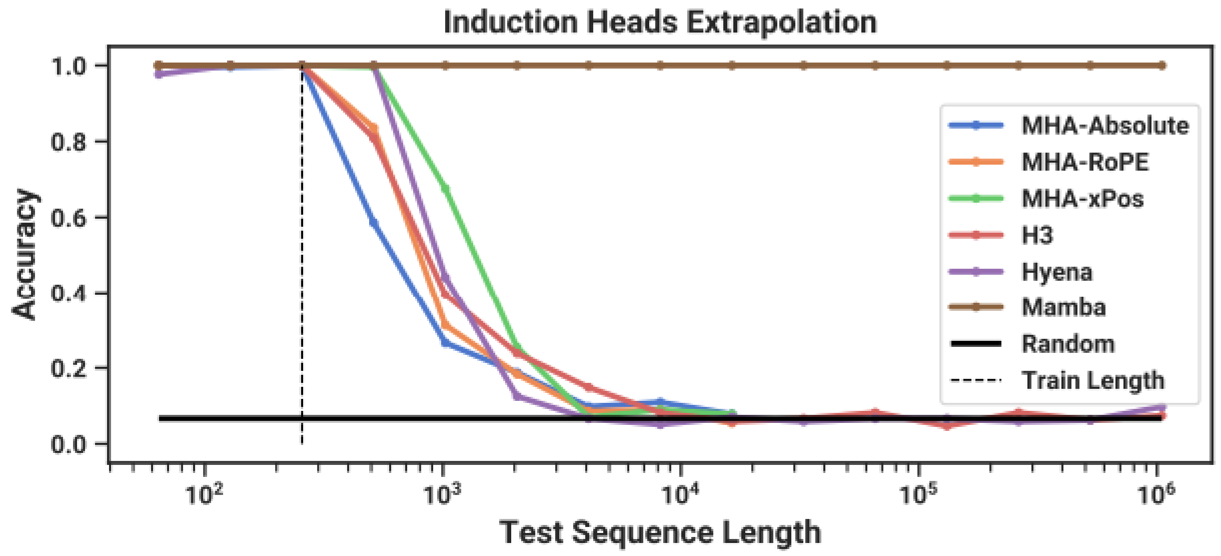
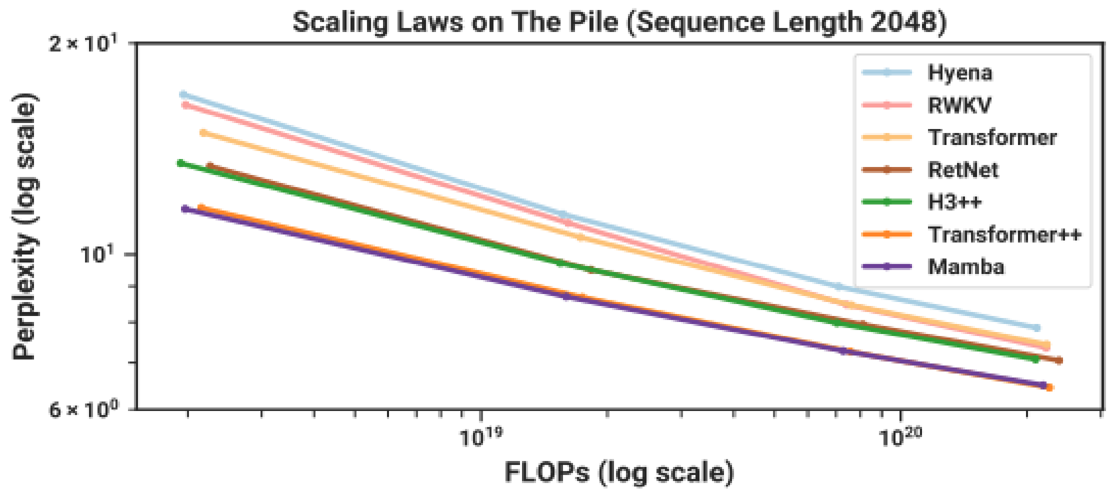
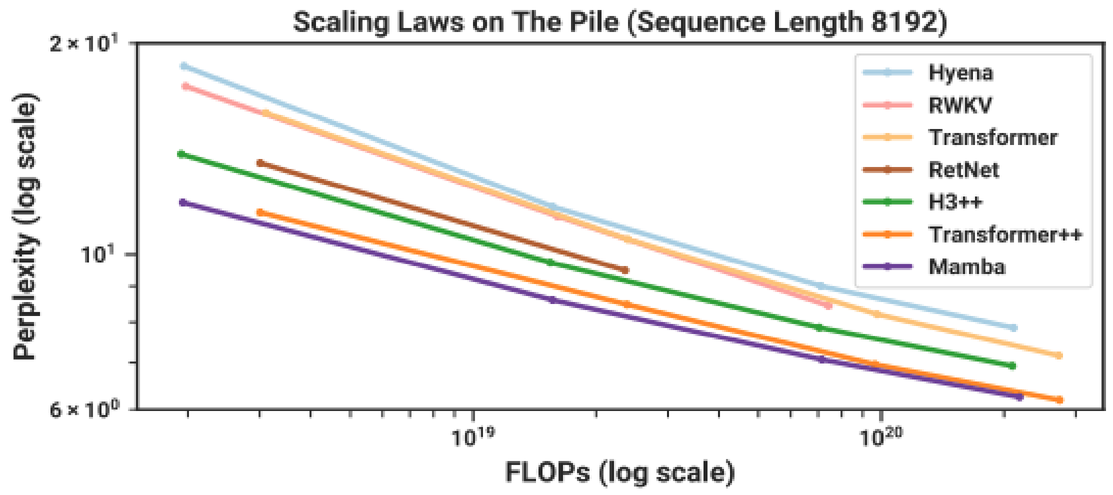


Table 2: (**Induction Heads**.) Models are trained on sequence length $2^8 = 256$, and tested on increasing sequence lengths of $2^6 = 64$ up to $2^{20} = 1048576$. Full numbers in Table 11.

- Models:
  - 2 layered models with variants of MHA and SSM
  - Mamba uses a 64-dimensional representation whereas MHA used D=128

Evaluation





Scaling Laws

- Evaluated on models ranging from ≈125M to ≈1.3B parameters.

Performance

- First attention-free model to match Transformer++ performance.
- Performs particularly well as sequence length increases.

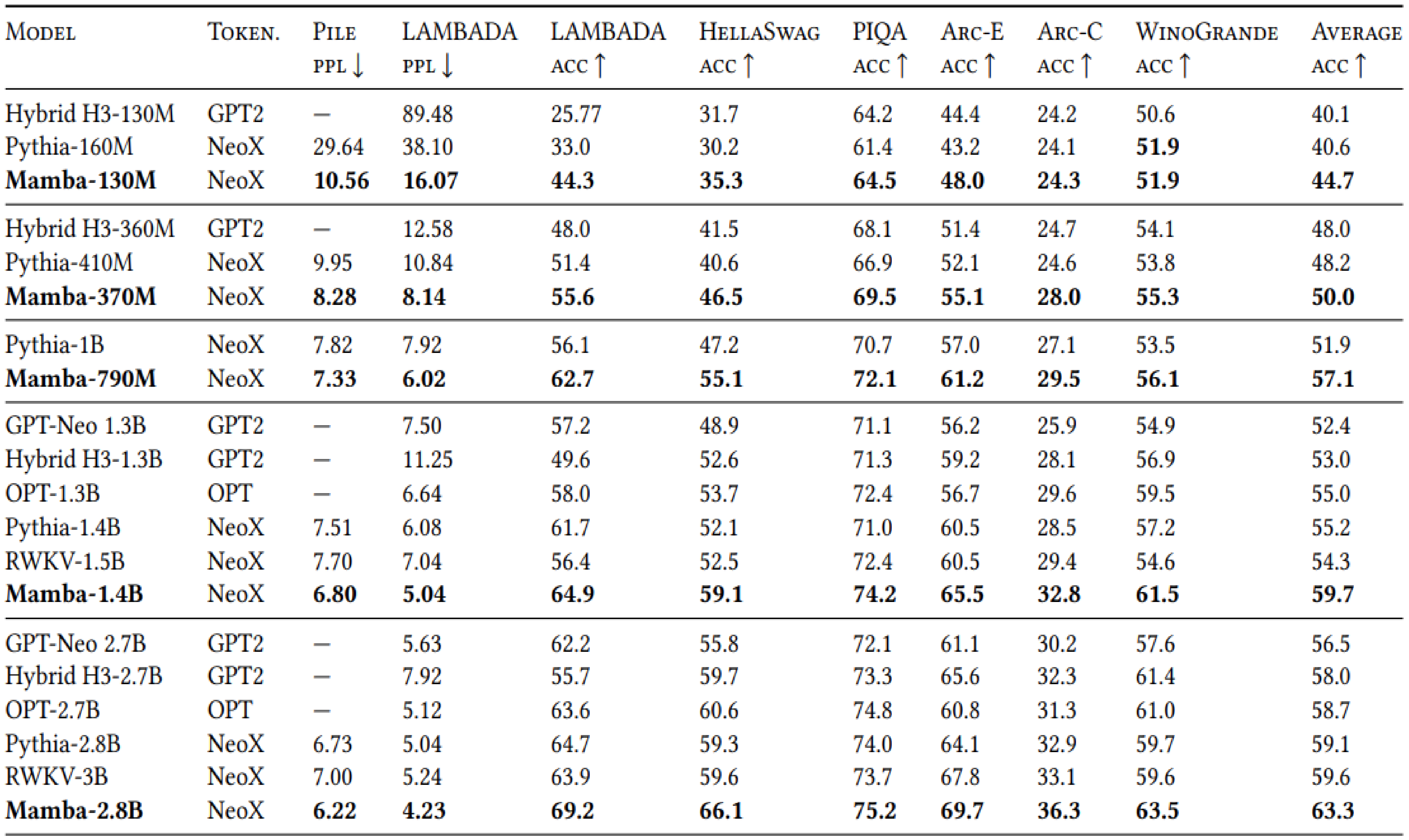
Evaluation

Downstream zero-shot evaluation

- All models trained with same tokenizer, dataset, and training length (300B tokens).

- Context length differences:
    - Mamba & Pythia: Trained with context length 2048.
    - RWKV: Trained with context length 1024.

Mamba is best-in-class on every single evaluation result



Model	Token.	Pile ppl ↓	LAMBADA ppl ↓	LAMBADA acc ↑	HellaSwag acc ↑	PIQA acc ↑	Arc-E acc ↑	Arc-C acc ↑	WinoGrande acc ↑	Average acc ↑
Hybrid H3-130M	GPT2	—	89.48	25.77	31.7	64.2	44.4	24.2	50.6	40.1
Pythia-160M	NeoX	29.64	38.10	33.0	30.2	61.4	43.2	24.1	**51.9**	40.6
Mamba-130M	NeoX	**10.56**	**16.07**	**44.3**	**35.3**	**64.5**	**48.0**	**24.3**	**51.9**	**44.7**
Hybrid H3-360M	GPT2	—	12.58	48.0	41.5	68.1	51.4	24.7	54.1	48.0
Pythia-410M	NeoX	9.95	10.84	51.4	40.6	66.9	52.1	24.6	53.8	48.2
Mamba-370M	NeoX	**8.28**	**8.14**	**55.6**	**46.5**	**69.5**	**55.1**	**28.0**	**55.3**	**50.0**
Pythia-1B	NeoX	7.82	7.92	56.1	47.2	70.7	57.0	27.1	53.5	51.9
Mamba-790M	NeoX	**7.33**	**6.02**	**62.7**	**55.1**	**72.1**	**61.2**	**29.5**	**56.1**	**57.1**
GPT-Neo 1.3B	GPT2	—	7.50	57.2	48.9	71.1	56.2	25.9	54.9	52.4
Hybrid H3-1.3B	GPT2	—	11.25	49.6	52.6	71.3	59.2	28.1	56.9	53.0
OPT-1.3B	OPT	—	6.64	58.0	53.7	72.4	56.7	29.6	59.5	55.0
Pythia-1.4B	NeoX	7.51	6.08	61.7	52.1	71.0	60.5	28.5	57.2	55.2
RWKV-1.5B	NeoX	7.70	7.04	56.4	52.5	72.4	60.5	29.4	54.6	54.3
Mamba-1.4B	NeoX	**6.80**	**5.04**	**64.9**	**59.1**	**74.2**	**65.5**	**32.8**	**61.5**	**59.7**
GPT-Neo 2.7B	GPT2	—	5.63	62.2	55.8	72.1	61.1	30.2	57.6	56.5
Hybrid H3-2.7B	GPT2	—	7.92	55.7	59.7	73.3	65.6	32.3	61.4	58.0
OPT-2.7B	OPT	—	5.12	63.6	60.6	74.8	60.8	31.3	61.0	58.7
Pythia-2.8B	NeoX	6.73	5.04	64.7	59.3	74.0	64.1	32.9	59.7	59.1
RWKV-3B	NeoX	7.00	5.24	63.9	59.6	73.7	67.8	33.1	59.6	59.6
Mamba-2.8B	NeoX	**6.22**	**4.23**	**69.2**	**66.1**	**75.2**	**69.7**	**36.3**	**63.5**	**63.3**

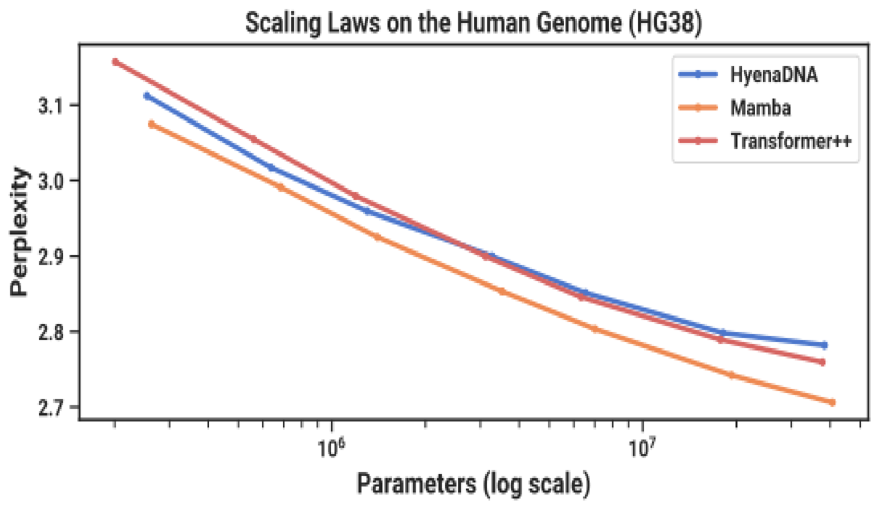
Evaluation

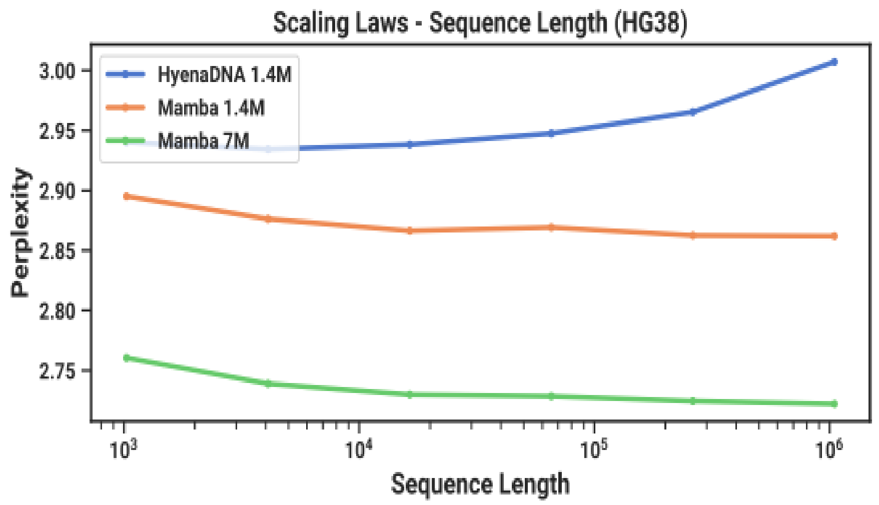
DNA Modeling

•Investigating Mamba as a backbone for both pretraining and fine-tuning in genomics.

•Evaluation performed in the same setting as recent long-sequence DNA models (Nguyen, Poli, et al. 2023).

•Dataset: HG38 human genome dataset, comprising about 4.5 billion tokens (DNA base pairs), following the HyenaDNA setup.





Scaling: Model Size

- Mamba's pretraining perplexity improves smoothly with model size.
- At ≈40M parameters, Mamba matches Transformer++ and HyenaDNA with roughly 3×–4× fewer parameters.

Scaling: Model Size

•Mamba benefits from longer context lengths
•HyenaDNA (LTI) model degrades in performance with increasing sequence length as it gathers information from all.

Evaluation

DNA Modeling

- Downstream task of classifying between 5 different species (share 99% of their DNA)
- Using randomly sampled contiguous segment of their DNA

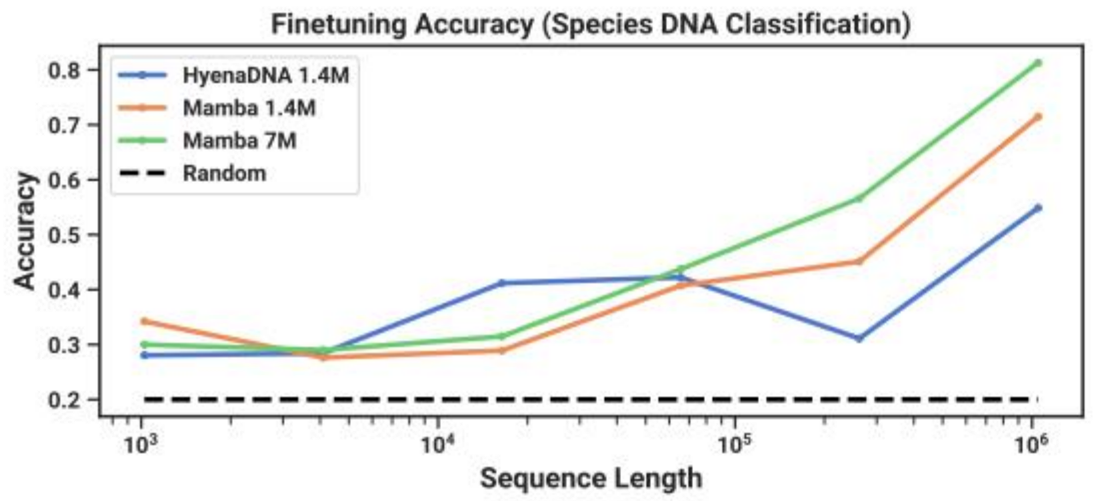


Figure 6: (**Great Apes DNA Classification**.) Accuracy after fine-tuning on sequences of length $2^{10} = 1024$ up to $2^{20} = 1048576$ using pretrained models of the same context length. Numerical results in Table 13.

Evaluation

Audio Modeling

- Evaluated pretraining quality using autoregressive next-sample prediction on YouTubeMix (DeepSound 2017).

- The main metric is bits per byte (BPB), which is a constant factor log(2) of the standard negative log-likelihood (NLL) loss for pretraining other modalities

Results:

•Both show consistent improvement with longer context lengths.

•Mamba outperforms SaShiMi .

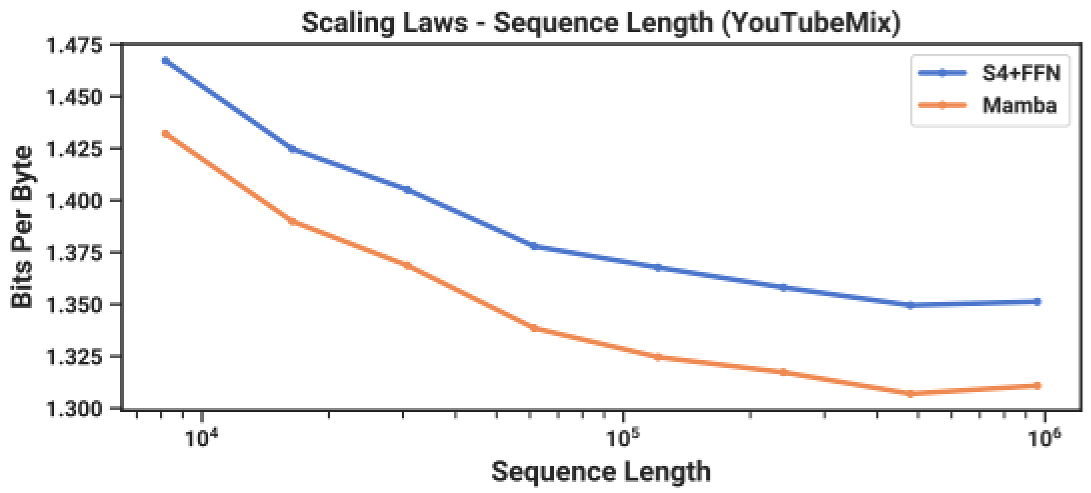


Figure 7: (**Audio Pretraining.**) Mamba improves performance over prior state-of-the-art (Sashimi) in autoregressive audio modeling, while improving up to minute-long context or million-length sequences (controlling for computation).

Evaluation

Audio Modeling

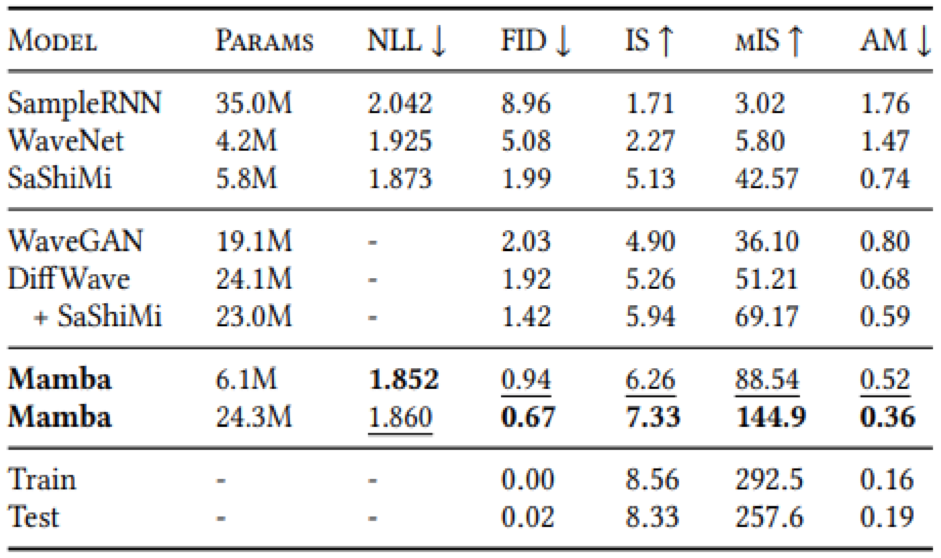
Dataset:
- SC09 is a benchmark speech generation dataset.
- 1-second clips sampled at 16000 Hz of the digits “zero” through “nine” with highly variable characteristics

Results:

•A small Mamba model outperforms the state-of-the-art GAN- and diffusion-based models, despite the latter being much larger.

•A larger Mamba model (parameter-matched to the baselines) significantly improves the metrics.

Table 4: (**SC09**) Automated metrics for unconditional generation or a challenging dataset of fixed-length speech clips. (*Top to Bottom* Autoregressive baselines, non-autoregressive baselines, Mamba, anc dataset metrics.



Model	Params	NLL ↓	FID ↓	IS ↑	mIS ↑	AM ↓
SampleRNN	35.0M	2.042	8.96	1.71	3.02	1.76
WaveNet	4.2M	1.925	5.08	2.27	5.80	1.47
SaShiMi	5.8M	1.873	1.99	5.13	42.57	0.74
WaveGAN	19.1M	-	2.03	4.90	36.10	0.80
DiffWave	24.1M	-	1.92	5.26	51.21	0.68
+ SaShiMi	23.0M	-	1.42	5.94	69.17	0.59
Mamba	6.1M	**1.852**	0.94	6.26	88.54	0.52
Mamba	24.3M	1.860	**0.67**	**7.33**	**144.9**	**0.36**
Train	-	-	0.00	8.56	292.5	0.16
Test	-	-	0.02	8.33	257.6	0.19

Evaluation

Audio Modeling

Combinations of different architectures for the outer stages and center stage in Sashimi

- In the outer blocks: Mamba consistently performs better than S4+MLP.
- In the center blocks: Performance ranking is Mamba > S4+MLP > MHA+MLP.

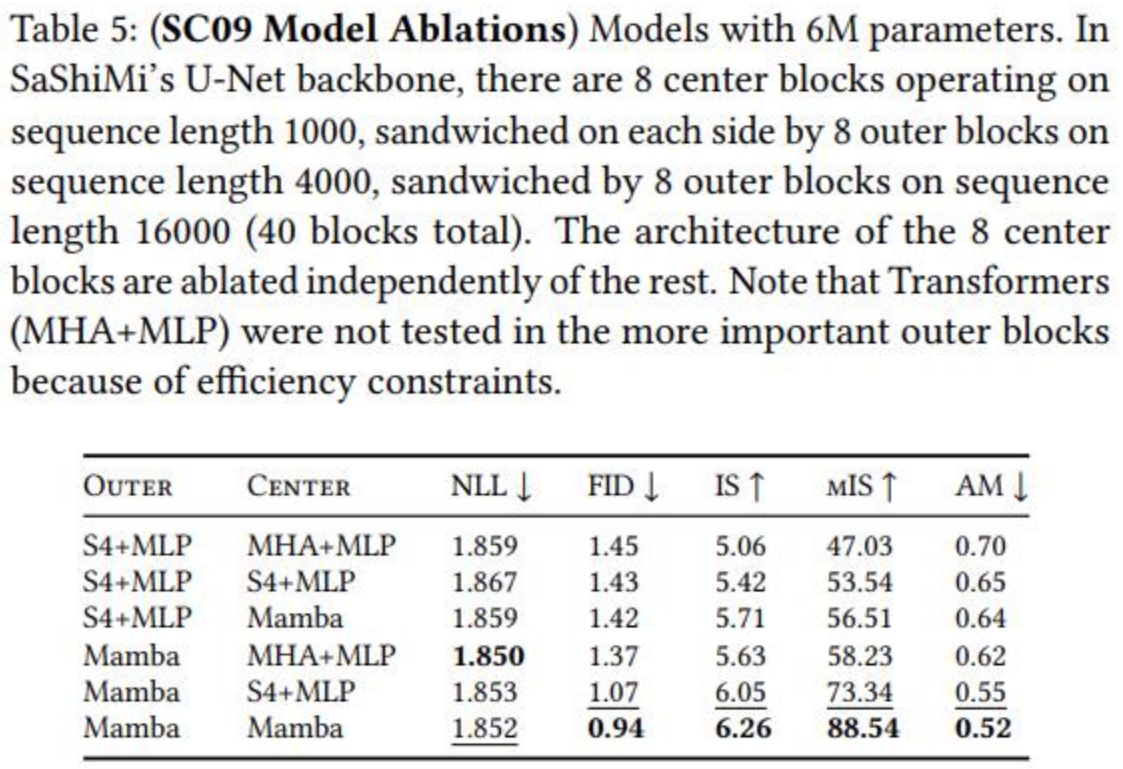


Table 5: (**SC09 Model Ablations**) Models with 6M parameters. In SaShiMi's U-Net backbone, there are 8 center blocks operating on sequence length 1000, sandwiched on each side by 8 outer blocks on sequence length 4000, sandwiched by 8 outer blocks on sequence length 16000 (40 blocks total). The architecture of the 8 center blocks are ablated independently of the rest. Note that Transformers (MHA+MLP) were not tested in the more important outer blocks because of efficiency constraints.

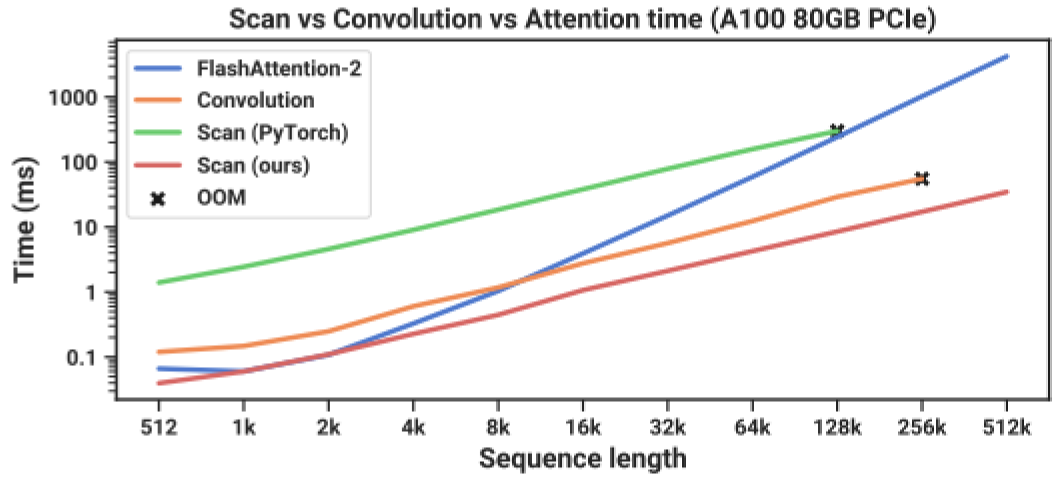
Outer	Center	NLL ↓	FID ↓	IS ↑	mIS ↑	AM ↓
S4+MLP	MHA+MLP	1.859	1.45	5.06	47.03	0.70
S4+MLP	S4+MLP	1.867	1.43	5.42	53.54	0.65
S4+MLP	Mamba	1.859	1.42	5.71	56.51	0.64
Mamba	MHA+MLP	**1.850**	1.37	5.63	58.23	0.62
Mamba	S4+MLP	1.853	1.07	6.05	73.34	0.55
Mamba	Mamba	1.852	**0.94**	**6.26**	**88.54**	**0.52**

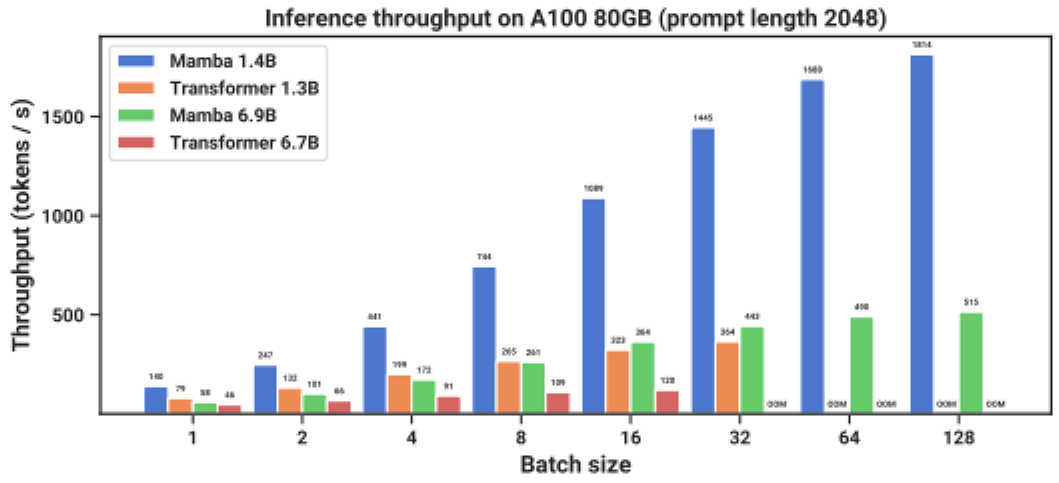
Evaluation

Speed and Memory

- The efficient SSM scan is faster than FlashAttention-2 (Dao 2024) when processing sequence lengths beyond 2K.

- Mamba achieves 4–5× higher inference throughput compared to a Transformer of similar size.





Evaluation

Ablations

Table 6: (**Ablations: Architecture and SSM layer.**) The Mamba block performs similarly to H3 while being simpler. In the inner layer, there is little difference among different parameterizations of LTI models, while selective SSMs (S6) provide a large improvement. More specifically, the S4 (real) variant is S4D-Real and the S4 (complex) variant is S4D-Lin.

MODEL	ARCH.	SSM LAYER	PERPLEXITY
Hyena	H3	Hyena	10.24
H3	H3	S4 (complex)	10.30
-	H3	S4 (real)	10.34
-	H3	S6	**8.95**

MODEL	ARCH.	SSM LAYER	PERPLEXITY
-	Mamba	Hyena	10.75
-	Mamba	S4 (complex)	10.54
-	Mamba	S4 (real)	10.56
Mamba	Mamba	S6	**8.69**

Table 7: (**Ablations: Selective parameters.**) Δ is the most important parameter (Theorem 1), but using multiple selective parameters together synergizes.

SELECTIVE Δ	SELECTIVE B	SELECTIVE C	PERPLEXITY
✗	✗	✗	10.93
✗	✓	✗	10.15
✗	✗	✓	9.98
✓	✗	✗	9.81
✓	✓	✓	8.71

Table 8: (**Ablations: Parameterization of A.**) The more standard initializations based on S4D-Lin (Gu, Gupta, et al. 2022) perform worse than S4D-Real or a random initialization, when the SSM is selective.

A_n INITIALIZATION	FIELD	PERPLEXITY
$A_n = -\frac{1}{2} + ni$	Complex	9.16
$A_n = -1/2$	Real	8.85
$A_n = -(n+1)$	Real	8.71
$A_n \sim \exp(\mathcal{N}(0,1))$	Real	8.71

Evaluation

Ablations

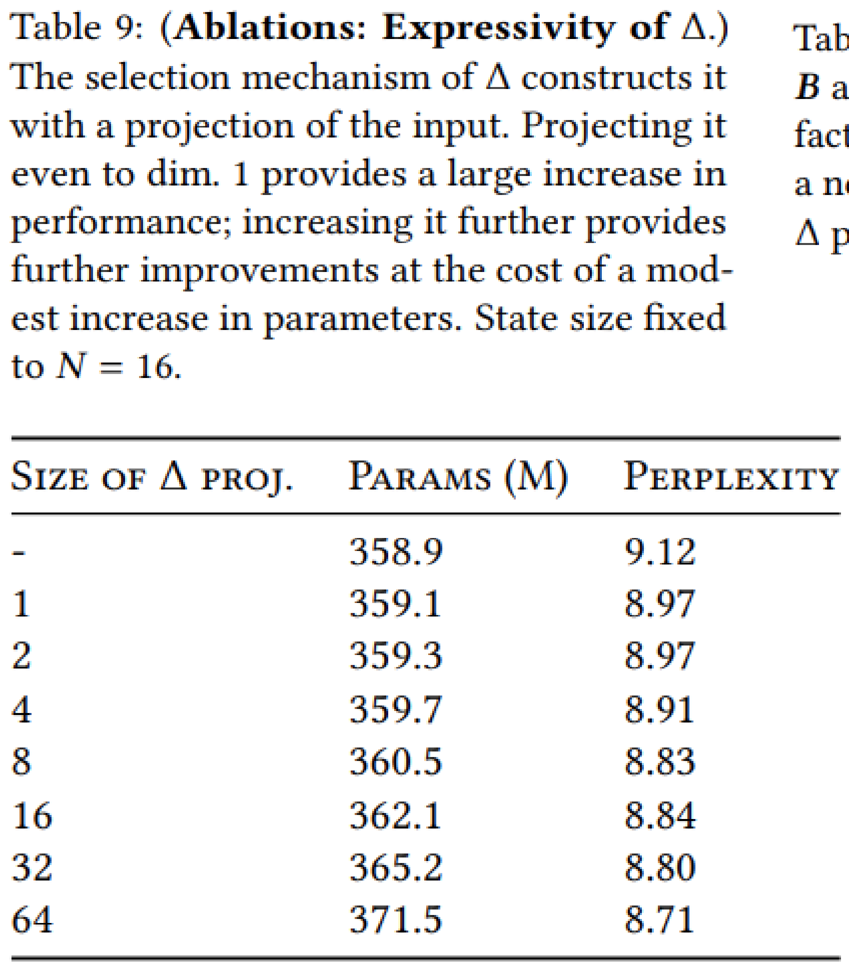


Table 9: (**Ablations: Expressivity of Δ.**) The selection mechanism of Δ constructs it with a projection of the input. Projecting it even to dim. 1 provides a large increase in performance; increasing it further provides further improvements at the cost of a modest increase in parameters. State size fixed to $N = 16$.

Size of Δ proj.	Params (M)	Perplexity
-	358.9	9.12
1	359.1	8.97
2	359.3	8.97
4	359.7	8.91
8	360.5	8.83
16	362.1	8.84
32	365.2	8.80
64	371.5	8.71

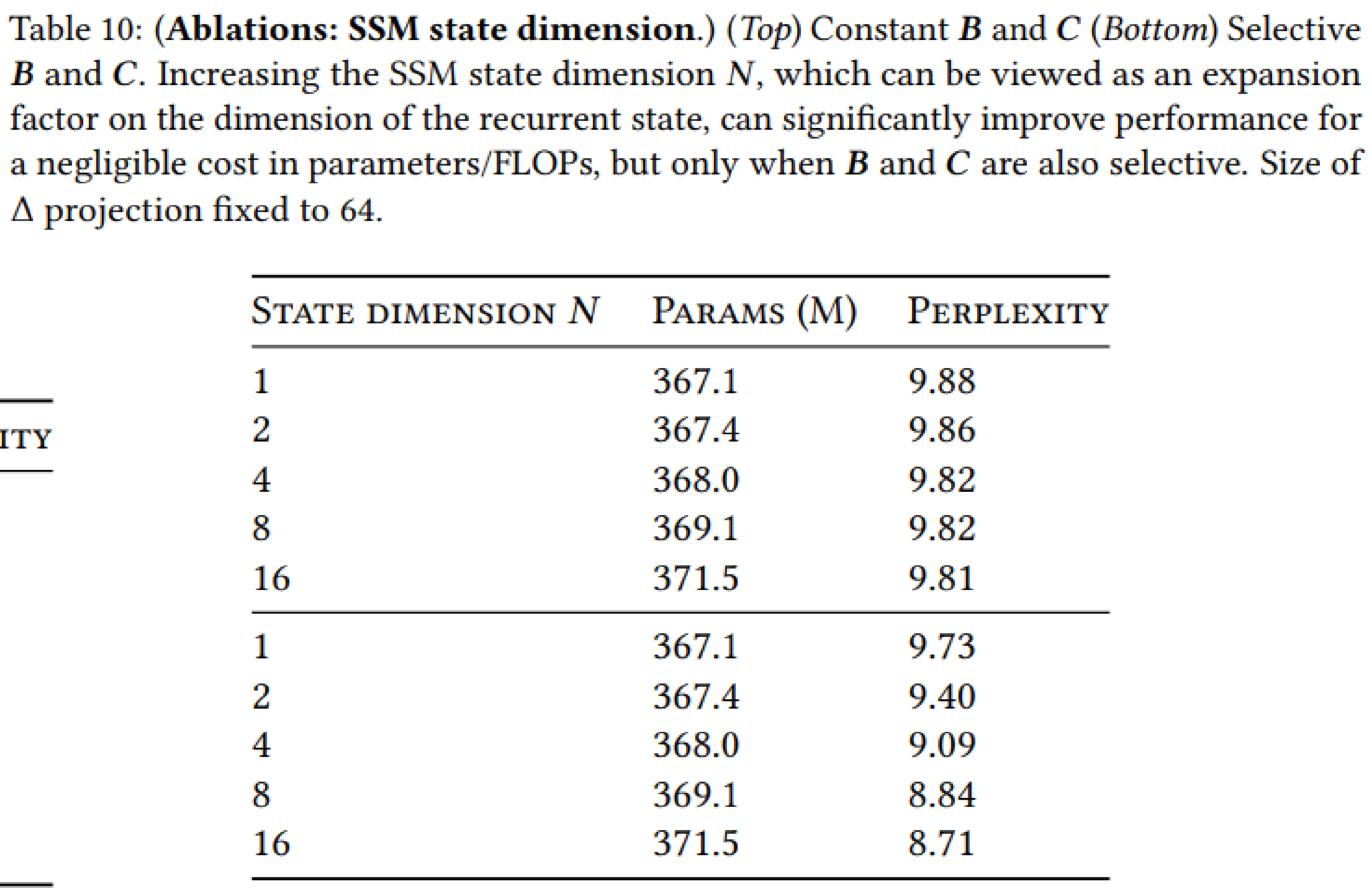


Table 10: (**Ablations: SSM state dimension.**) (*Top*) Constant ***B*** and ***C*** (*Bottom*) Selective ***B*** and ***C***. Increasing the SSM state dimension N, which can be viewed as an expansion factor on the dimension of the recurrent state, can significantly improve performance for a negligible cost in parameters/FLOPs, but only when ***B*** and ***C*** are also selective. Size of Δ projection fixed to 64.

State dimension N	Params (M)	Perplexity
1	367.1	9.88
2	367.4	9.86
4	368.0	9.82
8	369.1	9.82
16	371.5	9.81
1	367.1	9.73
2	367.4	9.40
4	368.0	9.09
8	369.1	8.84
16	371.5	8.71

Limitations

Tradeoff Between Modalities:
While the selection mechanism improves performance on discrete data (like text or DNA), it may reduce performance on continuous modalities (such as audio or video) where traditional LTI SSMs excel.

Downstream Integration and Affordances:
Unlike Transformer-based models that support a wide range of interactions (fine-tuning, prompting, in-context learning, etc.), it remains unclear whether SSMs (or Mamba) can offer similar downstream benefits.

Scaling Challenges:
The empirical evaluation has been limited to small model sizes. It is uncertain if the model will maintain favorable performance when scaled to larger sizes (e.g., 7B+ parameters) like other leading models, and scaling may introduce additional engineering challenges not addressed in the paper.

Conclusion

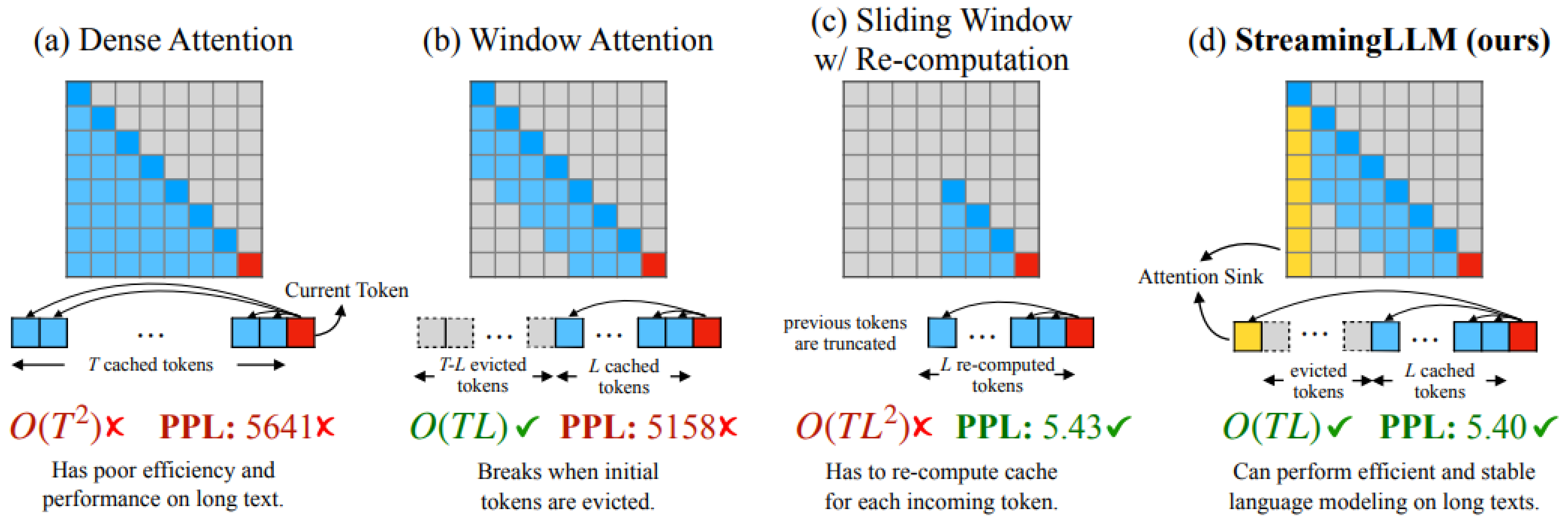
- Introduced a selection mechanism for structured state space models that enables context-dependent reasoning while scaling linearly with sequence length.
- Incorporated into a simple attention-free architecture, enabling efficient computation.
- Mamba achieves state-of-the-art results across diverse domains, matching or exceeding the performance of strong Transformer models.

Paper 2

Efficient Streaming Language Models with Attention Sinks

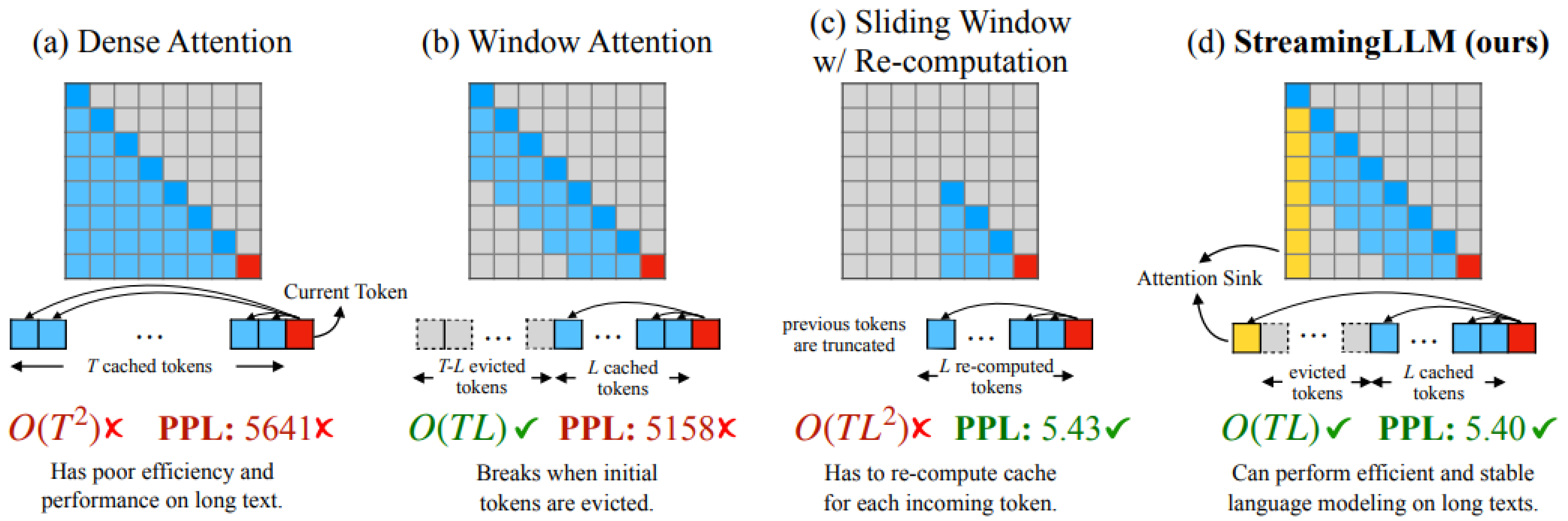
Challenges in Deploying LLMs for Streaming Applications

- **Memory Overhead** – Caching previous tokens' Key and Value (KV) states consumes excessive memory.
- **Limited Extrapolation** – LLMs cannot generalize to longer sequences than their training window.
- **Window Attention Limitation** – Caching only recent tokens leads to performance degradation when the sequence exceeds the cache size.



Challenges in Deploying LLMs for Streaming Applications

- **L**: The length of the text sequences the language model was pre-trained on. This determines the size of the context window the model was originally optimized for.
- **T**: The position of the token being predicted during inference. Since inference often involves generating tokens far beyond the pre-training length L, we typically assume T≫L, meaning we are dealing with much longer sequences.



(a) Dense Attention
Current Token
T cached tokens
O(T²)✗ PPL: 5641✗
Has poor efficiency and performance on long text.
(b) Window Attention
T-L evicted tokens
L cached tokens
O(TL)✓ PPL: 5158✗
Breaks when initial tokens are evicted.
(c) Sliding Window w/ Re-computation
previous tokens are truncated
L re-computed tokens
O(TL²)✗ PPL: 5.43✓
Has to re-compute cache for each incoming token.
(d) StreamingLLM (ours)
Attention Sink
evicted tokens
L cached tokens
O(TL)✓ PPL: 5.40✓
Can perform efficient and stable language modeling on long texts.

Challenges in Deploying LLMs for Streaming Applications

•**Dense Attention (Full Attention)**

Complexity:O(T^2)

This method attends to all previous tokens, leading to a quadratic time complexity in sequence length T.

It also has increasing memory demands due to a growing cache size.

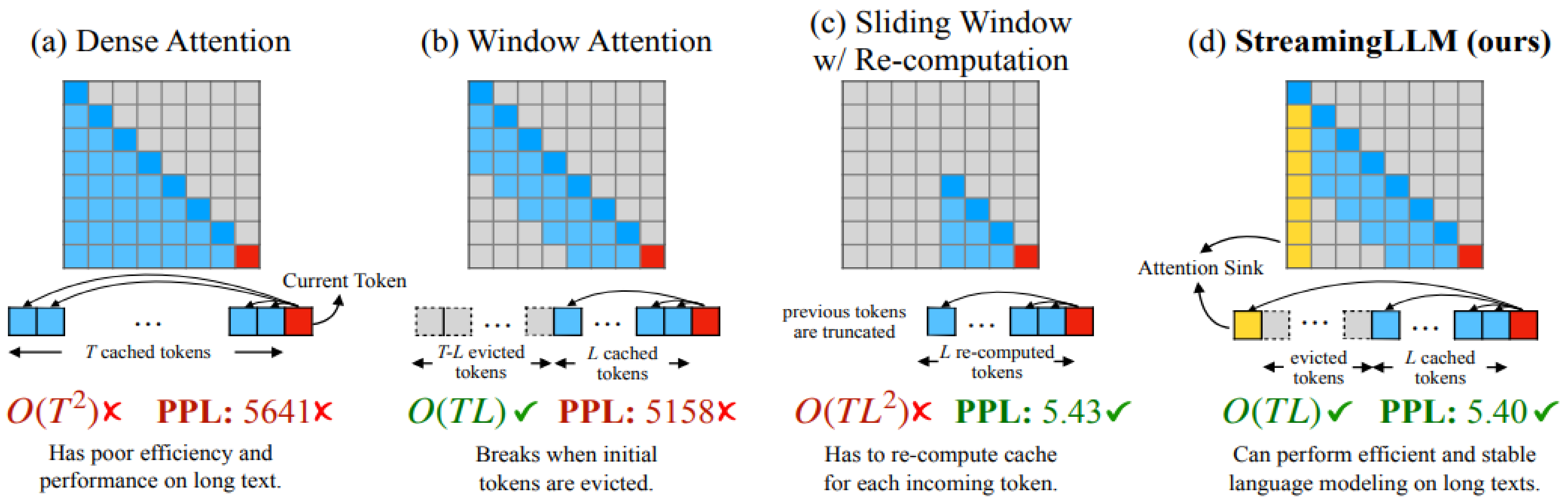
Performance drops when T>L because the model struggles with longer contexts.

•**Window Attention**

Complexity: O(TL)

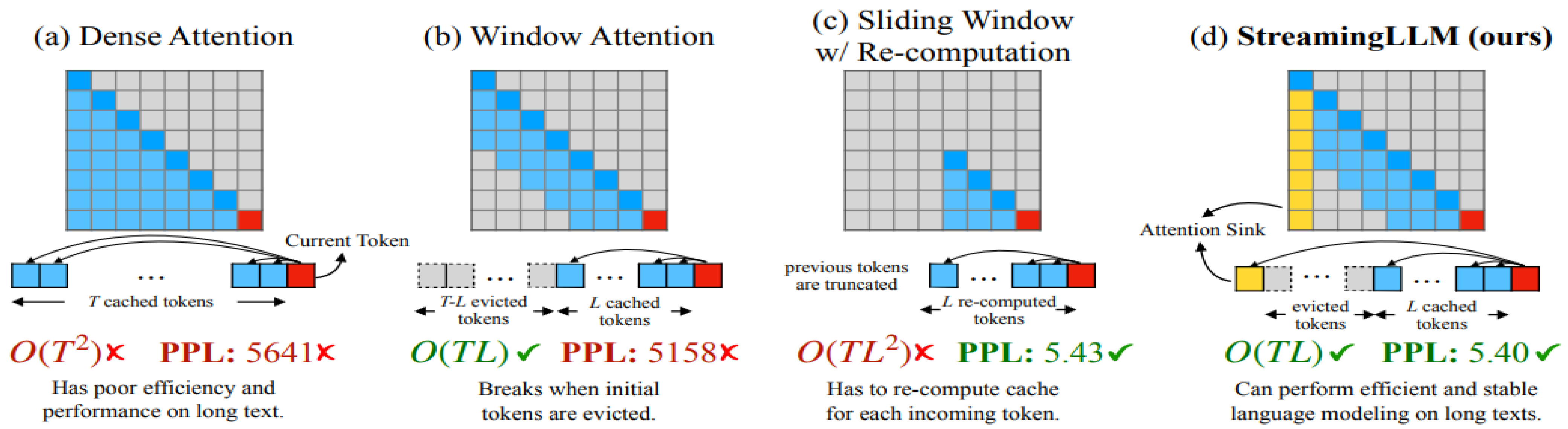
Only the most recent L tokens are cached for Key-Value (KV) storage.

While it is efficient in inference, performance declines sharply when the early tokens are evicted.


(a) Dense Attention
Current Token
T cached tokens
$O(T^2)$✗ PPL: 5641✗
Has poor efficiency and performance on long text.
(b) Window Attention
T-L evicted tokens
L cached tokens
$O(TL)$✓ PPL: 5158✗
Breaks when initial tokens are evicted.
(c) Sliding Window w/ Re-computation
previous tokens are truncated
L re-computed tokens
$O(TL^2)$✗ PPL: 5.43✓
Has to re-compute cache for each incoming token.
(d) StreamingLLM (ours)
Attention Sink
evicted tokens
L cached tokens
$O(TL)$✓ PPL: 5.40✓
Can perform efficient and stable language modeling on long texts.

Challenges in Deploying LLMs for Streaming Applications

•**Sliding Window with Re-computation**
Complexity: O(TL^2)
Instead of storing all KV pairs, this method reconstructs the KV cache using the most recent L tokens each time a new token is generated.
This enables better performance over long texts but suffers from significant computational overhead due to the quadratic O(L^2) recomputation.
•**StreamingLLM**
Maintains an "attention sink" (a small number of initial tokens) while keeping recent tokens for stable attention computation.
Avoids full recomputation and preserves long-term dependencies more effectively than windowed methods.



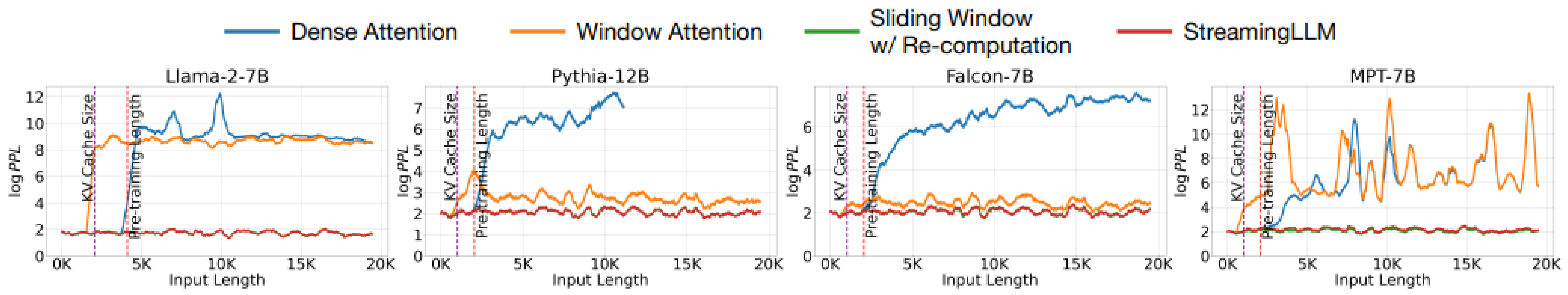
Existing Approaches and Attention Sink

Dense Attention Fails Beyond the Pretraining Window

In standard Transformer models with dense attention, each token attends to all previous tokens within a fixed window (e.g., 4K or 8K tokens in models like GPT-4 or LLaMA).

Issue: When the sequence length exceeds this pretraining window, the model struggles because it was not trained to handle such long-range dependencies.

Result: Perplexity increases sharply beyond this limit, meaning the model's predictions become significantly worse.

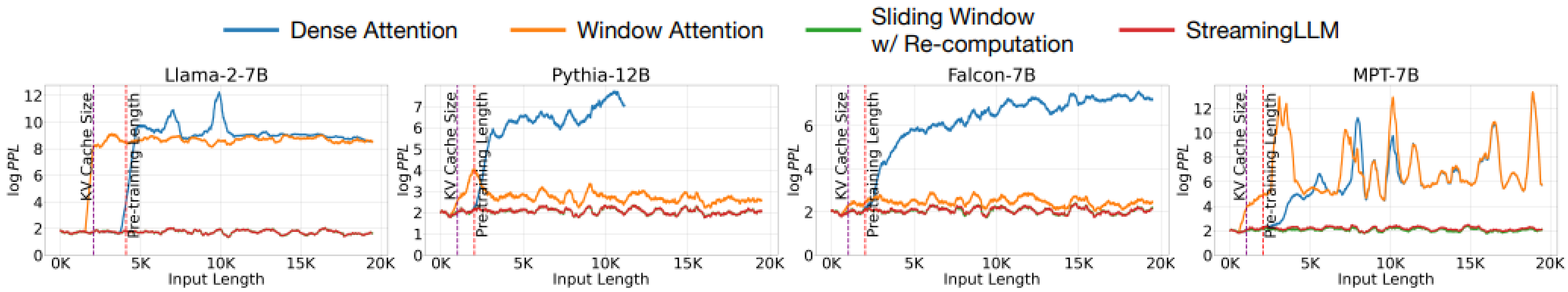


Existing Approaches and Attention Sink

Window Attention Collapses When Cache is Full
Keeps only the most recent tokens in a sliding window, discarding older tokens once the cache is full.

Issue: When the sequence length exceeds the cache size, the earliest tokens are evicted (i.e., their key-value (KV) pairs are removed from memory).

Finding: Perplexity spikes sharply the moment the first token is evicted, showing that even distant tokens are crucial for stable predictions.



Existing Approaches and Attention Sink

Why Removing Initial Tokens Breaks LLMs:

- Visualizing attention maps of Llama-2-7B reveals that, beyond the bottom layers, attention consistently focuses on initial tokens.
- Removing these tokens disrupts the SoftMax denominator, shifting attention score distributions.

Importance of Initial Tokens:

- Experiments replacing the first four tokens with linebreaks indicate that the absolute position of these tokens matters more than their semantic meaning.
- Reintroducing them restores perplexity to normal levels.

"Attention Sink" Phenomenon

- Initial tokens act as **"attention sinks"**, receiving high attention scores regardless of semantic importance.
- Removing these initial tokens disrupts attention distribution, causing performance degradation.

Existing Approaches and Attention Sink

	The	quick	brown	fox	jumps	over	the	lazy	dog
The	🔵	🔵	🔵	🔵	🔵	🔵	🔵	🔵	🔵
quick	🔵	🔵	🔵	🔵	🔵	🔵	🔵	🔵	🔵
brown	🔵	🔵	🔵	🔵	🔵	🔵	🔵	🔵	🔵
fox	🔵	🔵	🔵	🔵	🔵	🔵	🔵	🔵	🔵
jumps	🔵	🔵	🔵	🔵	🔴	🔵	🔵	🔵	🔵
over	🔵	🔵	🔵	🔵	🔵	🔴	🔵	🔵	🔵
the	🔴	🔵	🔵	🔵	🔵	🔵	🔵	🔵	🔵
lazy	🔴	🔵	🔵	🔵	🔵	🔵	🔵	🔴	🔵
dog	🔴	🔵	🔵	🔵	🔵	🔵	🔵	🔵	🔴

- 🔴 (High attention score)
- 🔵 (Low attention score)

Existing Approaches and Attention Sink

- Layer (0,1): Attention primarily focuses on recent tokens, exhibiting a "local attention" pattern.
- Deeper Layers: Attention shifts disproportionately toward the initial token, even when it lacks semantic significance

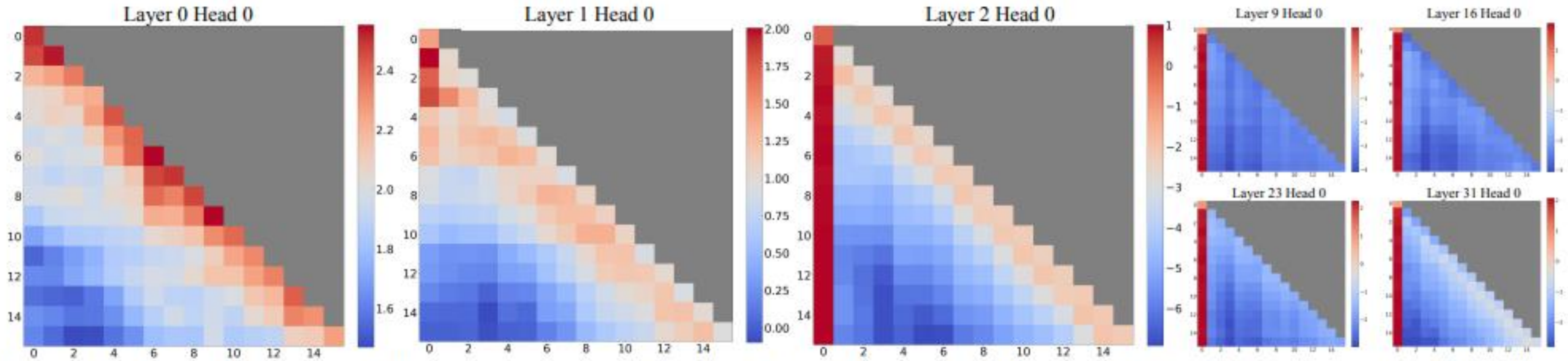


Figure 2: Visualization of the *average* attention logits in Llama-2-7B over 256 sentences, each with a length of 16. Observations include: (1) The attention maps in the first two layers (layers 0 and 1) exhibit the "local" pattern, with recent tokens receiving more attention. (2) Beyond the bottom two layers, the model heavily attends to the initial token across all layers and heads.

Proposed Approach

1. Attention sinks (four initial tokens)

To stabilize the attention computation;

- **Sink Token**: A trainable global token explicitly assigned to absorb unnecessary attention.
- **Zero Sink**: Replacing SoftMax with SoftMax-off-by-One, which does not enforce attention scores summing to one.

$$\text{SoftMax}_1(x)_i = \frac{e^{x_i}}{1 + \sum_{j=1}^{N} e^{x_j}},$$

Proposed Approach

2. Rolling KV Cache retains the most recent tokens, crucial for language modeling.

When determining the relative distance and adding positional information to tokens, StreamingLLM focuses on positions within the cache rather than those in the original text.

For instance, if the current cache (Figure 4) has tokens [0, 1, 2, 3, 6, 7, 8] and is in the process of decoding the 9th token, the positions assigned are [0, 1, 2, 3, 4, 5, 6, 7], rather than the positions in the original text, which would be [0, 1, 2, 3, 6, 7, 8, 9]

StreamingLLM' design is versatile and can be seamlessly incorporated into any autoregressive language model that employs relative positional encoding, such as RoPE and ALiBi

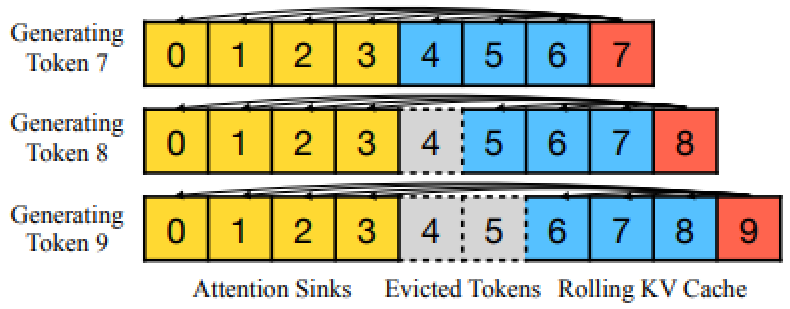


Figure 4: The KV cache of StreamingLLM.

Experimental Validation

- Three LLMs (160M parameters) trained under identical settings:
  - **Vanilla Model**: Uses standard SoftMax.
  - **Zero Sink Model**: Uses SoftMax-off-by-One.
  - **Sink Token Model**: Introduces a learnable placeholder token.
- **Results (Table 3)**:
  - **Zero Sink helps**, but initial tokens still act as attention sinks.
  - **Sink Token is highly effective**, stabilizing attention and improving evaluation perplexity.

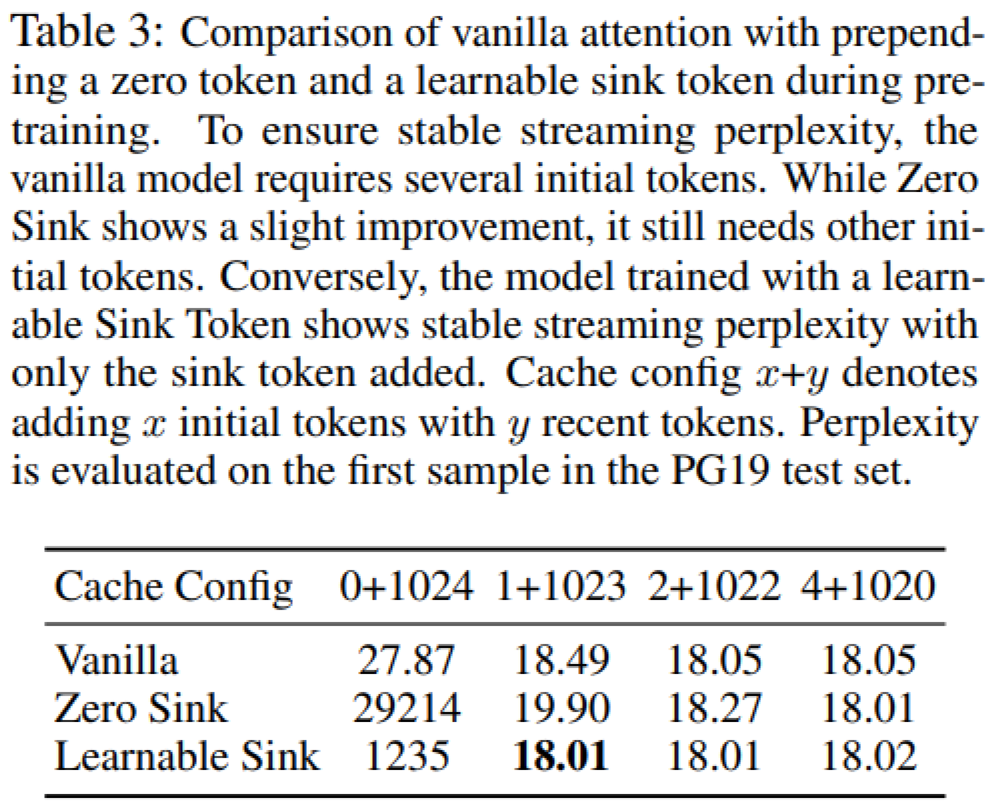


Table 3: Comparison of vanilla attention with prepending a zero token and a learnable sink token during pre-training. To ensure stable streaming perplexity, the vanilla model requires several initial tokens. While Zero Sink shows a slight improvement, it still needs other initial tokens. Conversely, the model trained with a learnable Sink Token shows stable streaming perplexity with only the sink token added. Cache config x+y denotes adding x initial tokens with y recent tokens. Perplexity is evaluated on the first sample in the PG19 test set.

Cache Config	0+1024	1+1023	2+1022	4+1020
Vanilla	27.87	18.49	18.05	18.05
Zero Sink	29214	19.90	18.27	18.01
Learnable Sink	1235	**18.01**	18.01	18.02

Experiment: Language modelling on long texts

Dataset & Cache Setup:

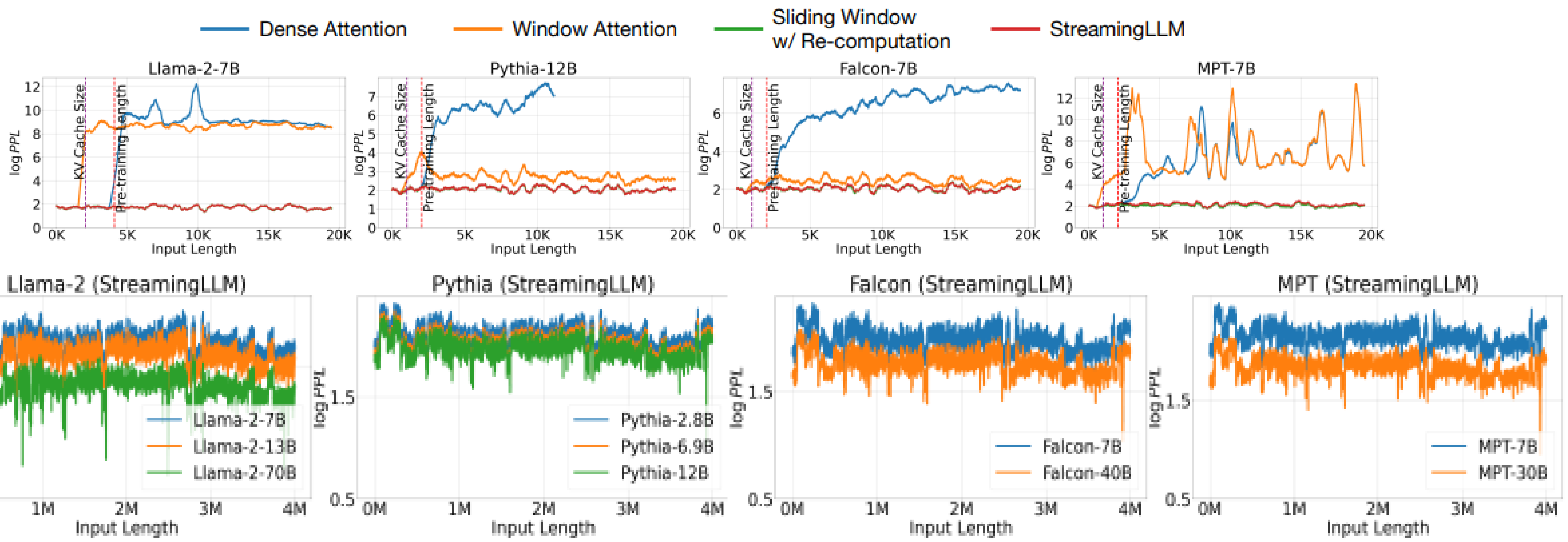
- Evaluated on the concatenated PG19 test set (100 long books).

Perplexity Performance:

- StreamingLLM matches the oracle baseline (sliding window with recomputation) for texts up to 20K tokens.

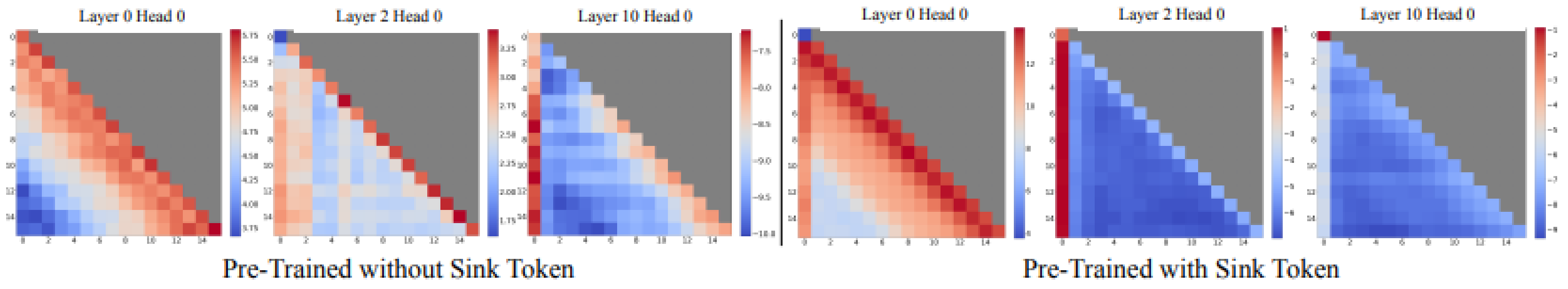
Scalability:

- Handles texts exceeding 4 million tokens across various model families and scales



Experiment: Pretraining with sink token

- **Without a sink token:**
  models show local attention in lower layers and increased attention to initial tokens in deeper layers.
- **With a sink token:**
  There is clear attention directed at it across all layers, effectively collecting redundant attention.
  With the presence of the sink token, less attention is given to other initial tokens, supporting the benefit of designating the sink token to enhance the streaming performance.



Experiment: Pretraining with sink token

- No negative impact on model convergence and subsequent performance on a range of NLP benchmarks

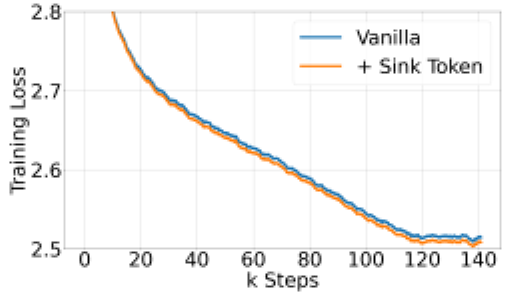
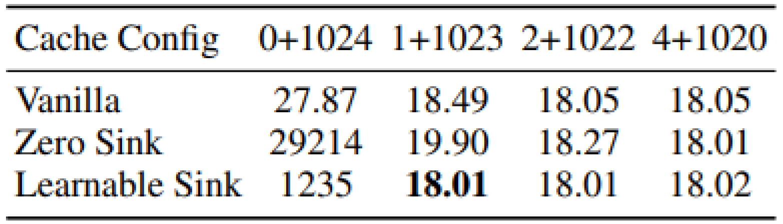


Figure 6: Pre-training loss curves of models w/ and w/o sink tokens. Two models have a similar convergence trend.

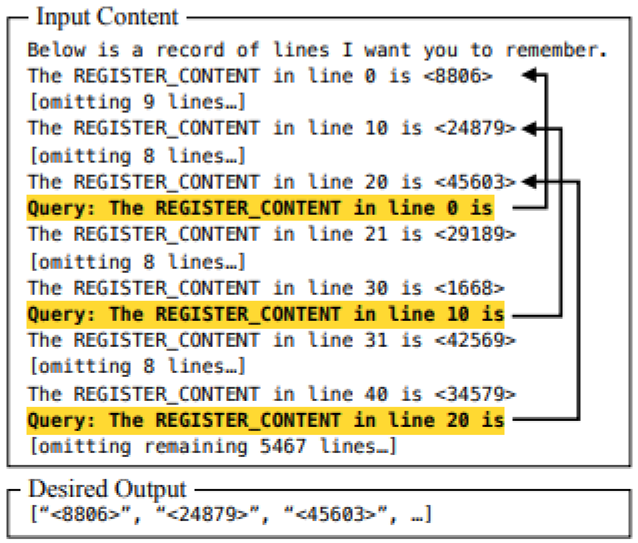
- Streaming perplexities differ between models trained using traditional methods and those augmented with a sink token



Cache Config	0+1024	1+1023	2+1022	4+1020
Vanilla	27.87	18.49	18.05	18.05
Zero Sink	29214	19.90	18.27	18.01
Learnable Sink	1235	**18.01**	18.01	18.02

Experiment: Streaming Question Answering

Introduced a dataset streameval.
StreamEval queries the model every 10 lines of new information.
•Querying Mechanism:
Each query's answer is found 20 lines prior in the text.
Mimics real-world scenarios where questions relate to recent context.



Input Content

```
Below is a record of lines I want you to remember.
The REGISTER_CONTENT in line 0 is <8806>
[omitting 9 lines…]
The REGISTER_CONTENT in line 10 is <24879>
[omitting 8 lines…]
The REGISTER_CONTENT in line 20 is <45603>
Query: The REGISTER_CONTENT in line 0 is
The REGISTER_CONTENT in line 21 is <29189>
[omitting 8 lines…]
The REGISTER_CONTENT in line 30 is <1668>
Query: The REGISTER_CONTENT in line 10 is
The REGISTER_CONTENT in line 31 is <42569>
[omitting 8 lines…]
The REGISTER_CONTENT in line 40 is <34579>
Query: The REGISTER_CONTENT in line 20 is
[omitting remaining 5467 lines…]
```

Desired Output

```
["<8806>", "<24879>", "<45603>", …]
```

Figure 8: The first sample in StreamEval.

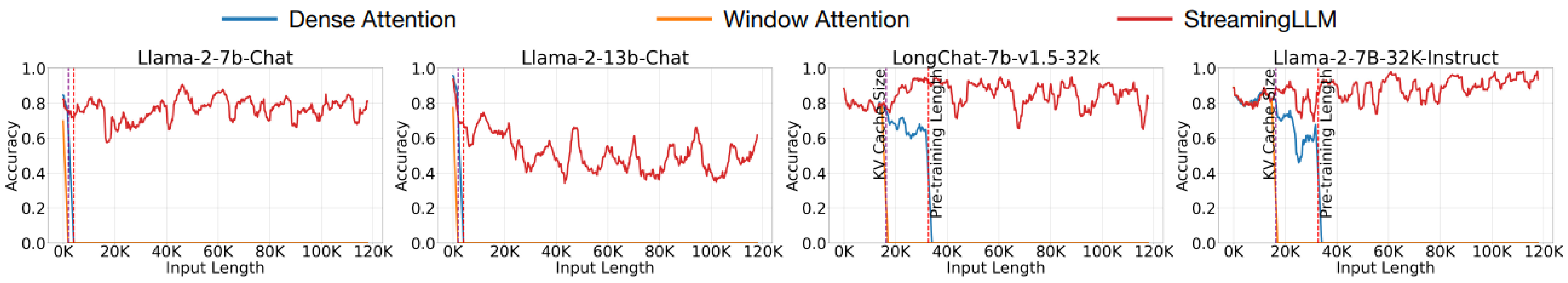


Figure 9: Performance on the StreamEval benchmark. Accuracies are averaged over 100 samples.

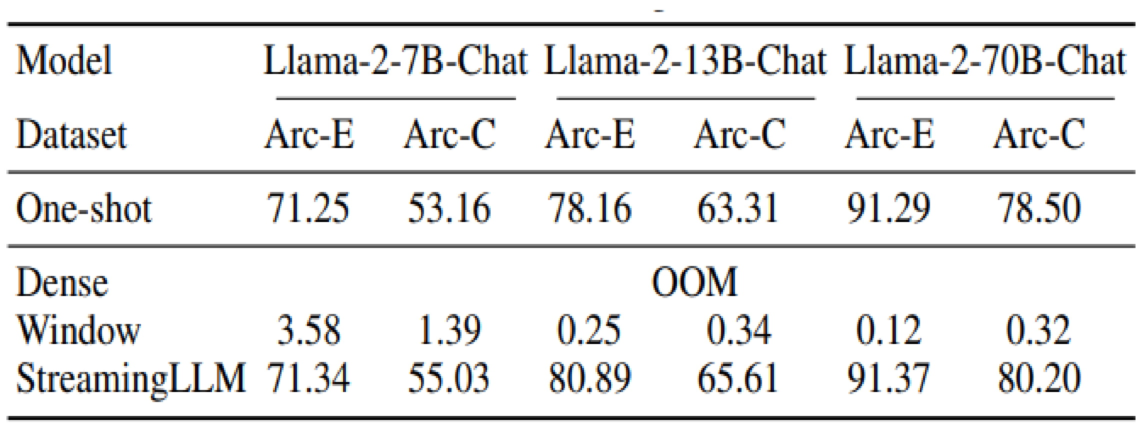
Experiment: ARC-Challenge

Evaluation Setup:

- Used ARC-[Challenge, Easy] datasets with concatenated question-answer pairs.
- Input streamed to Llama-2-[7,13,70]B-Chat models.
- Model completions assessed using exact match criterion.

Results:

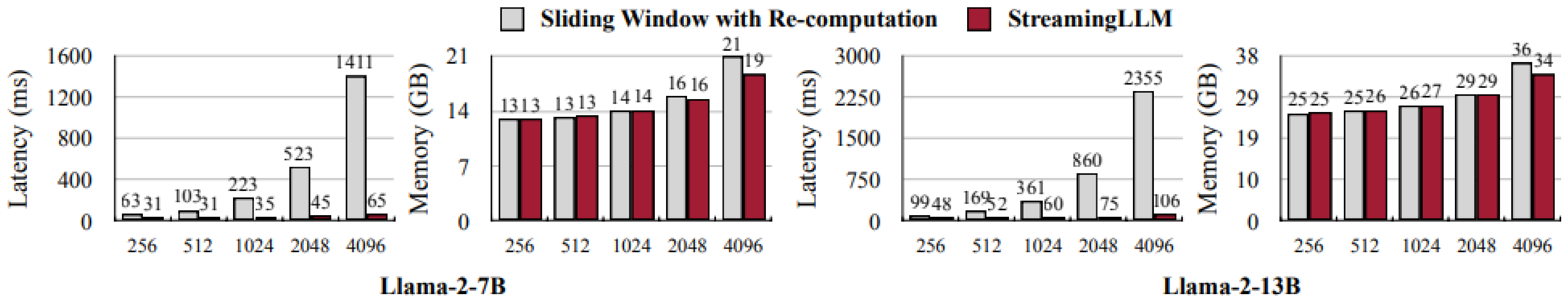
- Dense Attention: Causes Out-of-Memory (OOM) errors, making it unsuitable.
- Window Attention: Efficient but low accuracy due to randc outputs when input exceeds cache size.
- StreamingLLM: Handles streaming efficiently and matches one-shot baseline accuracy.



Model	Llama-2-7B-Chat		Llama-2-13B-Chat		Llama-2-70B-Chat	
Dataset	Arc-E	Arc-C	Arc-E	Arc-C	Arc-E	Arc-C
One-shot	71.25	53.16	78.16	63.31	91.29	78.50
Dense			OOM			
Window	3.58	1.39	0.25	0.34	0.12	0.32
StreamingLLM	71.34	55.03	80.89	65.61	91.37	80.20

Experiment: Efficiency

- **Models Tested**: Llama-2-7B and Llama-2-13B.
- **Decoding Speed**: StreamingLLM's decoding speed grows linearly with cache size, while the baseline shows a quadratic increase in latency.
- **Performance Gain**: StreamingLLM achieves up to **22.2× speedup per token**.
- **Memory Usage**: Despite lower latency, StreamingLLM maintains a memory footprint similar to the baseline.



Limitations

•**No Extension of Context Window**: StreamingLLM does not increase the model's context window or enhance long-term memory.

•**Cache Limitation**: The model operates only within its current cache and cannot retain information beyond it.

•**Unsuitability for Long-Term Tasks**: Not ideal for tasks requiring long-term memory and extensive data dependency, such as long document QA and summarization.

•**Effective for Short-Term Memory Tasks**: Performs well in tasks needing short-term memory, such as daily conversations and short document QA.

•**Strength in Recent Context**: Excels in generating coherent text from recent context without needing cache refreshment.

Conclusions

•Deploying LLMs in streaming applications is necessary but challenging due to efficiency limitations and performance degradation with longer texts.

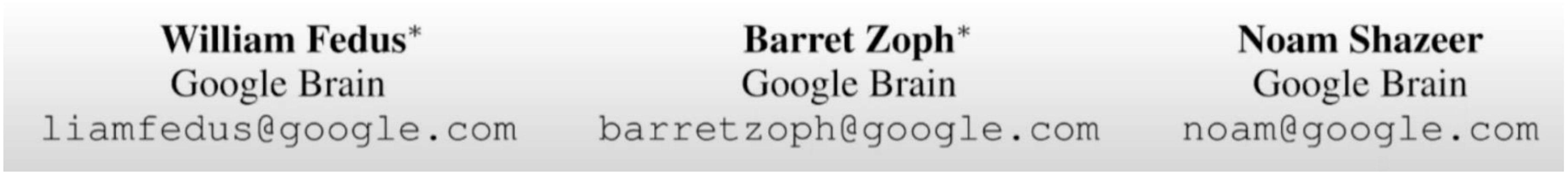
•Initial tokens act as "attention sinks," crucial for maintaining model performance.

•StreamingLLM is introduced as a simple and efficient framework for handling unlimited texts without fine-tuning.

•It incorporates attention sinks with recent tokens, enabling text modeling up to 4 million tokens.

Paper 3

Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity



William Fedus[*]
Google Brain
liamfedus@google.com

Barret Zoph[*]
Google Brain
barretzoph@google.com

Noam Shazeer
Google Brain
noam@google.com

GPT-3: 175 Billion params
Sparsity: not used in a traditional manner

Title Explanation (High-Level analysis)

Trillion & Sparsity:

Params:
- GPT-3: 175 Billion params
- The model has trillion params
- Not used in a traditional way
  - Sparsity params
  - The data used in different way

Switch Transformers:

Model structure:
- Transformers backbone
- Feed forward layers
  - (Divide up into mixture-experts)
- Switches to find the best experts

How does it work?
- Switch transformers only routes each token to only one expert only (sparsity).
- Previous Mixture-experts model required at least 2 experts to get a stable training signal.
- Scale the number of experts without making the model compute more.
- Make the training stable: selective dropout/casting of parameters to different precisions and a better initialization.
- We do not necessarily need trillion params.

Title Explanation

Switch Transformers:

- Simple architecture can surpass complicated algorithms by providing:
  - Generous computational budget
  - Data set size
  - Parameter count
- Sparsely-activated expert model
- Sparsity comes from activating a subset of the neural network weights for each incoming example

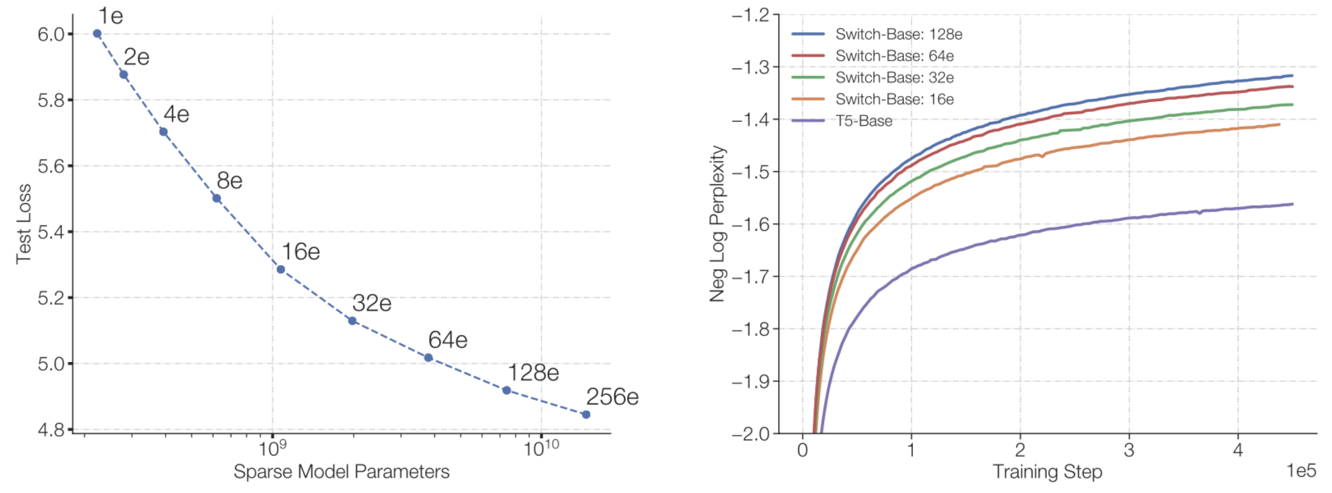
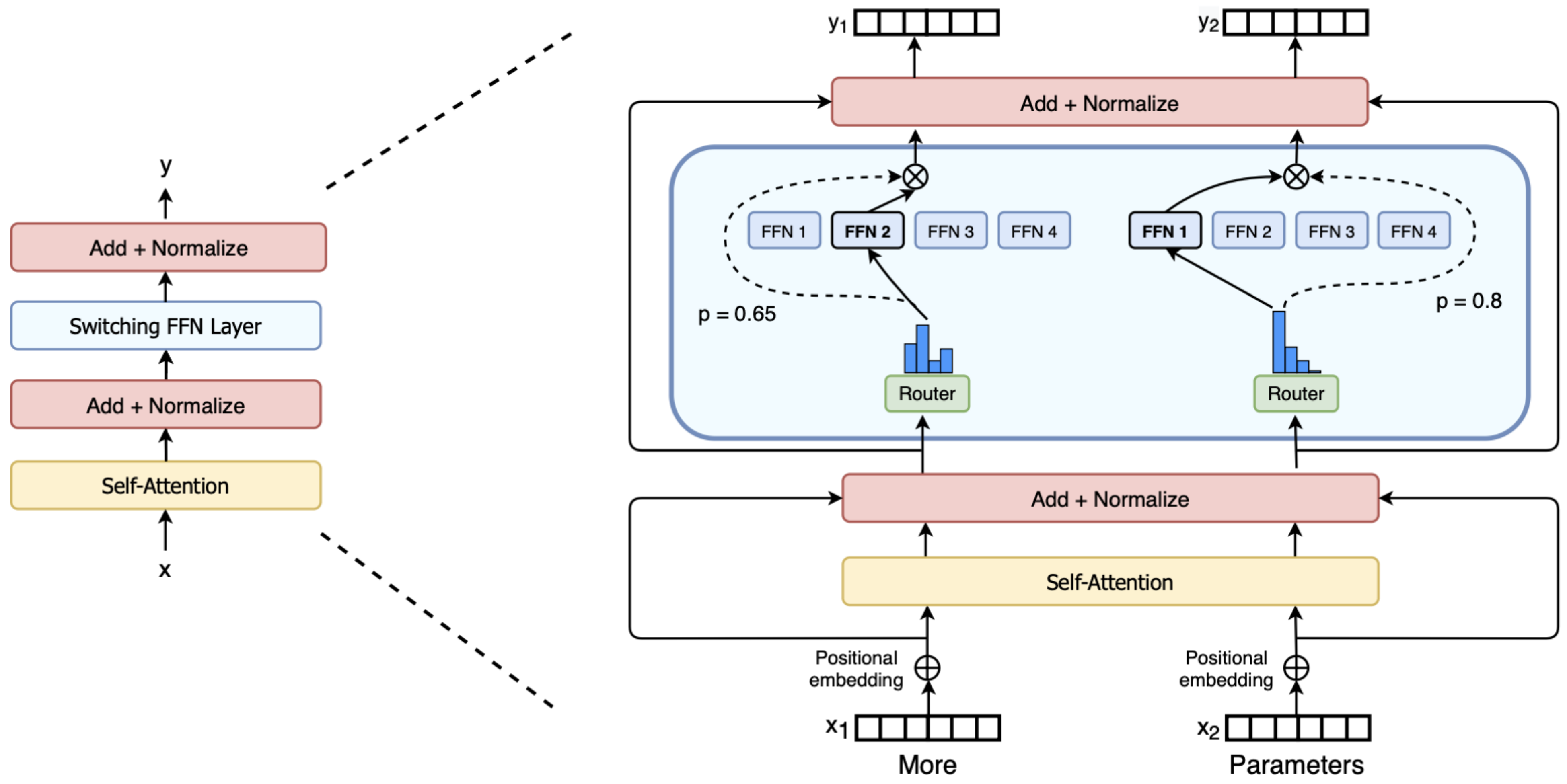


Figure 1: Scaling and sample efficiency of Switch Transformers. Left Plot: Scaling properties for increasingly sparse (more experts) Switch Transformers. Right Plot: Negative log perplexity comparing Switch Transformers to T5 (Raffel et al., 2019) models using the same compute budget.

Switch Transformer



Sparse Routing

Shazeer's MoE:

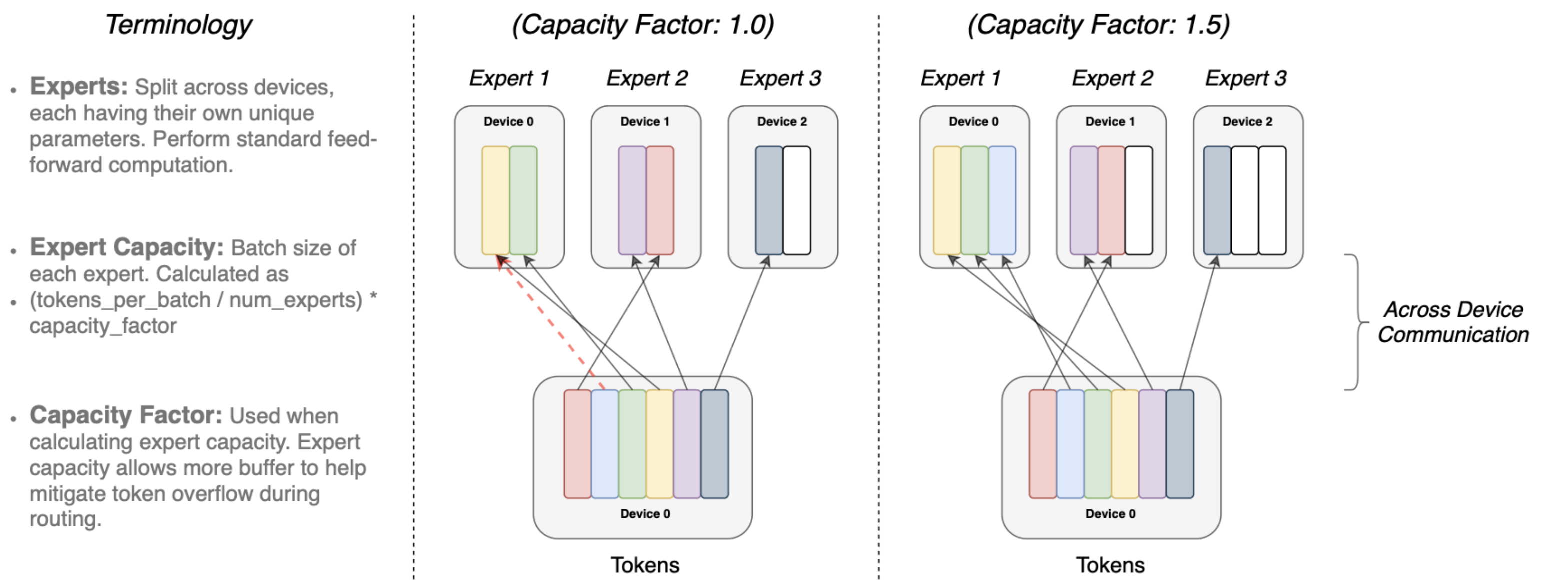
- Determine top-k experts for each token (k>1)
- Find: higher k-values in the lower layers are important for models with many routing layers
- With the router variable W_r

$$p_i(x) = \frac{e^{h(x)_i}}{\sum_j^N e^{h(x)_j}} \qquad y = \sum_{i \in \mathcal{T}} p_i(x) E_i(x)$$

$$h(x) = W_r \cdot x$$

Switch Routing:

- k = 1
- Find:
  - Preserves model quality
  - reduce routing computation


Terminology

- **Experts:** Split across devices, each having their own unique parameters. Perform standard feed-forward computation.
- **Expert Capacity:** Batch size of each expert. Calculated as
- (tokens_per_batch / num_experts) * capacity_factor
- **Capacity Factor:** Used when calculating expert capacity. Expert capacity allows more buffer to help mitigate token overflow during routing.

Sparse Routing

Pros:

- A capacity factor greater than 1.0 creates additional buffer to accommodate.
- Our computation is dynamic due to the routing decisions at training and inference (Expert capacity)

$$\text{expert capacity} = \left(\frac{\text{tokens per batch}}{\text{number of experts}}\right) \times \text{capacity factor}$$

Cons:

- High values will result in wasted computation and memory

Improvement:

- A differentiable load balance loss:

$$\text{loss} = \alpha \cdot N \cdot \sum_{i=1}^{N} f_i \cdot P_i \qquad f_i = \frac{1}{T} \sum_{x \in \mathcal{B}} \mathbb{1}\{\text{argmax}\, p(x) = i\}$$

$$P_i = \frac{1}{T} \sum_{x \in \mathcal{B}} p_i(x)$$

- Our computation is dynamic due to the routing decisions at training and inference (Expert capacity)

Preliminary Experiments

Compared to:

- T5
- MOE

Comparing variables:

- Capacity factors.
- Fixed amount of time and wall-clock time.
- Training speed.

Dataset: Colossal Clean Crawled Corpus (C4)
Task: predict missing tokens (15%)
Metric: neg-log-perplexity （Threshold: -1.5）

- High values will result in wasted computation and memory

Model	Capacity Factor	Quality after 100k steps (↑) (Neg. Log Perp.)	Time to Quality Threshold (↓) (hours)	Speed (↑) (examples/sec)
T5-Base	—	-1.731	Not achieved†	1600
T5-Large	—	-1.550	131.1	470
MoE-Base	2.0	-1.547	68.7	840
Switch-Base	2.0	-1.554	72.8	860
MoE-Base	1.25	-1.559	80.7	790
Switch-Base	1.25	-1.553	65.0	910
MoE-Base	1.0	-1.572	80.1	860
Switch-Base	1.0	-1.561	**62.8**	1000
Switch-Base+	1.0	**-1.534**	67.6	780

For Switch-Base+, they increase the model size until it matches the speed of the MoE model by increasing the model hidden-size from 768 to 896 and the number of heads from 14 to 16.

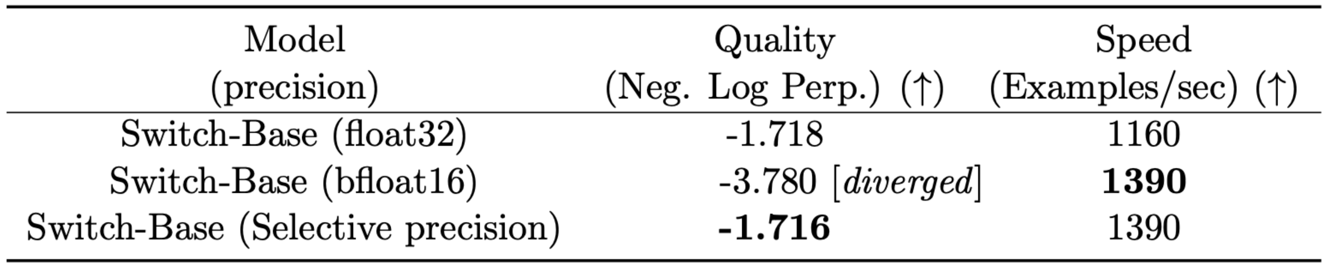
Improved Training and Fine-Tuning

Issues:

- Introduce training difficulties over a vanilla Transformer
- hard-switching (routing) decisions – instability
- Severe overfitting during fine-tuning

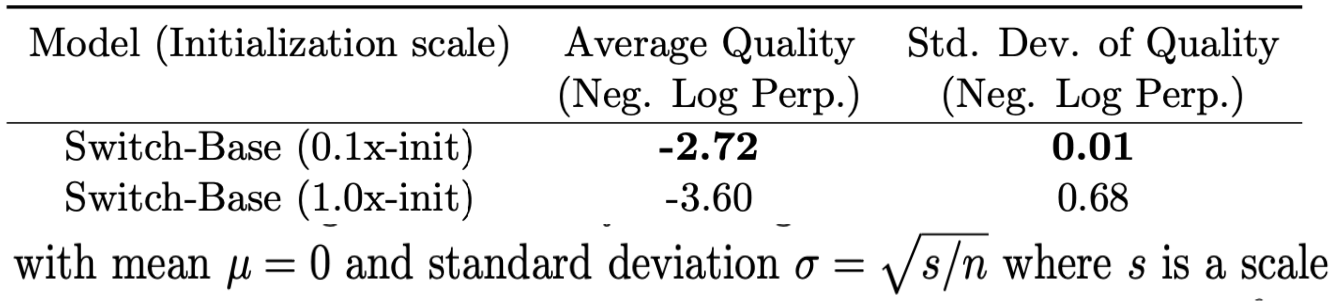
Methods:

- Low precision formats: bfloat16
- The float32 precision is only used within the body of the router function
- Smaller parameter initialization for stability （32 experts, 3.2k steps）
- Increase the dropout inside the experts-expert dropout (setting a smaller dropout rate (0.1) at non-expert layers and a much larger dropout rate (0.4)



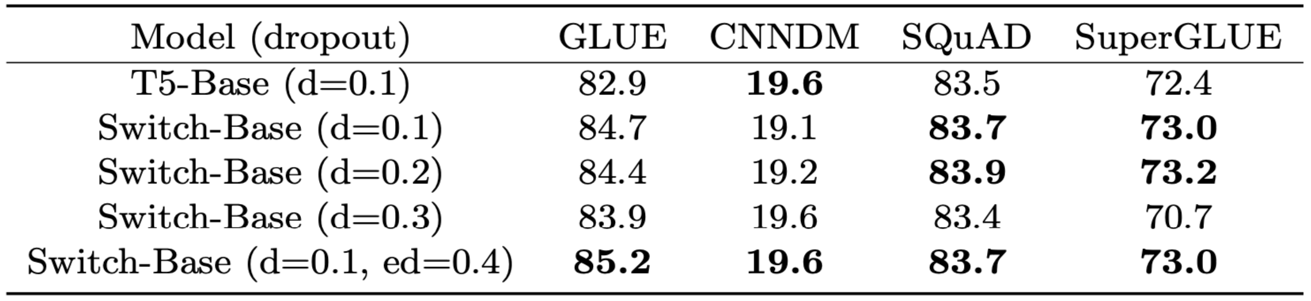
Model (precision)	Quality (Neg. Log Perp.) (↑)	Speed (Examples/sec) (↑)
Switch-Base (float32)	-1.718	1160
Switch-Base (bfloat16)	-3.780 [*diverged*]	**1390**
Switch-Base (Selective precision)	**-1.716**	1390

Cast the local routing operations to float32 while preserving bfloat16 precision elsewhere to stabilize the model while achieving nearly equal speed to (unstable) bfloat16-precision training.



Model (Initialization scale)	Average Quality (Neg. Log Perp.)	Std. Dev. of Quality (Neg. Log Perp.)
Switch-Base (0.1x-init)	**-2.72**	**0.01**
Switch-Base (1.0x-init)	-3.60	0.68

with mean $\mu = 0$ and standard deviation $\sigma = \sqrt{s/n}$ where s is a scale

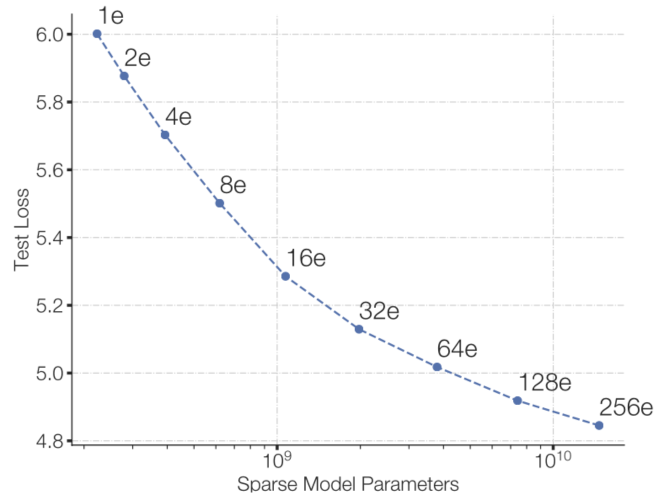


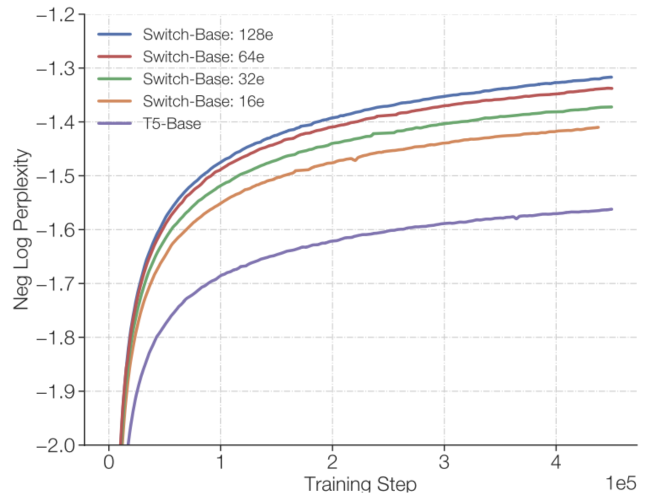
Model (dropout)	GLUE	CNNDM	SQuAD	SuperGLUE
T5-Base (d=0.1)	82.9	**19.6**	83.5	72.4
Switch-Base (d=0.1)	84.7	19.1	**83.7**	**73.0**
Switch-Base (d=0.2)	84.4	19.2	**83.9**	**73.2**
Switch-Base (d=0.3)	83.9	19.6	83.4	70.7
Switch-Base (d=0.1, ed=0.4)	**85.2**	**19.6**	**83.7**	**73.0**

Scaling Properties (Same computational budget)

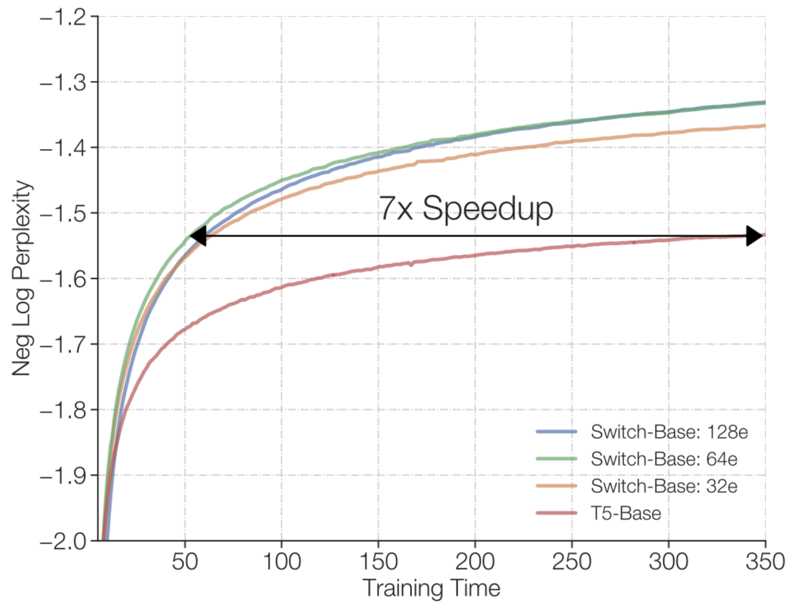
Findings:

- Step-Basis:
  - Keeping the FLOPS per token fixed, having more parameters (experts) speeds up training
  - Increasing the number of experts leads to more sample efficient models.
- Time-Basis:
  - The increased sample efficiency observed on a step-basis doesn't necessarily translate to a better model quality as measured by wall-clock.



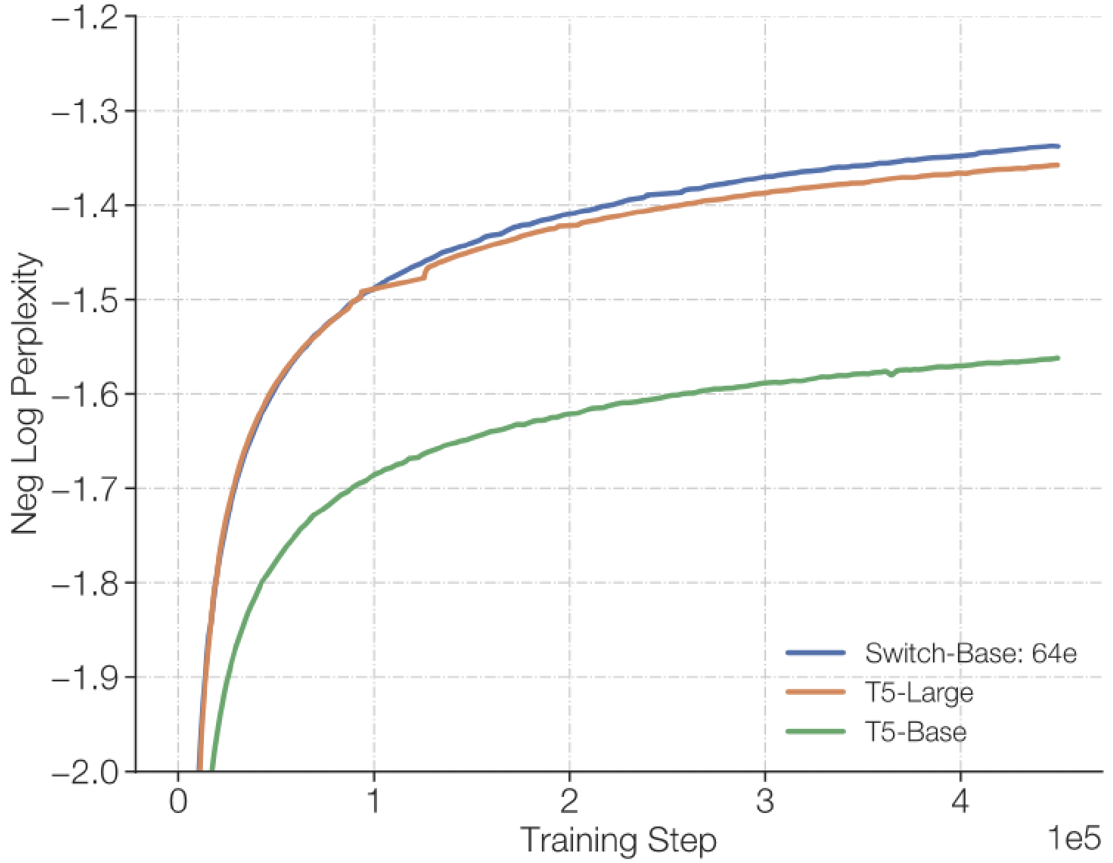


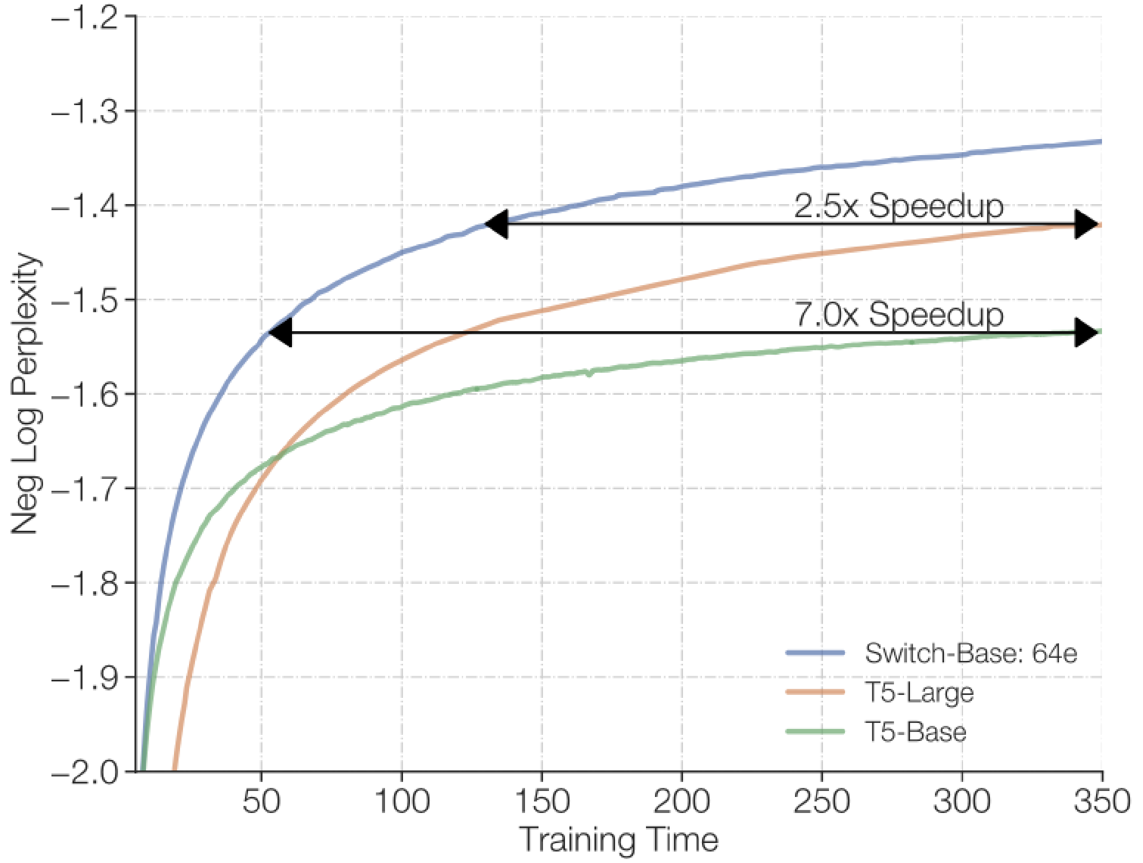
Switch-Base 64 expert model achieves the same performance of the T5-Base model at step 60k at step 450k, which is a 7.5x speedup in terms of step time.



UNIVERSITY of VIRGINIA

Scaling vs Larger Dense Models

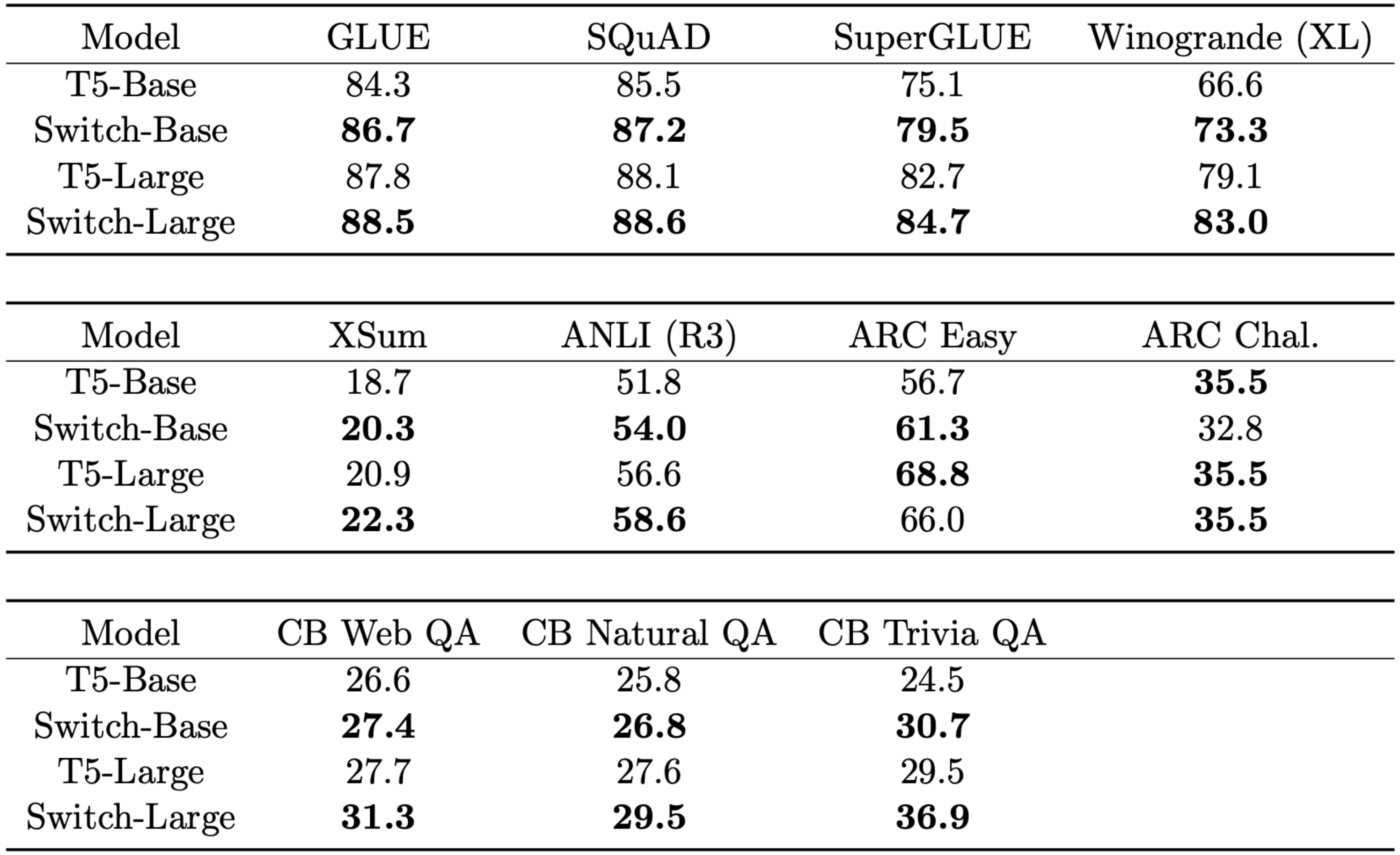




Left Plot: Switch-Base is more sample efficient than both the T5-Base, and T5-Large variant, which applies 3.5x more FLOPS per token. Right Plot: As before, on a wall-clock basis,we find that Switch-Base is still faster, and yields a 2.5x speedup over T5-Large.

UNIVERSITY of VIRGINIA

Downstream Tasks – Fine-tuning



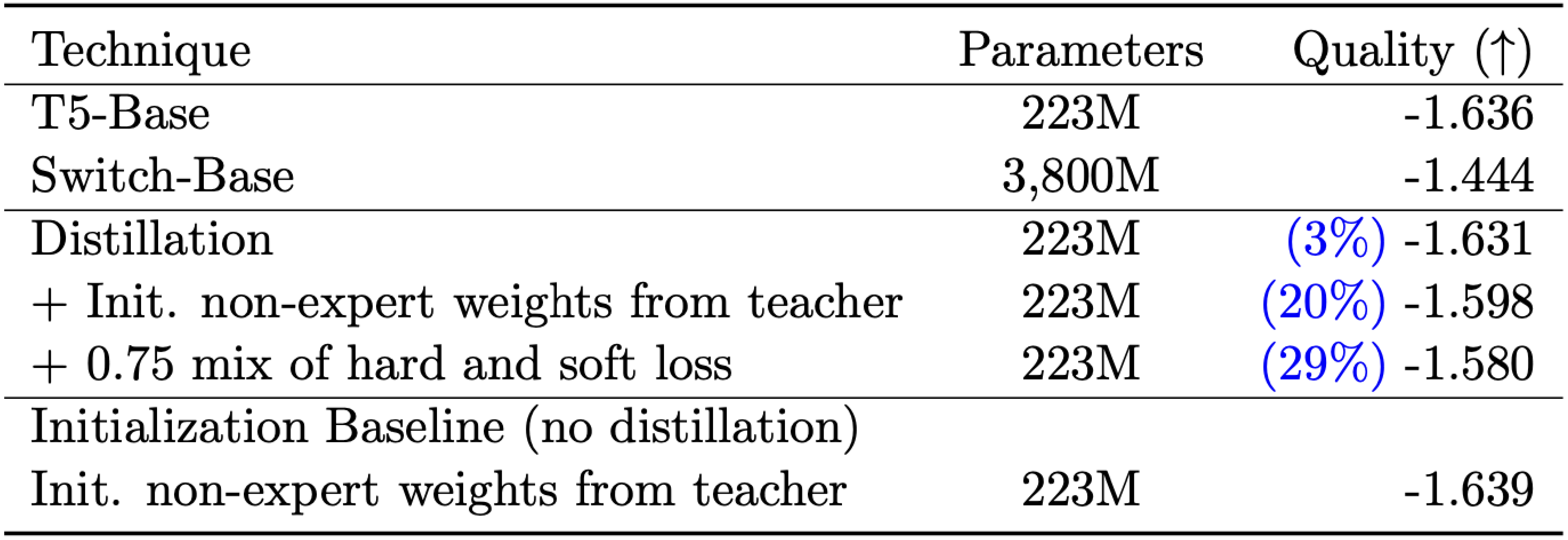
Model	GLUE	SQuAD	SuperGLUE	Winogrande (XL)
T5-Base	84.3	85.5	75.1	66.6
Switch-Base	**86.7**	**87.2**	**79.5**	**73.3**
T5-Large	87.8	88.1	82.7	79.1
Switch-Large	**88.5**	**88.6**	**84.7**	**83.0**

Model	XSum	ANLI (R3)	ARC Easy	ARC Chal.
T5-Base	18.7	51.8	56.7	**35.5**
Switch-Base	**20.3**	**54.0**	**61.3**	32.8
T5-Large	20.9	56.6	**68.8**	**35.5**
Switch-Large	**22.3**	**58.6**	66.0	**35.5**

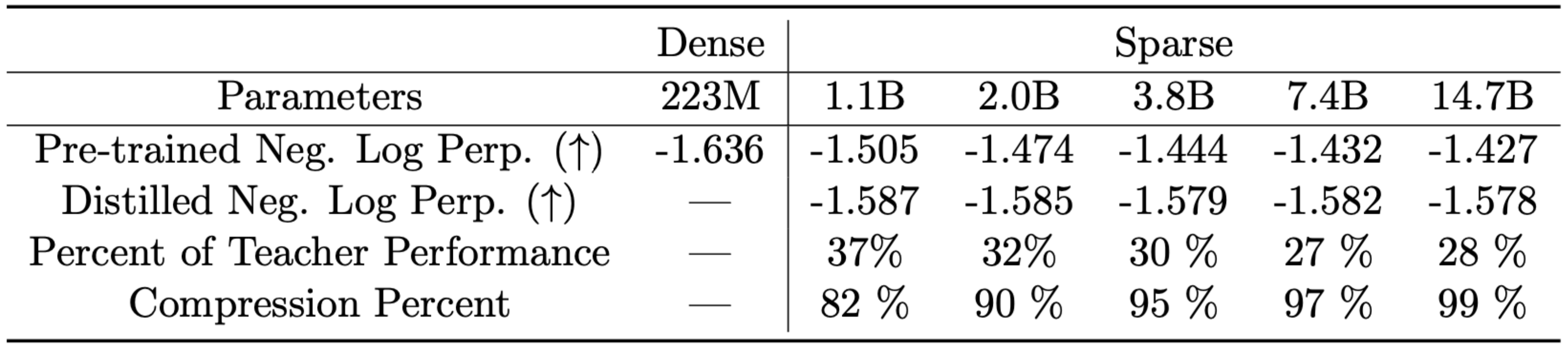
Model	CB Web QA	CB Natural QA	CB Trivia QA
T5-Base	26.6	25.8	24.5
Switch-Base	**27.4**	**26.8**	**30.7**
T5-Large	27.7	27.6	29.5
Switch-Large	**31.3**	**29.5**	**36.9**

Compare FLOP-matched Switch models to the T5-Base and T5-Large baselines.

Downstream Tasks – Distillation



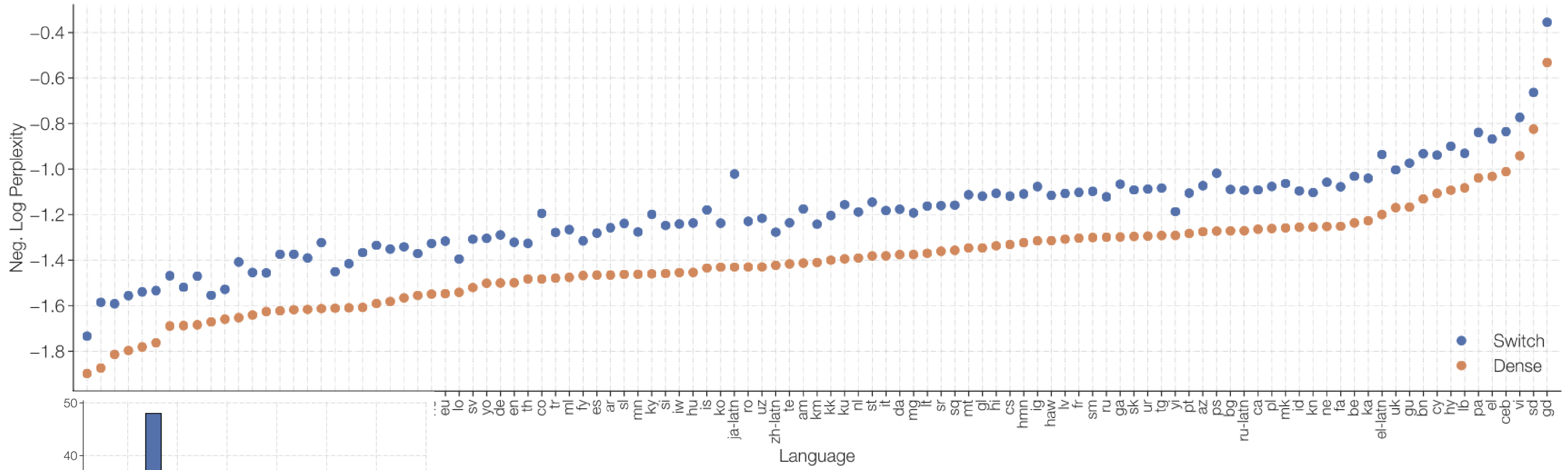
Technique	Parameters	Quality (↑)
T5-Base	223M	-1.636
Switch-Base	3,800M	-1.444
Distillation	223M	(3%) -1.631
+ Init. non-expert weights from teacher	223M	(20%) -1.598
+ 0.75 mix of hard and soft loss	223M	(29%) -1.580
Initialization Baseline (no distillation)		
Init. non-expert weights from teacher	223M	-1.639

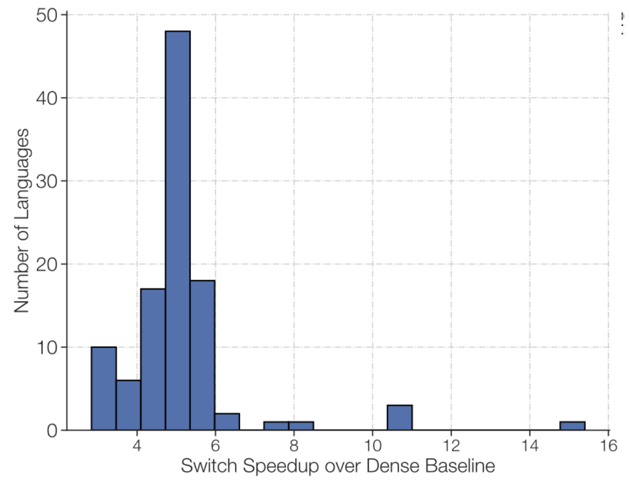


	Dense	Sparse				
Parameters	223M	1.1B	2.0B	3.8B	7.4B	14.7B
Pre-trained Neg. Log Perp. (↑)	-1.636	-1.505	-1.474	-1.444	-1.432	-1.427
Distilled Neg. Log Perp. (↑)	—	-1.587	-1.585	-1.579	-1.582	-1.578
Percent of Teacher Performance	—	37%	32%	30 %	27 %	28 %
Compression Percent	—	82 %	90 %	95 %	97 %	99 %

Distill a wide variety of sparse models into dense models.

Downstream Tasks – Multilingual Learning





The step speedup of Switch Transformers over the FLOP matched T5 dense baseline to reach the same quality.

Conclusions

- Sparse Scaling to Ultra-Large Models
- Simplified Routing Mechanism
- Significant Speedups in Training
- Stability and Generalization
- Improvements on downstream tasks

- Switch Transformers employ Mixture-of-Experts (MoE) with a sparse-activation scheme, distributing massive model parameters (billions to trillions) among multiple expert networks. This approach greatly increases the total parameter count without significantly increasing the computational cost, compared to dense models with the same FLOPs.

- By switching from "Top-k" (k>1) to "Top-1" routing, Switch Transformers reduce both the routing overhead and communication costs. An additional load-balancing loss ensures that tokens are distributed relatively evenly across experts.

- Under the same FLOPs budget, Switch Transformers often train substantially faster (e.g., 4x–7x speedups) and achieve superior or comparable performance to dense Transformer baselines on multiple benchmarks.

- When moving from large-scale pre-training to fine-tuning, Switch Transformers retain their advantages. Techniques like selective float32 routing, reduced initialization scale, and higher dropout on expert layers address training instability and overfitting, enabling large-scale training in bfloat16.

Discussion

- Trade-Offs in Load Balancing and Routing?
    - How to design?
- Sources of Training Instability
    - Hard routing decisions and large parameter counts can lead to unstable gradients
    - Initialization and dropout
- Non-Linear Returns from Scaling
    - The gains by increasing experts are not significant anymore.
    - Carefully discuss memory, communication… issues
- Increased memory requirements

  Although sparse models activate only a subset of parameters for each input, the total number of parameters (including all exp erts) is typically much larger than in dense models; this increases memory requirements for storing the model