Efficient Model Architectures

David Antrobius, Evan Conway, Brendan Malaugh

Overview

We cover three papers, each focusing on a different aspect of LLM efficiency:

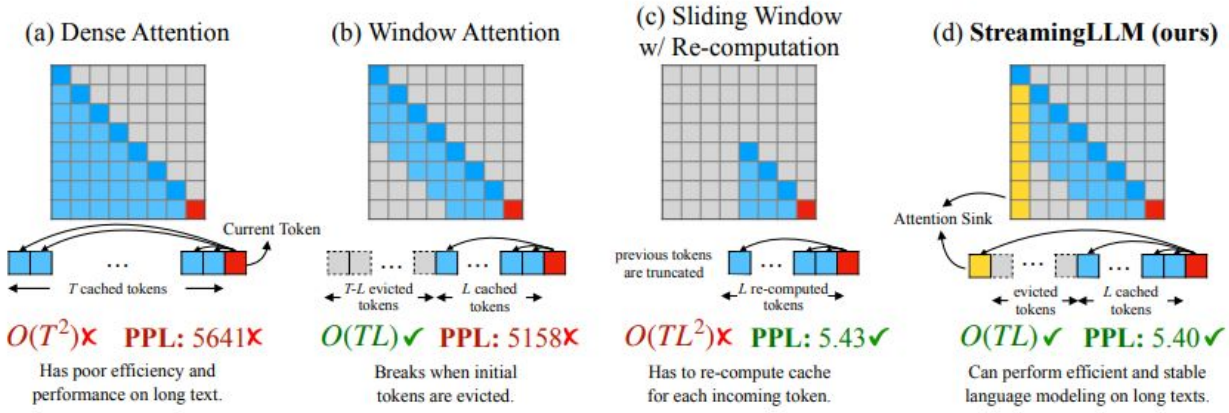
- **Attention Sinks** focuses on how we can help Transformers efficiently process long sequences without running out of memory or losing quality
- **Mamba** focuses on non-Transformer architectures that let us handle long sequences without the quadratic scaling of Transformers
- **Switch Transformers** focuses on designing sparse models that can be scaled without greatly increasing the amount of training FLOPs necessary

Efficient Streaming Language Models with Attention Sinks

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, Mike Lewis

Streaming difficulty

- In multi-round dialogue / continuous generation with unbounded context yields 2 problems: **KV-cache memory growth** and **length extrapolation failure**
- Authors identify **attention sinks** as a fundamental mechanism and propose **StreamingLLM** (no fine tuning)
- Yields stable language modeling up to ~4M tokens and 22.2x speedup



T = # of tokens processed so far

L = cache length

Generation memory and quality

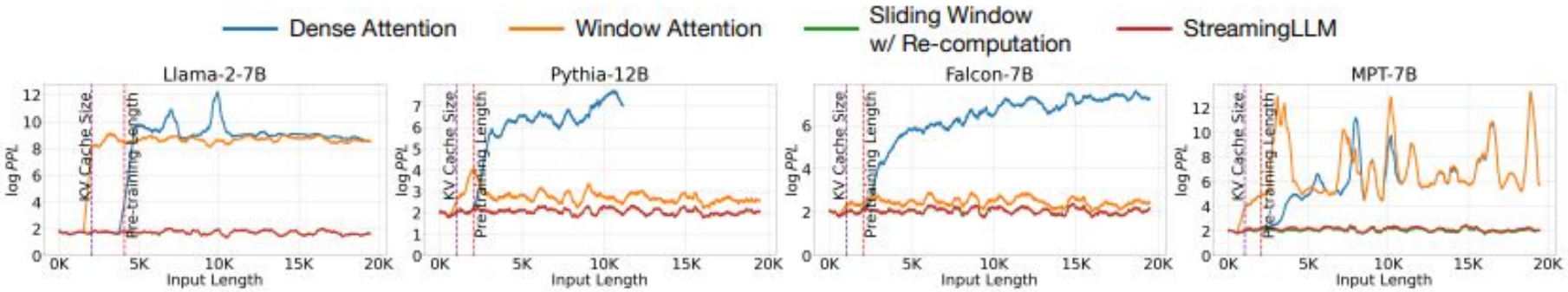
- In decoding, transformers cache Key/Value (KV) for past tokens so memory grows with sequence length
- Many LLMs' quality **degrades** beyond their training attention window
- The goal is to handle virtually infinite streams without fine tuning while maintaining memory/latency per token
- For autoregressive decoding, attention uses all previous keys/values for a token, t i.e. $\{k_i, v_i\}_{i \< t}$

$$\alpha_{t,i} = \mathrm{softmax}_i\left(\frac{q_t k_i^\top}{\sqrt{d}}\right), \quad h_t = \sum_{i \le t} \alpha_{t,i} v_i$$

- KV cache stores k_i and v_i for reuse so decoding avoids recomputing past states, but memory grows with t

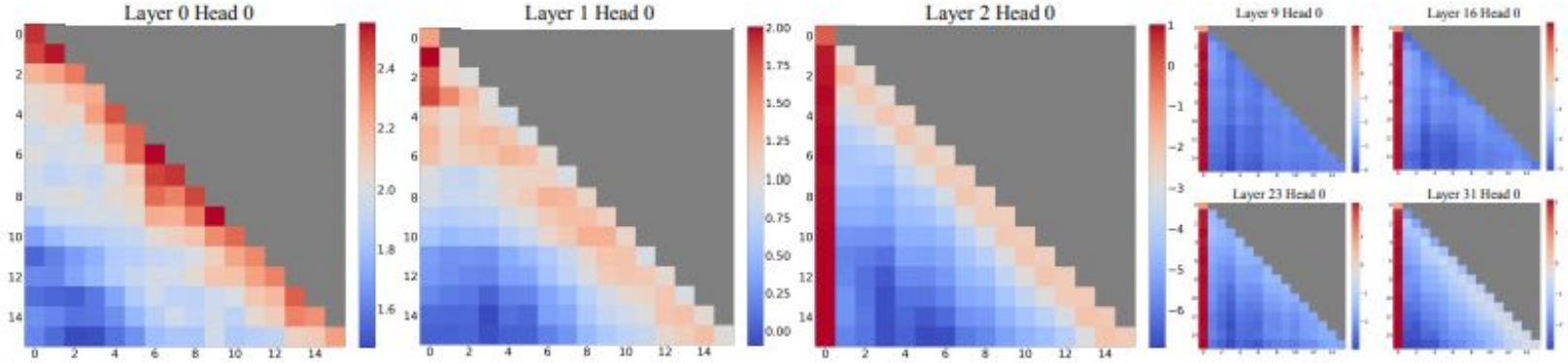
Window attention collapse

- Perplexity remains reasonable until the stream exceeds cache size
- When the cache starts evicting the earliest tokens, perplexity spikes rapidly
- Consistent across model families



Attention concentration

- Find that bottom layers are mostly local / recent token attention and the many higher layers/heads have remarkably strong attention to first tokens
- These early tokens are dubbed **attention sinks**



Softmax “denominator sensitivity”

- If early tokens receive large attention scores, removing them changes the softmax normalization everywhere
- Recall

$$\text{Softmax}(x)_i = \frac{e^{x_i}}{e^{x_1} + \sum_{j=2}^{N} e^{x_j}}$$

- If a few terms dominate the denominator, evicting them forces global renormalization yielding attention distribution shift away from what model expects

Position vs. meaning

- Two hypotheses: early tokens matter because of **content** or **position**
- Authors experiment by replacing the first few tokens with “\n”

Llama-2-13B	PPL (↓)
0 + 1024 (Window)	5158.07
4 + 1020	5.40
4"\n"+1020	5.60

- ⇒ Behavior is largely about being at the beginning, not semantic importance

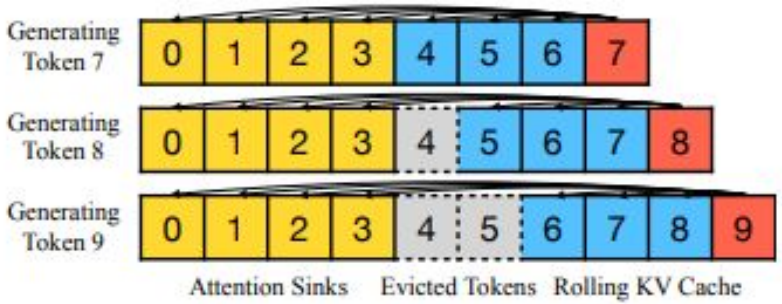
Sink token requirements

- Many LLMs use multiple early tokens as sinks (not just first)
- Authors run an ablation experiment finding that 1-2 initial tokens doesn't restore full perxplity; 4 suffices, more has diminishing returns

Cache Config	0+2048	1+2047	2+2046	4+2044	8+2040
Falcon-7B	17.90	12.12	12.12	12.12	12.12
MPT-7B	460.29	14.99	15.00	14.99	14.98
Pythia-12B	21.62	11.95	12.09	12.09	12.02
Cache Config	**0+4096**	**1+4095**	**2+4094**	**4+4092**	**8+4088**
Llama-2-7B	3359.95	11.88	10.51	9.59	9.54

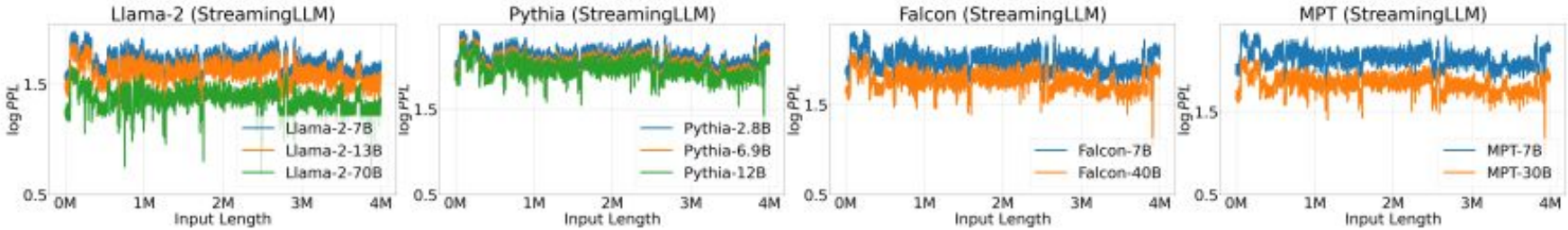
StreamingLLM

- Maintain a fixed size KV cache split into
  - Attention sink - keep first 4 tokens KV permanently
  - Rolling cache - keep the most recent L tokens KV
- Evict everything else
- Sinks “anchor” attention normalization so models attention distribution stays close its training regime



Cache relative positions

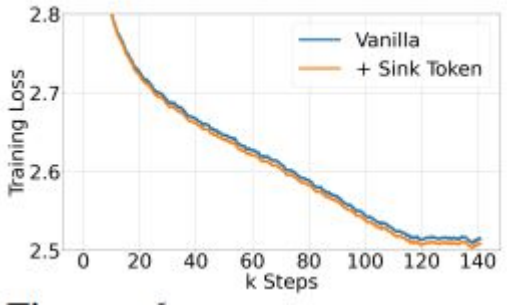
- StreamingLLM assigns positional indices within the cache, not the absolute original absolute token indices
  - Example - cache tokens [0,1,2,3,6,7,8] ⇒ position used for attn. [0,1,2,3,4,5,6]
- Two integrations
  - RoPE - cache keys before rotary transformation; apply rotation with cache relative positions each step
  - ALiBi - apply a contiguous linear bias to avoid “jumping” distances
- StreamingLLM maintains stable LM perplexity over very long streams



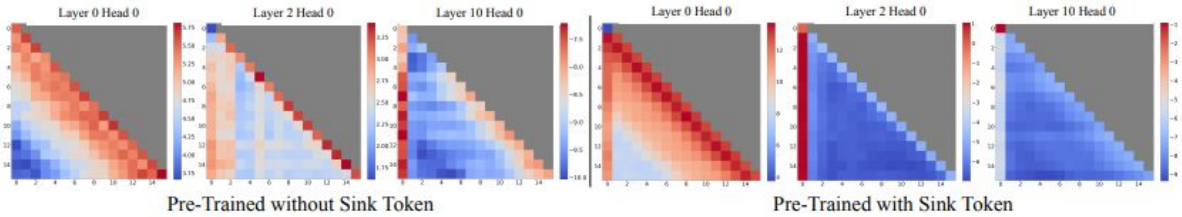
Training time improvement

- If the model lacks a consistent “always present” initial token, it may spread sink behavior across multiple early tokens
- So, pretrain with a learnable **sink token** prepended to every sequence

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



Streaming QA



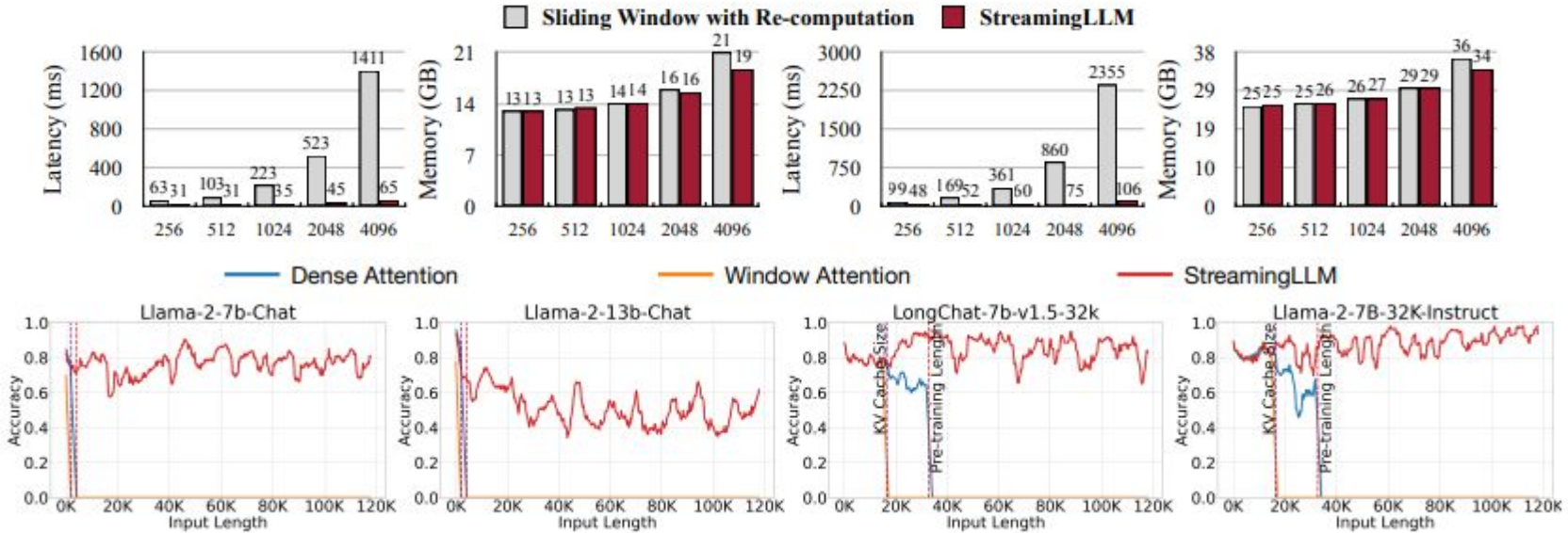
- Concatenate many QA pairs to mimic a multi-round interaction
- Dense attention often hits memory bound; window attention collapses; StreamingLLM matches one-shot baseline
- StreamingEval benchmark; query periodically about recent info; StreamingLLM maintains accuracy to very large input lengths

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

Methods	ARC-c	ARC-e	HS	LBD	OBQA	PIQA	WG
Vanilla	18.6	45.2	29.4	39.6	16.0	62.2	50.1
+Sink Token	**19.6**	**45.6**	**29.8**	**39.9**	**16.6**	**62.6**	**50.8**

StreamingLLM efficiency

- StreamingLLM:
  - Decoding latency scales approximately linearly with cache size
  - Achieves up to 22.2x per-token speedup
  - Memory footprint comparable to recomputation baseline



Limitations

- StreamingLLM does **not** increase content length; it generates based on what remains in the cache
- Not suitable for tasks requiring long term dependency
  - LongBench for example, using only 4 sink tokens underperforms truncation baselines when early prompt/document info is essential
  - Increasing # of preserved initial tokens restores performance but increases memory

Llama2-7B-chat	Single-Document QA		Multi-Document QA		Summarization	
	NarrativeQA	Qasper	HotpotQA	2WikiMQA	GovReport	MultiNews
Truncation 1750+1750	18.7	19.2	25.4	32.8	27.3	25.8
StreamingLLM 4+3496	11.6	16.9	21.6	28.2	23.9	25.5
StreamingLLM 1750+1750	18.2	19.7	24.9	32.0	26.3	25.9

Conclusion

- Authors identify attention sinks as key contributor to window attention collapse
- StreamingLLM - few initial sink tokens + rolling recent window yields stable streaming without fine-tuning
- Pretraining with dedicated sink token simplifies deployment
- Future work:
  - Adaptive sink selection (beyond “first k tokens”)
  - Combine with retrievela/summarization memory for long context tasks
  - Better attention mechanisms that avoid sink behavior (normalization alternatives)

Mamba: Linear-Time Sequence Modeling with Selective State Spaces

Albert Gu, Tri Dao

Two LM families and their properties

Transformers:

- Computation scales quadratically in terms of the sequence length, because each token attends to all previous tokens
- Training can be done in parallel

RNNs:

- Computation scale linearly in terms of the sequence length
- Training is sequential

The problem

- Transformers are not practical for long sequences, but RNNs cannot be trained efficiently enough to get the large models we need for strong performance
- We would like to find an RNN-like architecture that can be trained in parallel

Structured State Space Sequence Model (S4)

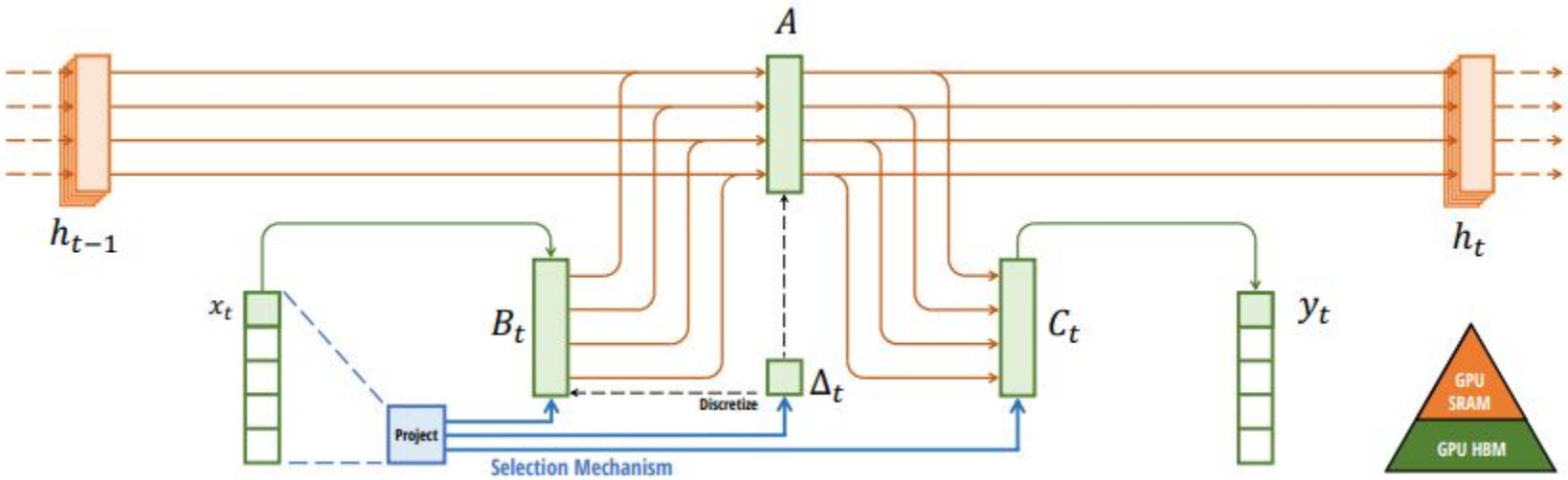
- Consider our input and output to be scalar-valued continuous functions of time, *x(t)* and *y(t)*. Similarly, there is a vector-valued hidden state *h(t)*
- For multi-dimensional input, apply the same model to each dimension
- Suppose that we model the interactions between these as follows:

$$h'(t) = \boldsymbol{A}h(t) + \boldsymbol{B}x(t) \qquad \text{(1a)}$$

$$y(t) = \boldsymbol{C}h(t) \qquad \text{(1b)}$$

- We discretize **A** and **B**, but not **C**, according to a discretization parameter Δ, which represents how often we sample

Visualization



Structured State Space Sequence Model (S4)

- We discretize using the zero-order hold rule:

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

- Discretization gives (with discretized **A** and **B**):

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

$$y_t = Ch_t \qquad (2b)$$

- This discretized version looks somewhat similar to how we might formulate an RNN
- This is sufficient to run linear inference, but will require sequential training

S4 With Convolution

- We can instead write the previous discretization in this form:

$$\overline{K} = (C\overline{B}, C\overline{AB}, \ldots, C\overline{A}^{k}\overline{B}, \ldots) \quad (3a)$$

$$y = x * \overline{K} \quad (3b)$$

- The * operator shown is convolution
- Intuition: we don't change **A**, **B**, or **C**, so we will be applying the same transformations to input values at many different positions
- This allows for efficient parallel training!

How to compress context?

- Issue: S4 models can only perform very inexact updates to the hidden state based on the input (can only add different multiples of **B**)

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

$$y_t = Ch_t \qquad (2b)$$

- This is because they are linear time invariant (LTI):
  - Linear: mapping from input x_t to output y_t is linear
  - Time invariant: same dynamics at each step in time
- LTI models have poor performance on tasks where we need to remember a specific piece of information and recall it far later

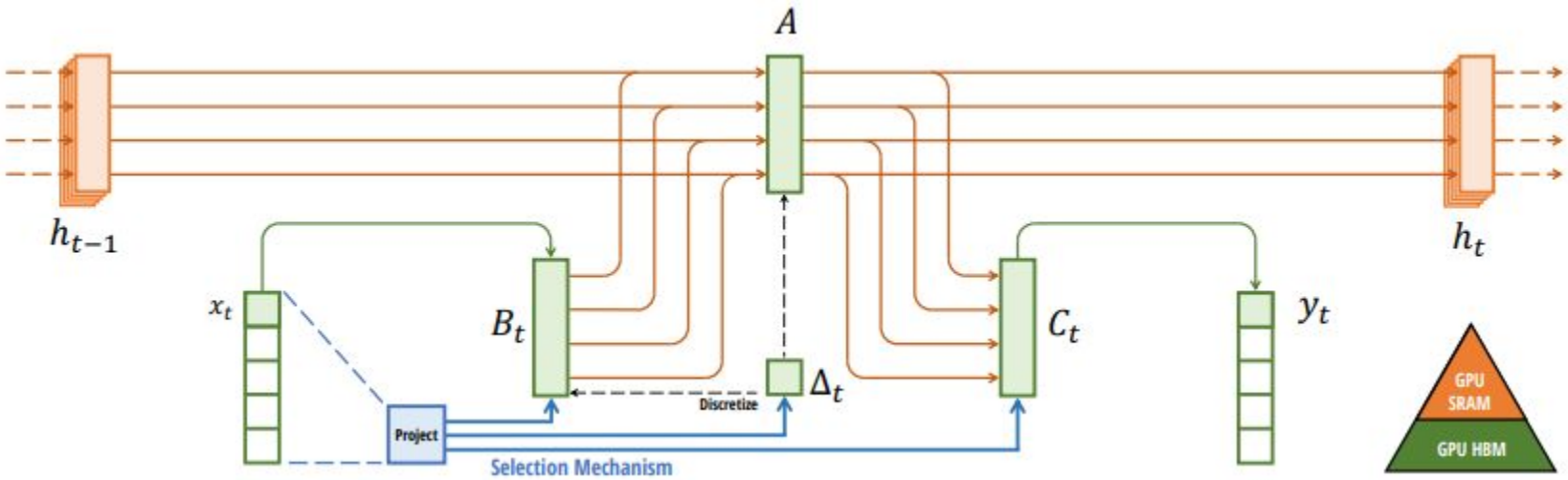
Why Selectivity Matters

- Selectivity: ability to filter out irrelevant information
- Transformers are able to select by immediately attending to all prior tokens
- As discussed previously, S4 models don’t really have selection (LTI)
- Some examples of where we want selection:
  - If there are irrelevant tokens between points of interest
  - If we want to ignore previous history. For example, if are receiving several documents back-to-back

Selective State Space Models

- Idea: make **B**, **C**, and Δ functions of the full input (all channels)
  - **B**: how the input contributes to elements of the hidden state
  - **C**: how elements of the hidden state contribute to the output
  - Δ: how much we focus on the current input (Δ = 0 means ignore)
- **B**, **C**, and Δ are all linear functions of the input
  - To ensure $\Delta \geq 0$, we pass it through Softplus(x) = $\ln(1 + e^x)$
- We lose convolution for parallel training, but we can instead use a parallel scan algorithm
  - Example: the contribution from all inputs before k steps ago to the current hidden state is $h_t = \boldsymbol{A}^k h_{t-k}$

Visualization



Algorithm With and Without Selection

Algorithm 1 SSM (S4)

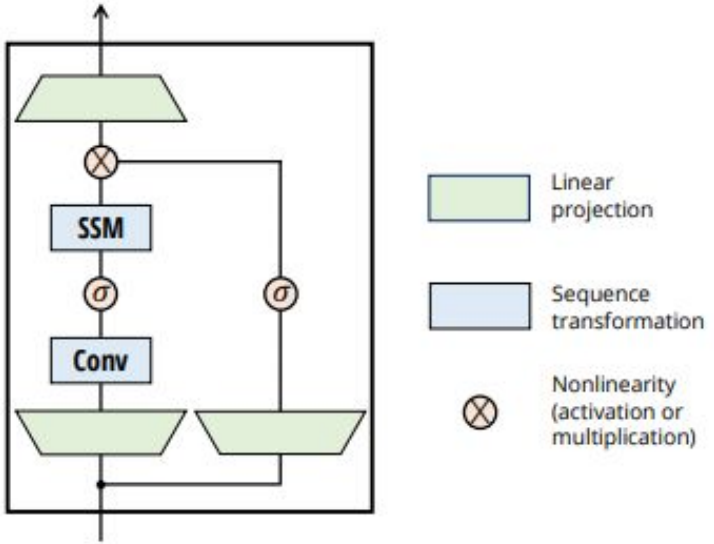
Input: $x : (\mathsf{B}, \mathsf{L}, \mathsf{D})$
Output: $y : (\mathsf{B}, \mathsf{L}, \mathsf{D})$
1: $\boldsymbol{A} : (\mathsf{D}, \mathsf{N}) \leftarrow$ Parameter
▷ Represents structured $N \times N$ matrix
2: $\boldsymbol{B} : (\mathsf{D}, \mathsf{N}) \leftarrow$ Parameter
3: $\boldsymbol{C} : (\mathsf{D}, \mathsf{N}) \leftarrow$ 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

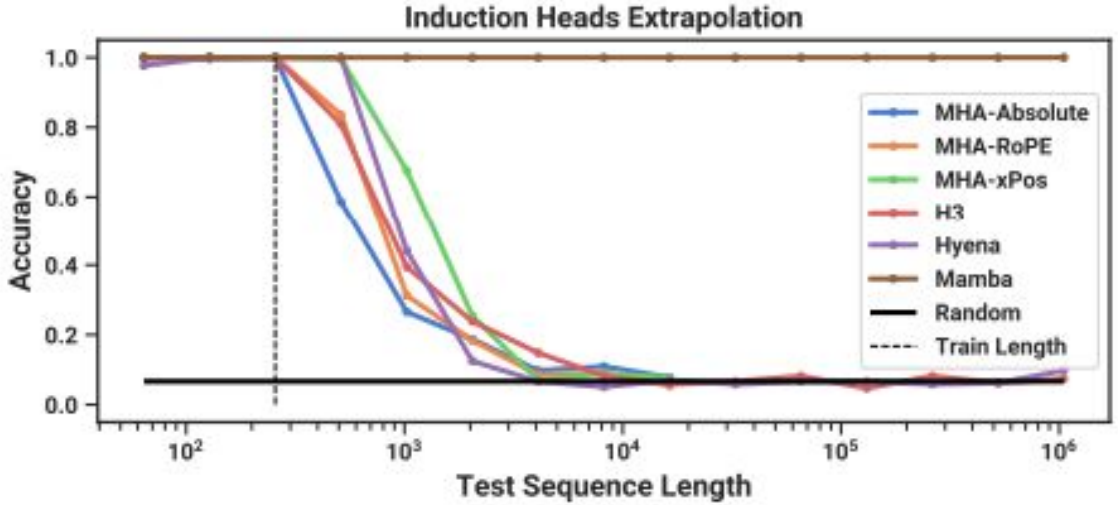
Mamba

- Selective SSMs give us efficient inference, efficient training, and selectivity
- Now we integrate it into a larger architecture
- Convolution is a small linear convolution
  - Captures immediate local context
- Input linear layers project from D to E*D
- Output linear layer projects back to D
- After this block, add & norm (like Transformer)
- Overall architecture is Transformer-like



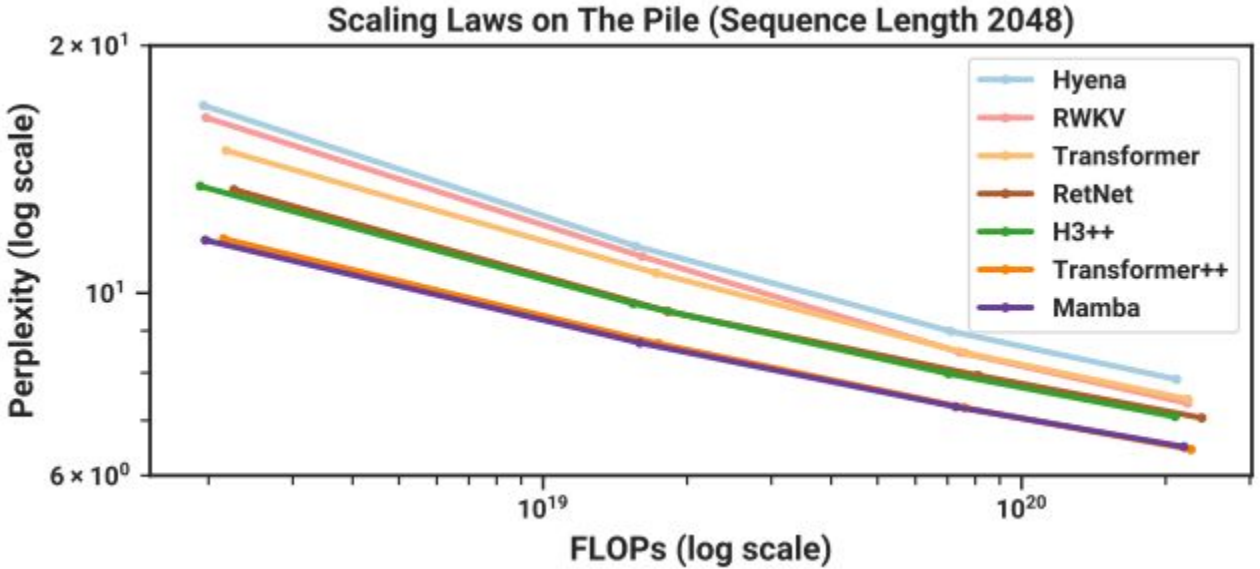
Testing Long-Context: Induction Heads

- Challenge: find the next element in the sequence, given that the preceding element has already appeared before (repeat the same pattern)
  - Ex: If “Harry Potter” has previously appeared, and we receive “Harry” then the next element will be “Potter”



Testing Short-Context: Scaling Laws

- The Pile is a common dataset for LLM training
- Challenge: achieve minimum validation perplexity on the Pile, given a certain FLOPs budget



Task Results

Long Context:

- Mamba maintains nearly perfect recall when compared to other long-range architectures, and is able to extrapolate to longer tasks

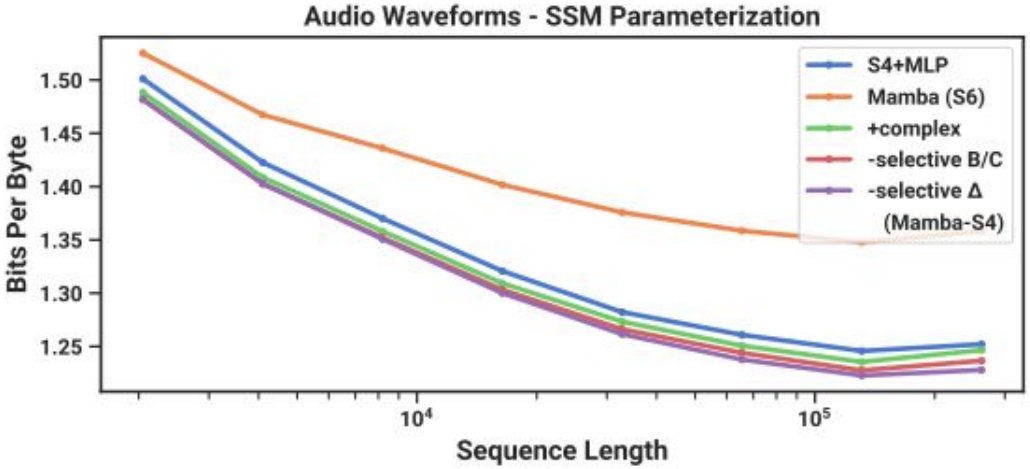
Short Context:

- Mamba scales as well as the Transformer++ architecture (Transformer with a wide variety of popular improvements)

Mamba also does very well on various downstream evaluations, like HellaSwag (commonsense reasoning) when compared to models with the same training

No Free Lunch: Continuous vs Discrete

- S4 models were originally developed for continuous modalities like audio
- Selectivity helps deal with discrete modalities like text
- Ablations indicate that moving from S4 models to Selective SSMs hurts performance on some audio tasks



Other Interesting Notes

For the sake of time, we're skipping over several sections of the results:

- Mamba does very well on DNA sequence learning, showing improved scaling over HyenaDNA and Transformer++
- Mamba's perplexity decreases for very long DNA sequences, while Hyena's increases, indicating that Mamba is better at dealing with long context
- Mamba outperforms the SOTA on speech generation
- Mamba's SSM scan is faster than the best attention implementations for sequence lengths greater than 2000

Conclusions

- Mamba presents an architecture with the following properties:
  - Linear inference in terms of context window size
  - Efficient parallel training
  - Strong empirical performance on long-range retrieval tasks compared to similar architectures that are specialized for long-context tasks
  - Strong empirical performance on short-range tasks compared to the strongest Transformer variants
- Future work: can Mamba be scaled up to the level of modern Transformers?
  - Biggest Mamba is 2.8B parameters, so scaling might not continue
  - Not clear whether things like instruction tuning will generalize to Mamba

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

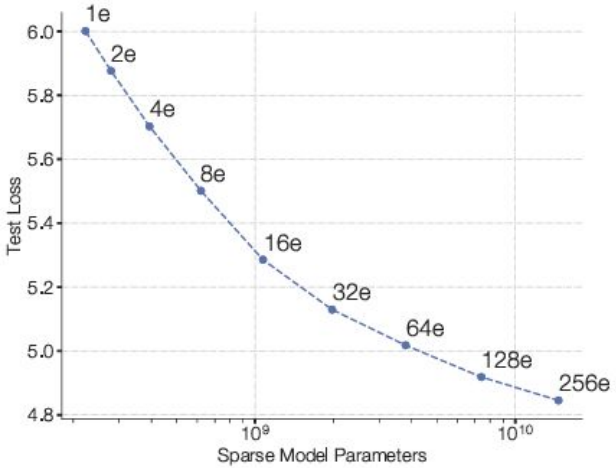
William Fedus, Barret Zoph, Noam Shazeer

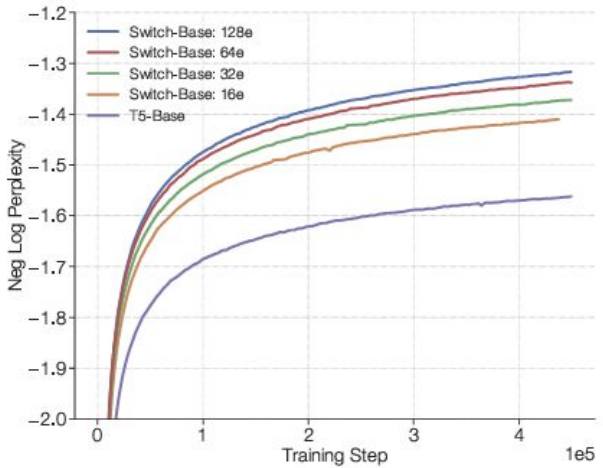
Overview

- Why dense Transformers are expensive to scale
- Mixture-of-Experts (MoE) → Switch (Top-1) routing
- How Switch avoids MoE pain points: simplicity, efficiency, stability
- Results: up to ~7× faster pretraining, strong downstream + multilingual gains

Motivation

- **Observation:** Bigger dense Transformers improve quality (scaling trend), but compute + memory costs become the bottleneck
- **Goal:** Increase parameter count without increasing compute per token
- **Idea:** Use sparsely activated layers: only a subset of weights used for each token





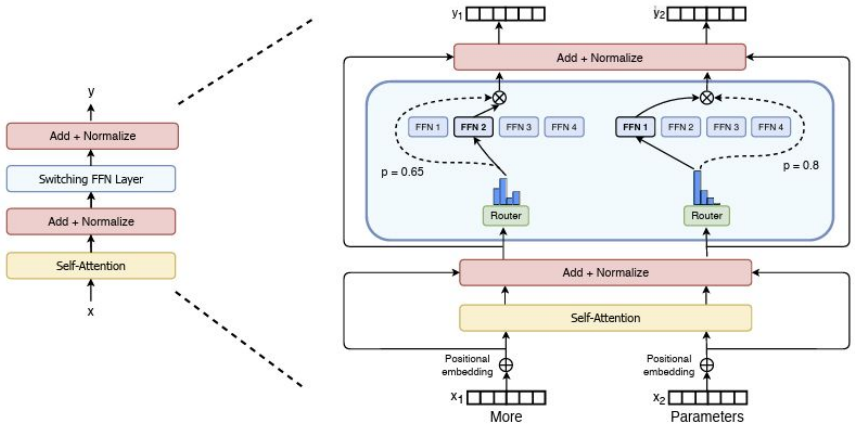
Mixture of Experts

- Router picks top-k experts; output is weighted combination of expert FFNs
- Very large models with constant-ish compute (since only k experts per token)
- Adoption was hard:
  - Complex routing
  - Cross-device communication cost
  - Training instability

Solution: Switch = MoE with Top-1 routing

- Switch Transformer change
  - Replace dense FFN with Switch FFN layer
  - Route each token to exactly 1 expert (k=1)

- Why k=1:
  - Simpler routing computation
  - Less communication
  - Smaller expert capacity needed
  - (Surprisingly) quality preserved / improved

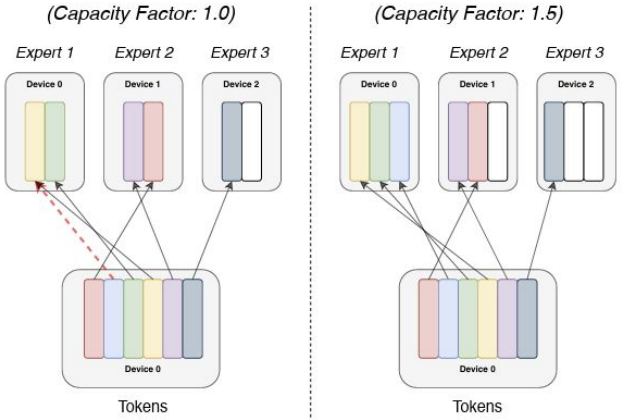


How Routing Works

- Router step (per token x)
  - Compute logits over experts: $h(x) = W_r \cdot x$
  - Softmax gives probabilities over N experts: $p_i(x)$
- Switch decision
  - Choose expert: $i^* = \text{argmax}_i \; p_i(x)$
  - Compute output: $y = p_{i^*}(x) \cdot E_{i^*}(x)$
- Importance
  - Gate value keeps gradients meaningful
  - Only one expert does FFN compute for each token

Expert Capacity and Token Dropping

- Expert capacity: $$capacity = \frac{\text{tokens\_per\_batch}}{\text{num\_experts}} \times \text{capacity\_factor}$$
- If an expert overflows
  - Extra tokens are dropped (skip expert compute)
  - Residual connection passes token onward
- Tradeoff
  - Larger capacity_factor → fewer drops, but more wasted compute/communication



Load Balancing Loss

- **Problem:** Router can send too many tokens to a few experts, causing other experts to be undertrained
- **Solution:** auxiliary load balancing loss

$$\text{loss} = \alpha \cdot N \cdot \sum_{i=1}^{N} f_i \cdot P_i$$

  - Encourage uniform routing across experts
  - Added per Switch layer during training
- Intuitively, we want each expert to receive approximately 1/N of the tokens

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

Training stability trick #1

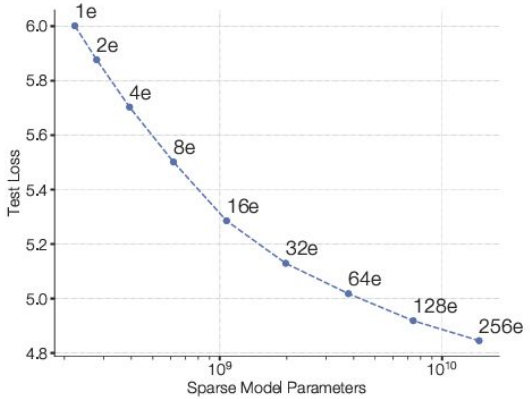
- **Problem:** Low precision (bfloat16) can destabilize routing softmax
- **Solution:**
  - Compute router internals in float32
  - Cast outputs back to bfloat16 so communication stays cheap
- Result: Stability like float32, speed like bfloat16

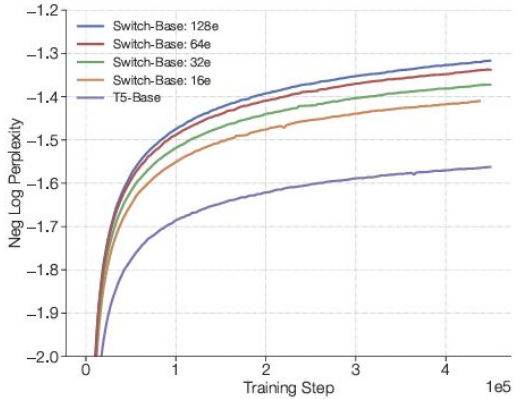
Training stability trick #2

- Smaller initialization
  - Reduce Transformer init scale (paper suggests 0.1×) → improves stability
- Fine-tuning regularization
  - Sparse models have many parameters → can overfit small downstream tasks
  - Use higher dropout inside expert FFNs (“expert dropout”)

Scaling with Experts

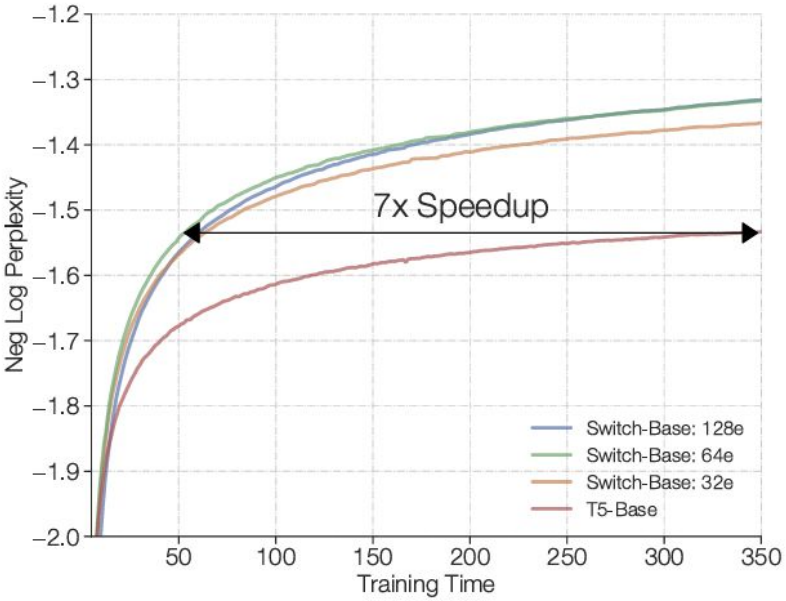
- Keep FLOPs/token ~fixed, increase #experts → more parameters
- Result: more experts ⇒ better perplexity/test loss at same compute budget
- Switch models become more sample-efficient as experts increase





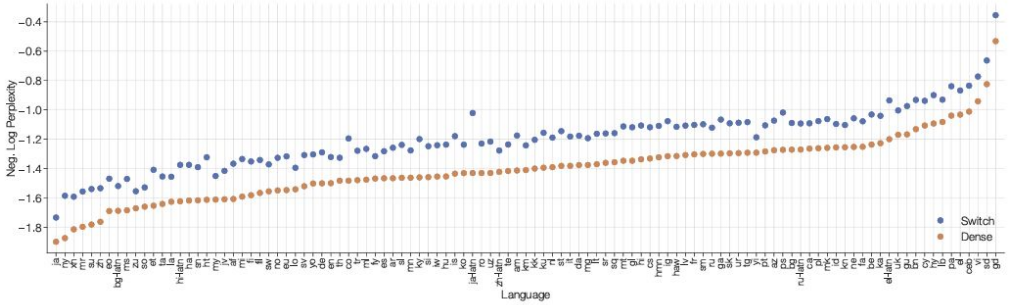
Wall-clock speed

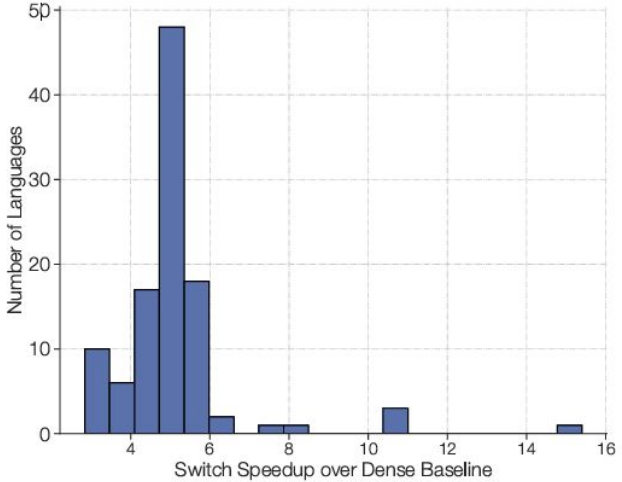
- Step gains translate to real training speedups
- Example: Switch-Base (64 experts) reaches similar quality in ~1/7th the time vs T5-Base
- Sparse wins on “speed-to-quality”



Multilingual Results

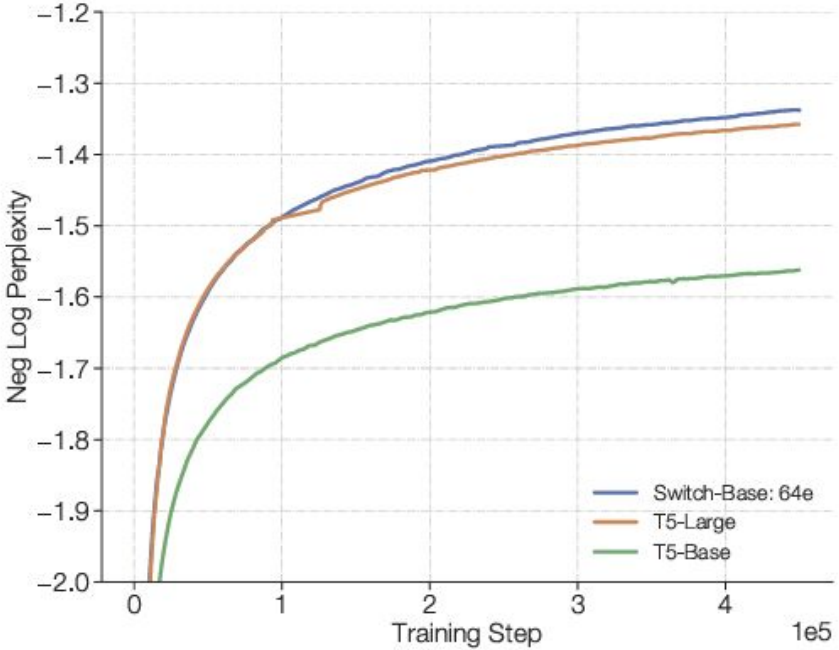
- Pretrain on mC4 (101 languages): Switch improves perplexity across all languages shown
- Speed: mean ~5× step speedup; 91% languages >=4× speedup

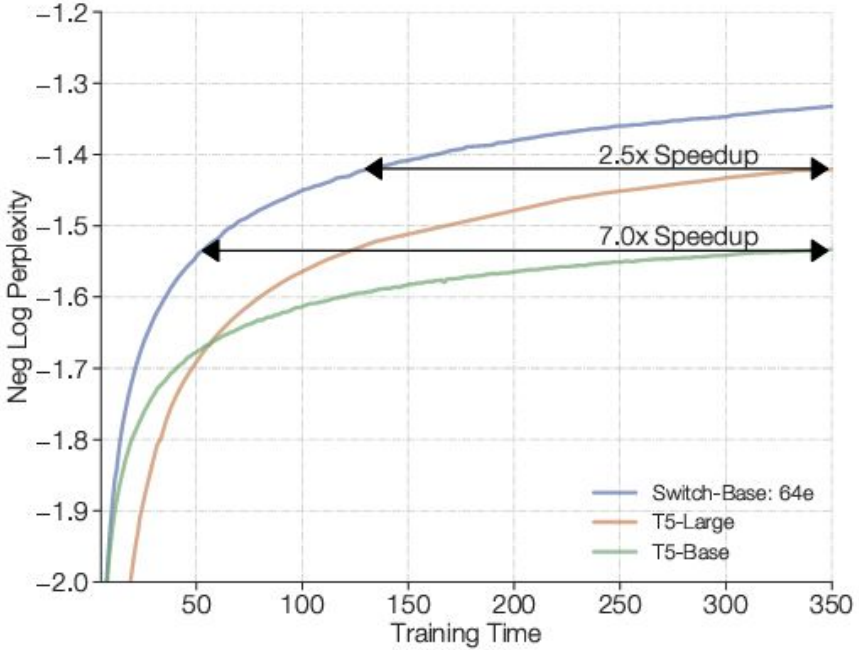




Scaling Versus a Larger Dense Model

If we spend more compute on a bigger dense model (T5-Large), is sparse still worth it?





Fine-tuning results

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**

Summary

- Introduces Switch Transformers: a Mixture-of-Experts Transformer that routes each token to one expert (top-1) for simplicity + efficiency.
- Adds training/engineering techniques to make sparse routing stable and scalable (e.g., load balancing loss, selective precision).
- Demonstrates strong scaling: more experts ⇒ many more parameters at ~constant FLOPs/token, yielding major speed-to-quality improvements.
- Gains transfer to fine-tuning (Table 5), distillation, and multilingual pretraining.

Limitations

- Systems/communication overhead: Routing requires cross-device expert communication; scaling is not “free” in wall-clock terms even if FLOPs/token are similar.
- Capacity & token dropping tradeoff: Expert capacity must be chosen carefully; overload can lead to dropped tokens (quality risk) while higher capacity wastes compute/memory.
- Training sensitivity: Routing introduces instability (e.g., precision issues in router softmax), requiring special tricks (selective float32 in routing, careful initialization).
- Load balancing is essential: Without balancing, experts can be underused/overused; requires auxiliary losses and tuning.
- Not universal improvements across tasks: Some benchmarks (e.g., ARC variants) show weaker/mixed gains vs dense baselines.

Summary

How does each improvement help make models efficient?

Improvement	Problem	Solution
Attention Sinks	LLM quality degrades outside of the initial training window.	Keep some of the earliest tokens as an “attention sink”.
Mamba	Existing models are either inefficient to train, scale poorly with sequence length, or lack selectivity.	Introduce selectivity into S4 models, and then integrate Selective SSMs into a Transformer-like architecture.
Switch Transformer	Dense Transformers are expensive to scale, but MoEs have several pain points.	Use only one expert, along with a load balancing loss and training stability tricks

Thank You!
Questions?