Efficient Streaming Language Models with Attention Sinks

Authors: Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, Mike Lewis
Presenters: Amandeep Gupta, Yiren Wang

Georgia Tech

Motivation

Fast and accurate inference

- Streaming LLM responses require:
    1. Outputs to generalize outside their pretraining context length
    2. Inference to be fast for real time streaming

Examples:

1. Multi-round dialogue/ChatBots: Persistent, continuous interaction over extended periods
2. Long horizon tasks & Code Completion: Generating long trajectories through continuous running agents

Georgia Tech

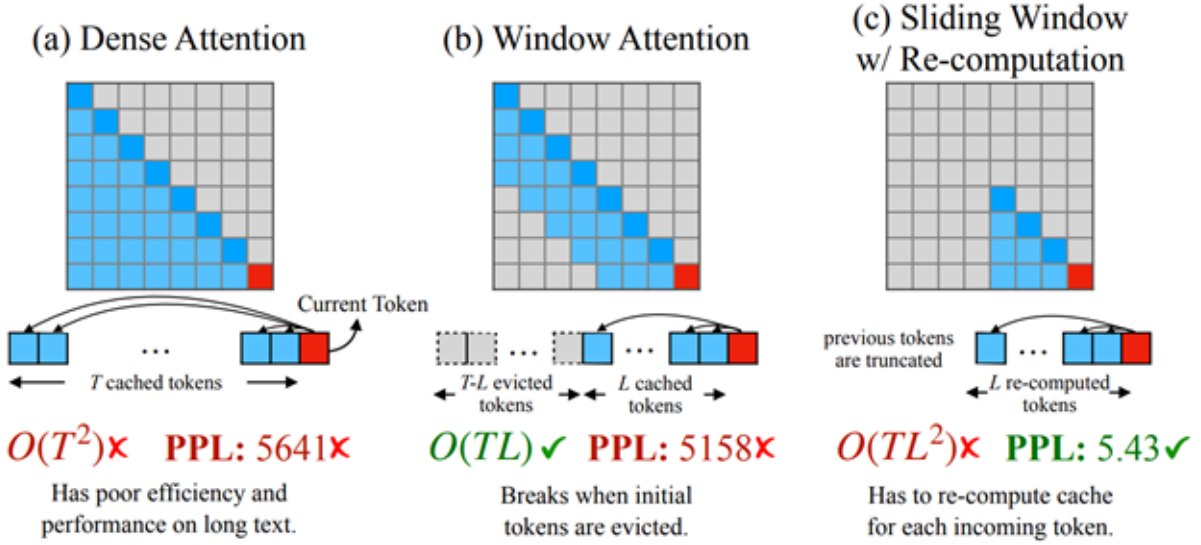
KV Caches

KV Caches

- Provide linear time next token prediction during inference
- Are limited by memory
- Old tokens need to be evicted - **Window Attention**

Current approaches

- Suffer from degraded outputs after eviction, or
- Require re-computation of all **Value** vectors after every eviction - $O(TL^2)$



Georgia Tech

Prior Work

Broad Categories of research

Length extrapolation

- Aims to enable language models trained on shorter texts to handle longer ones during testing
- Example: relative position encoding methods for transformers RoPE, ALiBi

Context Window Extension

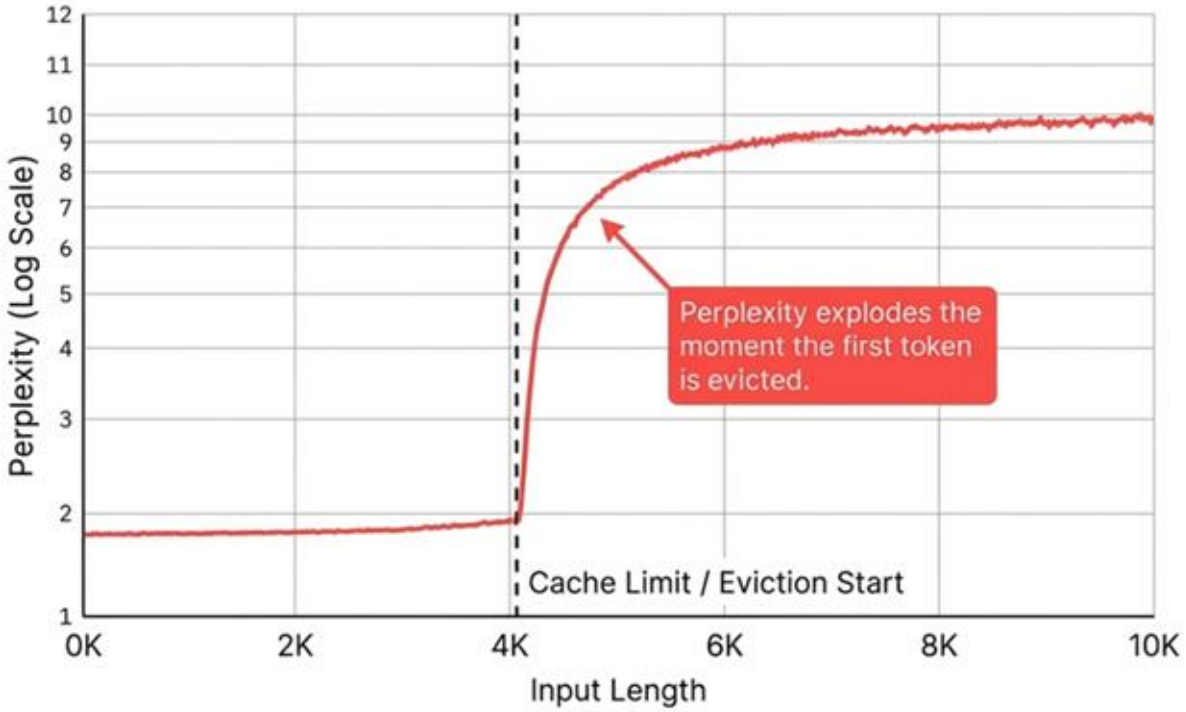
- Enabling the processing of more tokens in one forward pass
- Example: approximate attention methods, system-focused optimizations like FlashAttention

Improving LLMs' Utilization of Long Text

- Optimizes LLMs to better capture and employ the content within the context rather than merely taking them as inputs.

Georgia Tech

Window Attention



- Naive cache eviction: maintains only a fixed-size sliding window on KV states of **most recent** tokens
- This ensures constant memory and decoding speed after cache is filled but,
- The model collapses once sequence length exceeds cache size i.e. even evicting KV of first token

- Memory ✅
- Time ✅
- Perplexity ❌

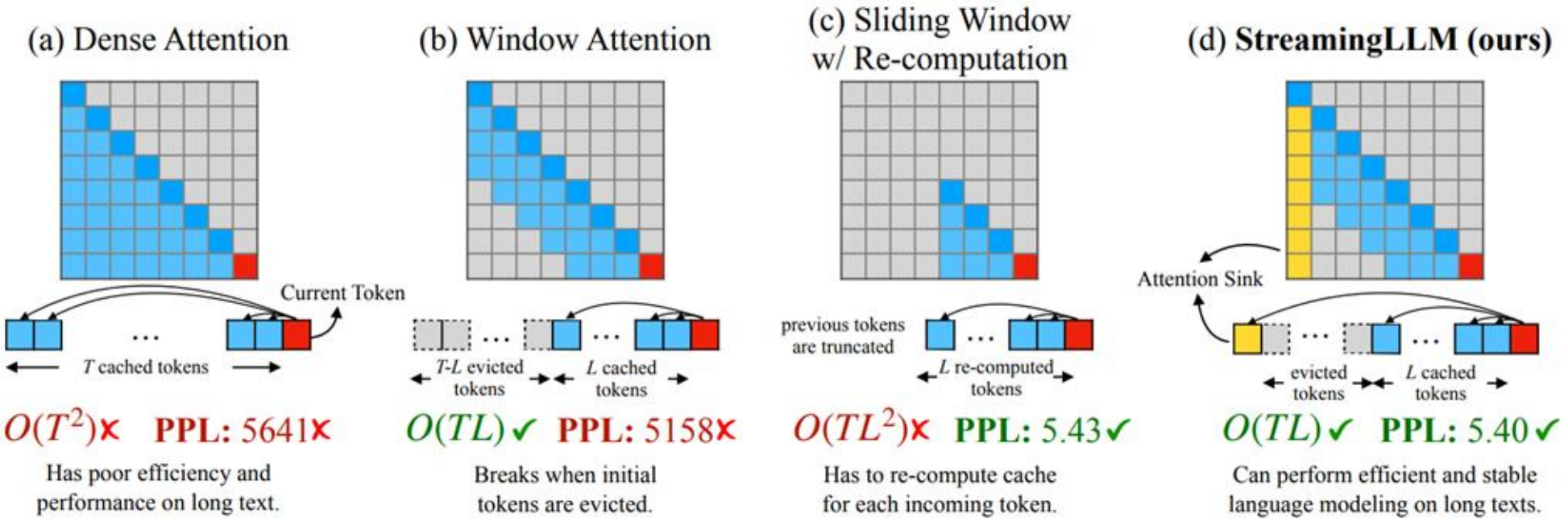
Georgia Tech

Sliding window with re-computation

- Memory ✓
- Time ✗
- Perplexity ✓

Window attention but

- Rebuilds the KV states of recent tokens in window - for every new token
- This preserves performance but is slower than even dense attention due to re-computations
- However the approach still requires only fixed memory



Georgia Tech

Key Technical Challenges

Challenge 1

Limited KV cache size vs. need for long-term context

Challenge 2

Performance degradation when cache eviction is naive

Challenge 3

Maintaining model stability and perplexity with streaming input

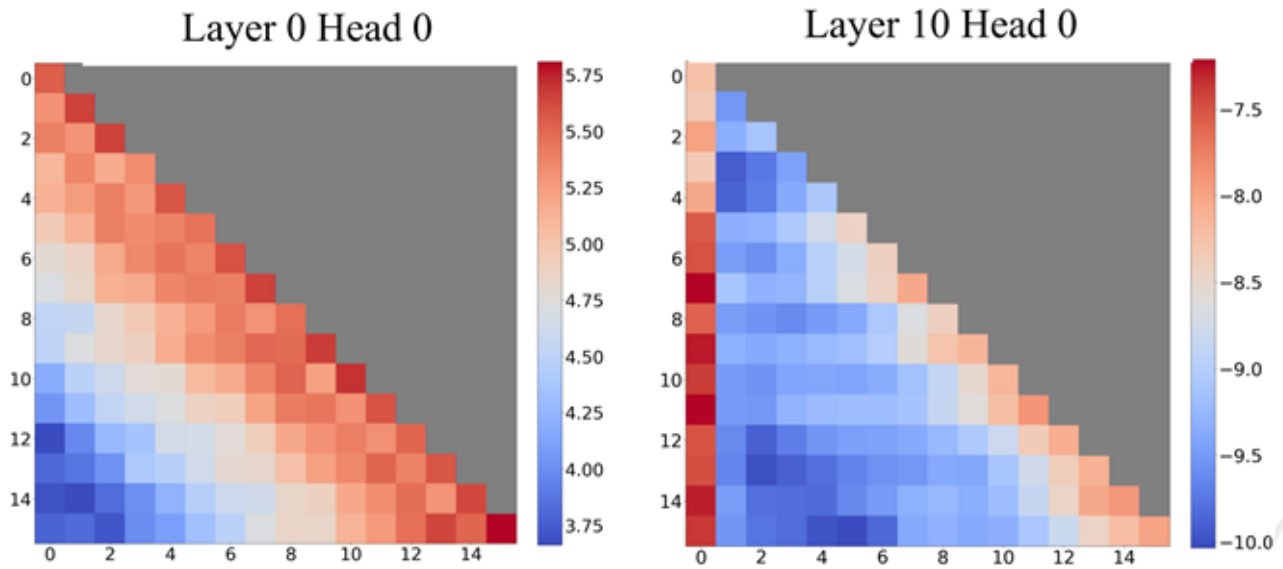
Georgia Tech

Solution: Attention Sinks

Main ideas

Perplexity spikes with the exclusion of initial tokens

- Suggest that initial tokens, regardless of their distance, are crucial
- Beyond the bottom two layers, the model consistently focuses on the initial tokens across all layers and heads
- Removing these token's KV will remove a considerable portion of the denominator in the SoftMax function - destorting output
- Either (1) their semantics are crucial or (2) model learns a bias towards their absolute position



Why are initial tokens crucial

Semantics or absolute position: What makes them special?

- Experiment: Fix first 4 tokens as linebreak (“\n”) tokens in the window
- Result: Model still significantly emphasizes these initial linebreak tokens

Absolute position of the starting tokens, rather than their semantic value, hold significance.

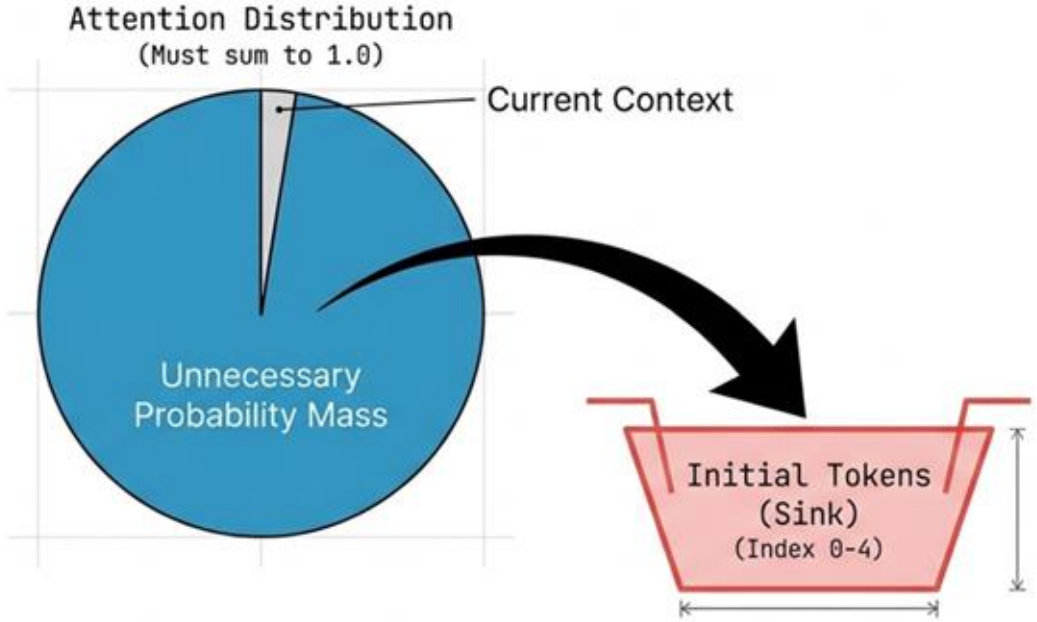
Table 1: Window attention has poor performance on long text. The perplexity is restored when we reintroduce the initial four tokens alongside the recent 1020 tokens (4+1020). Substituting the original four initial tokens with linebreak tokens “\n" (4"\n"+1020) achieves comparable perplexity restoration. Cache config x+y denotes adding x initial tokens with y recent tokens. Perplexities are measured on the first book (65K tokens) in the PG19 test set.

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

Georgia Tech

LLMs attend to Initial Tokens as Attention Sinks

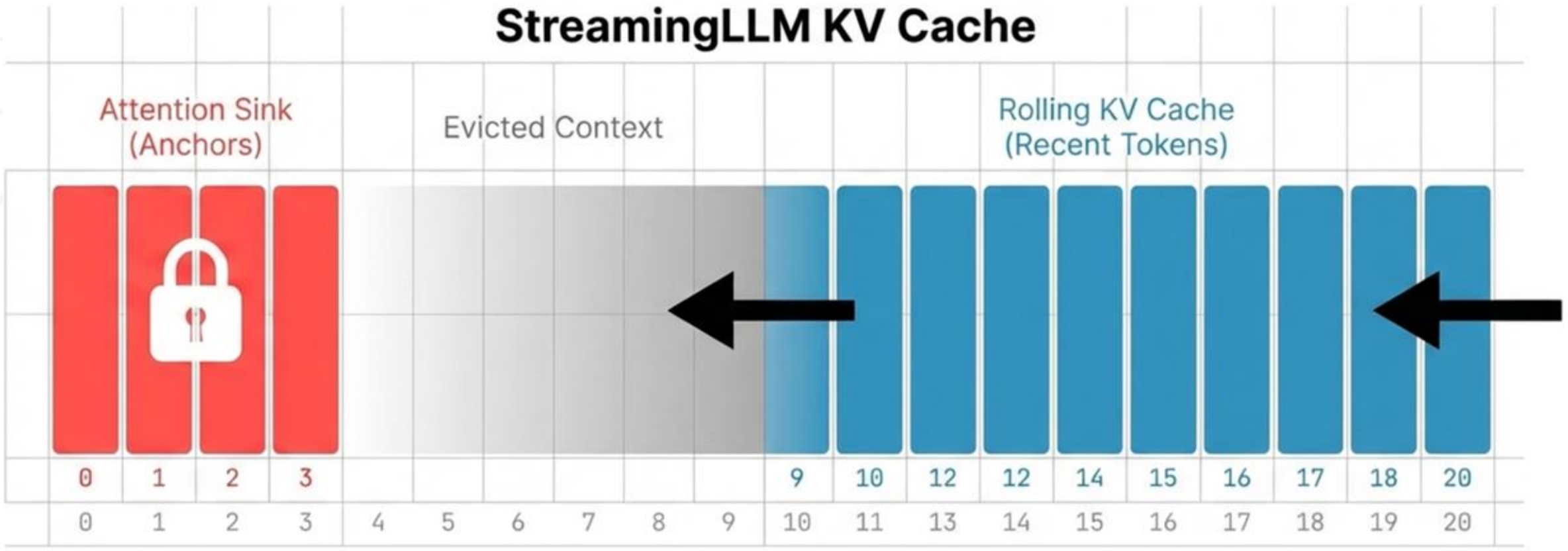
- Despite sufficient self-contained information, attention values must sum to 1
- Consequently, the model dumps unnecessary attention values to specific tokens
- Why initial tokens are the ‘sinks’: Due to the sequential nature of autoregressive language modeling, initial tokens are visible to all subsequent tokens
- LLMs typically utilize multiple initial tokens as attention sinks rather than just one
- SoftMax-Off-by-One (Miller, 2023) is also a potential remedy



Georgia Tech

StreamingLLM

Rolling KV cache with attention sinks



Georgia Tech

Some Details

- When adding positional information to tokens, StreamingLLM focuses on positions within the cache rather than those in the original text.
- For encoding like RoPE, the Keys of tokens are cached prior to introducing the rotary transformation.
- Then, position transformation is applied to the keys in the rolling cache at each decoding phase.
- This ensures that the model operates efficiently even beyond its pre-training attention window size.

Georgia Tech

Pre-training with Attention Sinks

- A potential remedy can be the intentional inclusion of a global trainable attention sink token, denoted as a “Sink Token”, which would serve as a repository for unnecessary attention scores.

- Alternatively, replacing the conventional SoftMax function with a variant like SoftMax-off-by-One (Miller, 2023)

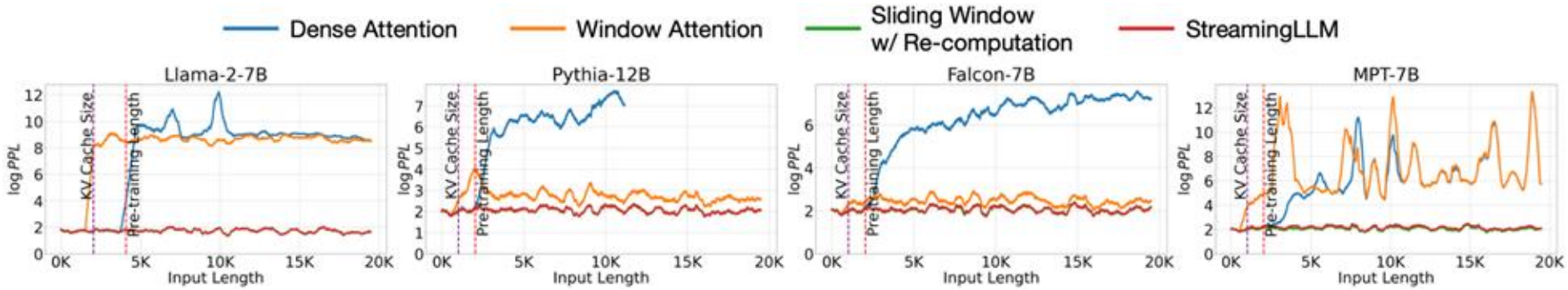
Table 3: Comparison of vanilla attention with prepending a zero token and a learnable sink token during pre-training. To ensure stable streaming perplexity, the vanilla model requires several initial tokens. While Zero Sink shows a slight improvement, it still needs other initial tokens. Conversely, 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

Georgia Tech

Evaluation & Results

Perplexity vs. Sequence Length

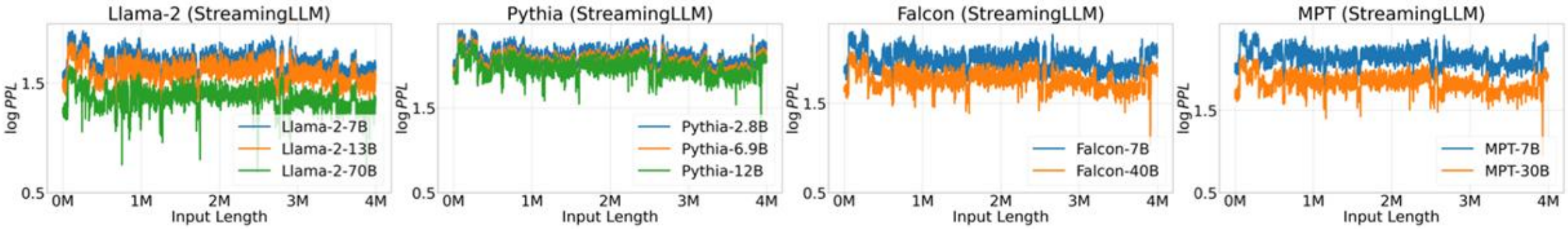


- Test Settings:
  - LLM: Llama-2-7B, Pythia-12B, Falcon-7B, MPT-7B
  - Benchmark: PG-19 Test Set
  - Input Length: 0 - 20K Tokens
  - Window Size: 2048 (Llama-2), 1024 (Falcon, Pythia and MPT)
- Dense Attention runs OOM around 20K tokens on an 80G A100.
- Window Attention crashes when the instant initial tokens leave the cache.
- StreamingLLM (4 sinks) maintains stable perplexity (PPL≈5.4).

Georgia Tech

4 Million Token Stability

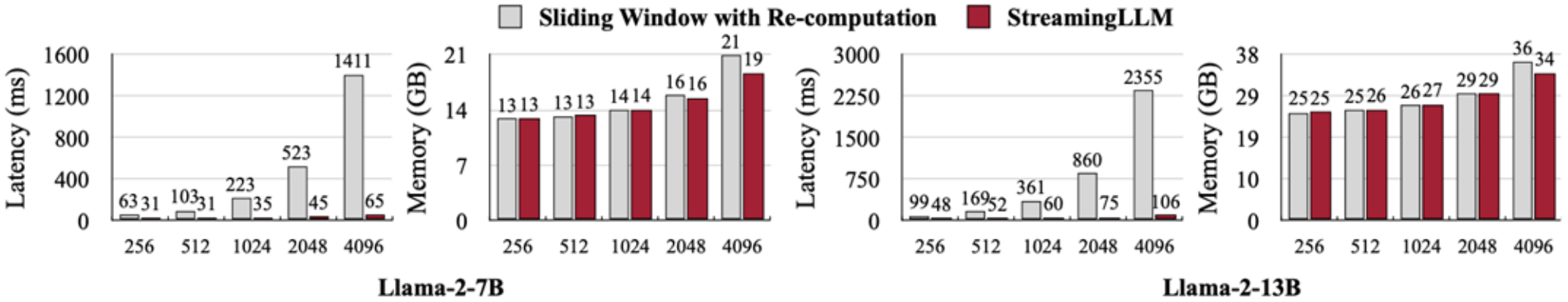
- Stress test: 10 models, continuously up to 4 million tokens
  - Models: Llama-2-7B/13B/70B, Falcon-7B/40B, MPT-7B/30B, Pythia-2.8B/6.9B/12B
  - Results: Stable perplexity throughout the 4M token range
- Baselines at 4M tokens:
  - Dense attention: OOM at ~ 20K tokens
  - Window Attention: crashes right after sink eviction
  - Recomputation: functionally correct but 22× slower



Georgia Tech

Efficiency: Speed & Memory

- Latency: 22.2× speedup
  - Recomputation: O(W) per token
  - StreamingLLM: O(1) per token
- Memory usage: O(N+W) constant throughout generation
  - N=4 sink tokens, W is the sliding window



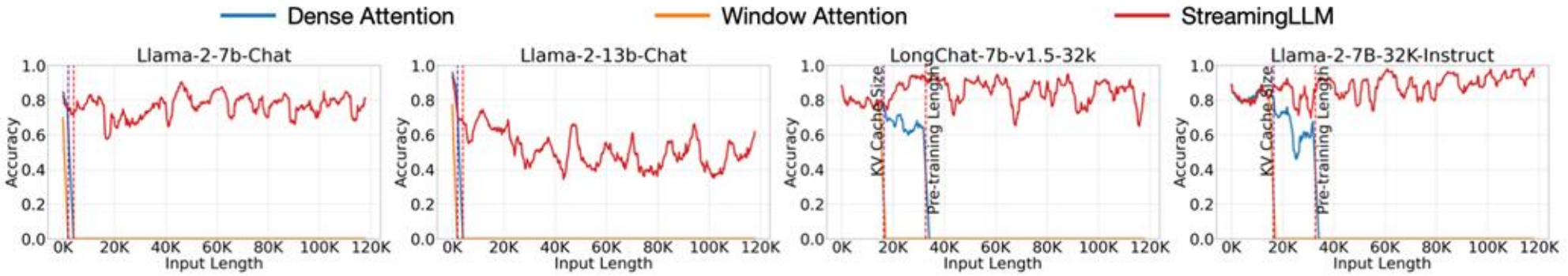
Georgia Tech

Streaming QA & Ablation Results

- StreamEval benchmark: multi-hop streaming QA across 120K token contexts
  - StreamingLLM: Accuracy matches one-shot baseline
  - Window Attention: Accuracy degrades to near-random at 10K+ tokens

- Ablation: # of sink tokens (N)
  - N=0: PPL=5158 — Instant Failure
  - N=1: PPL=5.8 — Near-stable
  - N=4: PPL=5.4 — Optimal

- Ablation: window size W
  - W=256: PPL=5.7 — Stable; Minimal memory
  - W=1000: PPL=5.4 — Sweet Spot



Georgia Tech

Research Evolution

Three Ways to Handle Attention Sinks

Accept & Preserve

StreamingLLM: Keep sink tokens in the KV cache. Attention sinks are a natural byproduct of Softmax — work around them rather than fighting the architecture.

Soften Softmax

Softmax1: Allow Softmax outputs to sum to less than 1. This reduces the pressure to dump excess attention, weakening sinks without a full architectural change.

Eliminate Sinks

Softpick: Replace Softmax Norm with a ReLU-based scheme that has no sum-to-one constraint. Attention weights can now sum to anything.

The Problem: Softmax Demands Attention Be Spent

- Root Cause: Softmax forces all attention weights to sum to exactly 1

- All three approaches target this same problem from different angles:
  - **StreamingLLM**: keeps sink tokens' KV in cache
  - **Softmax1**: truncates the normalization — minimal sinks
  - **Softpick**: removes sum-to-one via ReLU — no sinks at all

Georgia Tech

Approach 1: Preserve Sinks — StreamingLLM

- **Core idea**: accept that sinks exist and ensure they are always in the KV cache
  - Keep N=4 sink tokens (initial tokens) in cache alongside rolling window
  - Sinks are never evicted; window tokens are evicted oldest-first
- **Strengths**:
  - No modification to attention mechanism
  - Works on any existing model without retraining
  - 22× speedup, constant memory, proven stable at 4M tokens
- **Trade-offs**:
  - Attention is still “wasted” on semantically empty tokens
  - Does not improve quantization because of massive activation
  - Still bounded by window size W for effective context length

Georgia Tech

Approach 2: Weaken the Constraint — Softmax1

- **Core idea**:

Standard Softmax:

$$softmax(x)_i = \frac{\exp(x_i)}{\sum \exp(x_j)}$$

Softmax1 (Proposed):

$$softmax1(x)_i = \frac{\exp(x_i)}{1 + \sum \exp(x_j)}$$

- **Strengths**:
  - Attention maps become sparser and more semantically targeted
  - Initial tokens still accumulate some attention but significantly less
  - Can be combined with StreamingLLM for even fewer required sink tokens

- **Trade-offs**:
  - Requires fine-tuning or retraining from scratch — not training-free
  - Partial solution: sinks weakened but not fully eliminated

Georgia Tech

Approach 3: Eliminate Sinks — Softpick (Zuhri et al., 2025)

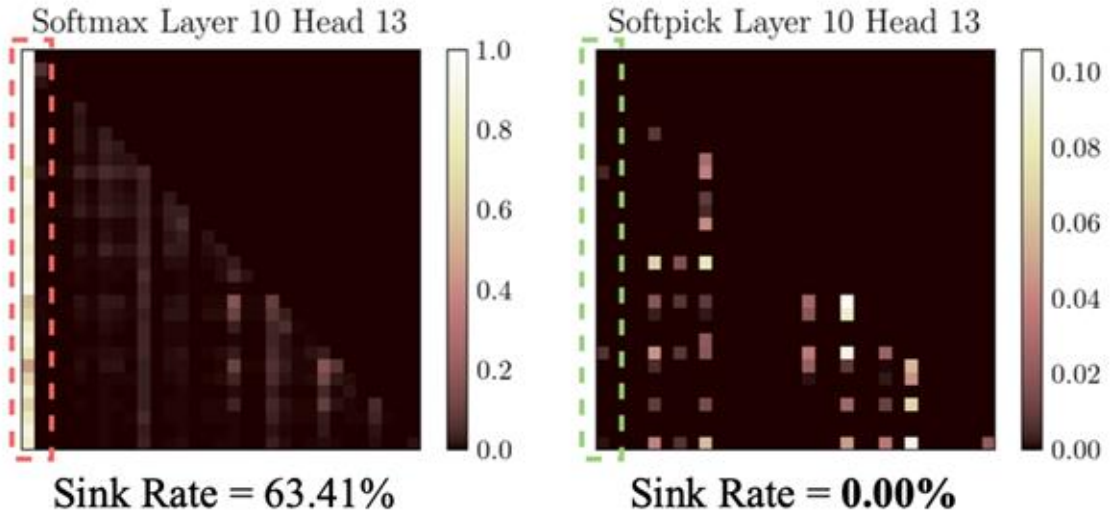
- **Core idea**:

Standard Softmax:

$$\text{softmax}(x)_i = \frac{\exp(x_i)}{\sum \exp(x_j)}$$

Softpick (Proposed):

$$\text{Softpick}(x)_i = \frac{\text{ReLU}(e^{x_i} - 1)}{\sum_{j=1}^{N} |e^{x_j} - 1|}$$



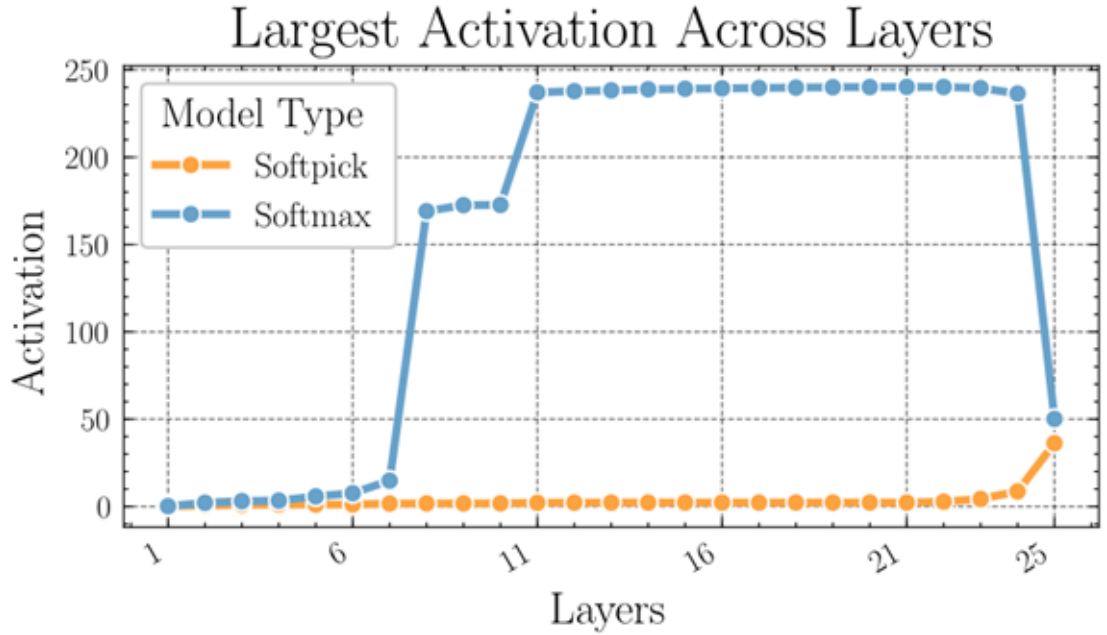
- **Strengths**:
  - Model can attend to nothing if irrelevant
  - Massive activations in initial tokens no longer dominate after ReLU zeroing
  - Sink tokens receive near-zero attention in most layers

- **Trade-off**:Requires training from scratch on Softpick architecture

Georgia Tech

Softpick: Results

- Attention Sinks: sink rate drops from 63.4% to 0.0%
- Massive Activations: completely absent in Softpick-trained models



- **Language Model Quality**:
  - WikiText-103: within 0.2 PPL of equivalent Softmax model
  - MMLU, HellaSwag, ARC: within noise margin of Softmax model
- **Quantization Advantage**:
  - Massive activations make 2/4-bit quantization difficult
  - Softpick models: significantly better quality at low-bit quantization (≤4 bit)
  - Practical implication: edge deployment becomes more practical

Georgia Tech

Comparing the Three Approaches

Feature	StreamingLLM	Softmax1	Softpick
Mechanism	Keep sinks in cache; Softmax unchanged	Add 1 to denominator; output sum ≤1; sparser attention	>0 Replace Softmax with ReLU-norm; no sum-to-one; no sinks
Training Cost	None (existing models)	Fine-tuning or pre-training	Full pre-training from scratch
Best For	Deploying existing open-source LLMs in streaming settings today	New training runs where some retraining cost is acceptable	Future models prioritizing quantization and truly sparse attention

Georgia Tech

Critical Perspective & Takeaways

Critical Perspective

⚠ StreamEval is too favorable

Each answer is placed exactly 20 lines (~460 tokens) before the query — always within the 1000-token window. It confirms in-window retrieval but never tests the evicted-context case.

⚠ Sink Token Choice Is Heuristic

N=1 already recovers 95% of performance. N=4 is empirical, not principled. For instruction-tuned models where system prompts occupy early tokens, what qualifies as a sink is ambiguous.

❓ Open Question: Highly Compressed Attention

Modern architectures use heavily compressed attention (MLA/DSA). Sink behavior in these compressed spaces is unstudied. Does StreamingLLM still need N=4 sinks when KV is compressed 8x?

Key Takeaways

Sinks Are Structural

Attention sinks arise because Softmax forces all attention to be spent. Massive activations in initial tokens make them the default dump. This is structural instead of a training artifact to be debugged away.

StreamingLLM

Keeping 4 sink tokens + a rolling window gives constant O(N+W) memory, 22× speedup, and stable PPL at 4M+ tokens without fine-tuning. Deployable on any open-source LLM.

Three Paths

StreamingLLM works around sinks; Softmax1 weakens sinks; Softpick eliminates sinks. Together, these papers show that the field is converging on a clear understanding and practical solutions.

Georgia Tech

Thank you!
Q&A