BLASST: Dynamic BLocked Attention Sparsity via Softmax Thresholding

Jiayi Yuan[*1], **Cameron Shinn**[*2], Kai Xu[3], Jingze Cui[3], George Klimiashvili[3], Guangxuan Xiao[3], Perkz Zheng[3], Bo Li[3], Yuxin Zhou[3], Zhouhai Ye[3], Weijie You[3], Tian Zheng[3], Dominic Brown[3], Pengbo Wang[3], Markus Hoehnerbach[4], Richard Cai[3], Julien Demouth[3], John D. Owens[2], Xia Hu[1], Song Han[3], Timmy Liu[3], Huizi Mao[3]

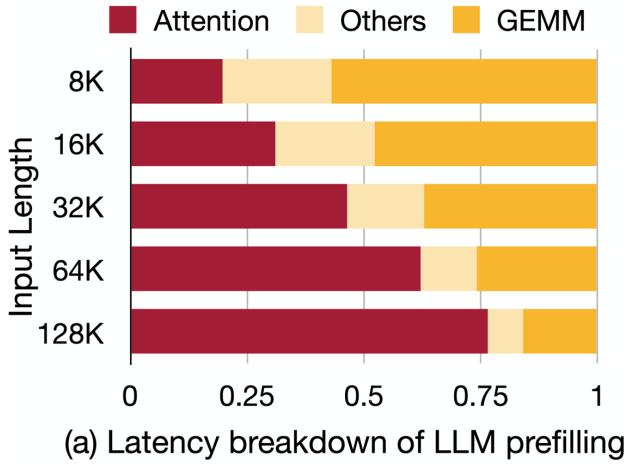
[1]Rice University, [2]UC Davis, [3]NVIDIA, [4]Meta

Background – LLMs Continue to Scale

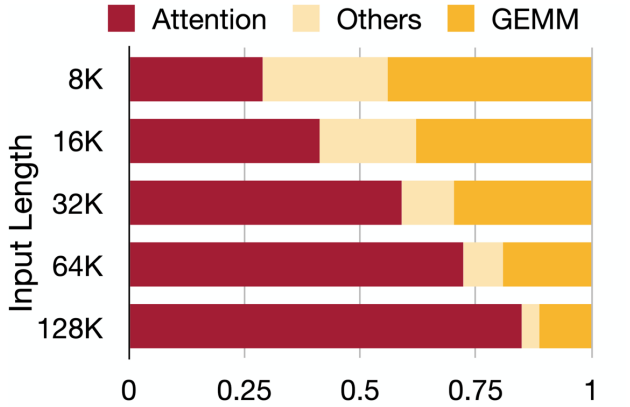
LLM inference context lengths are growing dramatically.

- System prompt tokens ⬆️
- Reasoning and chain-of-thought tokens ⬆️
- Content tokens (code, images, video) ⬆️

- VLM and diffusion models are rising in popularity and consume many tokens.



(a) Latency breakdown of LLM prefilling



(b) Latency breakdown of LLM decoding

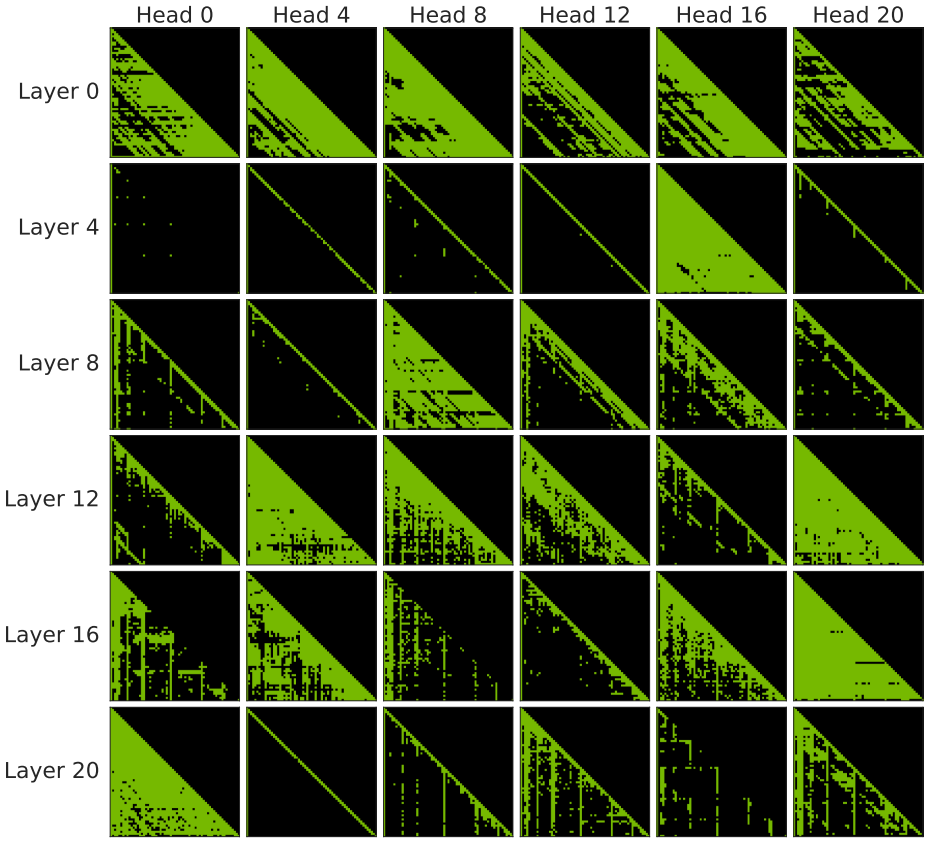
Attention dominates prefill and decode time as sequence length increases. Data from Llama-3-8B on A100 GPU.
Figure credit: LServe, Yang et al., 2025

Background – Attention Is Sparse

Attention is intrinsically sparse, with specific patterns being observed.

- Global, windowed and random sparsity (Big Bird, Zaheer et al., 2020).
- Attention sinks (StreamingLLM, Xiao et al., 2023).
- Λ-shape, vertical-slash, and block-sparse patterns (MInference, Jiang et al., 2024).
- DeepSeek dynamic sparse variants: HCA/CSA/DSA

Large body of existing work on how to leverage attention sparsity.



Llama 3.1 8B sparsity patterns from RULER sample

Barriers to Adoption

Existing works can be difficult to deploy for a few reasons...

Barriers to Adoption

Existing works can be difficult to deploy for a few reasons…

1. Pre-computation to decide sparsity pattern reduces speedup.

Barriers to Adoption

Existing works can be difficult to deploy for a few reasons…

1. Pre-computation to decide sparsity pattern reduces speedup.
2. Requiring training for added layers or a new architectures.

Barriers to Adoption

Existing works can be difficult to deploy for a few reasons…

1. Pre-computation to decide sparsity pattern reduces speedup.
2. Requiring training for added layers or a new architectures.
3. Exclusive to either prefill or decode phase.

Barriers to Adoption

Existing works can be difficult to deploy for a few reasons…

1. Pre-computation to decide sparsity pattern reduces speedup.
2. Requiring training for added layers or a new architectures.
3. Exclusive to either prefill or decode phase.
4. Not optimized or supported on modern GPUs.

Barriers to Adoption

Existing works can be difficult to deploy for a few reasons…

1. Pre-computation to decide sparsity pattern reduces speedup.
2. Requiring training for added layers or a new architectures.
3. Exclusive to either prefill or decode phase.
4. Not optimized or supported on modern GPUs.
5. Intrusive modifications to attention APIs that are hard for frameworks to adopt.

Barriers to Adoption

Existing works can be difficult to deploy for a few reasons…

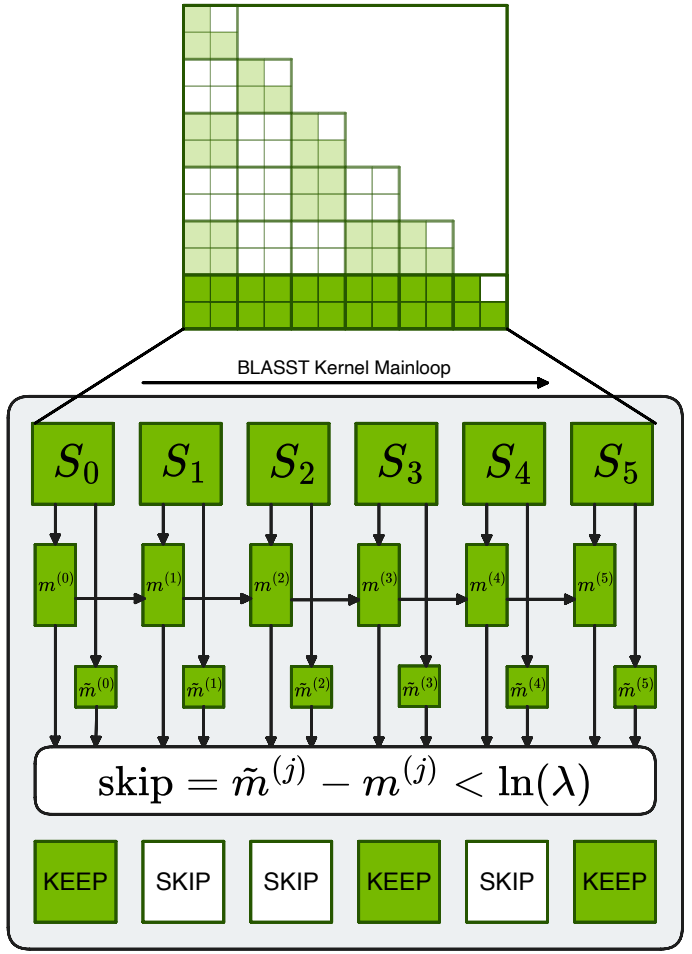
1. Pre-computation to decide sparsity pattern reduces speedup.
2. Requiring training for added layers or a new architectures.
3. Exclusive to either prefill or decode phase.
4. Not optimized or supported on modern GPUs.
5. Intrusive modifications to attention APIs that are hard for frameworks to adopt.

How can we avoid these and make sparse attention easier to use?

Our Solution: BLASST

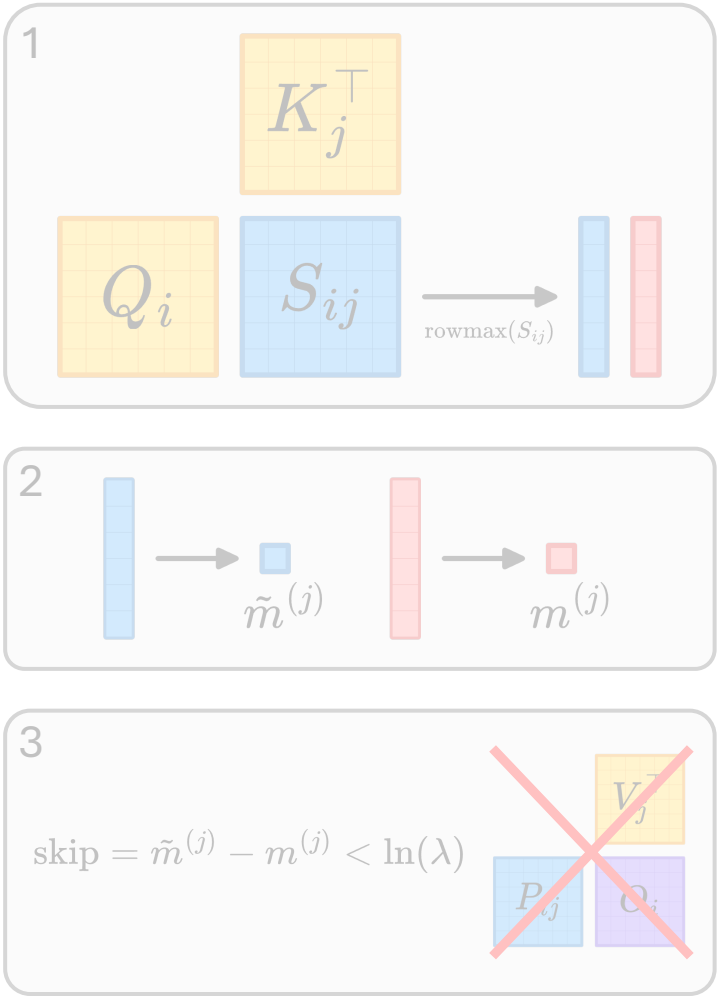
Dynamic block sparse attention using softmax thresholding

- Minimal impact on accuracy.
- Accelerates both **prefill and decode** stages with ~**no overhead**.
- **Training-free**.
- **Blackwell and Hopper** kernel support.
- **Seamless integration** existing frameworks, with a simple kernel backend swap.



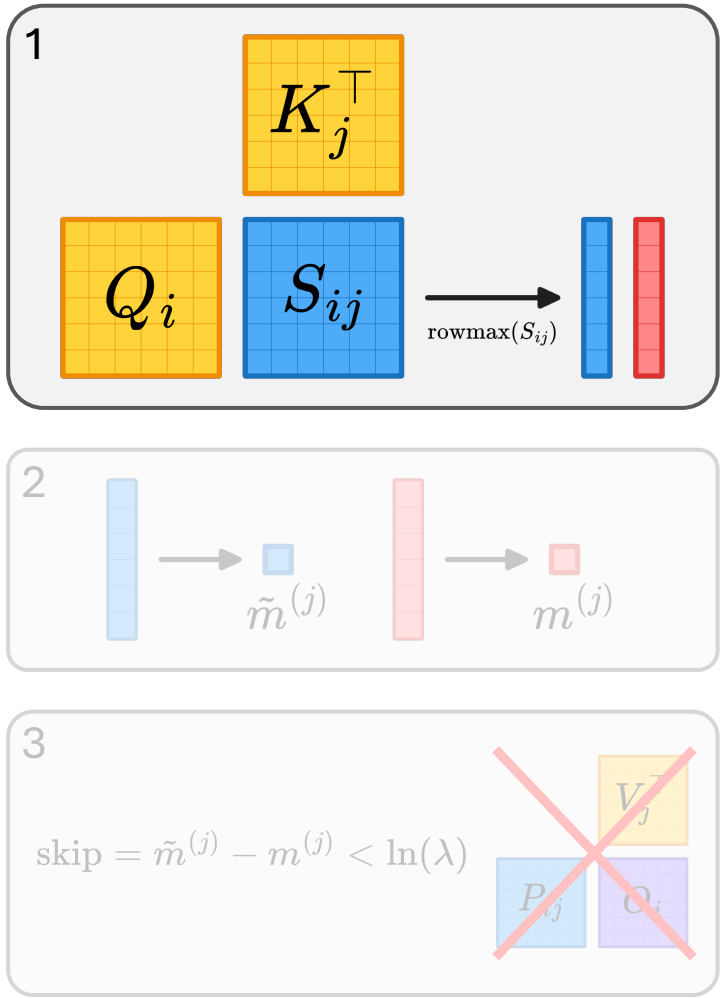
BLASST Algorithm

1. After $QK^\top$, online softmax already computes local row max of S, and updates running row max.
2. Take max across rows for both.
3. If local block max is below running block max by more than $\ln(\lambda)$: **Skip softmax, V load and $PV^\top$ matmul.**



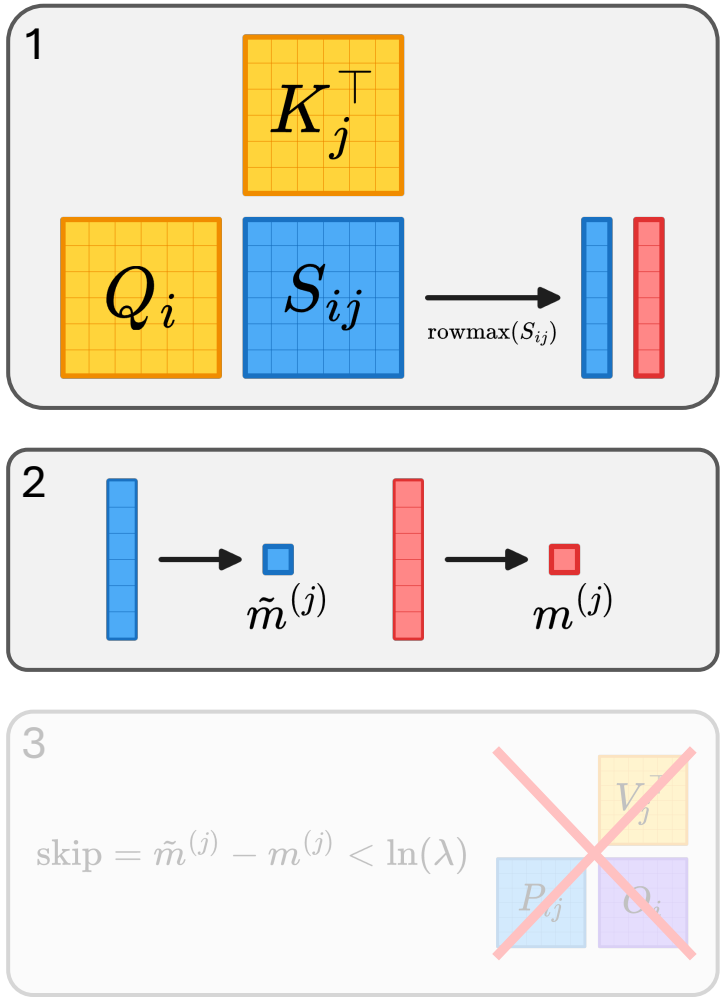
BLASST Algorithm

1. After $QK^\top$, online softmax already computes local row max of S, and updates running row max.
2. Take max across rows for both.
3. If local block max is below running block max by more than $\ln(\lambda)$: **Skip softmax, V load and $PV^\top$ matmul.**



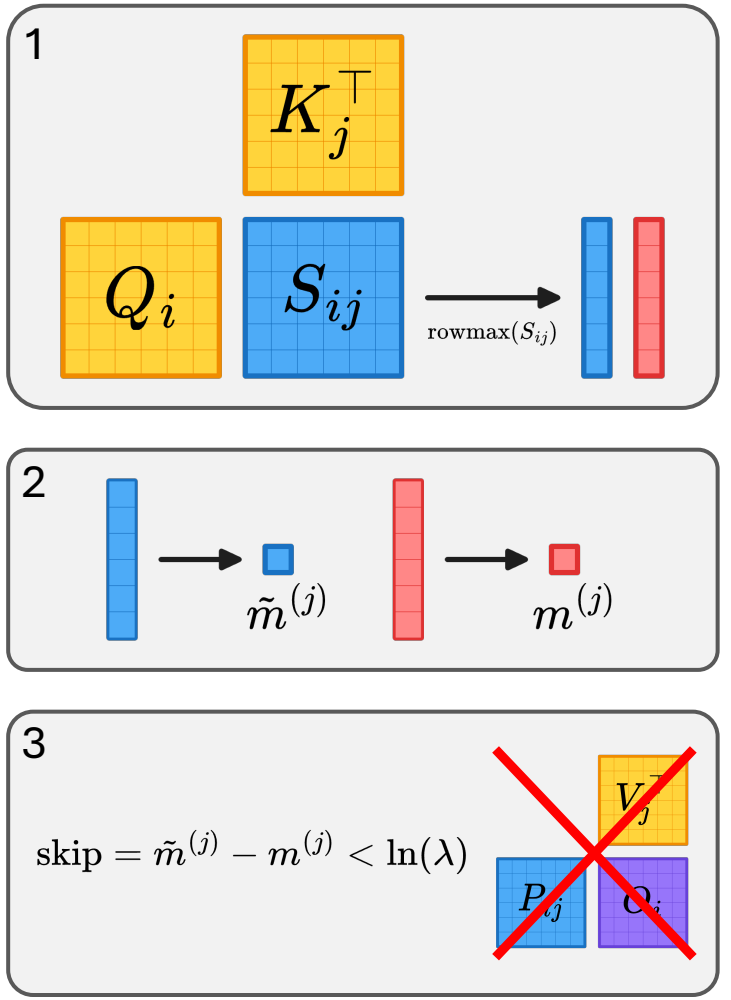
BLASST Algorithm

1. After $QK^\top$, online softmax already computes local row max of S, and updates running row max.
2. Take max across rows for both.
3. If local block max is below running block max by more than $\ln(\lambda)$: **Skip softmax, V load and $PV^\top$ matmul.**



BLASST Algorithm

1. After $QK^\top$, online softmax already computes local row max of S, and updates running row max.
2. Take max across rows for both.
3. If local block max is below running block max by more than $\ln(\lambda)$: **Skip softmax, V load and $PV^\top$ matmul**.



Minimal Changes to FlashAttention Algorithm

- Very minor algorithmic changes to support skipping (denoted in green).
- Aim to introduce as few extra instructions as possible. Re-use rowmax operation.
- Same block size as existing FA, maps directly onto tiled computation.

Algorithm 1 FlashAttention with BLASST

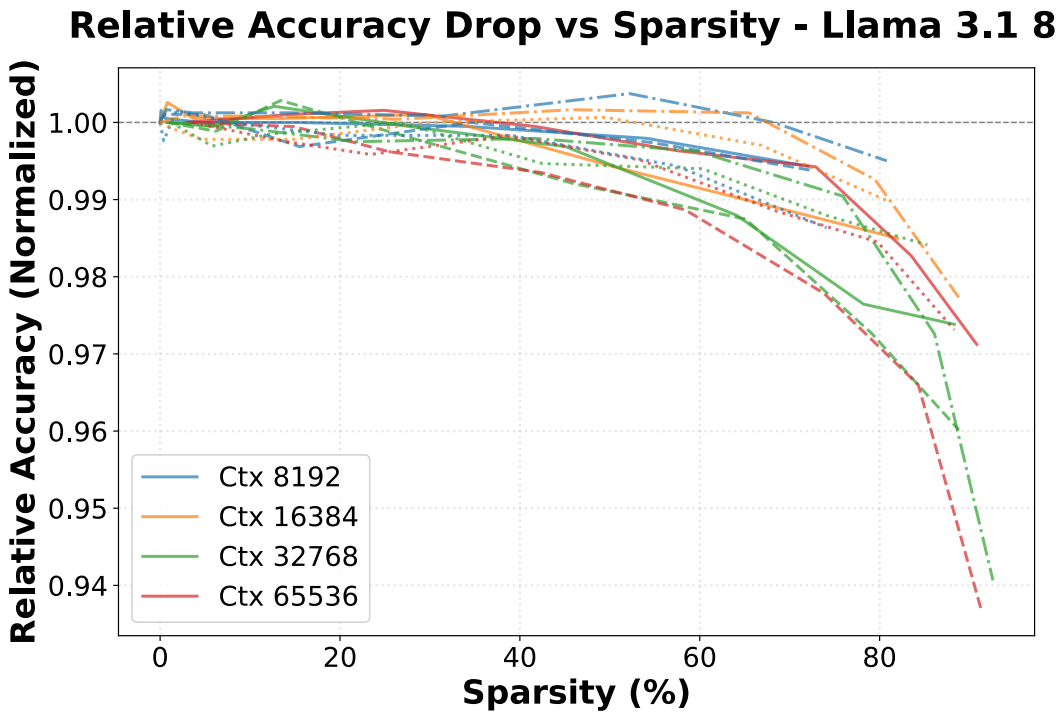
Require: Query blocks $\{Q_i\}_{i=1}^{T_r}$, Key blocks $\{K_j\}_{j=1}^{T_c}$, Value blocks $\{V_j\}_{j=1}^{T_c}$, threshold λ

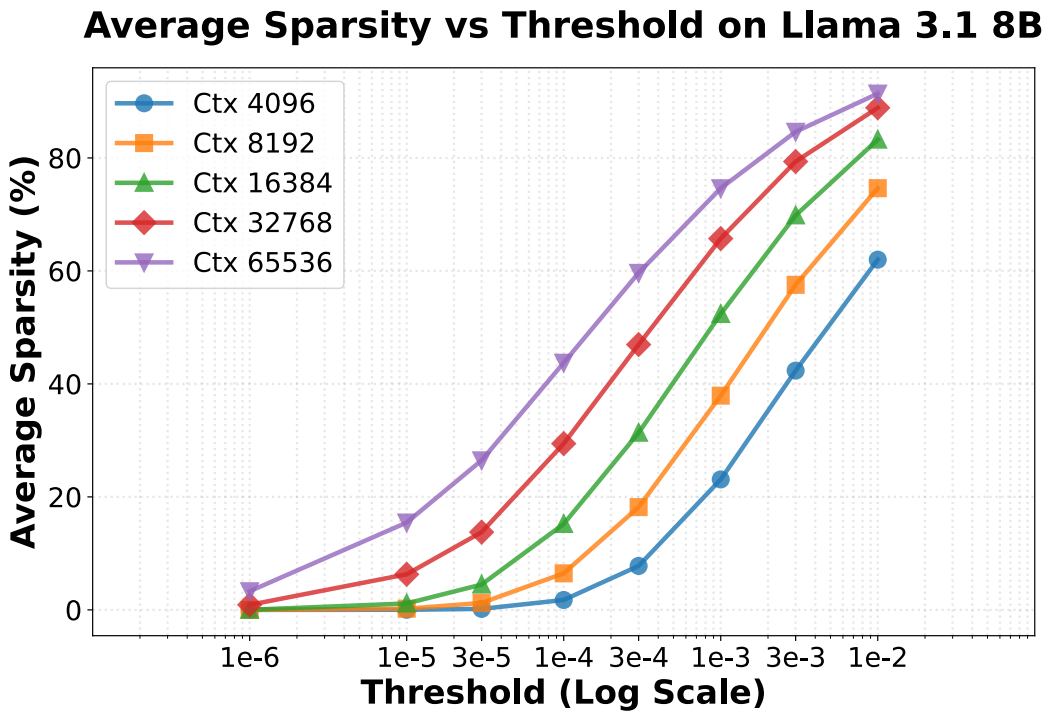
Ensure: Output blocks $\{O_i\}_{i=1}^{T_r}$

for $i = 1$ to T_r **do**
  Initialize $m_i^{(0)} = -\infty$, $O_i^{(0)} = 0$, $l_i^{(0)} = 0$
  for $j = 1$ to T_c **do**
    Compute $S_{ij} = Q_i K_j^\top$ ▷ Attention scores
    $\tilde{m}_i^{(j)} = \text{rowmax}(S_{ij})$ ▷ Local maximum
    $m_i^{(j)} = \max(m_i^{(j-1)}, \tilde{m}_i^{(j)})$ ▷ Running maximum
    if $\tilde{m}_i^{(j)} - m_i^{(j)} < \ln(\boldsymbol{\lambda})$ **then**
      continue ▷ ***Skip this block***
    end if
    $\tilde{P}_{ij} = \exp(S_{ij} - m_i^{(j)})$ ▷ Compute attn. weights
    $l_i^{(j)} = e^{m_i^{(j-1)} - m_i^{(j)}} l_i^{(j-1)} + \text{rowsum}(\tilde{P}_{ij})$
    $O_i^{(j)} = e^{m_i^{(j-1)} - m_i^{(j)}} O_i^{(j-1)} + \tilde{P}_{ij} V_j$
  end for
  $O_i = O_i^{(T_c)} / l_i^{(T_c)}$ ▷ Final normalization
end for
return $\{O_i\}_{i=1}^{T_r}$

How To Determine Threshold?

The right threshold depends on the **model**, **context length** and desired **accuracy**

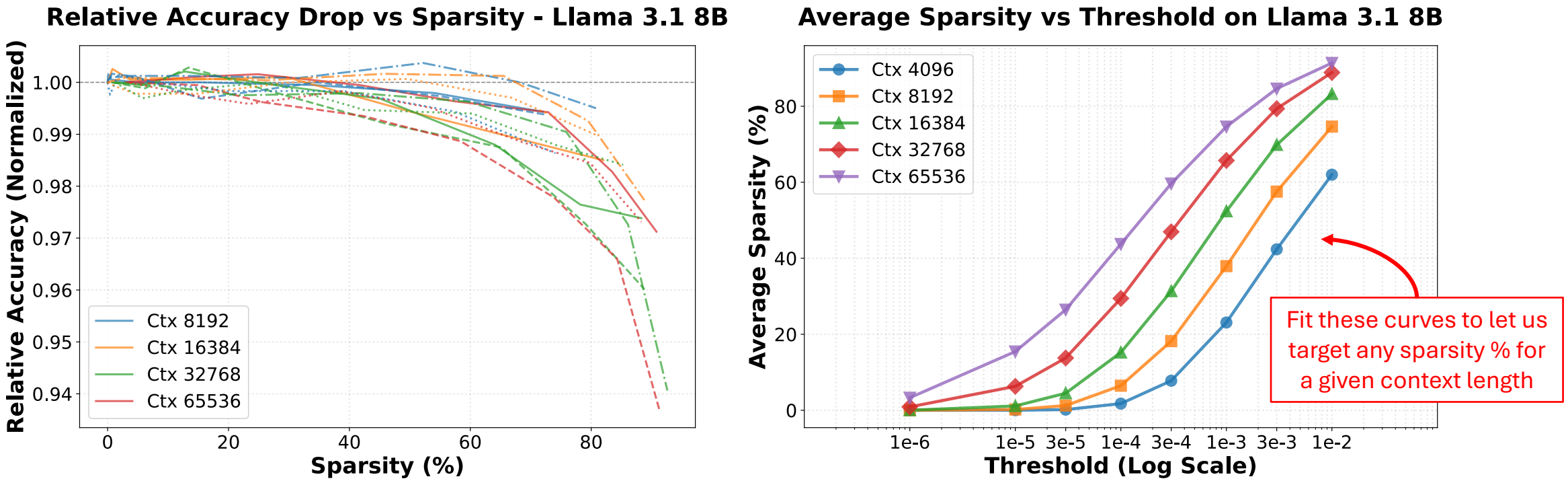




(Left) Relative accuracy drop across different datasets and context lengths shows consistent degradation patterns as observed sparsity increases. All curves are normalized to their initial accuracy. (Right) Relationship between threshold and observed sparsity levels across different sequence lengths, demonstrating the need for threshold calibration to maintain fixed sparsity across varying contexts.

How To Determine Threshold?

The right threshold depends on the **model**, **context length** and desired **accuracy**



(Left) Relative accuracy drop across different datasets and context lengths shows consistent degradation patterns as observed sparsity increases. All curves are normalized to their initial accuracy. (Right) Relationship between threshold and observed sparsity levels across different sequence lengths, demonstrating the need for threshold calibration to maintain fixed sparsity across varying contexts.

Threshold Calibration

One calibration pass → fit **λ · L = a · exp(b · S)** → dial in any target sparsity at runtime

Calibration Algorithm

1. **Sweep thresholds in one pass**

   Run the calibration set once, trying many candidate thresholds together.

2. **Record sparsity vs threshold**

   For each (threshold, sequence length), log the observed sparsity.

3. **Fit the curve**

   Fit λ · L = a · exp(b · S) — output is two scalars (a, b) per model.

4. **Deploy at any target sparsity**

   Pick S → threshold is set automatically; rescales with context length L.

Threshold Calibration

One calibration pass → fit **λ · L = a · exp(b · S)** → dial in any target sparsity at runtime

Calibration Algorithm

1. **Sweep thresholds in one pass**

   Run the calibration set once, trying many candidate thresholds together.

2. **Record sparsity vs threshold**

   For each (threshold, sequence length), log the observed sparsity.

3. **Fit the curve**

   Fit λ · L = a · exp(b · S) — output is two scalars (a, b) per model.

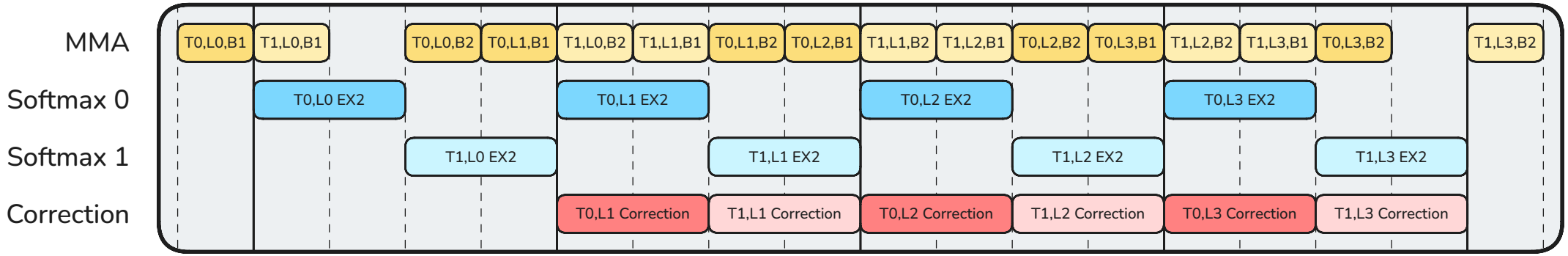
4. **Deploy at any target sparsity**

   Pick S → threshold is set automatically; rescales with context length L.

One scalar per model.
λ rescales with L — no per-context recalibration, predictable speedup.

Background – FlashAttention

- Avoid storing and loading N^2 attention matrix in HBM by fusing surrounding operators, softmax$(QK^\top)V$.
- Online softmax to avoid keeping entire attention row for rowmax.
- MMA and Softmax are on critical path for Blackwell: Specialize into their own warps (FlashAttention-4 , Zadouri et al., 2026).

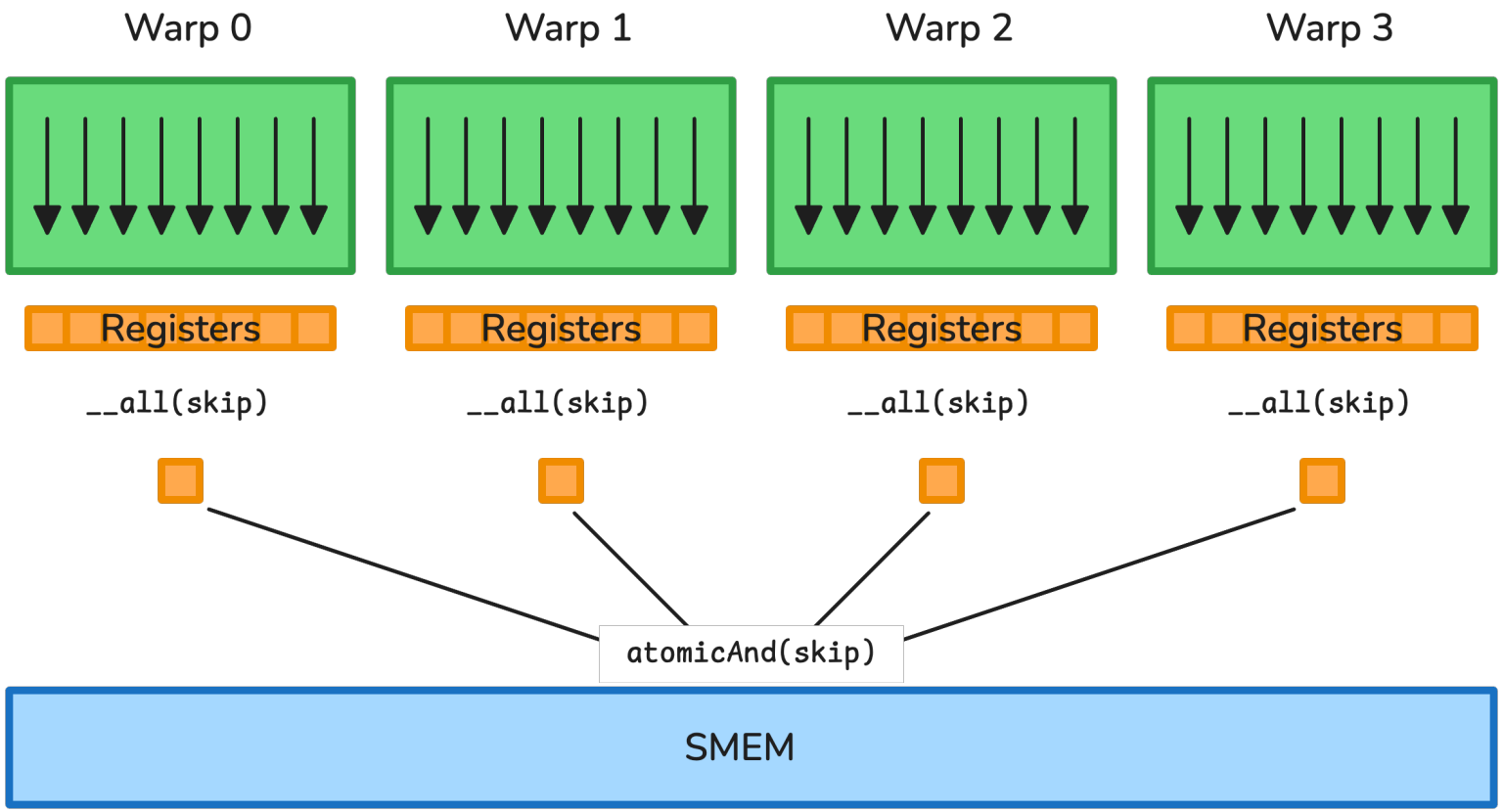


Skip Predicate Reduction

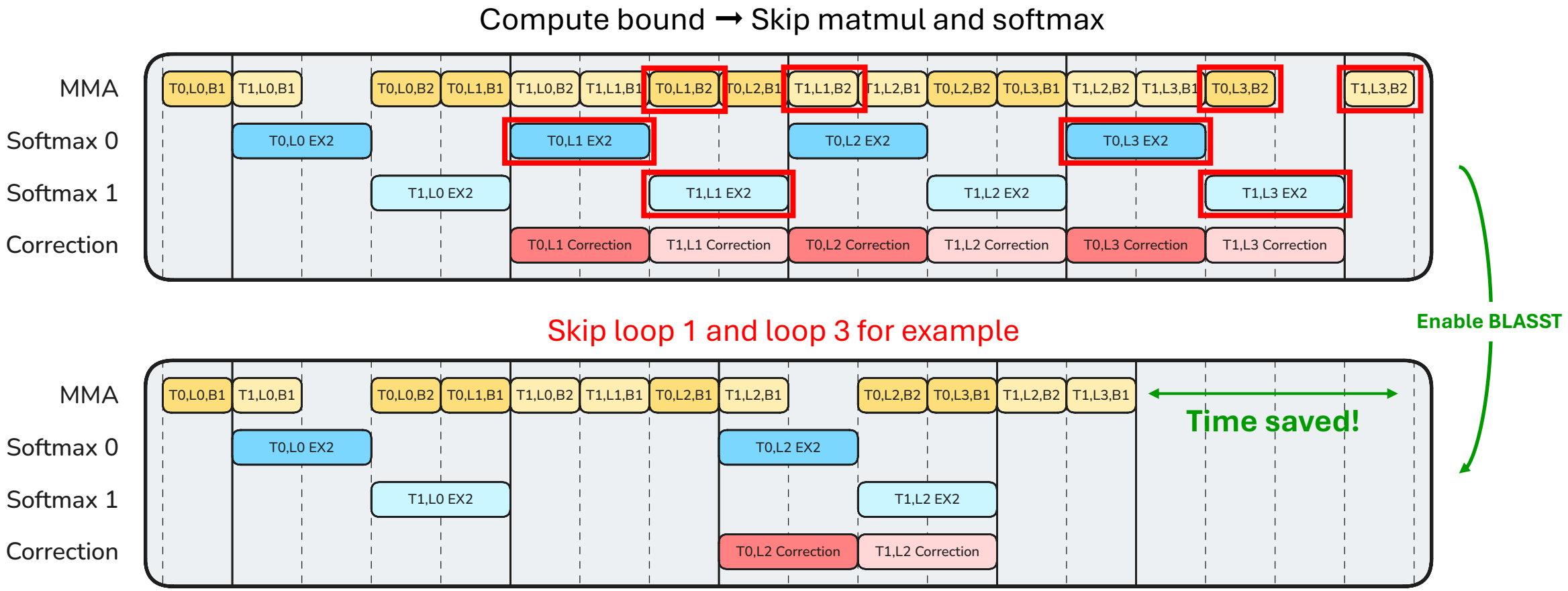
Compute block max off the critical path (in correction warpgroup)

Each thread computes skip predicate for its own block row.

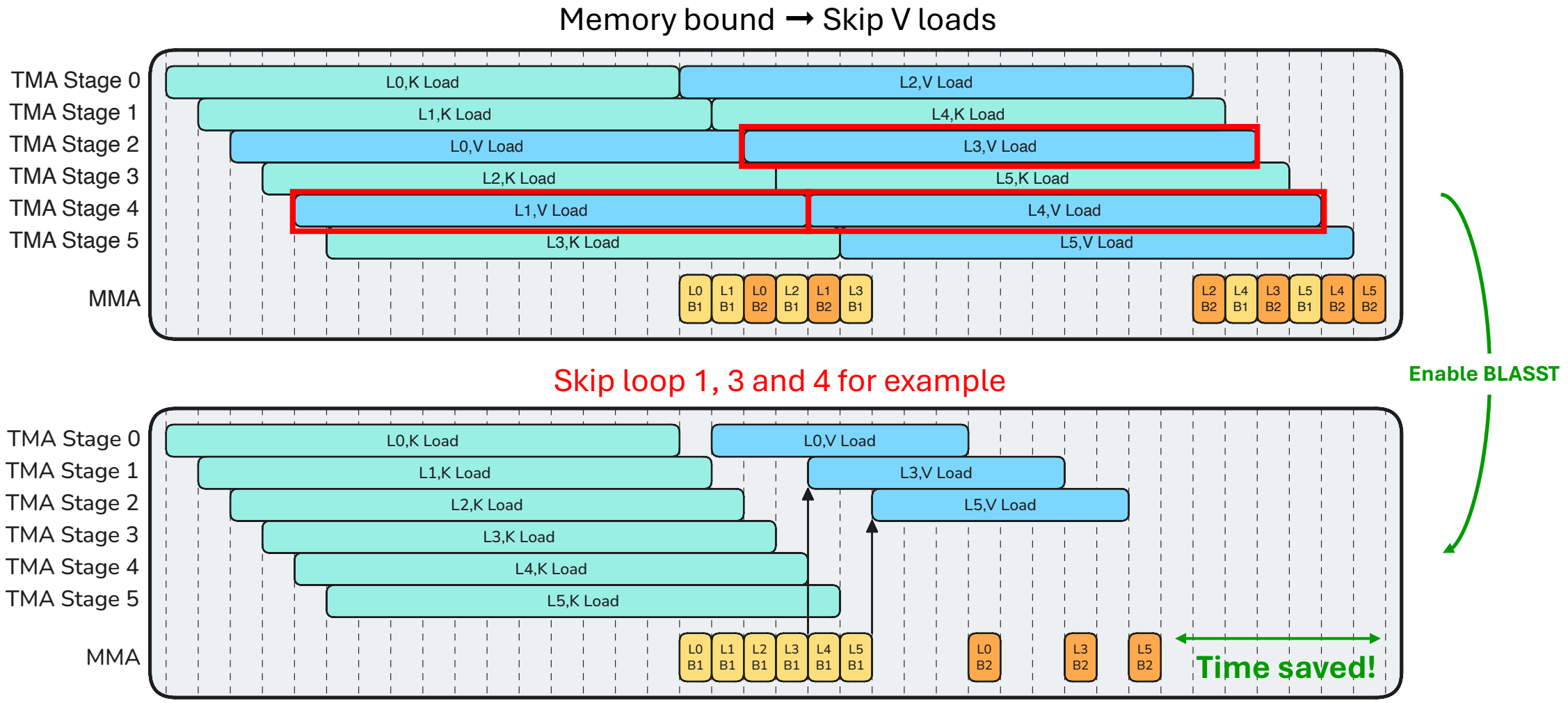
Warpgroup reduction using `VOTE` and `ATOMIC` instructions.



Prefill Pipeline Comparison



Decode Pipeline Comparison



*For compute bound decoding, e.g. MLA, we can also skip matmul and softmax

BLASST Preserves Accuracy

Model	Target Sparsity	Prefill Phase		Decode Phase			Prefill + Decode Phase	
		RULER-32K	LongBench	MATH500	AIME2024	GPQA	RULER-32K	LongBench
Llama-3.1-8B	Dense	92.33	31.40	73.40	46.66	46.71	92.33	31.40
	50%	91.81	31.80	73.71	46.15	46.31	91.79	32.40
	75%	91.67	31.80	73.89	46.01	45.95	91.67	31.80
Qwen3-8B	Dense	91.90	33.60	95.87	75.00	61.21	91.90	33.60
	50%	92.08	35.10	96.23	76.50	61.56	92.07	33.30
	75%	92.11	34.40	96.07	75.33	61.51	91.74	33.10

Performance of BLASST at different sparsity levels across all models and benchmarks. We evaluate on Llama-3.1-8B and Qwen3-8B across three deployment scenarios: prefill-only optimization (long-context tasks: RULER, LongBench); decode-only optimization (reasoning tasks: MATH500, AIME 2024, GPQA); and combined prefill+decode optimization. Results show minimal accuracy degradation even at ~75% sparsity, with occasional improvements over the dense baseline.

Prefill-Only Accuracy

Method	RULER						LongBench					
	4K	8K	16K	32K	64K	Average	Easy	Hard	Short	Medium	Long	Overall
Dense Attention	96.16	95.07	94.80	92.33	87.69	93.21	29.7	32.5	38.3	28.8	25.0	31.4
FlexPrefill	95.99	93.67	92.73	88.14	81.14	87.72	28.8	23.8	24.4	26.5	26.2	25.7
MInference	**96.54**	94.06	91.37	85.79	83.03	84.15	28.6	**32.8**	36.7	30.2	24.1	31.2
XAttention	96.37	94.47	94.48	**91.91**	85.01	92.44	29.2	31.5	**38.3**	26.0	**26.9**	30.6
BLASST (∼50%)	96.17	**94.70**	**94.61**	91.81	**87.06**	**92.87**	**30.7**	32.5	**38.3**	**29.8**	25.0	**31.8**

Prefill phase comparison on Llama-3.1-8B-Instruct across RULER and LongBench. Best (non-dense) score in each column is denoted in bold. Targeting 50% sparsity, BLASST achieves the best accuracy among all sparse attention methods, closely matching dense attention in addition to being the easiest to use.

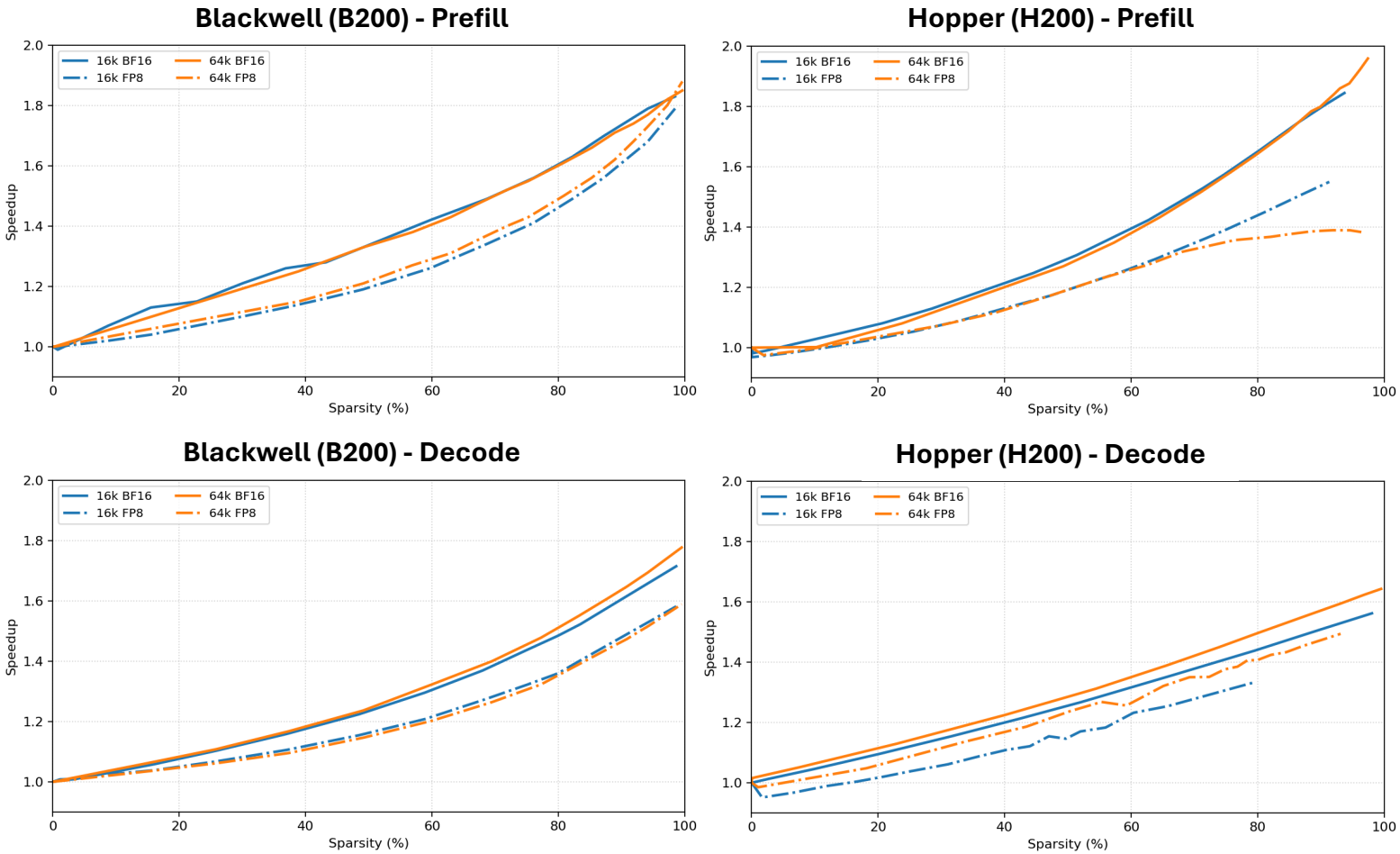
Decode-Only Accuracy

Method	RULER-32K	LongBench	MATH500	AIME 2024	LiveCodeBench	GPQA	Average
Dense Attention	91.90	33.60	95.87	75.00	53.83	61.21	68.57
Quest	56.23	30.30	94.18	71.50	52.17	60.12	60.75
RocketKV	87.89	30.60	95.88	73.54	53.10	60.50	66.91
BLASST ∼50%	**91.55**	**33.90**	**96.23**	**76.50**	**54.15**	**61.51**	**68.97**

Decode phase comparison on Qwen3-8B across diverse reasoning and generation tasks. Best (non-dense) score in each column is denoted in bold. Targeting 50% sparsity, BLASST matches or exceeds dense baseline on all benchmarks, including mathematical reasoning (MATH500, AIME 2024), graduate-level science (GPQA), and code generation (LiveCodeBench), while maintaining long-context performance (RULER, LongBench).

Kernel Performance

- Compared against SoL FlashAttention kernels in TensorRT LLM.
- We see **~1.5x BF16 speedup at 75% sparsity**, where accuracy remains good.
- **Minimal to no overhead at 0% sparsity** – skip computation costs are hidden.



Enable With a Few Lines of Code

Enable With a Few Lines of Code

1. ModelOpt finds threshold for your model

```
from modelopt.torch.sparsity.attention_sparsity.config \
import SKIP_SOFTMAX_CALIB

# Load your model
model = AutoModelForCausalLM.from_pretrained(
      "Qwen/Qwen3-30B-A3B-Instruct-2507",
      attention_implementation="eager",
      torch_dtype=torch.bfloat16
)

# Apply sparse attention
model = mtsa.sparsify(model, config=SKIP_SOFTMAX_CALIB)
```

Enable With a Few Lines of Code

1. ModelOpt finds threshold for your model

```
from modelopt.torch.sparsity.attention_sparsity.config \
import SKIP_SOFTMAX_CALIB

# Load your model
model = AutoModelForCausalLM.from_pretrained(
      "Qwen/Qwen3-30B-A3B-Instruct-2507",
      attention_implementation="eager",
      torch_dtype=torch.bfloat16
)

# Apply sparse attention
model = mtsa.sparsify(model, config=SKIP_SOFTMAX_CALIB)
```

2a. Deploy on TensorRT LLM server

```
from tensorrt_llm import LLM
from tensorrt_llm.llmapi import SkipSoftmaxAttentionConfig

sparse_attn_config = SkipSoftmaxAttentionConfig(
    threshold_scale_factor=1000.0
)

# Or separate threshold for prefill/decode
sparse_attn_config = SkipSoftmaxAttentionConfig(
    threshold_scale_factor={
        "prefill": 1000.0, "decode": 500.0
    }
)

llm = LLM(
   model="Qwen/Qwen3-30B-A3B-Instruct-2507",
   sparse_attention_config=sparse_attn_config,
   # Other arguments...
)
```

Enable With a Few Lines of Code

1. ModelOpt finds threshold for your model

```
from modelopt.torch.sparsity.attention_sparsity.config \
import SKIP_SOFTMAX_CALIB

# Load your model
model = AutoModelForCausalLM.from_pretrained(
      "Qwen/Qwen3-30B-A3B-Instruct-2507",
      attention_implementation="eager",
      torch_dtype=torch.bfloat16
)

# Apply sparse attention
model = mtsa.sparsify(model, config=SKIP_SOFTMAX_CALIB)
```

2b. Deploy on SGLang server

```
SGLANG_SKIP_SOFTMAX_PREFILL_THRESHOLD_SCALE_FACTOR=1000.0 \
SGLANG_SKIP_SOFTMAX_DECODE_THRESHOLD_SCALE_FACTOR=500.0 \
python3 -m sglang.launch_server \
    --model-path "Qwen/Qwen3-30B-A3B-Instruct-2507" \
    # Other arguments...
```

2a. Deploy on TensorRT LLM server

```
from tensorrt_llm import LLM
from tensorrt_llm.llmapi import SkipSoftmaxAttentionConfig

sparse_attn_config = SkipSoftmaxAttentionConfig(
    threshold_scale_factor=1000.0
)

# Or separate threshold for prefill/decode
sparse_attn_config = SkipSoftmaxAttentionConfig(
    threshold_scale_factor={
        "prefill": 1000.0, "decode": 500.0
    }
)

llm = LLM(
   model="Qwen/Qwen3-30B-A3B-Instruct-2507",
   sparse_attention_config=sparse_attn_config,
   # Other arguments...
)
```

Framework Support

Kernels available in TensorRT LLM and FlashInfer.

NVIDIA TensorRT LLM

Link

FlashInfer

Link

Threshold calibration tool and recipes available in Model Optimizer.

NVIDIA ModelOpt

Link

Framework Support

SGL

Link

Coming soon

vLLM

Link

References

Yang, S., Guo, J., Tang, H., Hu, Q., Xiao, G., Tang, J., Lin, Y., Liu, Z., Lu, Y. and Han, S., 2025. **Lserve: Efficient long-sequence llm serving with unified sparse attention**. In *Proceedings of Machine Learning and Systems*, *7*.

Zaheer, M., Guruganesh, G., Dubey, K. A., Ainslie, J., Alberti, C., Ontanon, S., Pham, P., Ravula, A., Wang, Q., Yang, L., et al. **Big Bird: Transformers for longer sequences**. In *Advances in Neural Information Processing Systems*, volume 33, pp. 17283–17297, December 2020. doi: 10.48550/arXiv.2007.14062.

Xiao, G., Tian, Y., Chen, B., Han, S., and Lewis, M. **Efficient streaming language models with attention sinks**. In *The Twelfth International Conference on Learning Representations*, May 2024b. doi: 10.48550/arXiv.2309.17453

Jiang, H., Li, Y., Zhang, C., Wu, Q., Luo, X., Ahn, S., Han, Z., Abdi, A. H., Li, D., Lin, C.-Y., Yang, Y., and Qiu, L. **MInference 1.0: Accelerating prefilling for long-context LLMs via dynamic sparse attention**. In *Advances in Neural Information Processing Systems*, volume 37, pp. 52481–52515, December 2024. doi: 10.48550/arXiv.2407.02490. URL https://openreview.net/forum?id=fPBACAbqSN.

Zadouri, T., Hoehnerbach, M., Shah, J., Liu, T., Thakkar, V., and Dao, T. **FlashAttention-4: Algorithm and kernel pipelining co-design for asymmetric hardware scaling**. *CoRR*, abs/2603.05451, March 2026. doi: 10.48550/arXiv.2603.05451.

BLASST Is Robust and Easy to Use

Method	Accelerates Prefill	Accelerates Decode	No Training	No Pre-Computation
H2O	❌	✅	✅	✅
SnapKV	❌	✅	✅	✅
RocketKV	❌	✅	✅	❌
Quest	❌	✅	✅	❌
DuoAttention	✅	✅	❌	✅
DSA	✅	✅	❌	✅
MInference	✅	❌	✅	❌
SpargeAttention	✅	❌	✅	❌
XAttention	✅	❌	✅	❌
BLASST	✅	✅	✅	✅