NEURIPS 2026 · PARIS

UniPrefill: Universal Long-Context Prefill Acceleration

via Block-wise Dynamic Sparsification

Qihang Fan Huaibo Huang Zhiying Wu Bingning Wang Ran He

CASIA · UCAS · WeChat, Tencent

ONE LINE

Up to 7.06× faster time-to-first-token at 1M tokens, accuracy within noise, and no model weight changes.

THE PROBLEM

Sparse attention only fixes one layer

- Sparse-attention methods (MInference, FlexPrefill, XAttention) put all of their speedup inside the full-attention layer.
- Modern models are hybrids: Qwen3-Next interleaves linear and full attention 3:1, Gemma-3 mixes sliding window and full. At most one layer in four can be accelerated.
- The FFN and GEMM budget — dominant once attention is sub-quadratic — is never touched.
- They assume a static batch, so they do not fit vLLM continuous batching.

A 3:1 linear / full hybrid

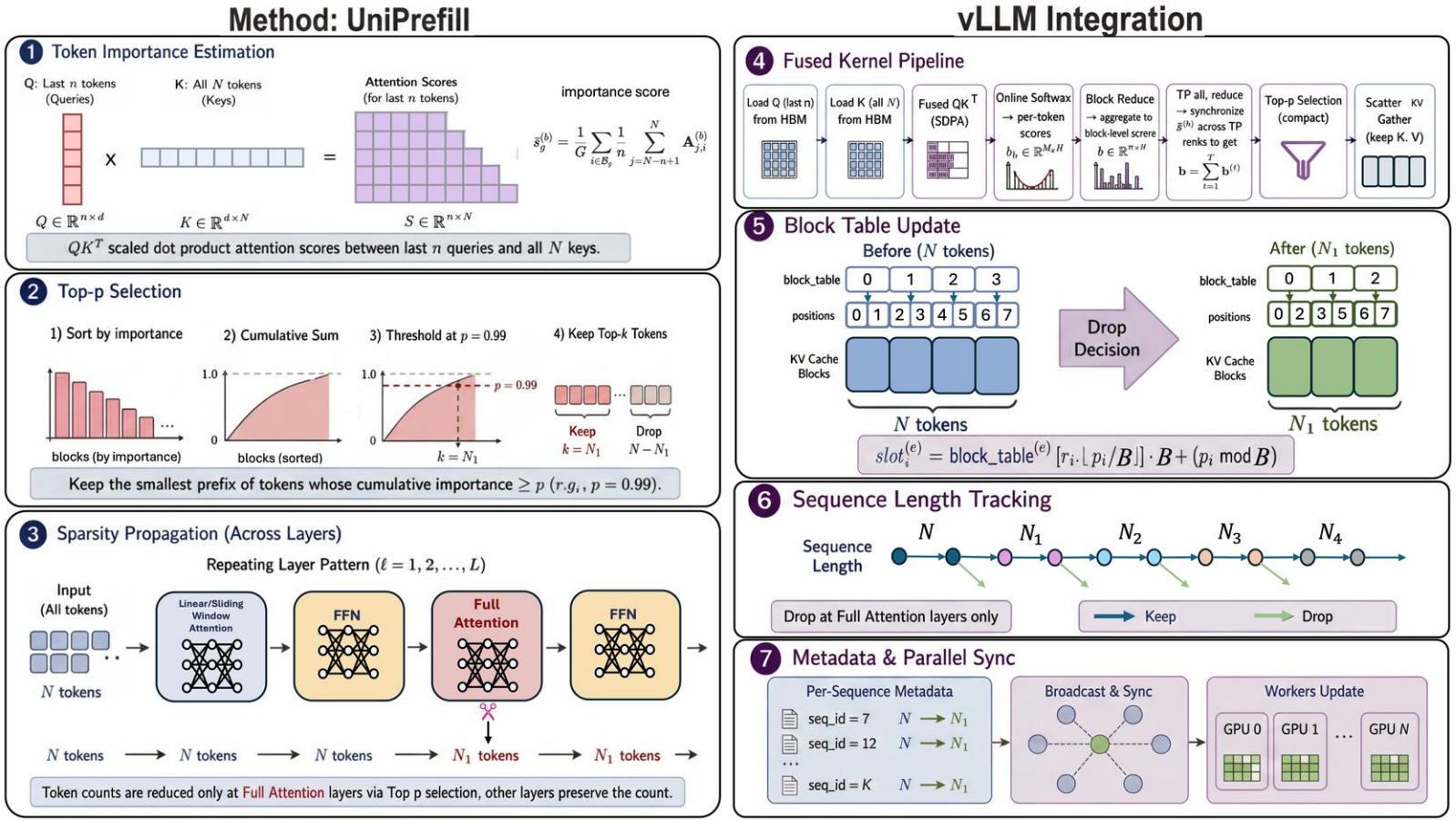
only this one can be accelerated

What each approach buys you

	sparse	UniPrefill
layers sped up	1 in 4	**all below a drop**
FLOPs cut	attention	**attention + GEMM**
batching	static	**native in vLLM**

So the question is not how to sparsify attention — it is how to make the whole layer stack cheaper.

Drop tokens, and every layer below pays less



Irreversible by design

A dropped token never comes back: full attention, linear attention, sliding window and FFN all run on a strictly smaller set, so one decision at one layer saves FLOPs at every layer below it.

retention at 128K

1.00 → 0.42 → 0.05 → 0.4%

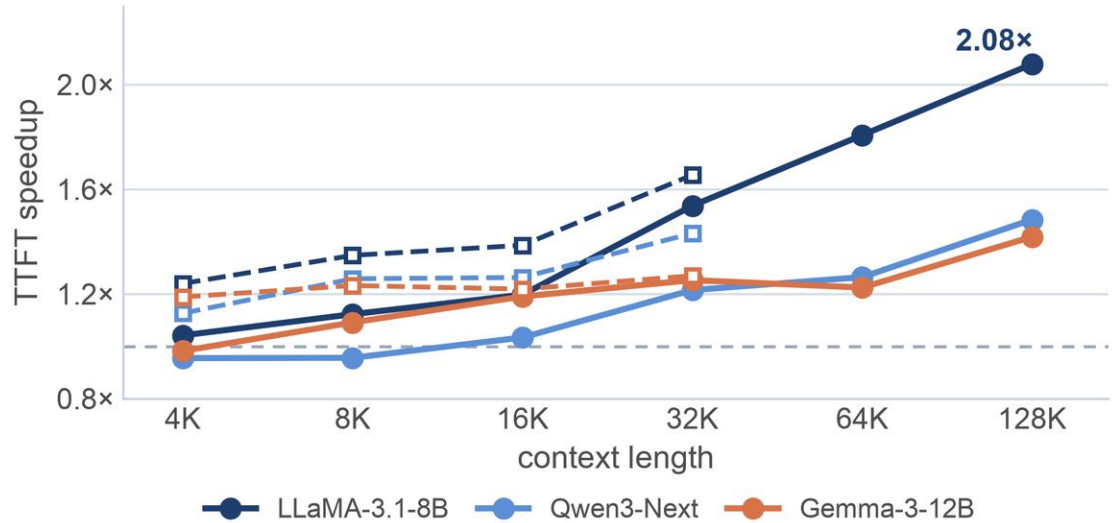
Left: (1) score blocks, (2) keep until mass ≥ p, (3) propagate the drop.
Right: (4)–(7) fused kernels and vLLM scheduling.

RESULTS · RULER

Accuracy holds, TTFT drops

Model	dense avg	ours avg	TTFT
LLaMA-3.1-8B full attention	90.36	90.45	**2.26×**
Qwen3-Next-80B linear / full	94.76	93.94	**1.68×**
Gemma-3-12B window / full	79.99	78.87	**1.49×**

MInference keeps accuracy too, but reaches only 1.34× at 128K — it can only touch one layer in four.



TTFT speedup vs. context length; solid = batch 1, dashed = batch 64.

Beyond RULER: 17-task LongBench 48.08 → 47.80, LongBench Pro 51.54 → 51.18.

Operator overhead at 128K: 1.3 – 2.2% of prefill. vLLM v0.16, 8 × H20, TP = 8.

RESULTS · BEYOND 128K

The longer the context, the bigger the win

7.06×
TTFT at 1M
Qwen3-Next

7.61×
TTFT at 512K
Seed-OSS

0.1%
tokens kept
at 1M

- Longer inputs are more redundant, so top-p keeps fewer tokens: 0.5% at 128K, 0.1% at 1M.
- Each drop is permanent and a model has five of them, so the savings compound with depth.

Model	Length	dense	ours	kept	TTFT
Qwen3-Next	128K	92.09	91.41	0.5%	**1.48×**
	256K	91.21	90.82	0.3%	**2.13×**
	512K	85.31	85.44	0.1%	**3.78×**
	1M	**79.02**	**79.27**	**0.1%**	**7.06×**
Seed-OSS	128K	94.42	94.08	0.6%	**2.41×**
	256K	86.08	85.31	0.3%	**4.09×**
	512K	78.33	79.64	0.2%	**7.61×**

Sparse attention only ever discounts the N^2 term, at one layer. UniPrefill also discounts the $M \cdot N \cdot d^2$ GEMM term, at every layer below a drop — so the saving grows with depth and with length.

TAKEAWAY

Three things to remember

1

Drop tokens, not attention blocks

token-level sparsity, one decision per block

2

Sparsity propagates

all later sub-layers, including FFN and GEMM, run on the kept set

3

Ships inside vLLM

continuous batching, PD co-processing, TP; 1.3–2.2% overhead

Code

github.com/qhfan/UniPrefill

Thank you