



Twilight: Adaptive Attention Sparsity with Hierarchical Top- p Pruning

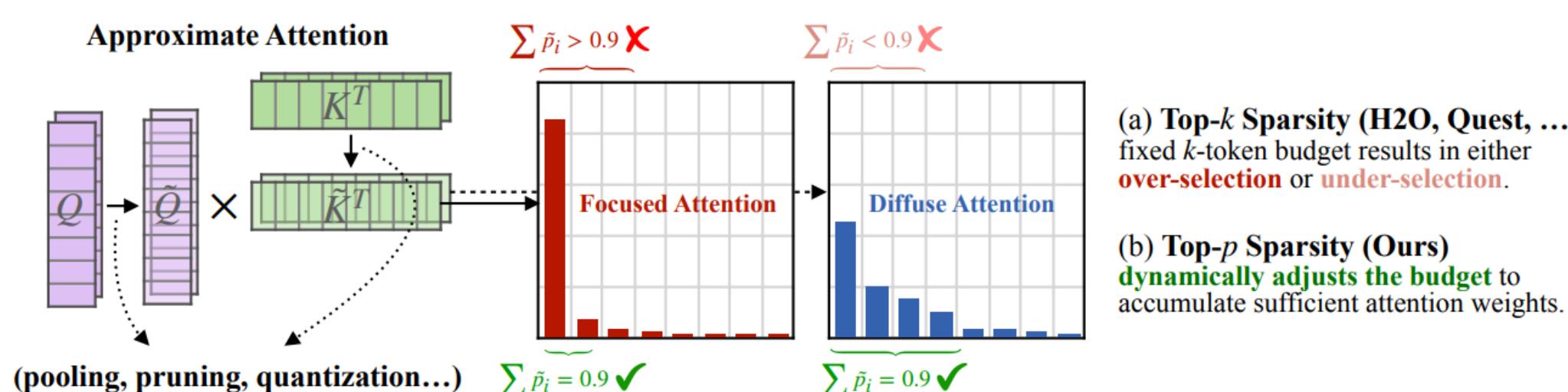
Chaofan Lin, Jiaming Tang, Shuo Yang, Hanshuo Wang, Tian Tang, Boyu Tian, Ion Stoica, Song Han, Mingyu Gao



Tsinghua University, MIT, UC Berkeley

Introduction

- Top- k sparse attention accelerates long context LLM inference by saving memory loading costs.
- However, fixed budget can lead to over-selection or under-selection.
- We propose **Twilight**, a composable optimizer to accelerate any top- k sparse attention through hierarchical top- p pruning, making them efficient and budget-adaptive.



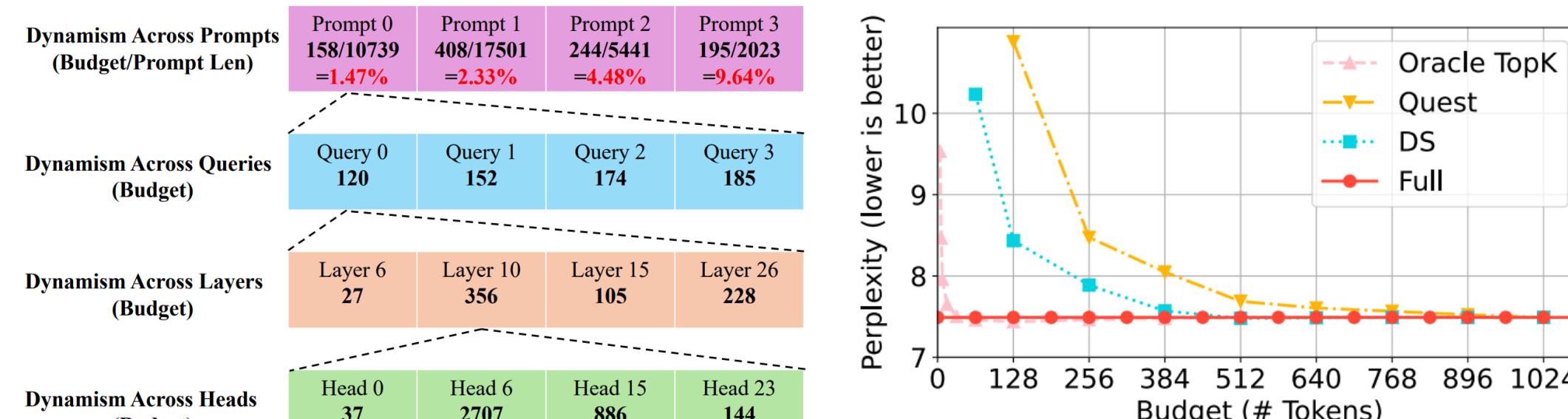
Top- k Sparse Attention for Long Context Large Language Model Inference

- For long context LLM inference, attention dominates the latency. Top- k sparse attention is proposed to save memory loading cost.

Definition 3.2 (Oracle Top- k Sparse Attention). Given the budget B ,

$$\mathcal{I} = \arg \max_{\mathcal{I}} \sum_{i \in \mathcal{I}} \mathbf{W}[i] \quad \text{s.t. } |\mathcal{I}| = B$$

- However, the best budget (a.k.a, k) choices are dynamic.
- The main challenge of top- k sparse attention is to find a universally applicable budget to all scenarios.



The best budget choices vary dynamically across different levels.

Bringing Top- p Sampling to Sparse Attention

- The core reason for budget dynamism is the dynamic nature of the **attention weight distributions** at runtime.
- Inspired by nucleus sampling, we propose **top- p sparse attention**.

Definition 3.3 (Oracle Top- p Sparse Attention). Given the threshold p ,

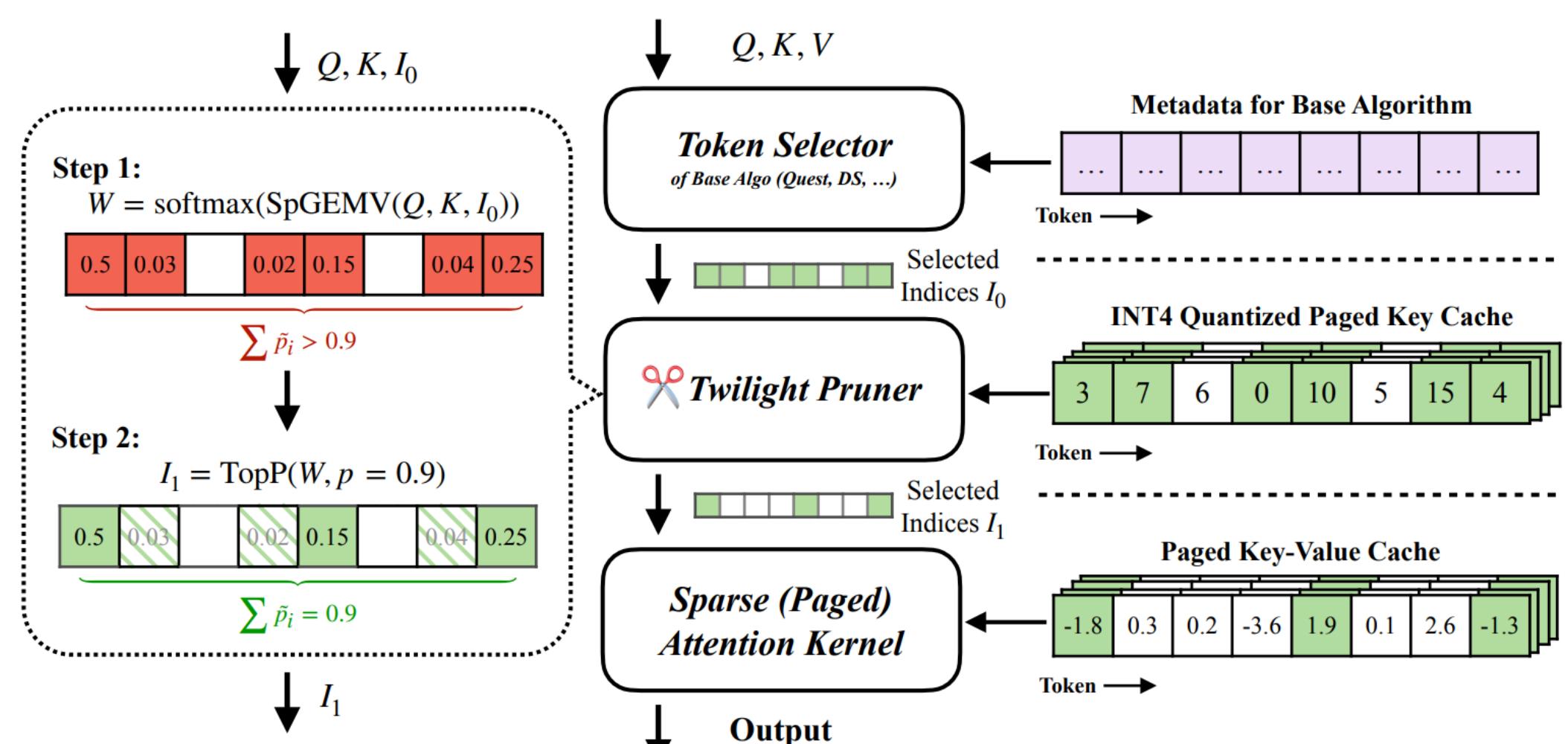
$$\mathcal{I} = \arg \min_{\mathcal{I}} |\mathcal{I}| \quad \text{s.t. } \sum_{i \in \mathcal{I}} \mathbf{W}[i] \geq p$$

- Top- p efficiently uses minimal budget to achieve the error requirement, thus achieving adaptive sparsity.



Twilight Design

- Twilight adopts a hierarchical **Select-then-Prune** architecture to accelerate existing sparse attention methods.



- Three **key kernel optimizations** to accelerate Twilight Pruner:
 - Efficient **SpGEMV** with 4-bit Quantization of Key Cache.
 - Efficient Sorting-Free **Top- p** via Binary Search.
 - Load Balancing** Attention Kernel with Awareness of Head Dynamism and GQA adaption.

Accuracy Evaluation

- Twilight achieves nearly no **accuracy loss** on three medium-context benchmarks and two long-context benchmarks (LongBench, RULER).

Table 2: Average scores on 12 different tasks from Longbench. We report relative error changes (**improvement** or **degradation**) when integrating Twilight with each base algorithm. Detailed results are in Table 5 in Appendix C.

	Budget	LongChat-7B	LLM4-3.1-8B
Full	32k	36.78	52.01
Twilight	38.52 (+4.7%)	51.64 (0.7%)	
MagicPIG	K=8, L=75 K=10, L=150	-	51.70 51.32
Quest	256 1024 4096 8192	31.26 36.85 37.33 37.10	47.79 50.79 51.44
Twilight	38.04 (+2.5%)	51.57 (0.3%)	
DS	256 1024 4096 8192	35.32 35.96 36.31 36.62	45.74 49.43 50.98 51.14
Twilight	38.71 (+5.7%)	51.73 (+1.2%)	

Table 3: Average scores on RULER.

	Budget	16k	32k	64k	96k	Avg.
Full	100%	92.88	89.42	85.17	85.23	88.18
Twilight	93.13	89.10	84.64	83.10	87.49	
MagicPIG	K=8, L=75 K=10, L=150	92.22 91.38	89.37 88.20	84.07 83.34	82.58 82.02	87.06 86.23
Quest	4% 8%	79.35 87.31	79.88 80.82	78.64 75.28	73.22 81.62	77.75
Twilight	91.53	87.97	84.12	82.96	86.65	
DS	4% 8%	92.04 92.89	88.11 88.70	84.43 84.39	82.56 82.72	86.79 87.18
Twilight	93.54	89.24	85.91	82.81	87.88	

Table 4: Results on 3 medium-context benchmarks.

	GSM8K(flexible/strict)↑	COQA(em/f1)↑	PG-19 Perplexity↓
Full	0.2290/0.2282	0.5935/0.7511	7.503
Quest	0.0523/0.0508	0.5710/0.7425	14.15
DS	0.2191/0.2190	0.5855/0.7401	7.622
Twilight	0.2153/0.2115	0.6088/0.7642	7.600
(Twilight Avg. Budget)	90.82	91.86	102.58

Efficiency Evaluation

- Speedup on self-attention operator compared to FlashInfer and Quest without Twilight.

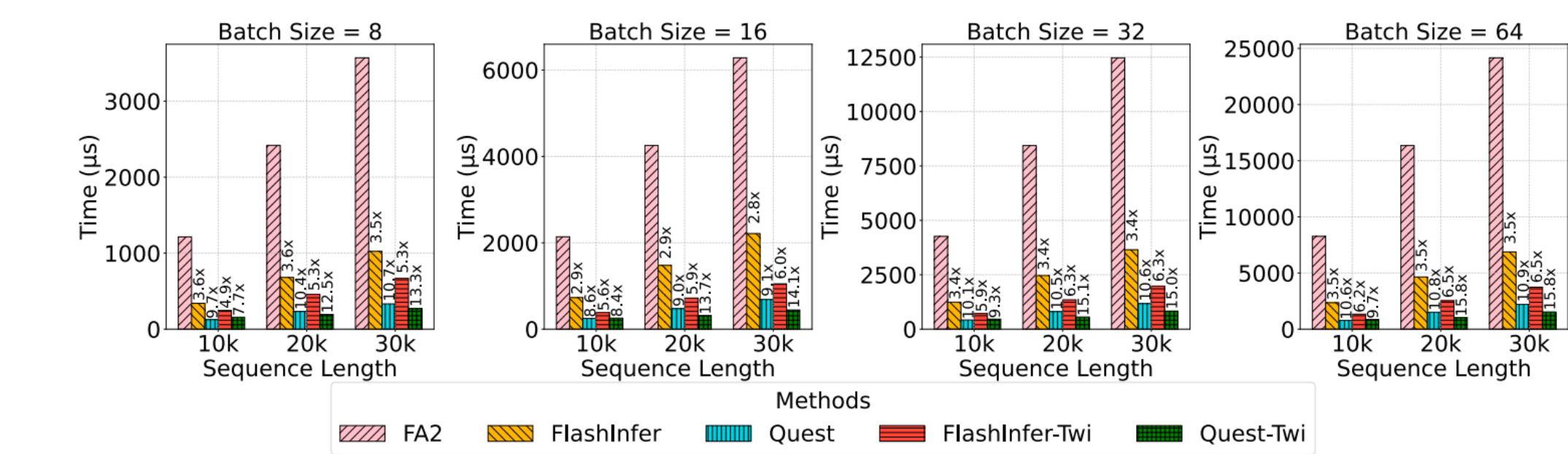


Figure 7: Latencies and speedups of self-attention at different sequence lengths and batch sizes.

- Latency breakdown of Quest with Twilight.

