DEFT: Decoding with Flash Tree-attention for Efficient Tree-structured LLM Inference

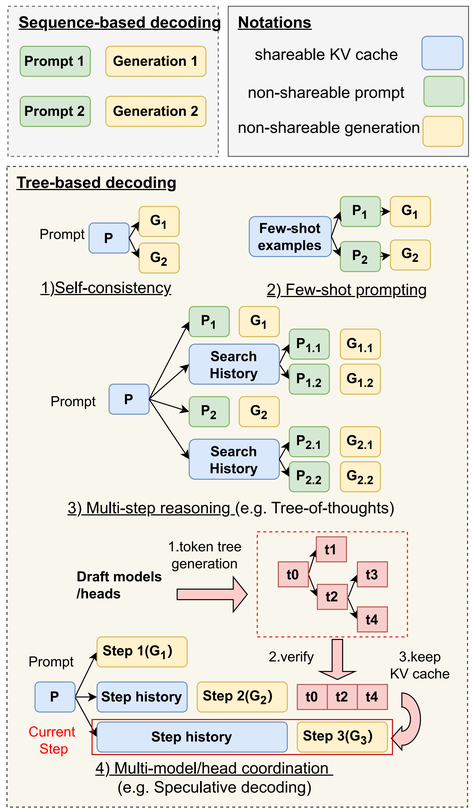
Jinwei Yao[1,4,*] **Kaiqi Chen**[2,*] Kexun Zhang[3] Jiaxuan You [4,†]
Binhang Yuan[5] Zeke Wang[2,†] Tao Lin[1,†]

[1]Westlake University [2]Zhejiang University [3]Carnegie Mellon University [4]University of Illinois Urbana-Champaign [5]Hong Kong University of Science and Technology

ICLR Spotlight

Motivation&Challenges

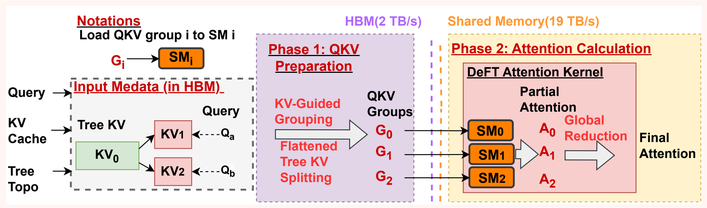
Large language models (LLMs) are increasingly employed for complex tasks that process multiple generation calls in a tree structure with shared prefixes of tokens, including few-shot prompting, multi-step reasoning, speculative decoding.



How make use of shared generation calls for acceleration is challenging:

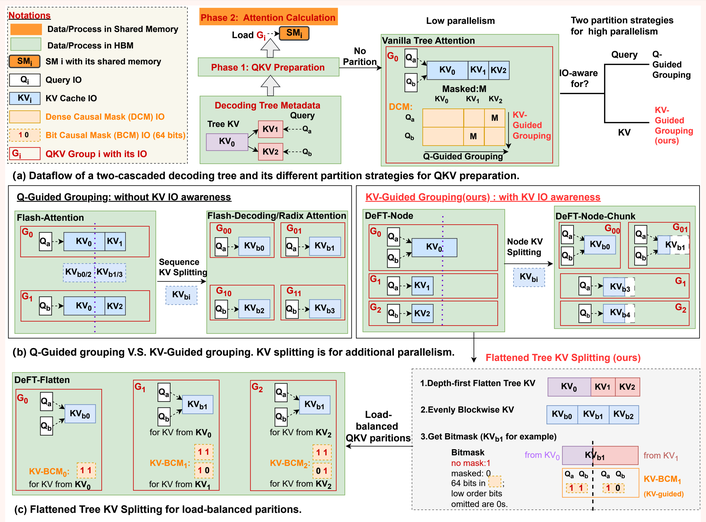
- C1: How to ensure prefix-awareness in memory access of KV cache?
- C2: How to split the tree-structured KV cache for load balancing and high GPU utilization?

DeFT Overview



- QKV Preparation Phase. QKV will be grouped logically to partitions with IO-awareness of shared prefixes' KV cache and load-balancing.
- Attention Calculation Phase. The attention calculation will be executed in QKV groups with a global reduction afterward.

DeFT Algorithm Design



1. KV-Guided Grouping. DeFT reduces redundant KV loads by grouping queries based on shared KV blocks, enabling prefix-aware attention.
2. Flattened Tree KV Splitting. DeFT improves load balance by chunking large KV blocks, ensuring better SM utilization across layers.

Workloads&Baselines

We compare DeFT with Flash-Decoding, Tree Attention-Medusa, and Radix Attention across three tasks including few-shot prompting, multi-step reasoning, and speculative decoding.

Table 1. Comparison of QKV partitioning strategies for baselines and DeFT. For IO redundancy, significant issues are highlighted in red, while negligible ones are in blue. "Q" refers to queries, and "KV" refers to the KV cache. "DCM" stands for Dense Causal Mask (a matrix), and "BCM" refers to Bit Causal Mask (a set of 64-bit integers). "PA" represents partial results during attention calculations, including $\mathbf{QK}^T$, Softmax, etc. More $\star$ symbols indicate better-balanced workloads for QKV partitions.

Attention Algorithm	Grouping Indicator	KV Split Granularity	IO Redundancy	Load-balancing Level
Flash-Attention	Q-guided	-	KV	$\star$
Flash-Decoding	Q-guided	by block	KV	$\star\star\star$
Radix Attention	Q-guided	by block	KV	$\star\star\star$
Tree Attention-S	Q-guided	by block	KV and BCM	$\star\star\star$
Tree Attention-M	entire tree	by GEMM in PyTorch	DCM and PA	$\star\star\star$
Vanilla Tree Attention	entire tree	no split	DCM and PA	$\star$
DeFT-Node	KV-guided	by tree node	Q	$\star$
DeFT-Node-Chunk	KV-guided	by tree node, then by block	Q	$\star\star$
DeFT-Flatten	KV-guided	by block	Q and BCM	$\star\star\star$

Table 2. Workloads generation. ToT-BFS stands for Tree-of-Thoughts using breadth-first search. APPS is a competitive programming problem dataset. Medusa is a speculative decoding framework. "GoT" stands for Graph-of-Thoughts, which contains iteration records using GPT-3.5 for complex reasoning tasks within ToT-BFS.

Task	Prompt Dataset	Decoding Tree Source	Decoding Tree Collection Method	Stopping Criteria
Few-shot prompting	APPS	-	Pad the prompt to 4000 tokens	400 iterations
Multi-step reasoning	4 tasks in GoT	ToT-BFS	Reconstruct from interaction records with GPT 3.5 in GoT	End of task(~ 3500 iterations)
Speculative decoding	APPS	Medusa	Record token tree shape and accepted token length per step	~1000 steps(max length=6000)

Efficiency Results

Table 3. Comparison of DeFT-Flatten and baselines in average decoding latency (in seconds) for tree-based decoding. Here, b represents the tree width, and t denotes the token tree size (i.e., the number of tree-structured queries). The fastest method is in bold, and the second fastest is underlined. *Radix Attention* is the best baseline in decoding latency. $\star$ denotes out-of-memory (OOM) errors for the A100 80GB GPU. *Speedup Upper-bound (no attention)* refers to the maximum speedup we could achieve for *Radix Attention* if we exclude the attention computation and only run other components including MLP.

Memory	Method	Few-shot Prompting			Multi-Step Reasoning				Speculative Decoding			
		b=20	b=30	b=50	*Sorting*	*Document*	*Keyword*	*Set*	t=32	t=64	t=128	t=256
Unpaged	Flash-Decoding	78.96	131.19	191.09	429.65	241.20	32.75	51.76	574.50	1128.45	$\star$	$\star$
	Tree Attention-Medusa	52.58	103.90	144.07	380.87	236.86	33.52	50.10	263.40	483.35	924.97	1881.51
Paged	*Radix Attention*	12.37	14.08	16.54	104.79	69.61	11.25	17.03	54.66	69.75	108.56	188.66
	DeFT-Flatten	**9.98**	**10.99**	**12.48**	**94.67**	**66.95**	**10.90**	**16.10**	**42.23**	**46.60**	**56.96**	**84.27**
	Attention Speedup over Radix Attention	1.73×	1.63×	1.70×	1.39×	1.15×	1.21×	1.34×	1.96×	2.41×	3.11×	3.59×
	Decoding Speedup over Radix Attention	1.24×	1.28×	1.33×	1.10×	1.03×	1.03×	1.05×	1.29×	1.50×	1.91×	2.23×
	Speedup Upper-bound(no attention)	1.71×	2.08×	2.51×	1.96×	1.82×	1.70×	1.76×	1.89×	2.89×	3.34×	4.36×

DeFT-Flatten can achieve up to $2.23\times$ end-to-end latency speedup thanks to $3.59\times$ attention speedup, with a reduction of $73-99\%$ KV cache IO.

Ablations

Table 4. [Different KV Splitting Strategies] Comparison of DeFT-Node, DeFT-Node-Chunk and DeFT-Flatten in average attention latency (second) with NVIDIA A100 (80GB) for Llama3-8B model(GQA). The fastest method is in bold, and the second fastest is underlined. Radix Attention is the best baseline in decoding latency.

Memory	Method	Few-shot Prompting			Multi-Step Reasoning				Speculative Decoding			
		b=20	b=30	b=50	*Sorting*	*Document*	*Keyword*	*Set*	t=32	t=64	t=128	t=256
Paged	Radix Attention	5.99	7.30	9.96	39.37	24.69	3.11	5.13	25.73	40.47	76.10	145.43
	DeFT-Node	10.59	10.62	10.85	42.96	33.29	6.16	9.58	34.59	34.41	34.96	41.78
	DeFT-Node-Chunk	8.52	9.69	13.45	49.63	36.37	4.77	7.40	14.54	20.28	32.57	57.26
	DeFT-Flatten	**3.47**	**4.07**	**5.87**	**28.41**	**21.45**	**2.57**	**3.83**	**13.15**	**16.79**	**24.46**	**40.56**

- DeFT-Flatten achieves the best performance across all tree structures.
- DeFT-Node-Chunk improves over DeFT-Node by splitting large nodes, enhancing load balance.
- However, DeFT-Node-Chunk suffers when many small nodes exist (e.g., t=256), due to extra QKV groups and GPU kernel launches.

The influence of prompt length for attention speedup.

As prompt length increases, DeFT-Flatten achieves increasingly higher speedups over Radix Attention. For example, when prompt length = 20K and Queries = 64, DeFT-Flatten is up to 4.3× attention speedup compared with Radix Attention.