BatchLLM: Optimizing Large Batched LLM Inference with Global Prefix Sharing and Throughput-oriented Token Batching

Zhen Zheng, Xin Ji, ***Taosong Fang***,
Fanghao Zhou, Chuanjie Liu, Gang Peng

Large-batched / Offline Inference Matters

- **Many industry tasks are batched or even offline**
  - Search engine: snippet generation, offline ranking, etc.
  - Ads: title rewrite
  - …
- **Throughput is the target**
  - The gap: the community focuses on TTFT/TPOT, but batched/offline jobs want high throughput

Common Pattern: Single Heavy Prefix

Several examples here:

Snippet generation (search engine)	• *Prompt*: generate a short snippet based on the given Document to answer the given Query. **Document: <⋯>**, Query: <⋯> • Common document between queries
Offline ranking (search engine and recommendation)	• *Prompt*: please evaluate the relevance of the document for the query, **Query: <⋯>**, Document: <⋯> • Common query between documents
Ads title rewrite	• *Prompt*: given the Query and the Landing Page info, please refine the old title 'xxx' to get a better ad title. **Landing Page: <⋯>**, Query: <⋯> • Common landing page between queries

The Gap - 1

The popular inference engines are online-oriented

- **Sub-optimal for prefix sharing**
  - Will evict tokens in KV cache with only local information, causing KV cache thrashing of can-be-shared prefix

Insight 1: Global Prefix Sharing

- **Ahead-of-time prefix identification**
  - **Build the prefix information globally**, thus maximize the KV cache reuse

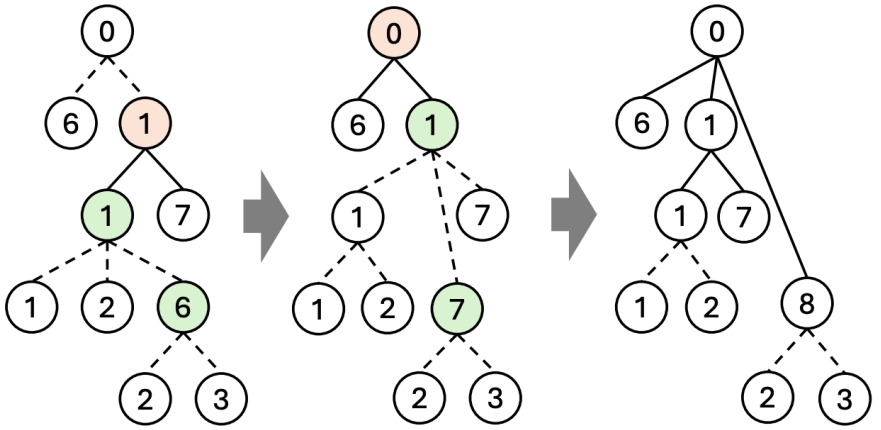
- **Prefix-aware Attention kernel optimization**
  - Using separate kernels to compute the partial Attention on the prefix and the distinct KV context, make the prefix part Attn more memory efficient by merging Q on the shared KV
  - **Challenge**: the dynamic multiple-level prefix makes the prefix-aware Attn kernel hard to optimize

Global 1st-level Prefix Enlargement

Insight: refactor the prefix-tree to enlarge the 1st level prefix, as "single heavy prefix" is common in industry.

Algorithm 1 Dynamic Programming on the prefix tree to maximize the first-level token reusing of node D.

```
1:  ▷ The 1st-level prefixes of a node are its children
2:  function MaximizeReuse(D)
3:      if D has no children then
4:          return
5:      for child ∈ D.children do
6:          MaximizeReuse(child)        ▷ Solve sub-problems
7:          for gchild ∈ child.children do
8:              gain ← (leaves(gchild) - 1) × tokens(gchild)
9:              penalty ← tokens(child)
10:             if gain > penalty then
11:                 Fork child and merge with gchild
```



Pseudo tokens 1: a b c d e a
Pseudo tokens 2: b a b
Pseudo tokens 3: b a c d
Pseudo tokens 4: b a a b c d e a b c
Pseudo tokens 5: b a a b c d e a d e a
Pseudo tokens 6: b b c d e a b c

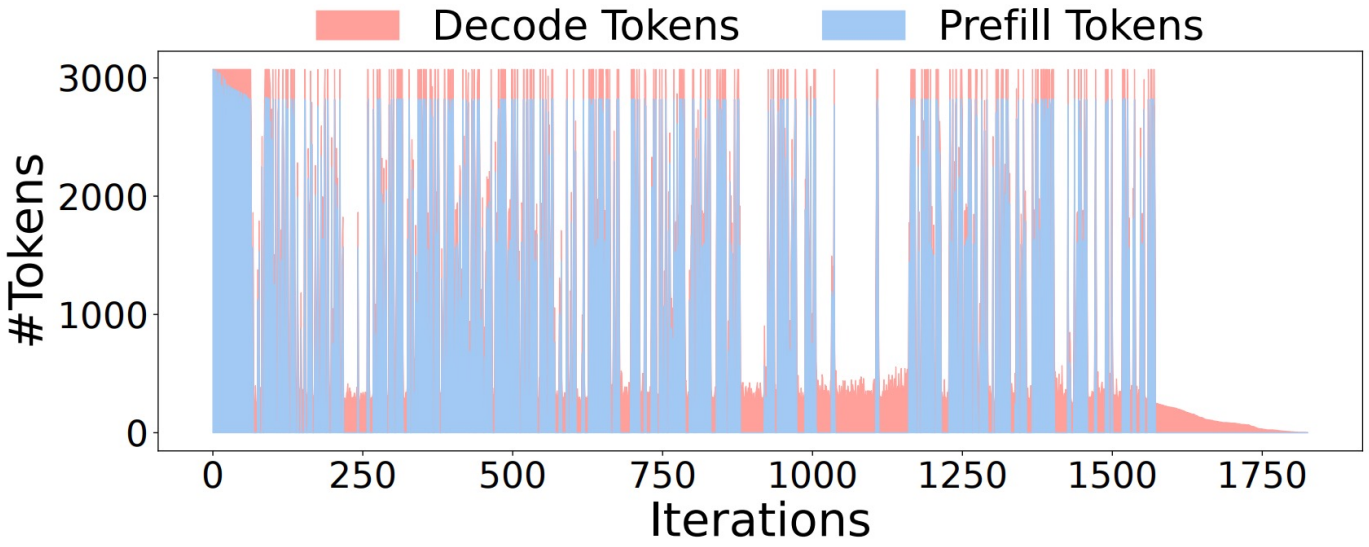
d. the pseudo tokens of 6 prompts

Figure 4: The preprocessing to maximize the first level prefix reusing. Each circle represents a node containing a number of tokens. The number in the circle is the token number of the node.

The Gap - 2

The popular inference engines are online-oriented

- **Sub-optimal for throughput**
  - The “eager” continuous batching leads to ”valleys” of batch size. Does not leverage the global length distribution within large batch.



Insight 2: Token Mixing with Global Info

- **Mix decoding with prefill is helpful for throughput**
  - KV memory consumption of per decoding (prefill) is high (low)
  - Mix prefill and decode will make the per-batch KV memory consumption smaller

- **The order is important for token mixing**

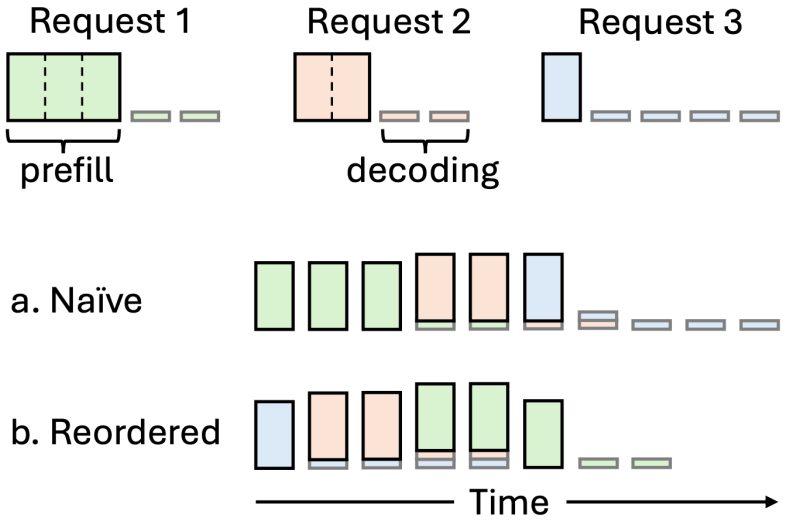


Figure 1: The effect of processing order of requests with chunked-prefill enabled. Given the three requests with different prefill/decoding length characteristics, the naive token batching in the coming order of the requests has worse token mixing of decoding and prefill chunks.

Token Batching with Global Information

To make the scheduling prefix-sharing aware

- **Prefix-sharing Group Based Scheduling**
  - Prefix-sharing group: requests sharing the same prefix
  - The unit of the task scheduling is prefix-sharing group
  - Three queues of the to-be-batched tokens: common prefix queue, distinct prompt queue, and decoding queue

Token Batching with Global Information

To mix the tokens of prefill and decode to the best

- **Request Groups Reordering**
  - Groups with longer prefills will be scheduled later, so that they have the chance to mix with the existing decoding tokens in the continuous batch.
  - Estimate ratio of len(decoding)/len(prefill)

$$R_{group} = \frac{1}{L_{prefix} + \sum L_{distinct}}$$

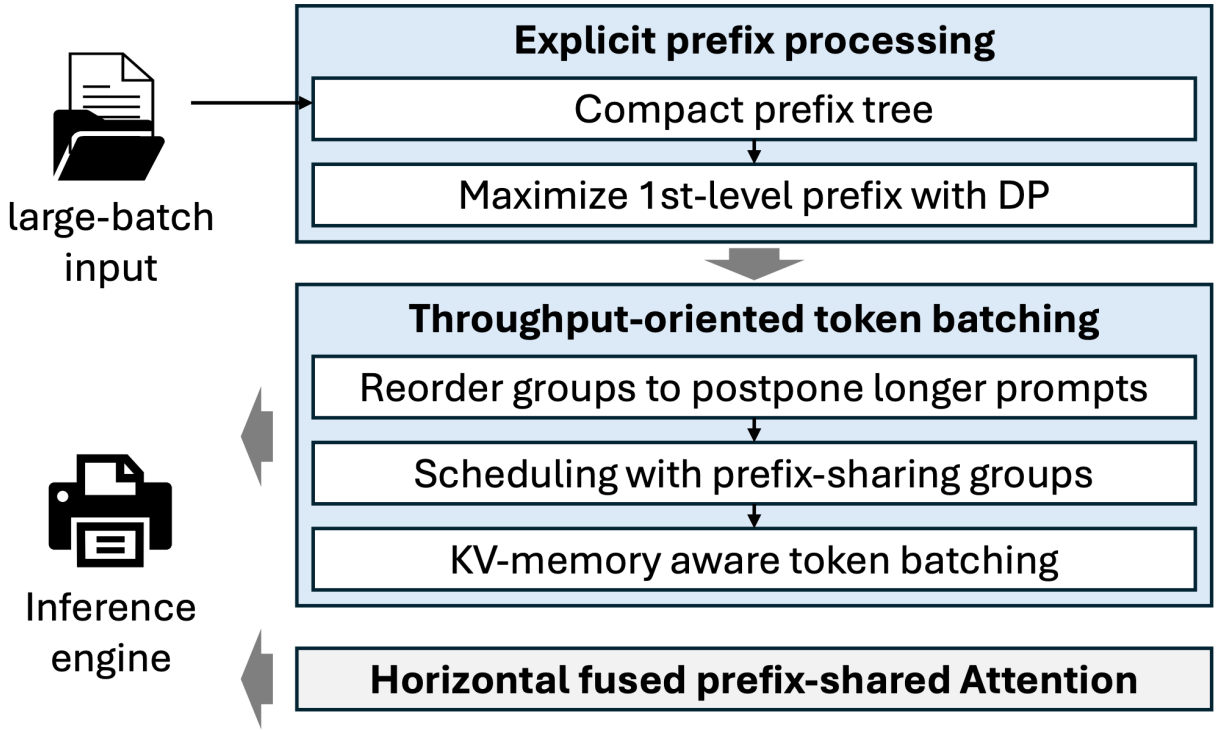
Token Batching with Global Information

To maximize the token batch size

- **Memory-centric Token Batching to Saturate GPU**
  - Remove the request number bound in each batch
  - Introduce memory threshold $M_{threshold}$ as the new bound

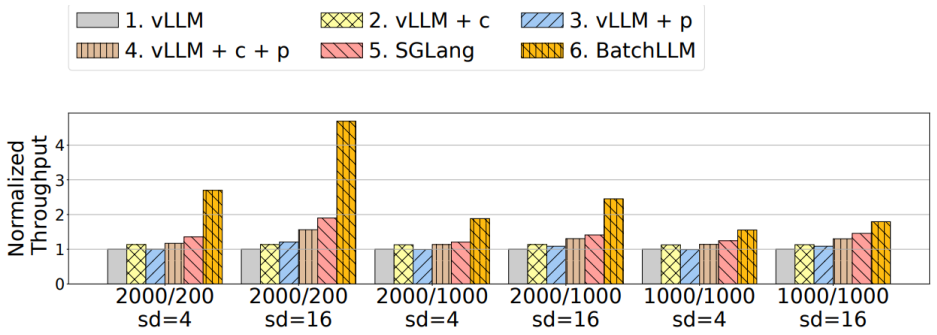
System Implementation

- **Easy integration & kernel optimization**
  - Token batching scheduler
  - Attention backend with horizontal fusion of prefix-aware attn

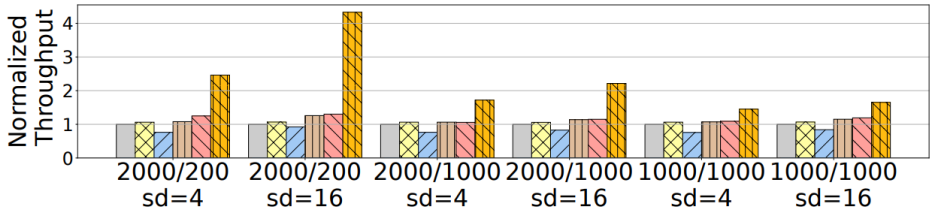


Evaluation

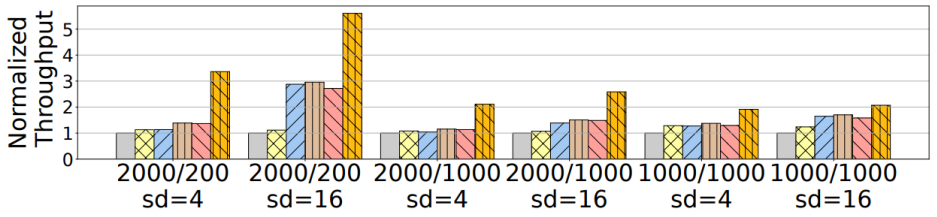
Outperforms vLLM and SGLang by **1.3× to 10.8×**



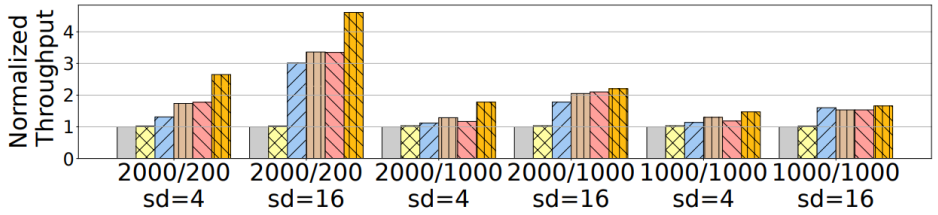
(a) Llama 3.1 8B throughput on A100, #requests = 6400



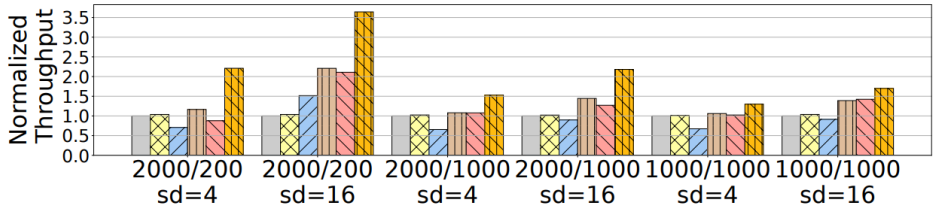
(b) Qwen 2.5 14B throughput on A100, #requests = 6400



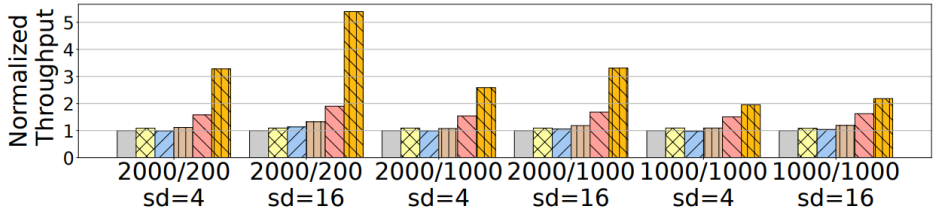
(c) Llama 3.1 70B throughput, on A100, #requests = 800



(d) Llama 3.1 70B throughput, on A100-TP2, #requests = 800



(e) Llama 3.1 8B throughput with Poisson perturbed lengths on A100, #requests = 6400



(f) Llama 3.1 8B throughput, on MI200, #requests = 6400

THANKS

https://github.com/microsoft/MixLLM/tree/batchllm_vllm_064