FlashDecoding++: Faster Large Language Model Inference on GPUs with Asynchronization, Flat GEMM Optimization, and Heuristics

Ke Hong*[1,3], Guohao Dai*[2,3], Jiaming Xu*[2,3], Qiuli Mao[1,3], Xiuhong Li[3,4], Jun Liu[1,3], Kangdi Chen[3], Yuhan Dong[1], Yu Wang[1]

Equal contribution*, Tsinghua University[1], Shanghai Jiao Tong University[2], Infinigence-AI[3], Peking University[4]

Correspondence to:

Guohao Dai <daiguohao@sjtu.edu.cn>

Yu Wang <yu-wang@tsinghua.edu.cn>

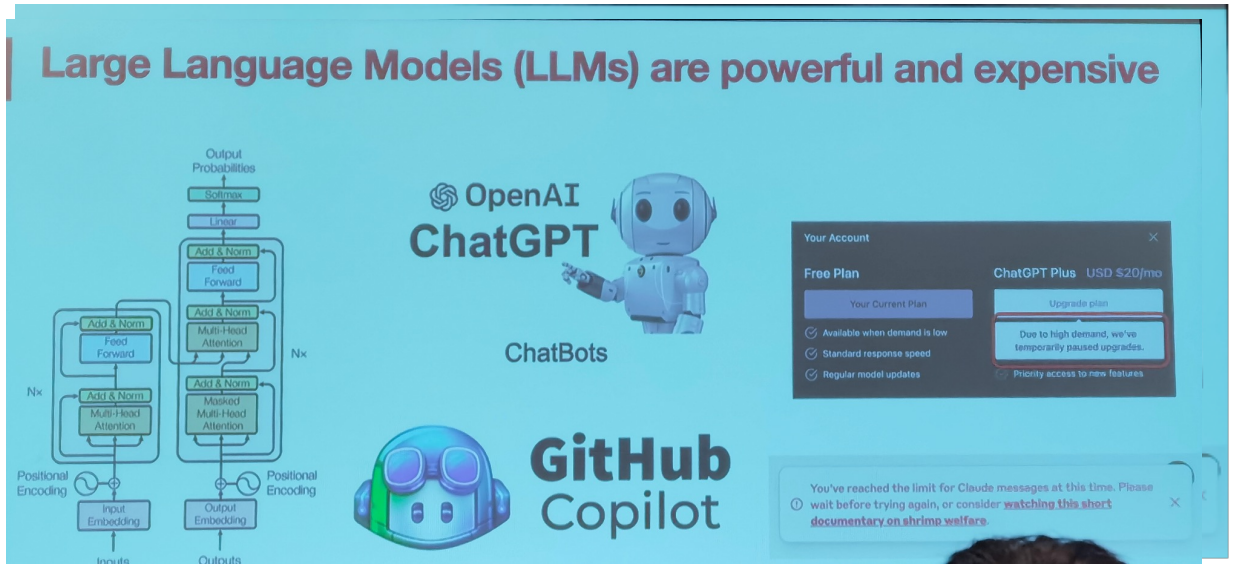


Outline

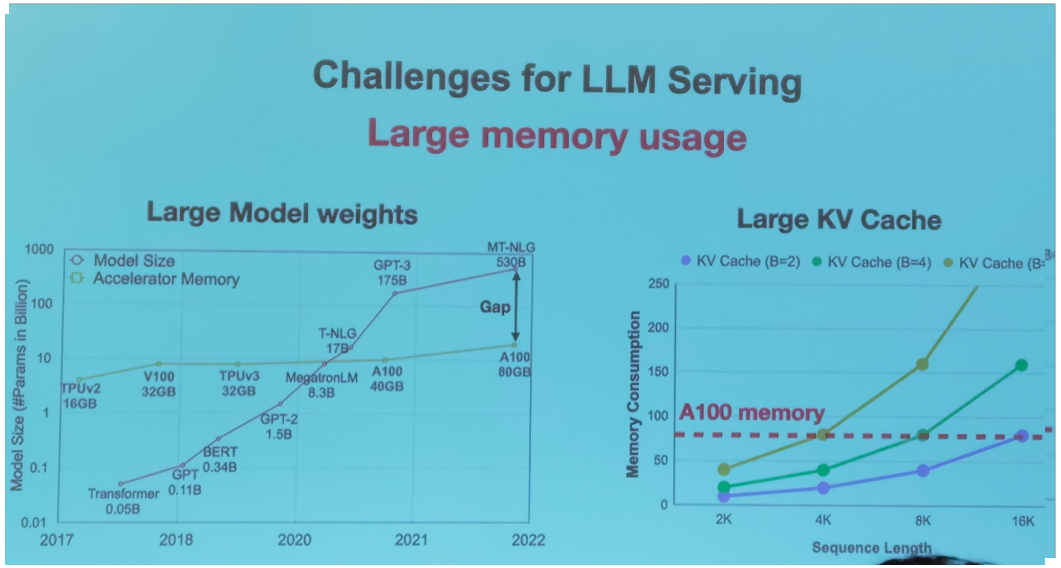
- Backgrounds and Motivations
- Related Work
- Challenges and Techniques
  - Overview
  - Attention Operator
  - Linear Operator
  - Compilation Tuning
- Experiment Results
- Extension Works

Era of LLMs

Large Language Models have attracted much attention.



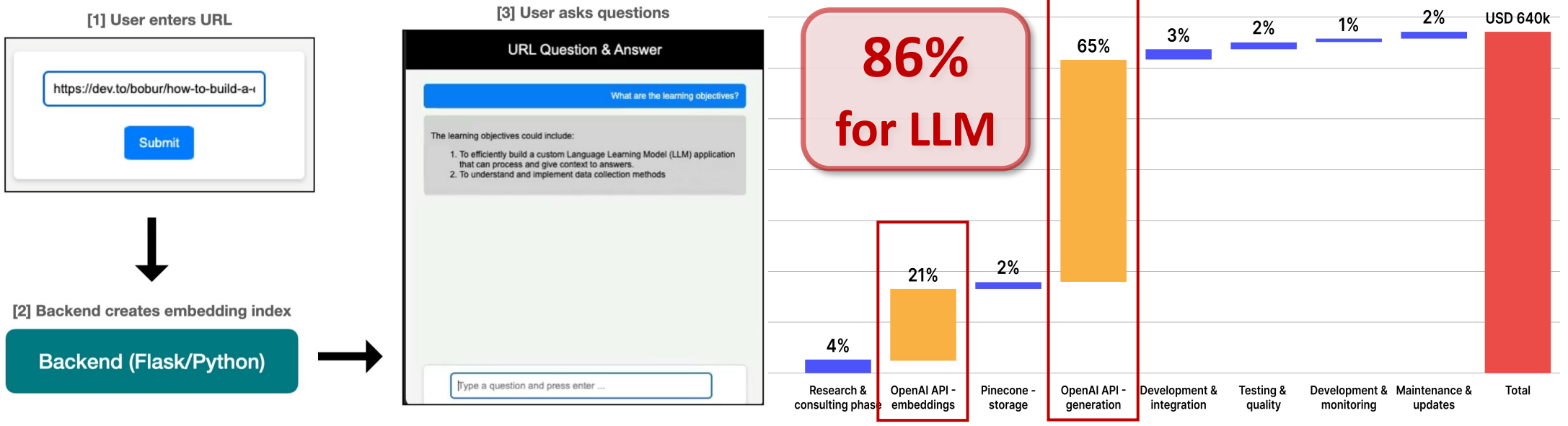
AWQ [MLSys'24 Best Paper]



Atom [MLSys'24]

Expensive Cost

Most budge of downstream applications are for LLM.

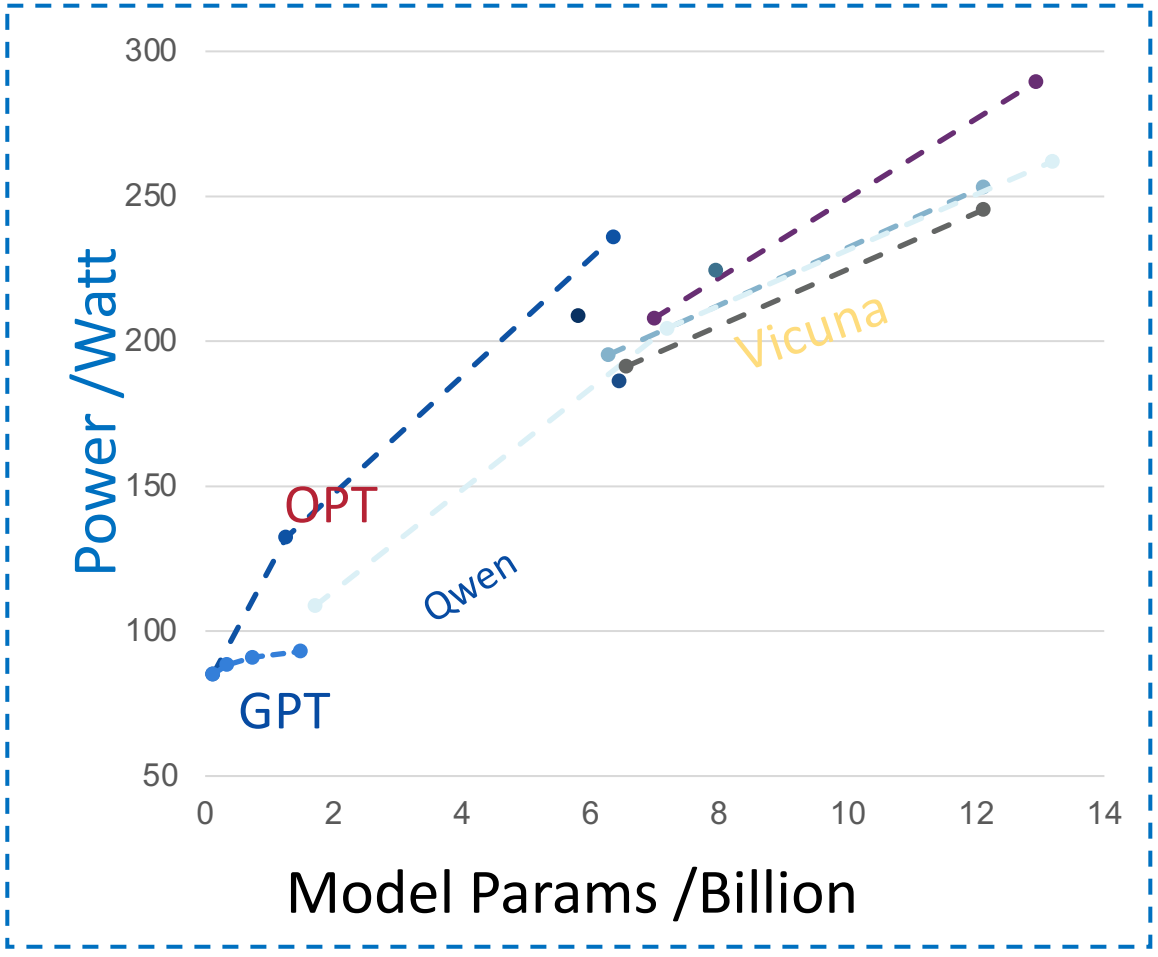


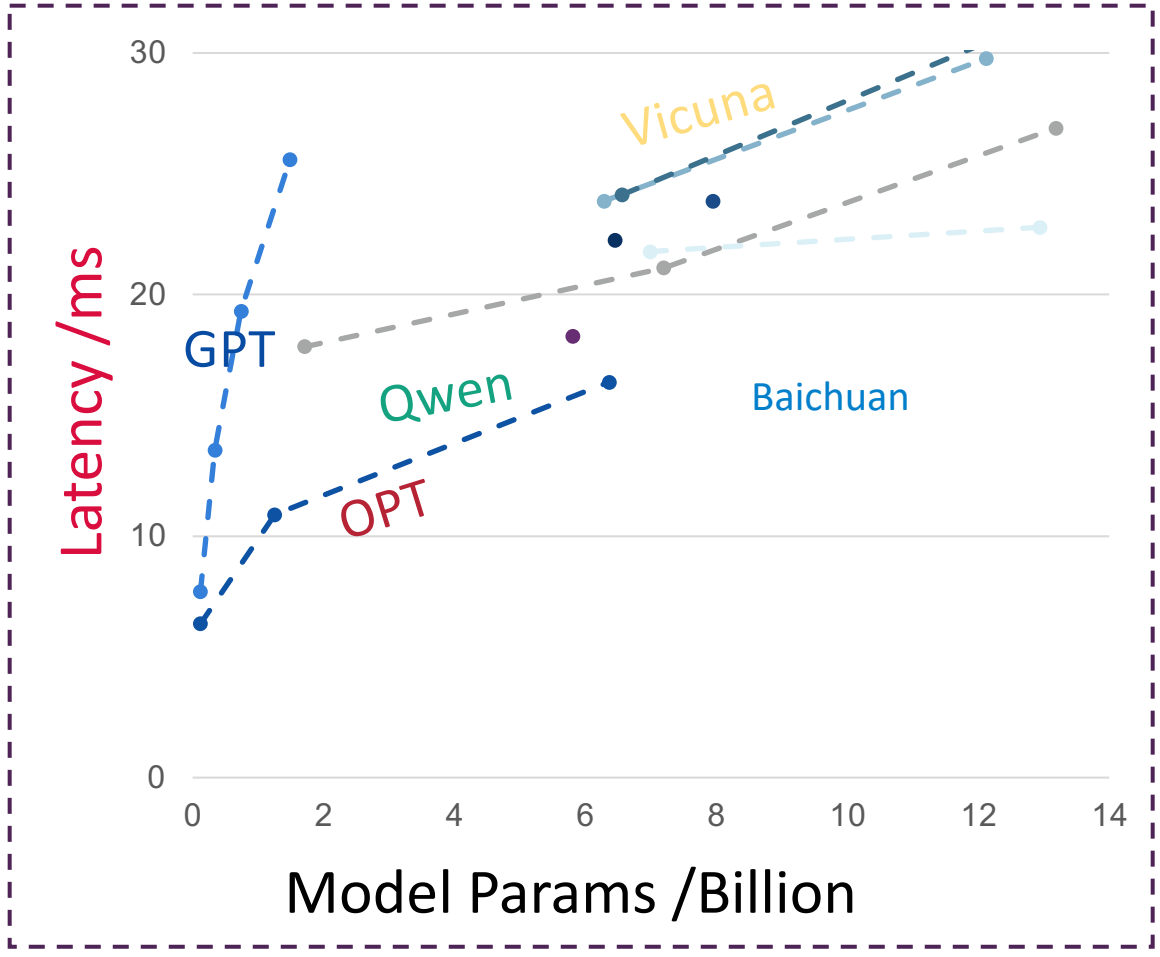
Total Cost of LLM application based on OpenAI[1]

[1] https://www.nebuly.com/blog/understanding-the-total-cost-of-openai

Expensive Cost

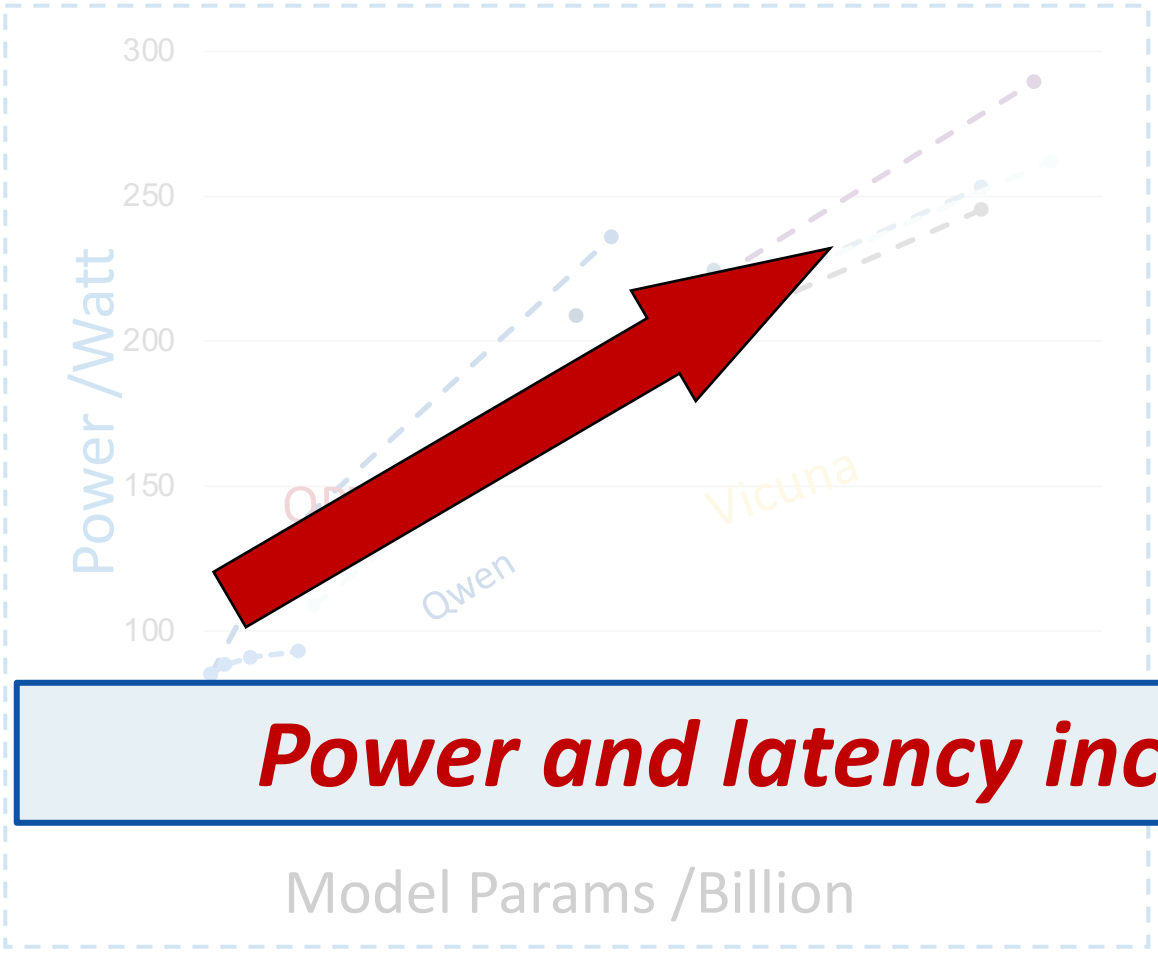
Energy Cost = Power × Latency × Price

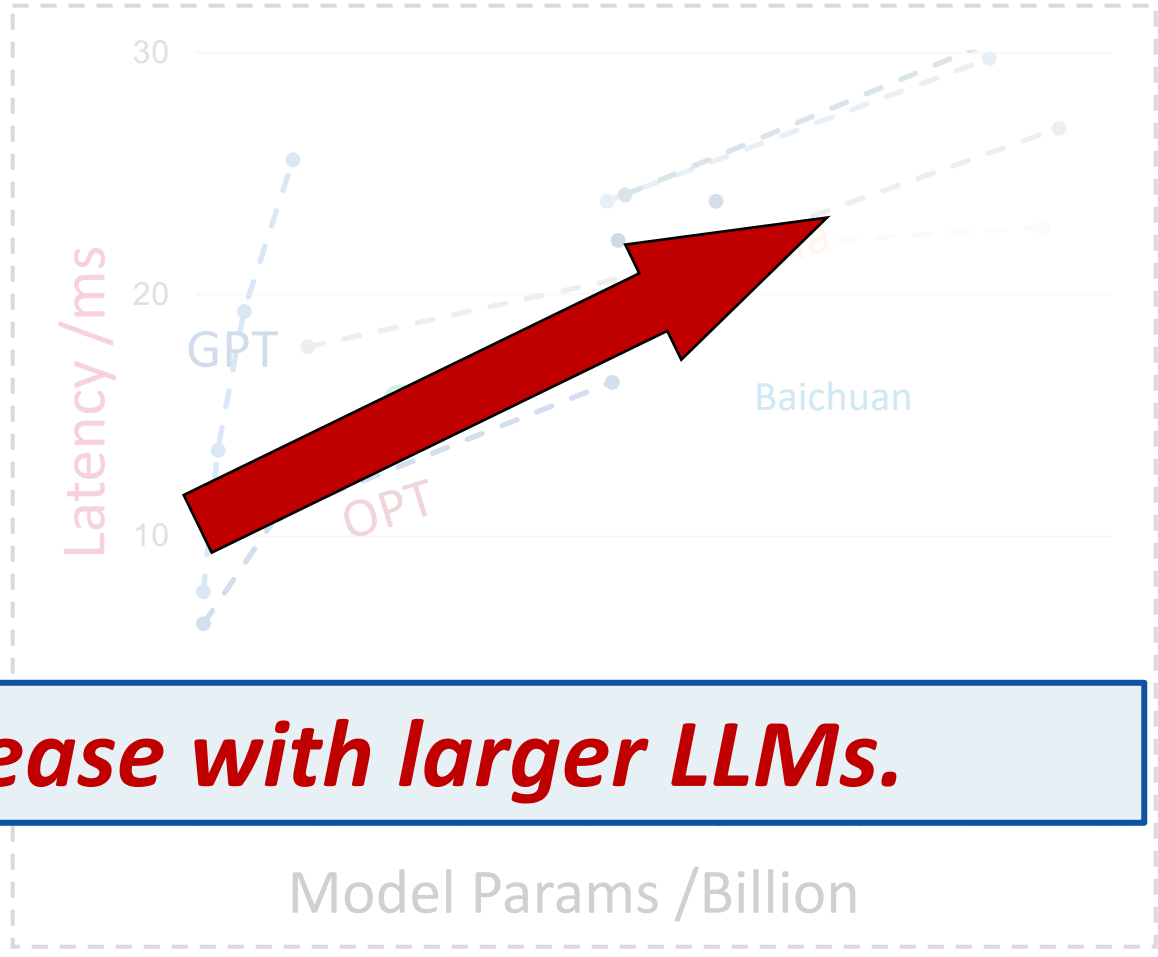




Expensive Cost

Energy Cost = Power × Latency × Price





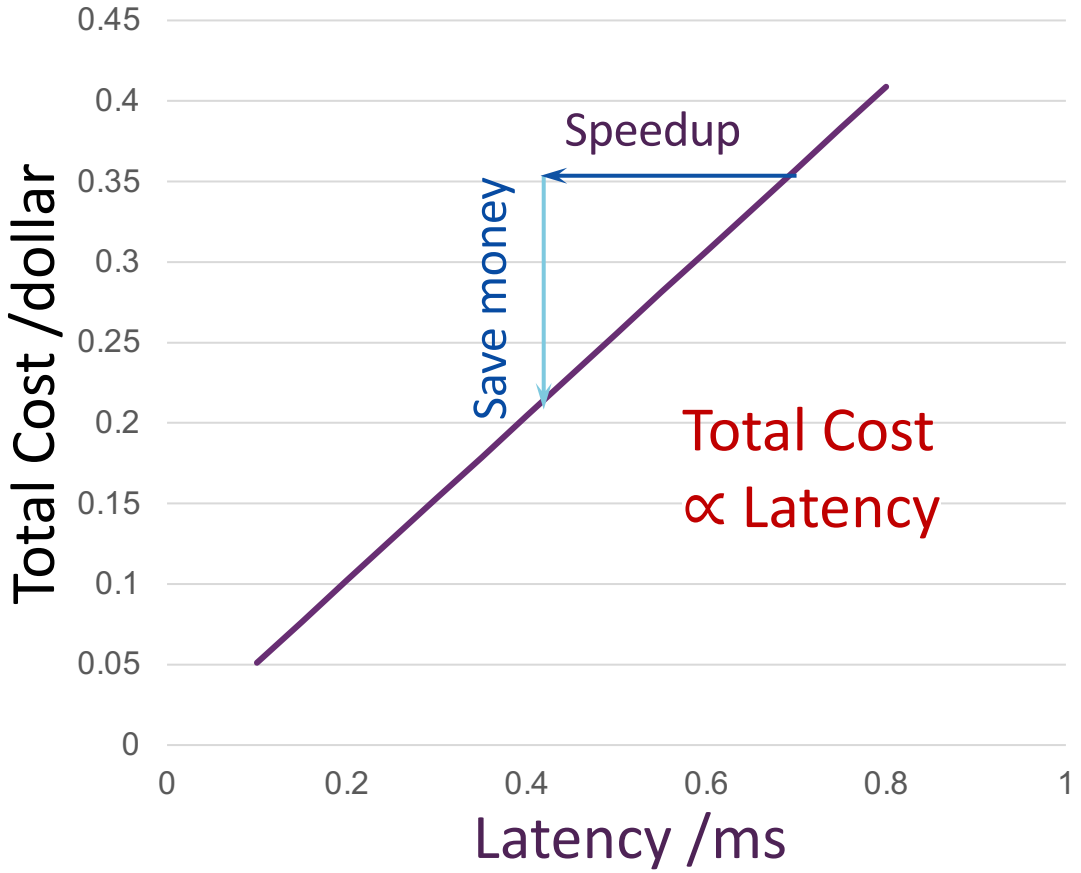
Power and latency increase with larger LLMs.

Expensive Cost

Total Cost = Energy Cost + Hardware Cost
= Latency × (Energy Price + Hardware Depreciation)

Total Cost

NVIDIA A100 SXM 80GB	$17900
Active Time	5 years
Hardware Depreciation	$0.409/hour
Power	400W
Energy Price ($0.256/kWh)	$0.102/hour



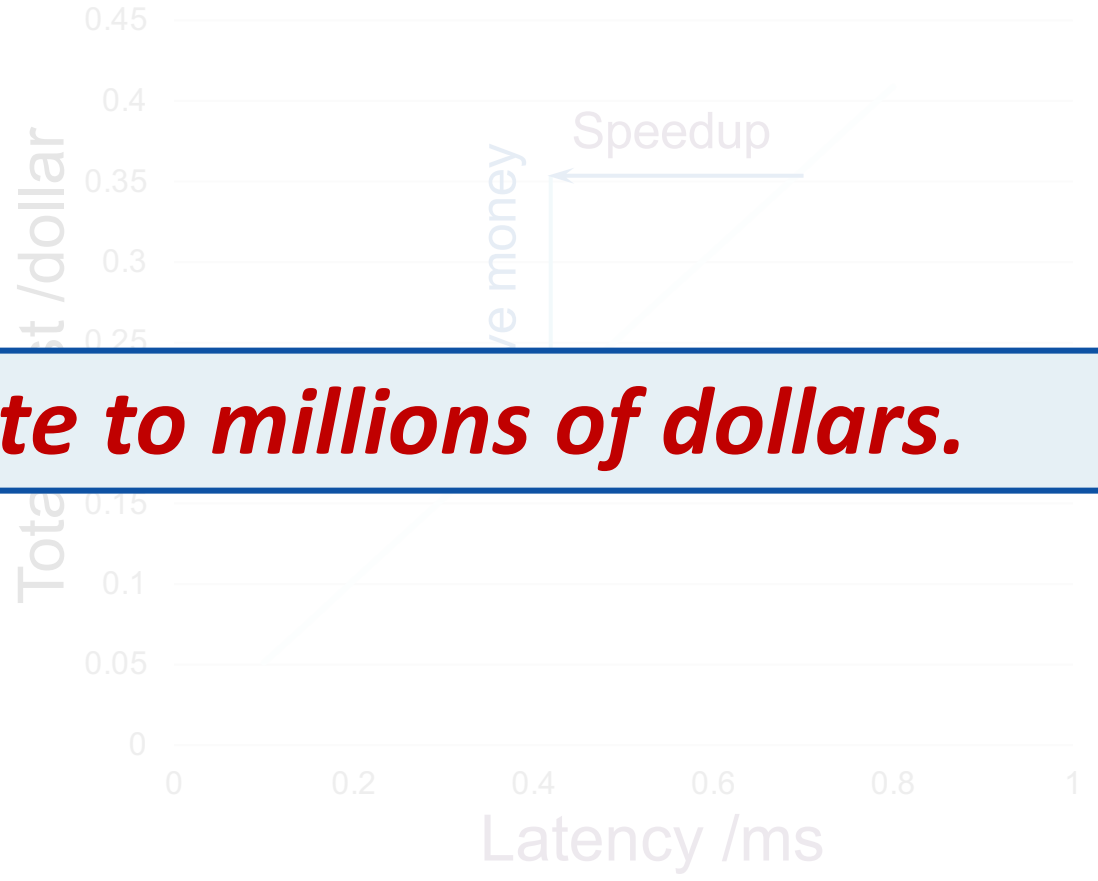
Expensive Cost

Total Cost = Energy Cost + Hardware Cost

= Latency × (Energy Price + Hardware Consumption)

Total Cost

NVIDIA A100 SXM 80GB	$17900
Consumption	ur
Power	400W
Energy Price ($0.256/kWh)	$0.102/hour

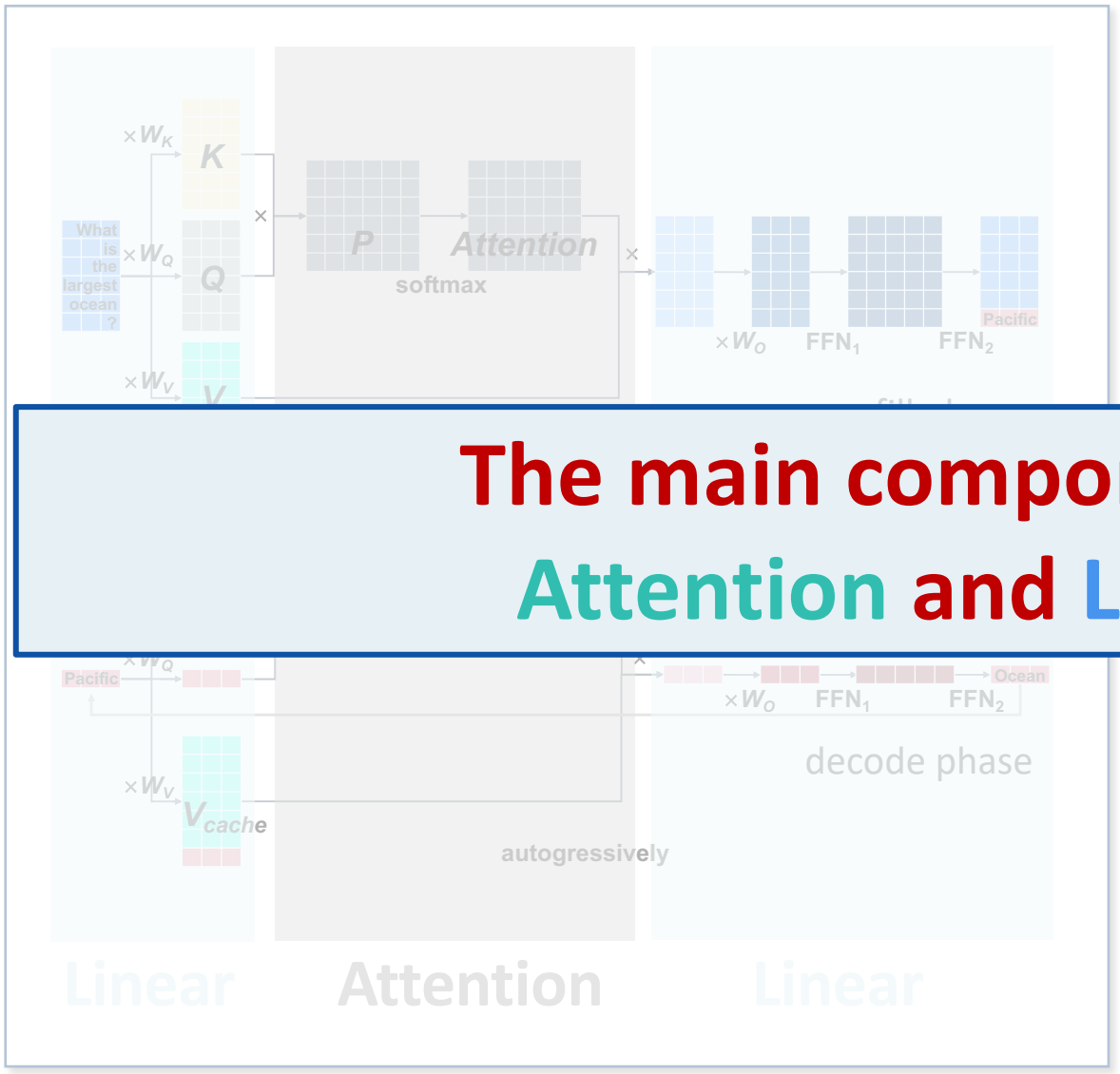


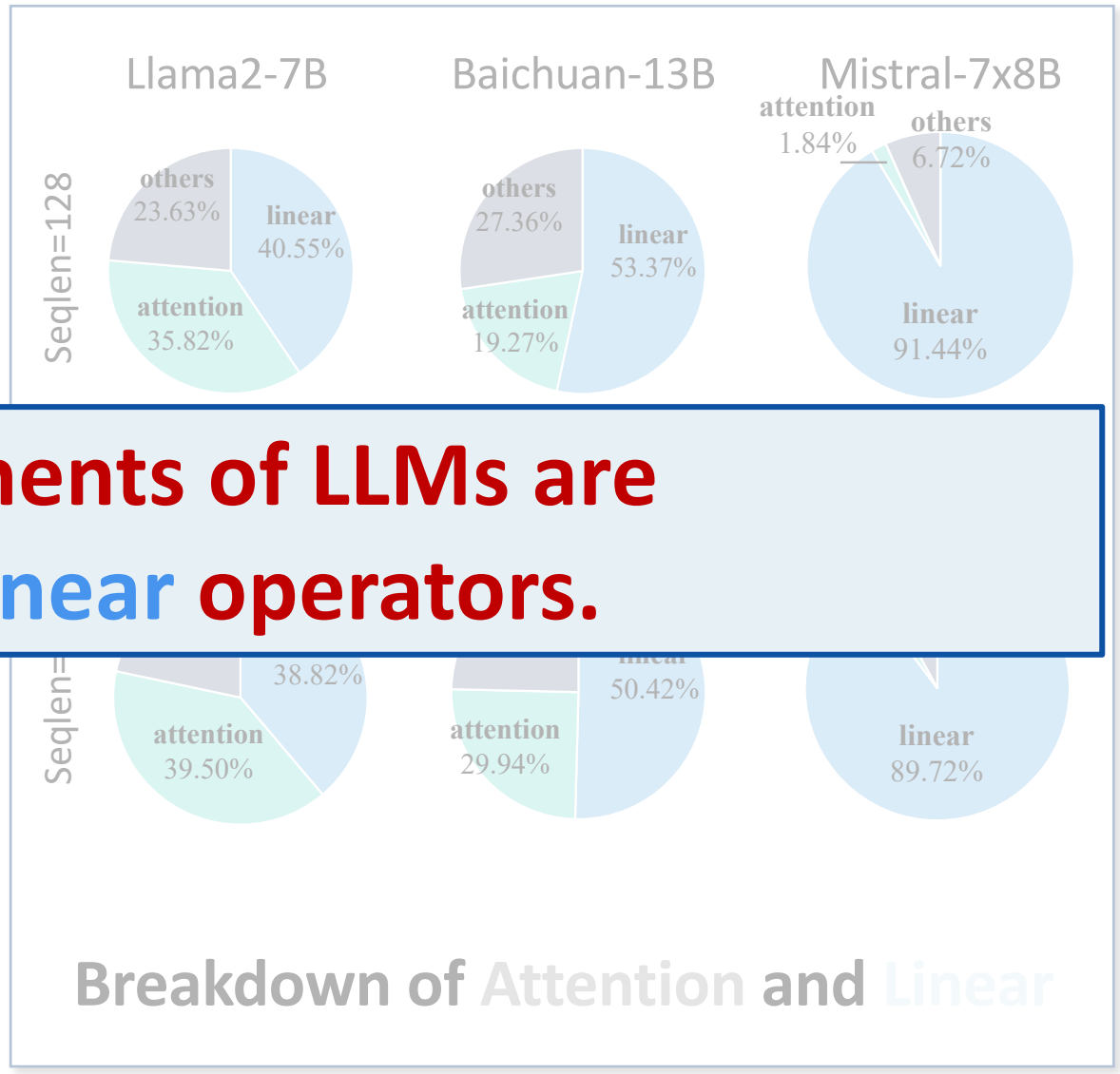
Speedup can translate to millions of dollars.

Outline

- Backgrounds and Motivations
- Related Work
- Challenges and Techniques
  - Overview
  - Attention Operator
  - Linear Operator
  - Compilation Tuning
- Experiment Results
- Extension Works

LLM Dataflow

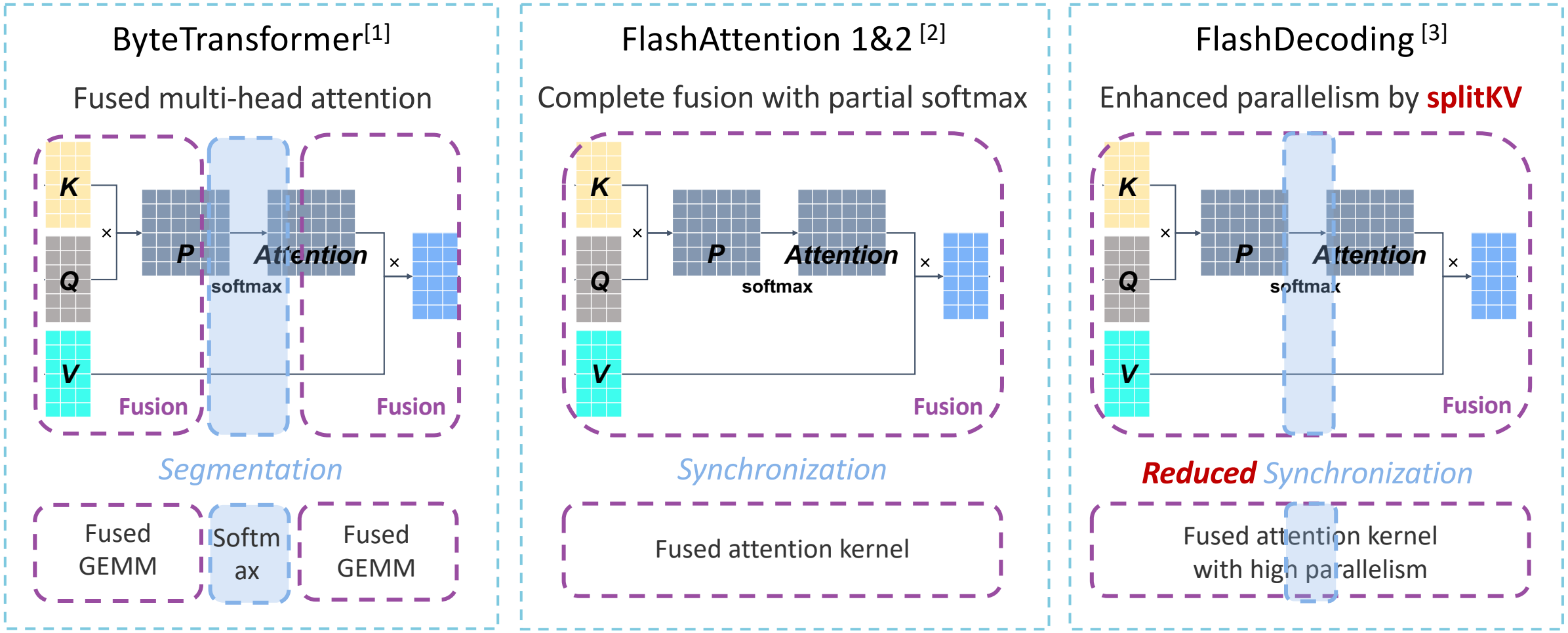




The main components of LLMs are Attention and Linear operators.

Related Work

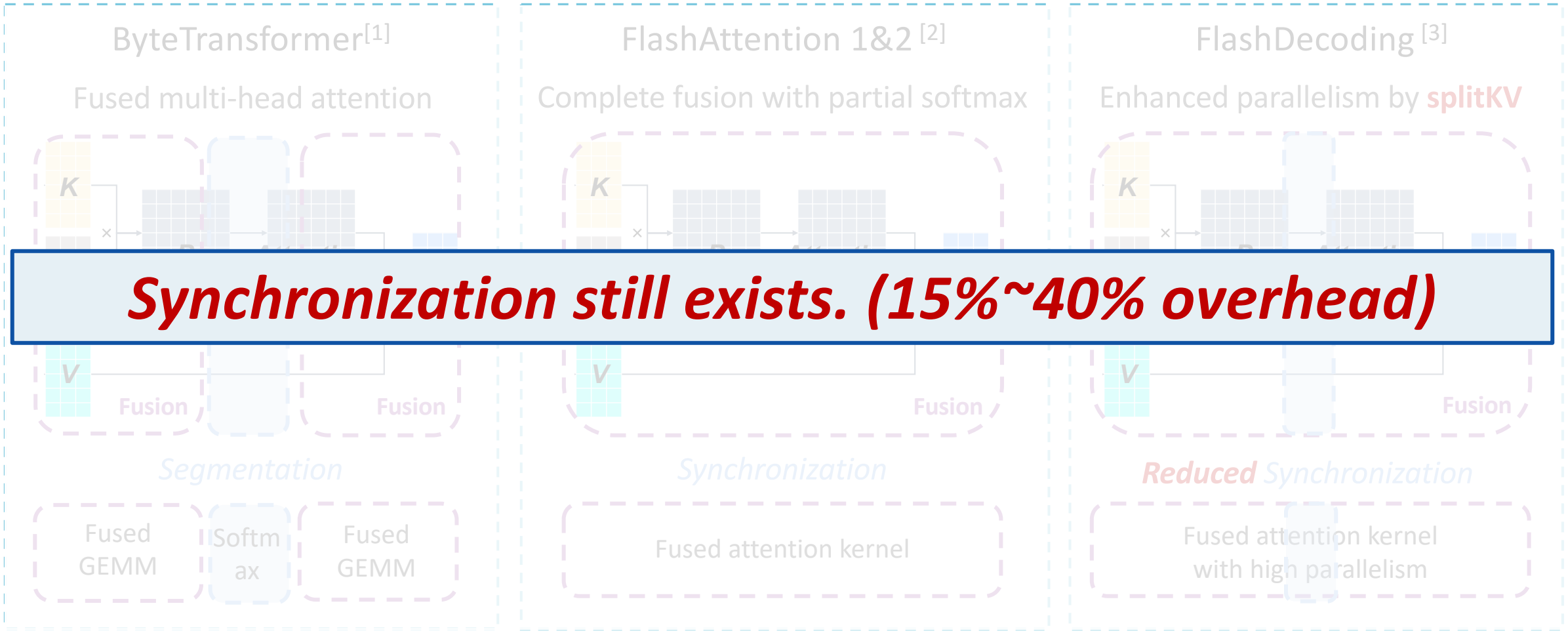
Attention Operator Optimization



[1] Zhai Y, Jiang C, Wang L, et al. Bytetransformer: A high-performance transformer boosted for variable-length inputs[C]//2023 IPDPS
[2] Dao T, Fu D, Ermon S, et al. Flashattention: Fast and memory-efficient exact attention with io-awareness[C]//2022 NeurPIS
[3] https://crfm.stanford.edu/2023/10/12/flashdecoding.html

Related Work

Attention Operator Optimization



Synchronization still exists. (15%~40% overhead)

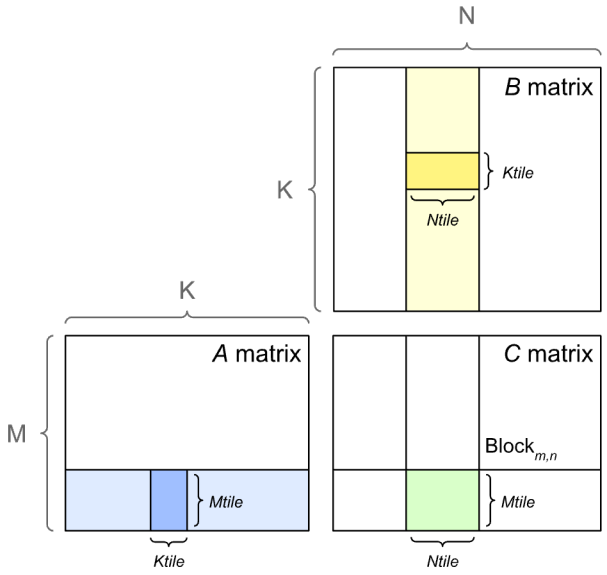
[1] Zhai Y, Jiang C, Wang L, et al. Bytetransformer: A high-performance transformer boosted for variable-length inputs[C]//2023 IPDPS
[2] Dao T, Fu D, Ermon S, et al. Flashattention: Fast and memory-efficient exact attention with io-awareness[C]//2022 NeurPIS
[3] https://crfm.stanford.edu/2023/10/12/flashdecoding.html

Related Work

Linear Operator Optimization

cuBLAS[1]

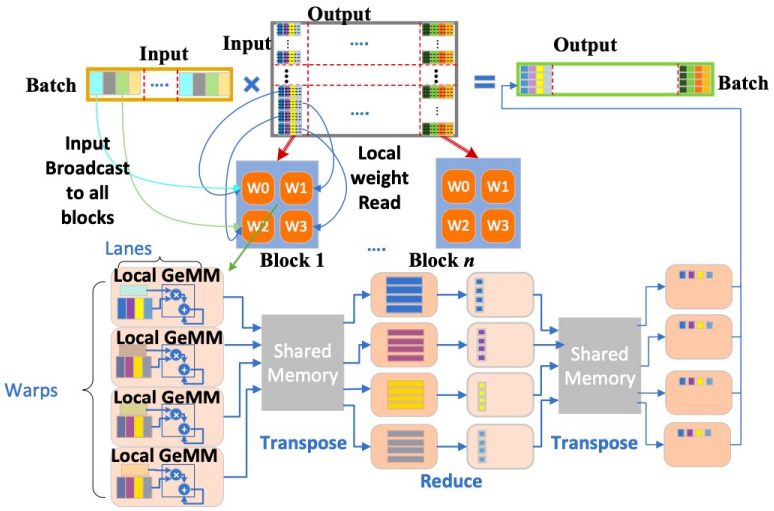
General Linear Operator



Lack optimization *for LLM*

DeepSpeed[2]

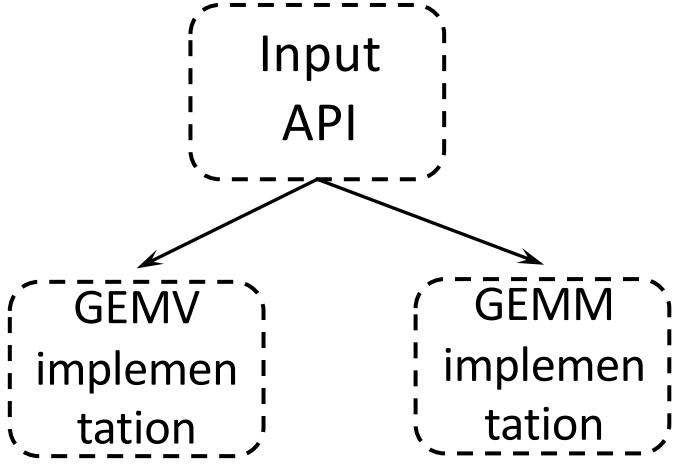
Skinny Linear Operator



Custom Linear for ***small batchsize*** *but with suboptimal tiling*

TensorRT-LLM [3]

Input-aware Linear Operator



Only add ***the scenario of batchsize=1***

[1] NVIDIA, “cublas: Basic linear algebra on nvidia gpus,” [Online], 2017, https://developer.nvidia.com/cublas.

[2] Aminabadi R Y, et al. Deepspeed-inference: enabling efficient inference of transformer models at unprecedented scale[C]//SC22

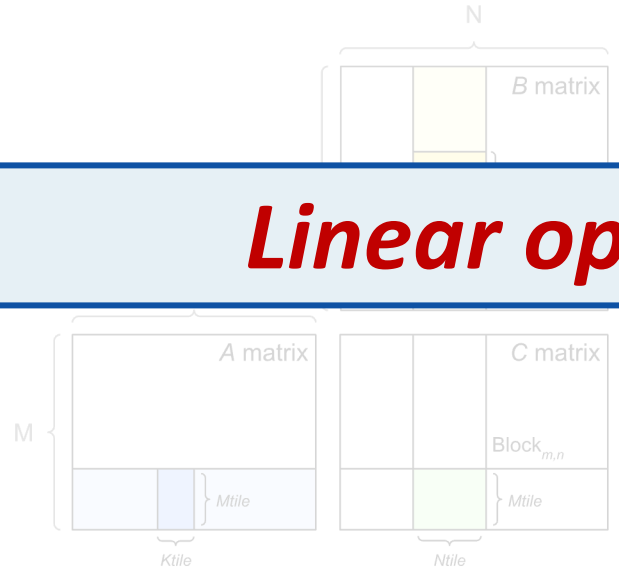
[3] N. Vaidya, F. Oh, and N. Comly, “Optimizing inference on large language models with nvidia tensorrt-llm, now publicly available,” 2023

Related Work

Linear Operator Optimization

cuBLAS[1]

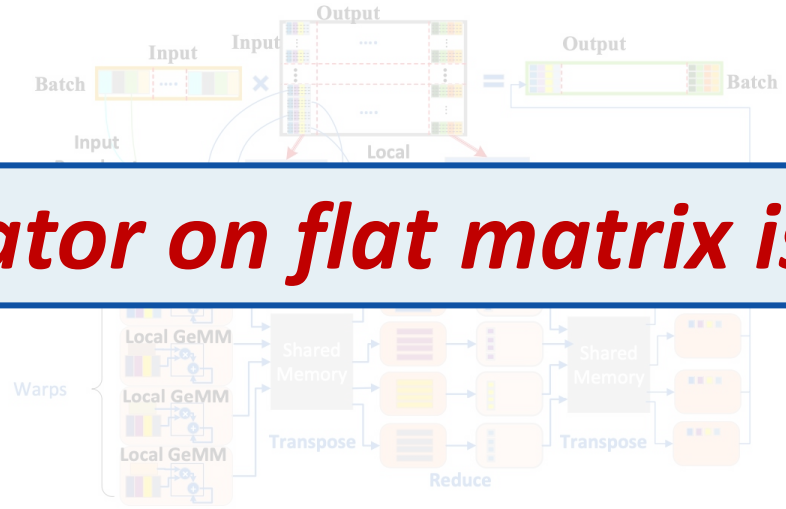
General Linear Operator



Lack optimization for LLM

DeepSpeed[2]

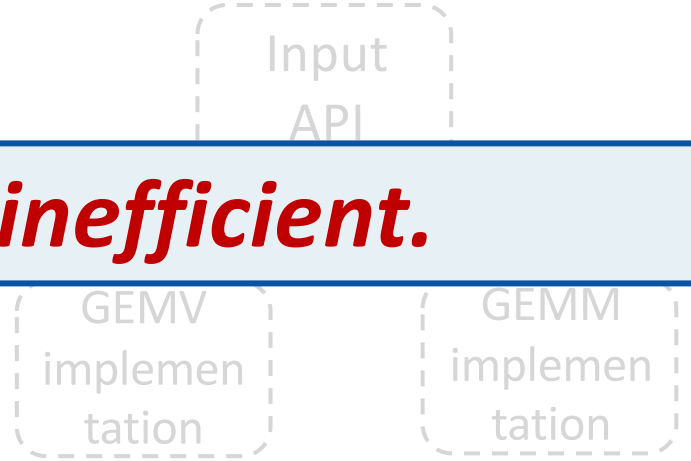
Skinny Linear Operator



Custom Linear for small batch but lack optimization on Tensor Core

TensorRT-LLM [3]

Input-aware Linear Operator



Only add the scenario of batchsize=1

Linear operator on flat matrix is inefficient.

[1] NVIDIA, “cublas: Basic linear algebra on nvidia gpus,” [Online], 2017, https://developer.nvidia.com/cublas.

[2] Aminabadi R Y, et al. Deepspeed-inference: enabling efficient inference of transformer models at unprecedented scale[C]//SC22

[3] N. Vaidya, F. Oh, and N. Comly, “Optimizing inference on large language models with nvidia tensorrt-llm, now publicly available,” 2023

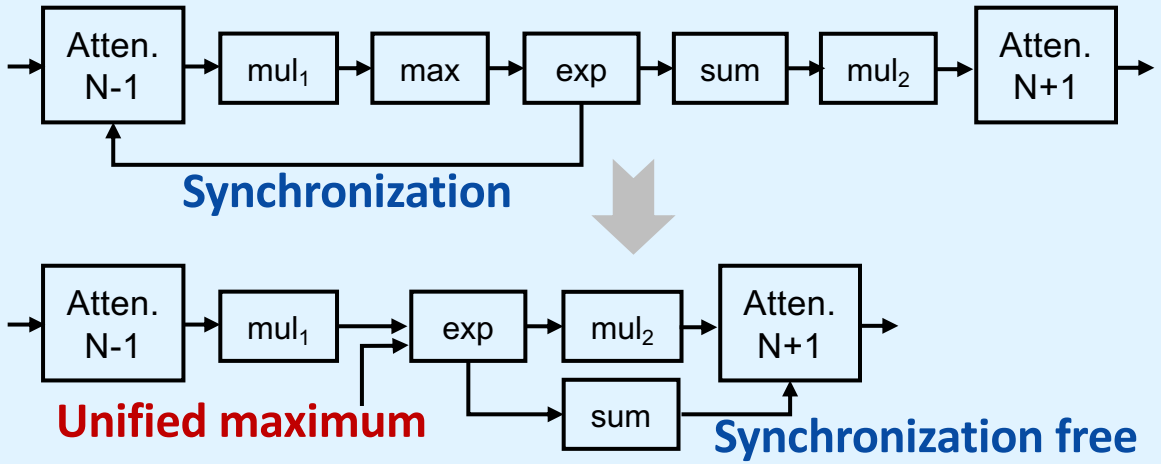
Outline

- Backgrounds and Motivations
- Related Work
- Challenges and Techniques
  - Overview
  - Attention Operator
  - Linear Operator
  - Compilation Tuning
- Experiment Results
- Extension Works

Techniques Overview

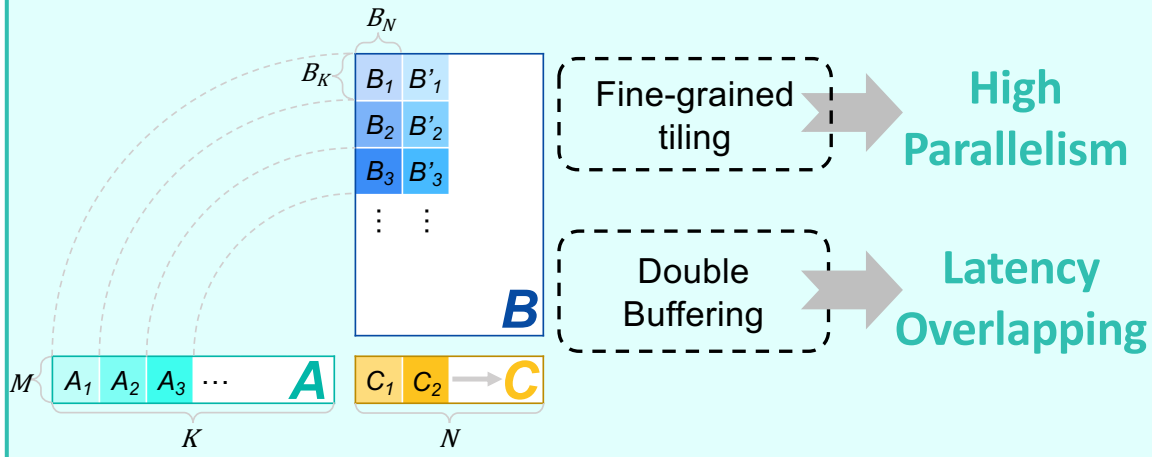
Tech. 1: Attention Operator

Eliminate synchronization with *unified maximum*



Tech. 2: Linear Operator

Customized designs for *GEMMs with flat shape*



GEMM with flat shape (Flat GEMM)

Tech. 3: Compilation Tuning

Find the optimal implementation for *diverse linear workloads*



Technique 1: Attention Operator

Challenge **Attention operator based on the partial softmax introduces overhead for updates.**

Original softmax

$softmax(x)$

x_1 x_2 x_3 x_4 ⋯ x_{d-1} x_d

$$m(x) = max(x_i)$$

$$f(x) = \left(e^{x_1 - m(x)}, e^{x_2 - m(x)}, \dots, e^{x_d - m(x)}\right)$$

$$l(x) = \sum_i f_i(x)$$

$$softmax(x) = \frac{f(x)}{l(x)}$$

Partial softmax

$softmax(x')$ $softmax(x'')$

x_1 x_2 x_3 x_4 ⋯ x_{d-1} x_d

Calcutate $m(x')$, $f(x')$, $l(x')$, $softmax(x')$

Calcutate $m(x'')$, $f(x'')$, $l(x'')$, $softmax(x'')$

Update $softmax(x')$ with $m(x')$, $f(x')$, $l(x')$, $m(x'')$, $f(x'')$, $l(x'')$

Update $softmax(x'')$ with $m(x')$, $f(x')$, $l(x')$, $m(x'')$, $f(x'')$, $l(x'')$

Process next partial vector

Update Overhead

Additional update is due to the different maximums in different blocks.

Is it feasible to use a unified maximum for all blocks?

Technique 1: Attention Operator

Approach

A unified maximum for partial softmax operation and a recomputation mechanism to ensure accuracy.

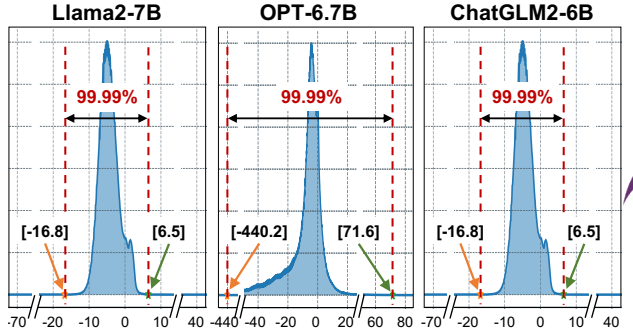
① The unified maximum preserves the mathematical equivalence of softmax

$$softmax(x) = \frac{[e^{x_1 - m(x)}, ..., e^{x_d - m(x)}]}{\sum_i e^{x_i - m(x)}}$$

$$= \frac{[e^{x_1 - \phi}, ..., e^{x_d - \phi}]}{\sum_i e^{x_i - \phi}} \quad \forall \phi \in \mathbb{R}$$

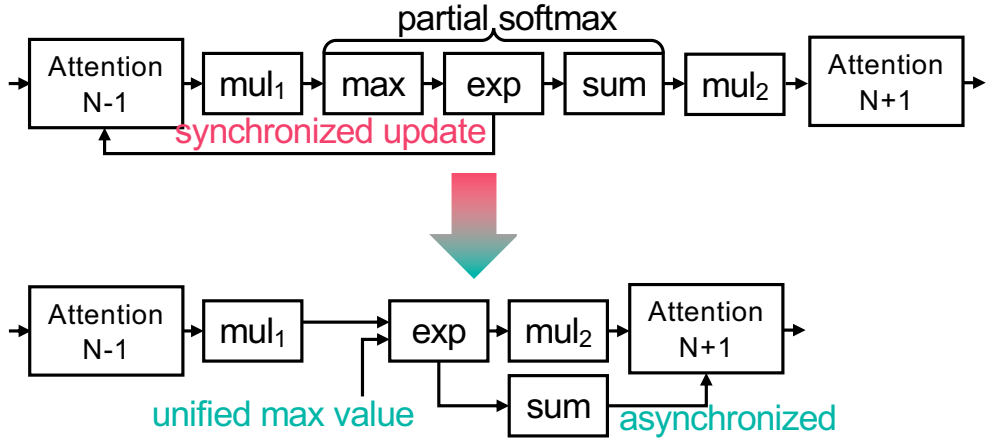
Unified maximum

② The unified "max" value guarantees the accuracy of FP32 calculation in 99.99% cases



Computational Feasibility (FP32)

Synchronized partial softmax update



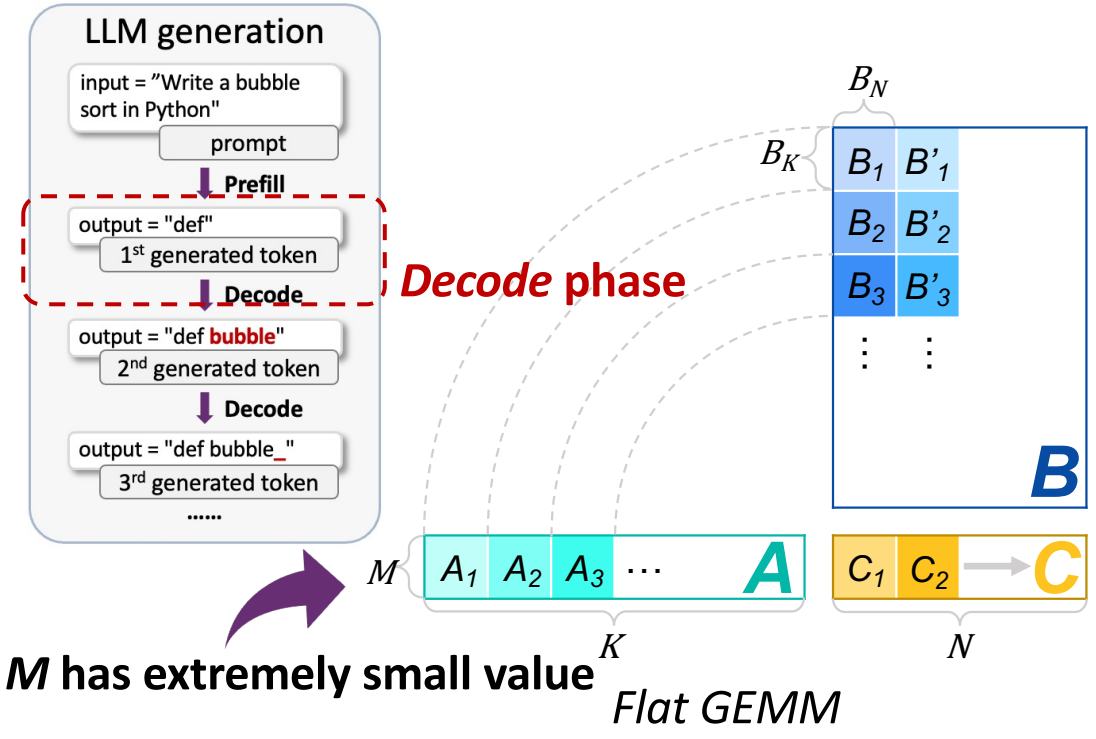
Asynchronized softmax with unified max value

Up to 2.02x speedup over FlashDecoding

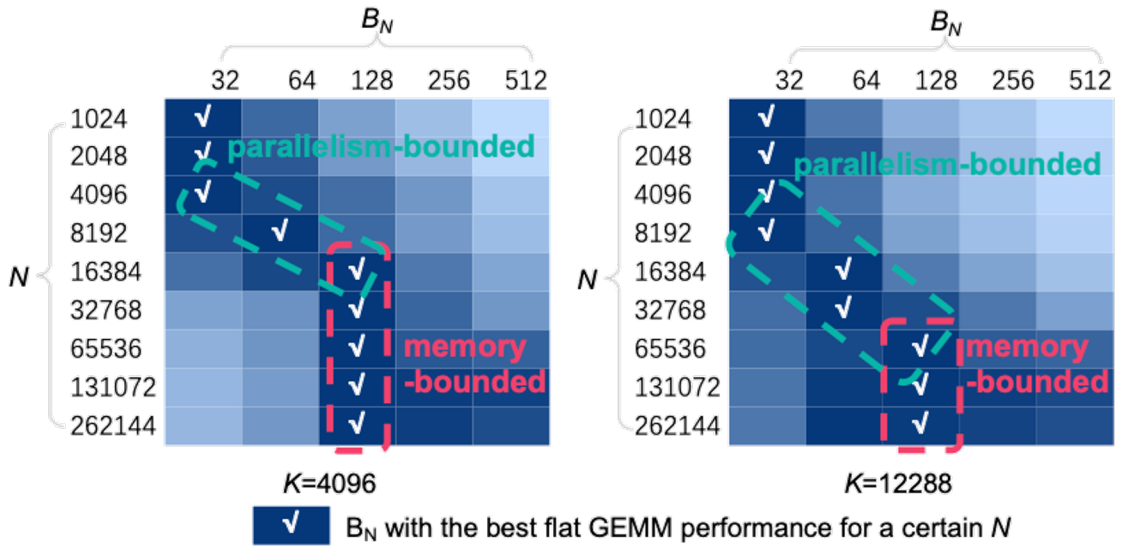
Technique 2: Linear Operator

Challenge **The flat GEMMs need careful designs from the perspective of parallelism and arithmetic intensity.**

GEMM at *decode* phase: Flat GEMM



The bottleneck analysis with different *N*



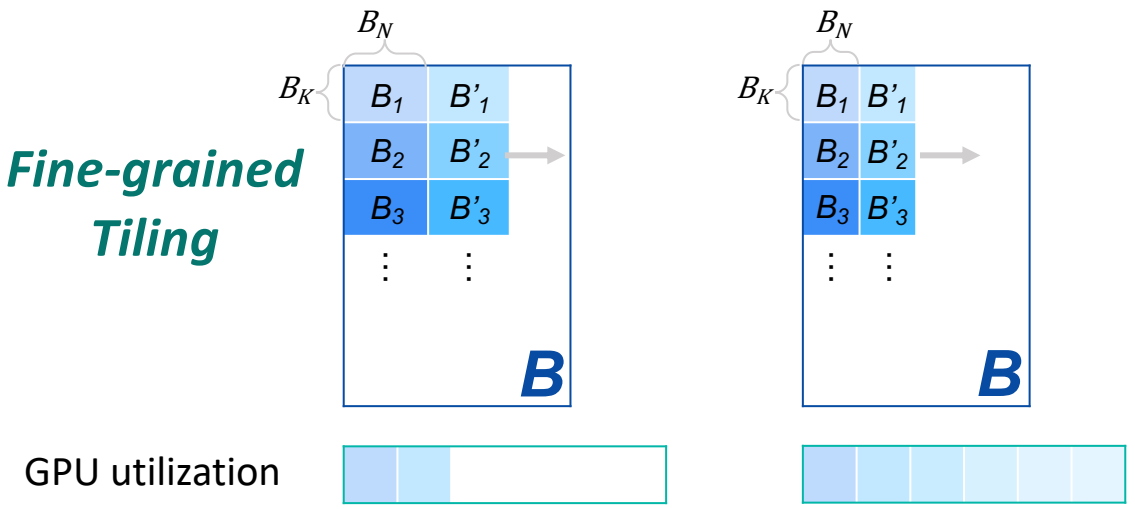
Small *N*: Parallelism-bounded

Large *N*: Memory-bounded

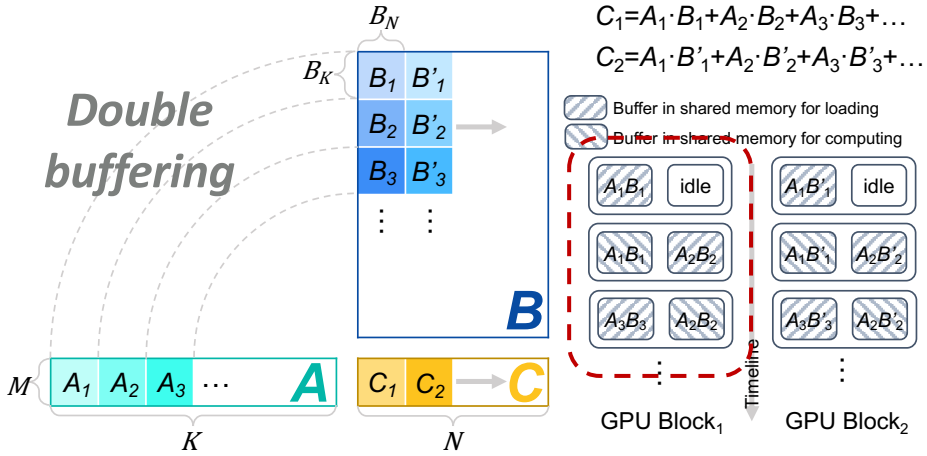
Technique 2: Linear Operator

Approach Use fine-grained tiling to improve parallelism with small *N* and double buffering with large *N*.

Improve parallelism with small *N*



Alleviate memory bottleneck with large *N*



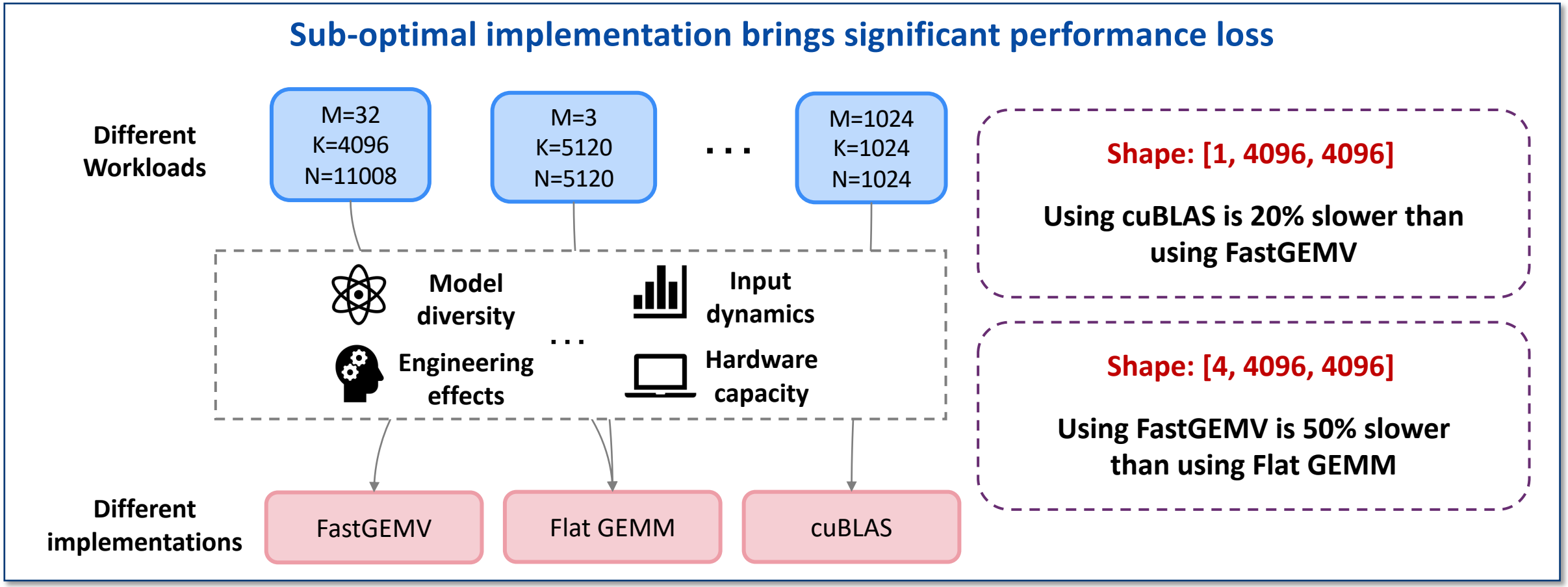
Overlap computation and data movement

1.17x average speedup over cuBLAS

Technique 3: Compilation Tuning

Challenge

For a given linear workload, it is non-trivial to find the optimal underlying implementation.



Technique 3: Compilation Tuning

Approach

Build the mapping table offline and choose the optimal implementation accordingly at runtime.

Reduce the workload variation space

Observation 1
Only *M* varies with the input

Observation 2
Given a model, [*N*, *K*] has only 4 choices

	Operation	M	N	K
Prefill phase	K, Q, V projection	SeqLen*B	HD*3	HD
	O projection	SeqLen*B	HD	HD
	FFN1	SeqLen*B	FD	HD
	FFN2	SeqLen*B	HD	FD
Decode phase	K, Q, V projection	B	HD*3	HD
	O projection	B	HD	HD
	FFN1	B	FD	HD
	FFN2	B	HD	FD

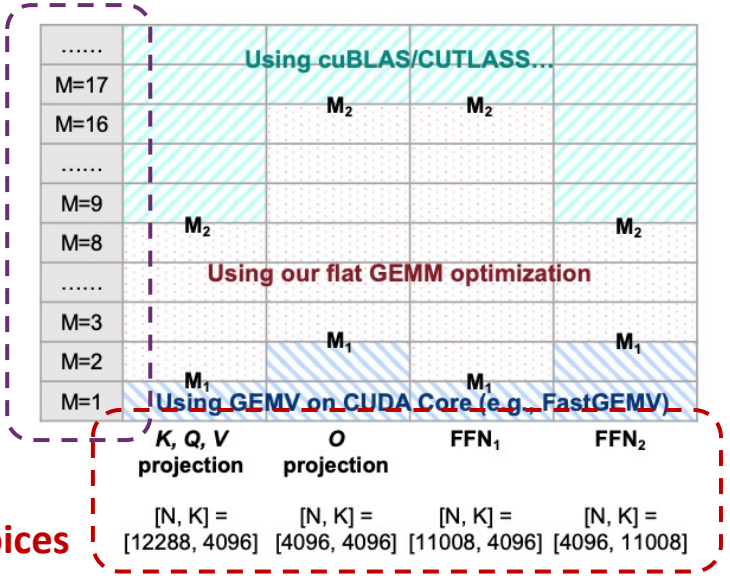
HD: Hidden dimension size
FD: Dimension size after the first FFN
B: Batch size
SeqLen: Input sequence length

Only 4 shapes!

Build the table to map the optimal impl.

Observation 1
Only *M* varies with the input

Observation 2
Given a model, [*N*, *K*] has only 4 choices



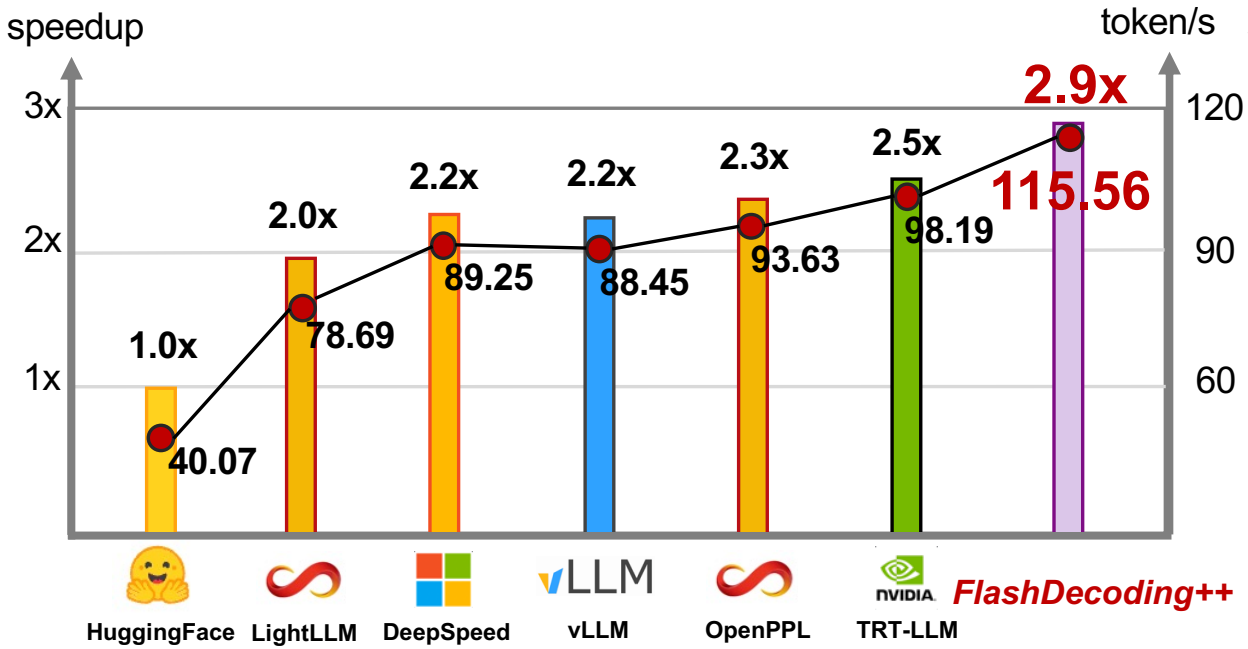
Up to 1.29x speedup over the static implementation

Outline

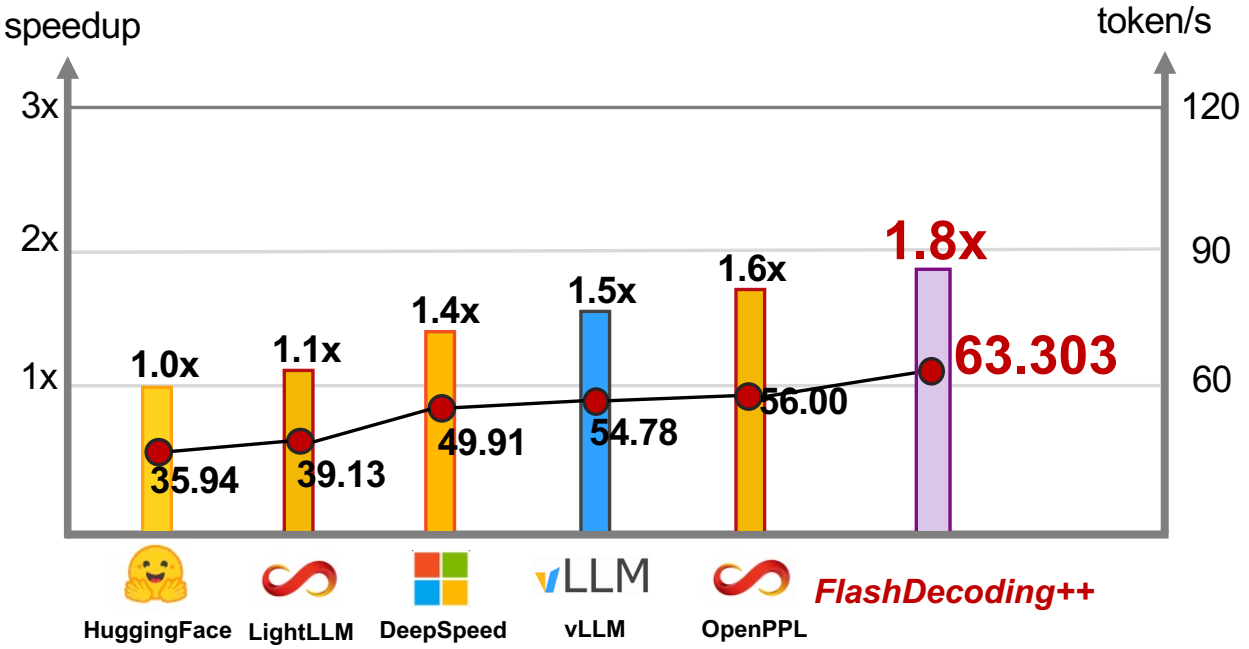
- Backgrounds and Motivations
- Related Work
- Challenges and Techniques
  - Overview
  - Attention Operator
  - Linear Operator
  - Compilation Tuning
- Experiment Results
- Extension Works

Results on NVIDIA GPUs

Experiment Setup		
	NVIDIA A100 SXM 80GB	CUDA 12.1
	NVIDIA RTX 3090 24GB	CUDA 11.6



NVIDIA Tesla A100



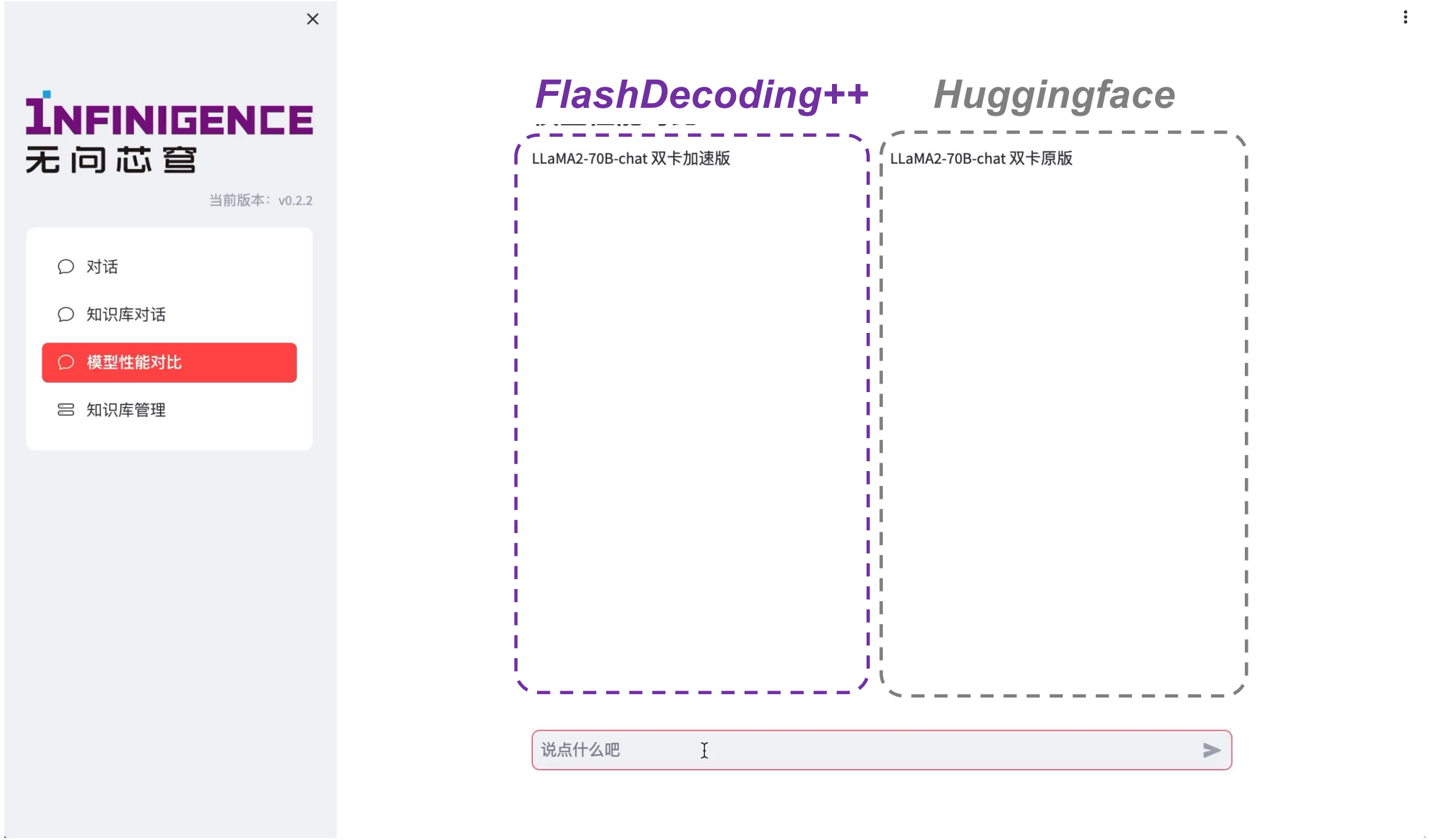
NVIDIA RTX 3090

Llama2-7B, batch size=1, input=128 tokens, output=128 tokens,
the throughput of inference engines on a single GPU

1.88x average speedup over HuggingFace (FP16)

1.56x average speedup over HuggingFace (FP16)

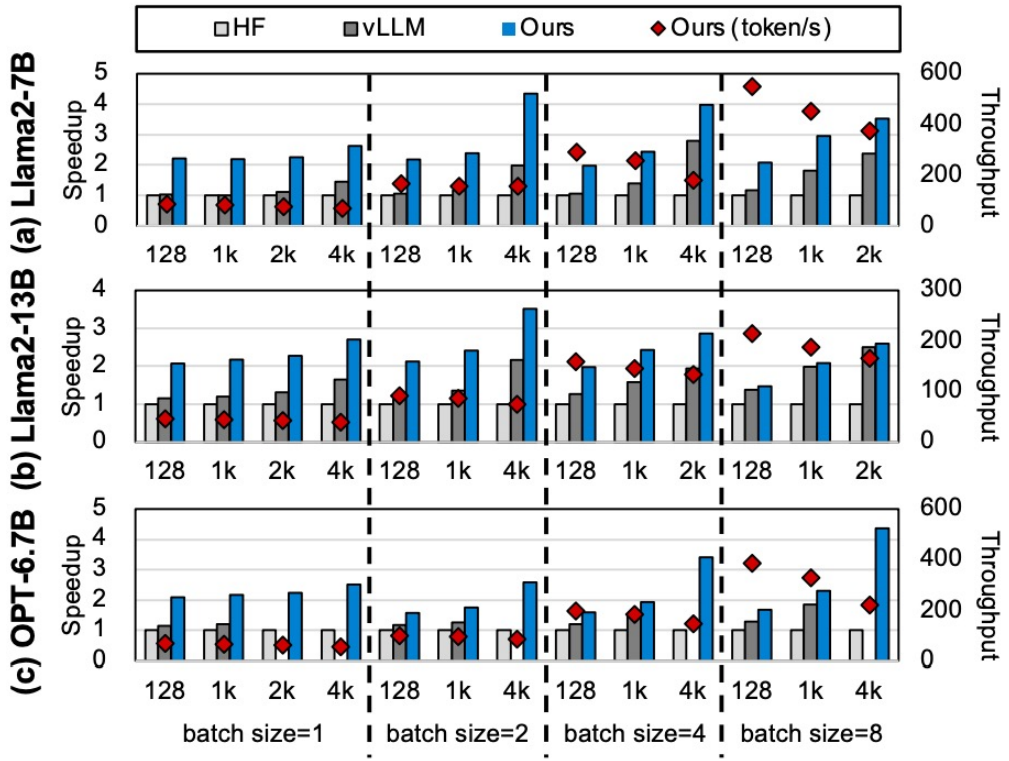
Llama2-70B-chat demo on NVIDIA A100*2



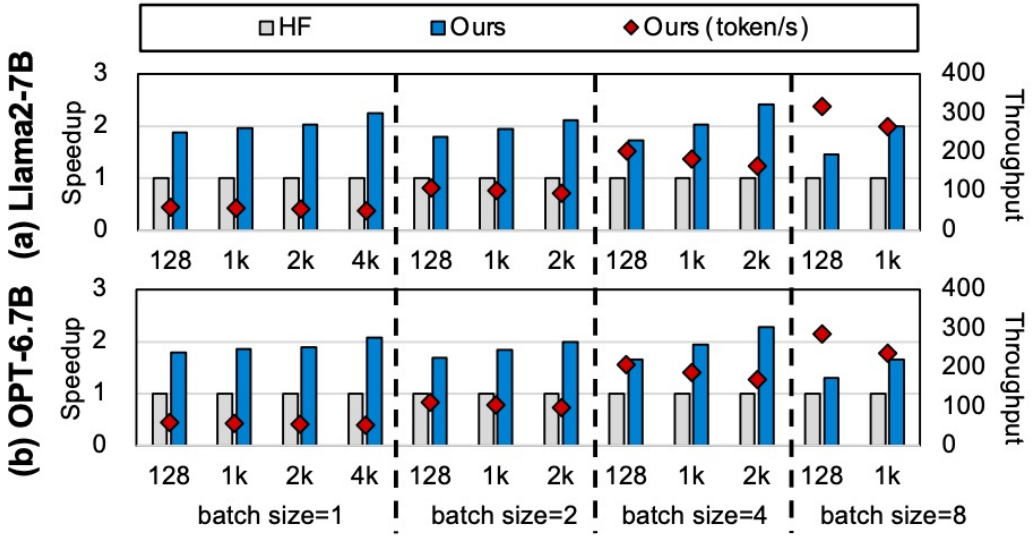
Results on AMD GPUs

Experiment Setup

AMD MI210 64GB	ROCm 5.7
AMD RX7900XTX 24GB	ROCm 5.6



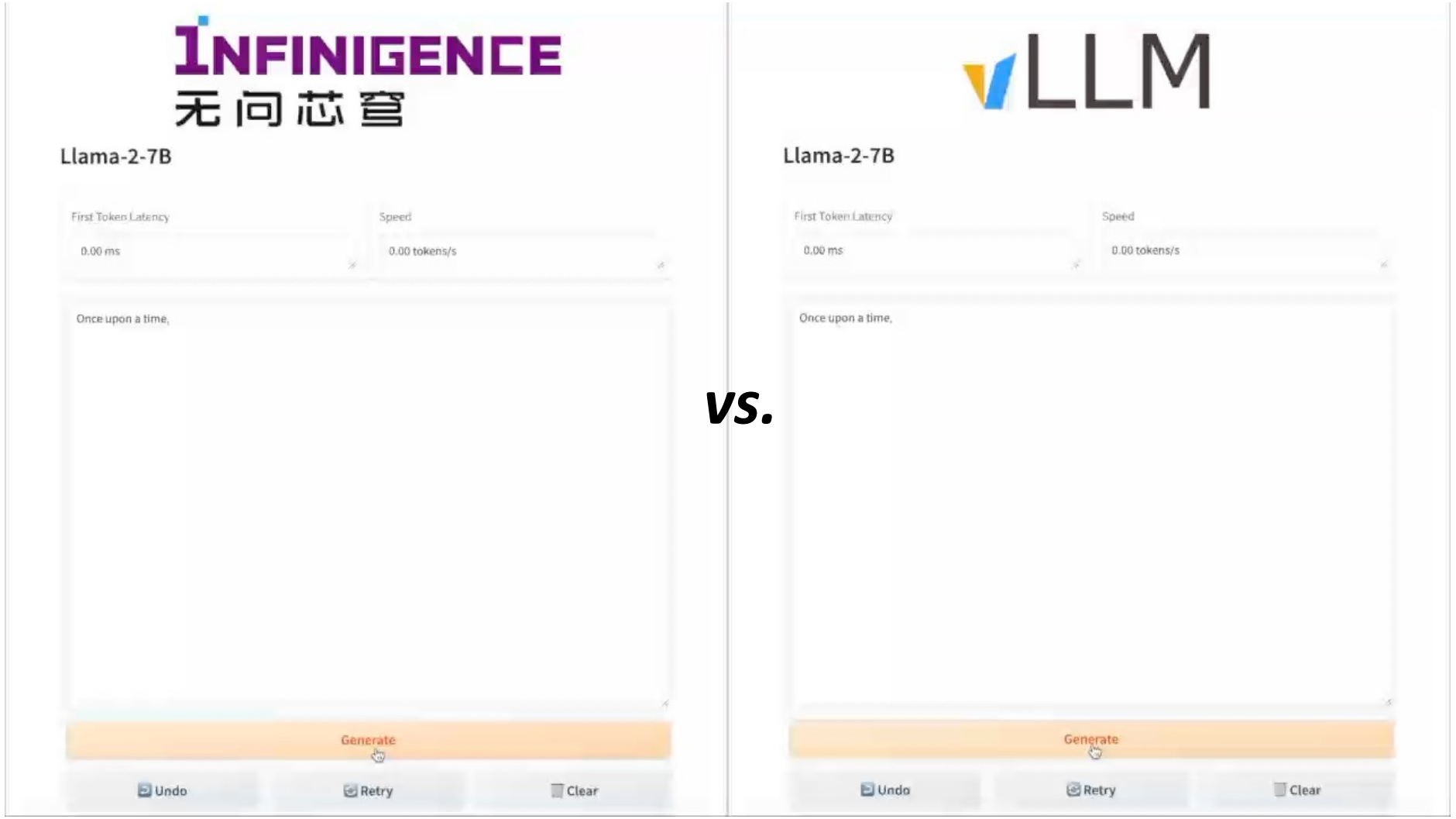
AMD MI210



AMD RX7900XTX

Up to 2.41x speedup on AMD RX7900XTX and 4.35x speedup on AMD MI210

Llama2-7B demo on AMD MI210



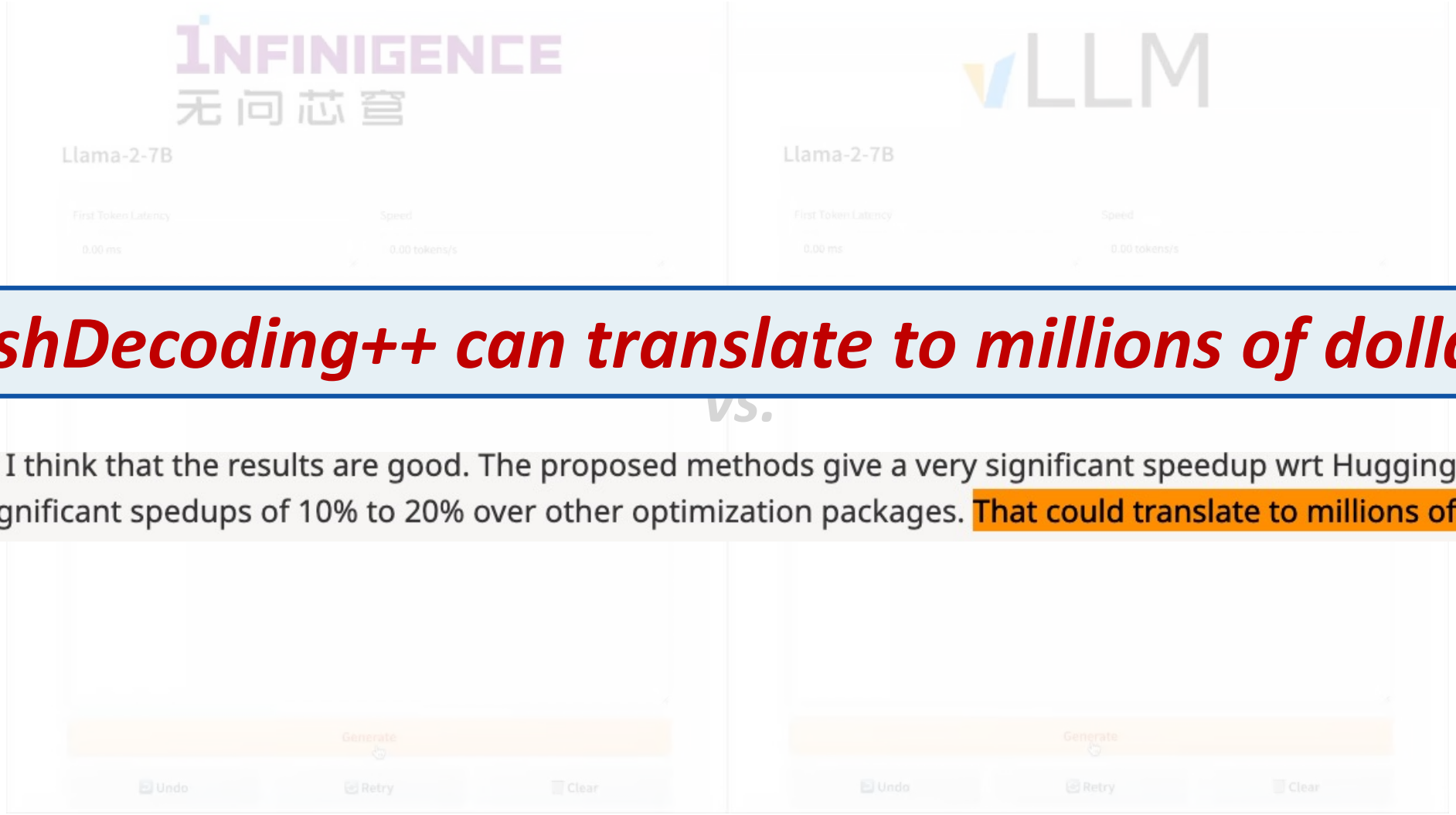
FlashDecoding++

vLLM framework

Summary

FlashDecoding++ can translate to millions of dollars.

Strengths: I think that the results are good. The proposed methods give a very significant speedup wrt HuggingFace, but even give significant spedups of 10% to 20% over other optimization packages. That could translate to millions of dollars.



FlashDecoding++ *vLLM framework*

Outline

- Backgrounds and Motivations
- Related Work
- Challenges and Techniques
  - Overview
  - Attention Operator
  - Linear Operator
  - Compilation Tuning
- Experiment Results
- Extension Works

Extension Works: Full Stack LLM Optimization

- Application -



Chat



Translate



Search

……

- Collaborators

- Algorithm -

- Low-bit quantization
- Sparse Attention/Activation
- ……

- Quantization [NIPS'23]
- Sparsification [under review]
- ……

4x Lower cost

- System -

- High performance operator
- Adaptive serving
- ……

- **FlashDecoding++ [MLSys'24]**
- Quantization GPU kernel [arXiv]
- ……

5x Lower cost

- Cluster -

- Unified schedule
- Offline and online tidal mixing
- ……

- Scheduling/Engineering
- Startup issues

10x Lower cost

- Hardware -

- GPU
- FPGA
- ……

- LLM FPGA acc [FPGA'24]
- Sparse Tensor Core [ICCAD'23]
- ……

10-100x Lower cost

Goal

10^4 x Lower Cost

Technique stack of LLMs

Our work for cost-efficient LLMs

Infinigence

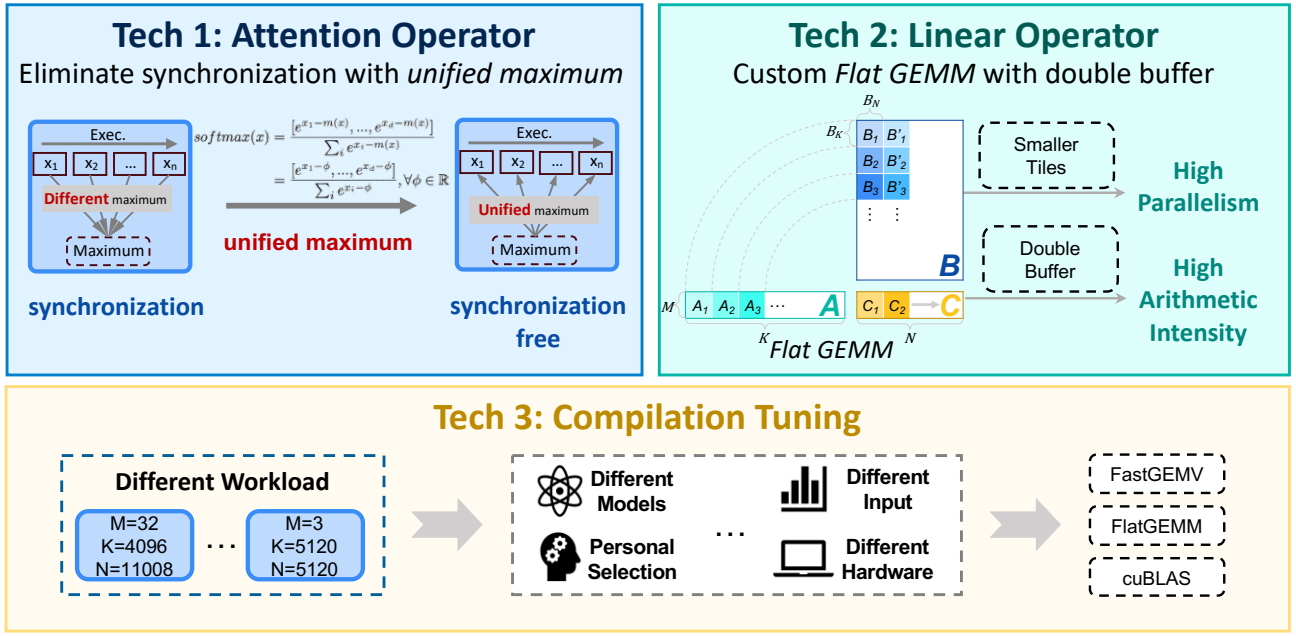
Infinigence: Infinity Computing Accelerates the Future of AI



Building M×N Middle Layers for LLMs with AI Infrastructure Capability

FlashDecoding++

https://arxiv.org/abs/2311.01282



Thank you
Q&A