

HPC Techniques For Efficient LLM Serving

SCA: HPC for Machine Learning

Juan Jose Olivera
November 2025



TurboTransformers: An Efficient GPU Serving System For Transformer Models

Jiarui Fang
Pattern Recognition Center, Wechat AI, Tencent Inc.
Beijing, China
jiarui.fang@tencent.com

Chengduo Zhao
Pattern Recognition Center, Wechat AI, Tencent Inc.
Beijing, China
chengduo.zha@tencent.com

Yang Yu
Pattern Recognition Center, Wechat AI, Tencent Inc.
Beijing, China
yang.yu@tencent.com

Jie Zhou
Pattern Recognition Center, Wechat AI, Tencent Inc.
Beijing, China
jie.zhou@tencent.com

CCS Concepts: Computing methodologies; Concurrent computing methodologies; Natural language generation; Parallel algorithms.

Keywords: Transformers, Deep Learning Runtime, Serving System, GPU

1 Introduction
The recent success of Natural Language Processing (NLP) field in recent years. Under the trend of NLP models becoming larger and larger, inference latency can be up to 100ms on dimensions of sequence lengths in parallel, therefore leads to better accuracy on long sequence lengths. However, serving large language models as web services in data centers equipped with GPUs are not easy. First, more computations introduced by transformer structures make the computation cost higher. Second, there are two main constraints of serving. Second, NLP tasks take in sequences of variable length. The variability of input dimensions brings a

DeepSpeed Inference: Enabling Efficient Inference of Transformer Models at Unprecedented Scale

Reza Yazdani Aminabadi, Samyam Rajbhandari, Minja Zhang, Ammar Ahmad Awan, Cheng Li, De Li, Elton Zheng, Jeff Rasley, Shaden Smith, Olatunji Rewase, Yuxing He
Microsoft Corporation
[\[reza.yazdani.reta.samymr.aminabadi.ammar.awan.cheng.li.de.li.li.tun.zheng.jeff.rasley.shaden.smith.olatunji.rewase.yuxhe\]@microsoft.com](mailto:[reza.yazdani.reta.samymr.aminabadi.ammar.awan.cheng.li.de.li.li.tun.zheng.jeff.rasley.shaden.smith.olatunji.rewase.yuxhe]@microsoft.com)

Abstract
The past several years have witnessed the success of transformer-based models, and their scale and application areas have been rapidly expanding. The underlying architecture of transformer models is increasingly diverse: the model size varies from tens of millions to billions of parameters, and the model characteristics differ due to the quantity introduced by the attention mechanism. The inference latency of a transformer model is latency-critical or throughput-oriented; the deployment hardware is memory and storage oriented, and the system requirements are memory and storage, etc. With such increasing diversity and complexity of transformer models, designing a highly efficient inference system for them is a challenging task.
In this paper, we present DeepSpeed Inference, a comprehensive system solution for transformer model inference to address the challenges mentioned above. The key idea is to decompose the inference of a multi-GPU inference solution to minimize latency while maintaining efficiency. The system can support two types of models when it is to aggregate GPU memory, and (2) a memory-aware inference solution for models that do not require memory access in addition to the GPU memory and compute to enable latency requirements, and thus the batch sizes used are generally small. For small batch sizes, inference latency of a model is mainly determined by the time required to move the model parameters from memory to registers. Meeting the latency requirements of a transformer model inference, therefore, is highly dependent on the algorithmic design of the inference system. Maximizing effective memory bandwidth at small batch sizes requires reading memory at near peak memory bandwidth for most of the time. The system also needs to read the majority of the model weights, while also minimizing kernel launch and data movement overhead of other operations like layernorm and softmax. The system also supports two types of kernels designed for training primarily focus on maximizing compute utilization at large batch sizes and are sub-optimal for latency-critical inference.
In addition, for large models, even the peak memory bandwidth of a single device may not be sufficient to meet inference

FLASHDECODING++: FASTER LARGE LANGUAGE MODEL INFERENCE ON GPUs

Ke Hong¹
Tsinghua University
& Infiniti-AI
Guhao Dai^{2*}
Shanghai Jiao Tong University
& Infiniti-AI
Xiaohui Li³
Peking University
Kangqi Chen⁴
Infiniti-AI
Aihua Dong⁵
Tsinghua University
Yu Wang⁶
Tsinghua University

¹daiгуahuо@bjtu.edu.cn, daiguhao@infini-ai.com, yu-wang@tsinghua.edu.cn

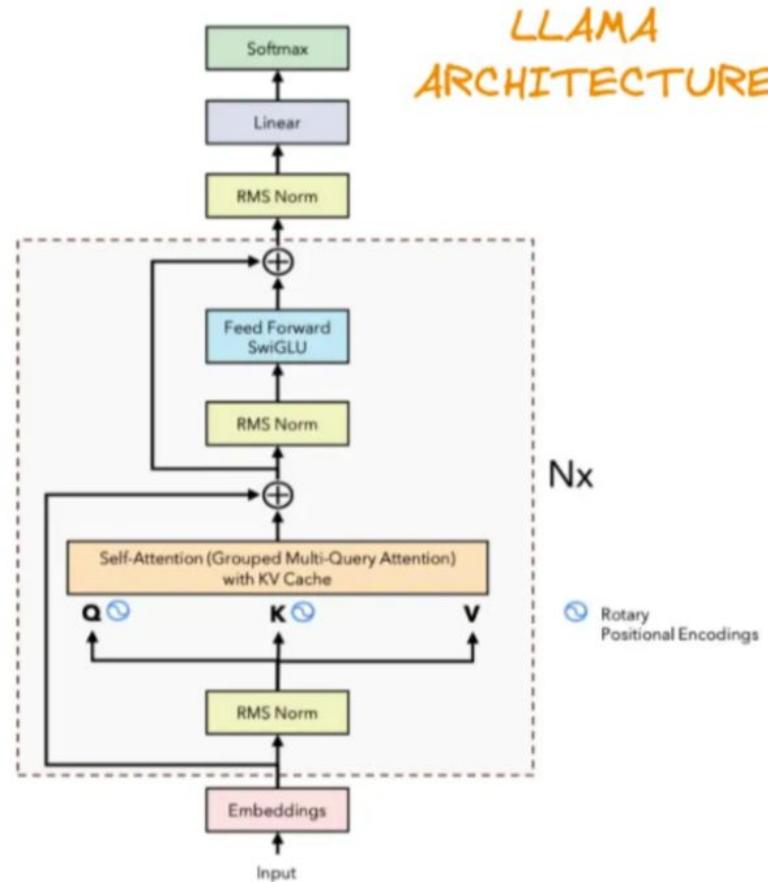
ABSTRACT
As the Large Language Model (LLM) becomes increasingly important in various domains, the performance of LLM inference is crucial to massive LLM applications. However, the following challenges still remain unsolved in accelerating LLM inference: (1) Synchronized partial softmax computation for multiple heads in parallel, which leads to significant performance overhead and redundant computation. The proposed FlashDecoding++ can achieve 1.5x performance improvement compared to the state-of-the-art. (2) The computation of GEMM in LLM inference is flat-

Agenda

- 1. LLM Inference 101**
- 2. Performance Metrics**
- 3. Motivation and Papers**
- 4. (Four) Optimization Techniques**
 - a. Problem**
 - b. Insights**
 - c. Solution**
- 5. Conclusions**
- 6. Questions**

LLM INFERENCE 101

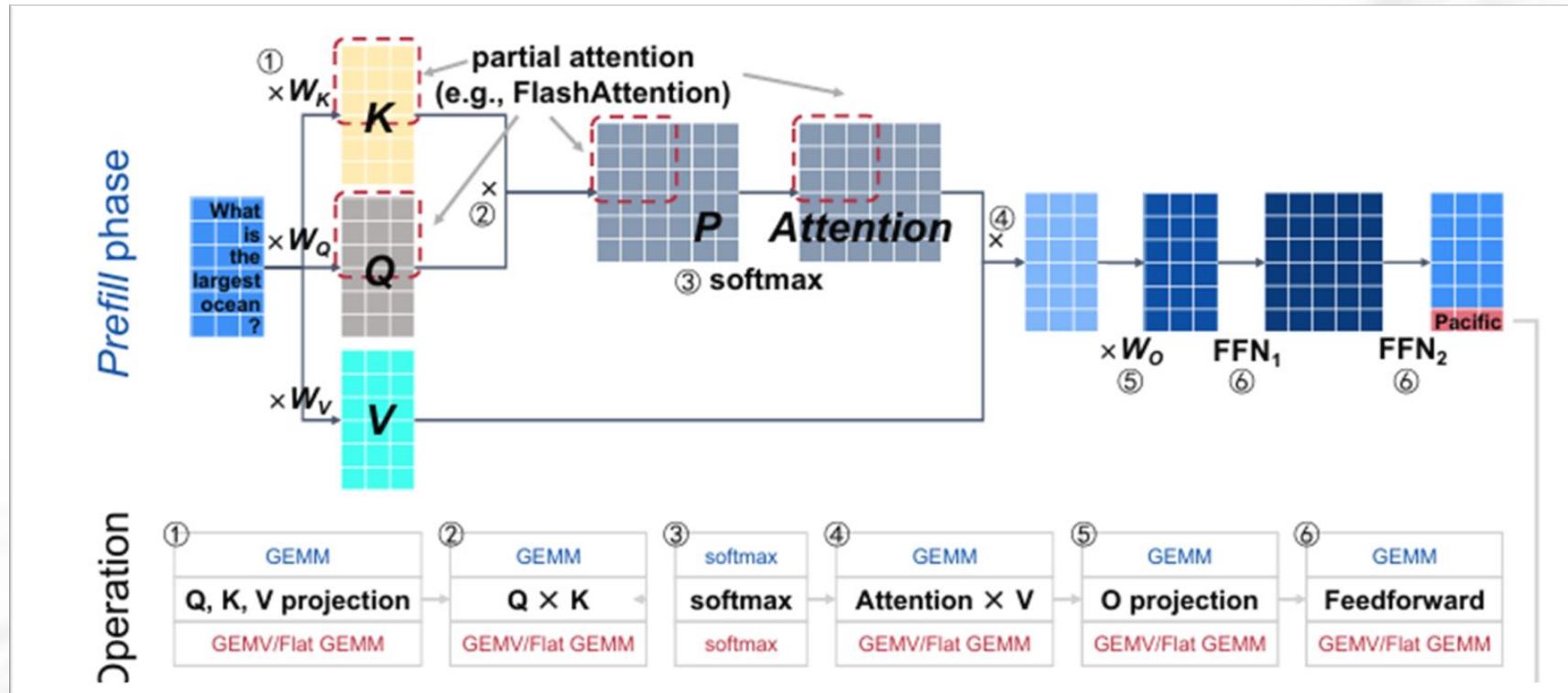
Modern GPT LLMs Architectures



LLM INFERENCE 101

Phase 1 of 2: Prefill

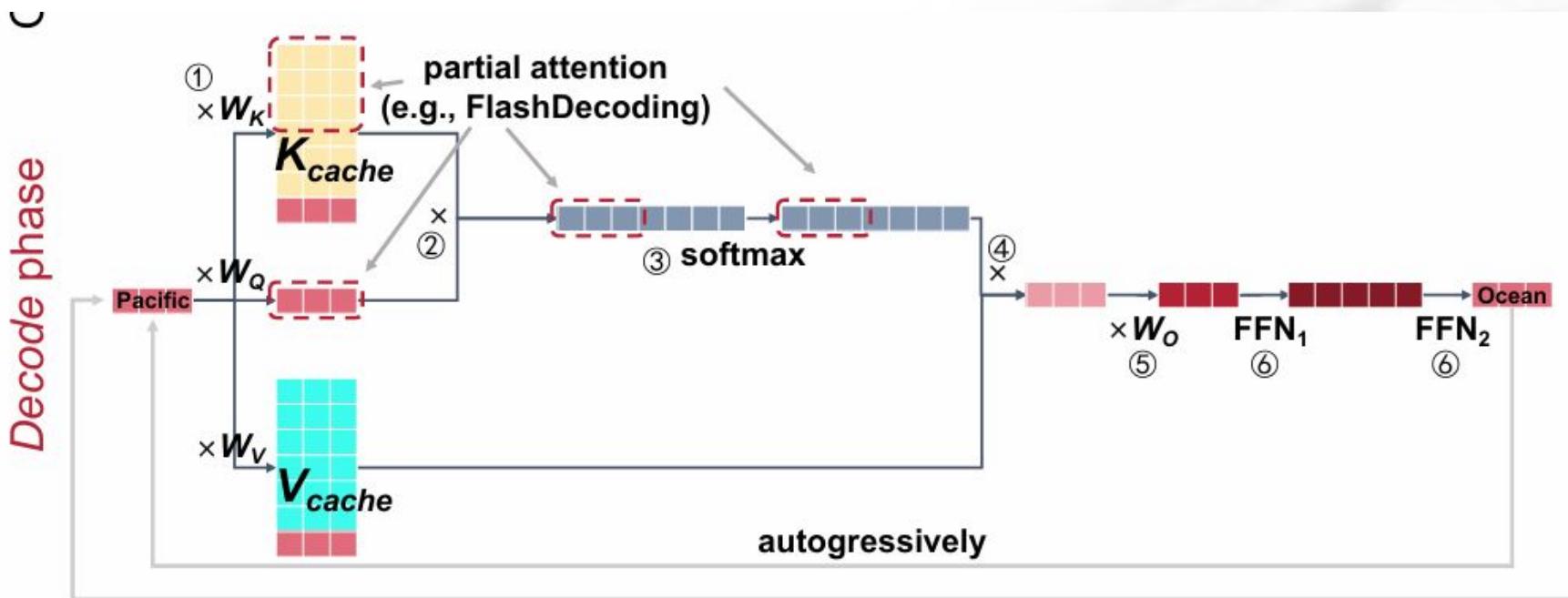
- 4 GEMMs + Softmax
- Big matrices



LLM INFERENCE 101

Phase 2 of 2: Decode

- 4 GEMMs + Softmax
- Slim matrices



Performance Metrics

Latency

End-to-end token generation time [seconds]

Throughput

Amount of tokens generated [tokens/s]

Compute Efficiency

FLOPs achieved against theoretical max [%]

Memory Efficiency

Footprint Size | No. mem allocations

Throughput Scalability

Throughput Increase Efficiency [%]

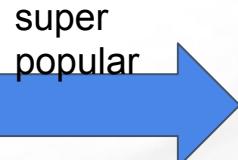
Model Scalability

Model Increase Efficiency [%]

Motivation: Per què?

**What is currently out there?
What is being researched?**

Motivation Use Case



Accelerating LLMs with llm.cpp on NVIDIA RTX Systems



Oct 02, 2024

+17 Like Discuss (0)

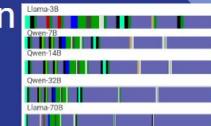


Thus, worth
analyzing



LLM Model Size Impact on
Application Performance

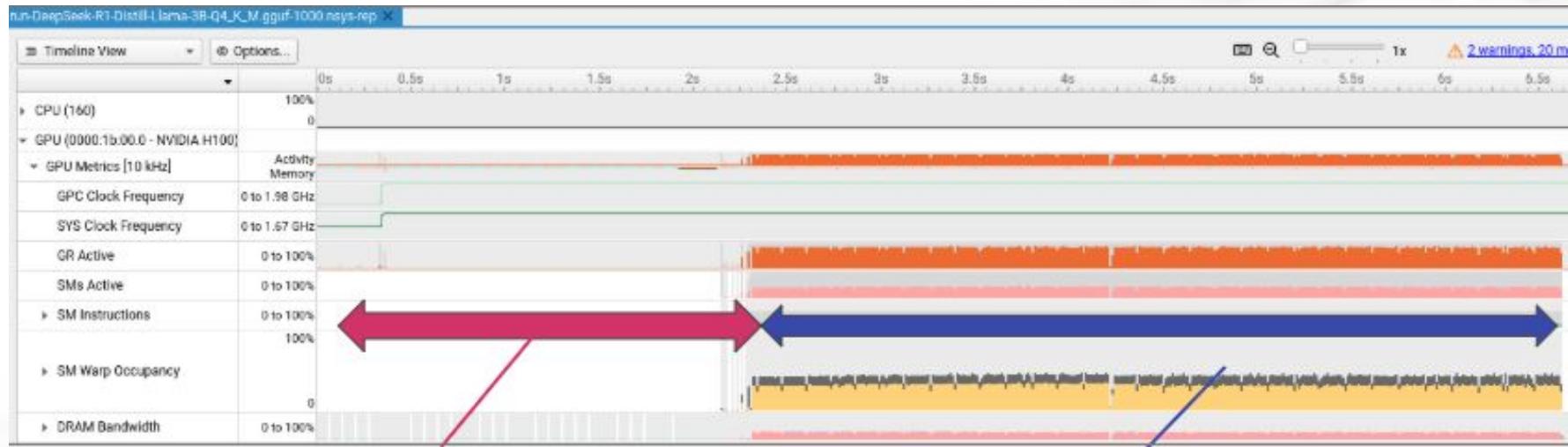
Juan Jose Olivera Loyola



PPTM Final Project - Professor Jesus Jose Labarta
TMIRI-HPC

Motivation Use Case

Overall execution of llama.cpp of 1000 tokens with Llama-3B

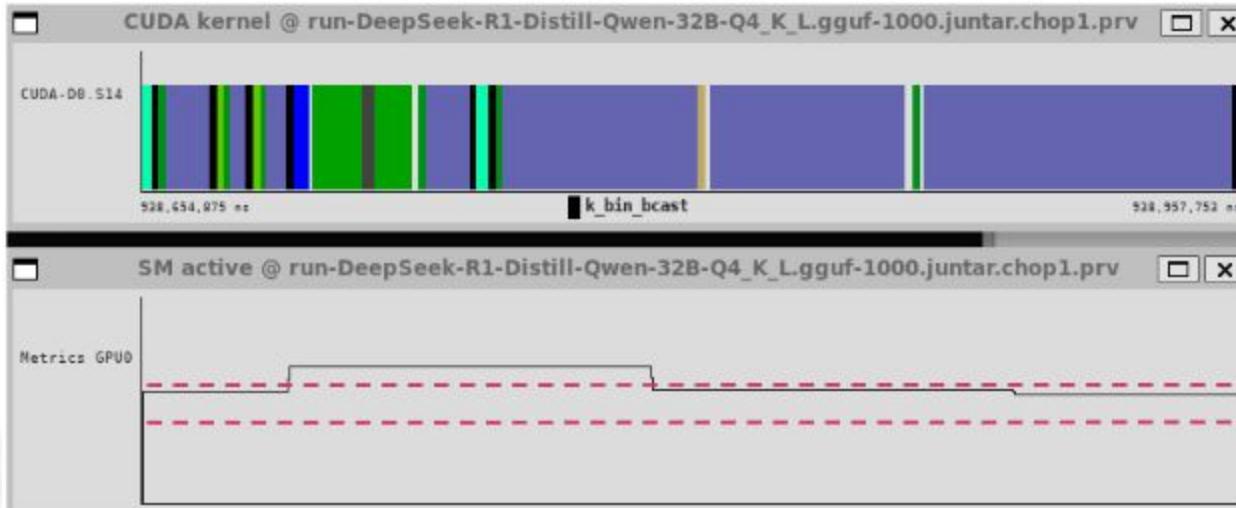


Initialization/Model Loading

Real Work: Token Prediction

Motivation Use Case

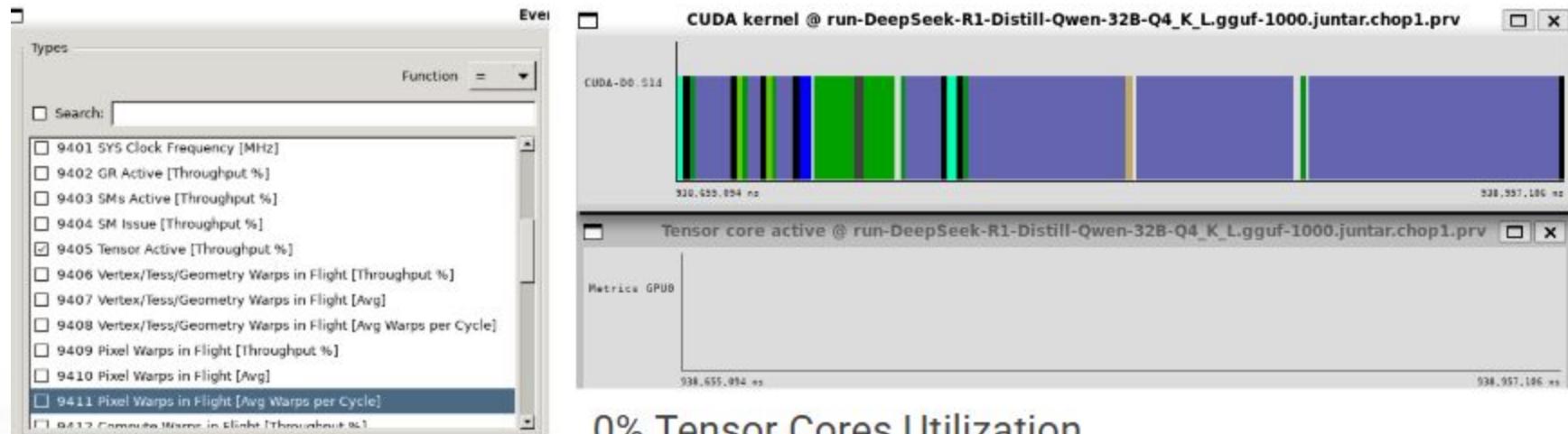
% Stream Multiprocessors (SM) Active



90%
72%

Motivation Use Case

NO Tensor Cores Active per Iteration



0% Tensor Cores Utilization

HPC CRIME!!!

Is it really?

Source: "LLM Model Size Impact on Application Performance"

Motivation & Papers

How implementations differ? What HPC techniques can be applied?



TurboTransformers: An Efficient GPU Serving System For Transformer Models

Jiarui Fang
Pattern Recognition Center, Wechat AI, Tencent Inc
Beijing, China
jiarufang@tencent.com

Yang Yu
Pattern Recognition Center, Wechat AI, Tencent Inc
Beijing, China
josephy@tencent.com

Chengduo Zhao
Pattern Recognition Center, Wechat AI, Tencent Inc
Beijing, China
florianzhao@tencent.com

Abstract
The transformer is the most critical algorithm innovation of the Natural Language Processing (NLP) field in recent years. Unlike the Recurrent Neural Network (RNN) models, transformer is able to process sequences of arbitrary lengths in parallel, thereby leads to better accuracy on long sequences. However, efficient deployments of them for online services in data centers equipped with GPUs are not easy. First, the computation cost by transformer is very high, which makes it more challenging to meet latency and throughput constraints of serving. Second, NLP tasks take in sentences of variable length. The variability of input dimensions brings a

CCS Concepts: • Computing methodologies: Concurrent computing methodologies; Natural language generation; Parallel algorithms.

Keywords: Transformers, Deep Learning Runtime, Serving System, GPU

TurboTransformers
Feb 2021
Multiple GPUs
Serving/Scheduling

DeepSpeed Inference: Enabling Efficient Inference of Transformer Models at Unprecedented Scale

Reza Yazdani Aminabadi, Samyam Rajbanshi, Minjia Zhang, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Jeff Rasley, Shaden Smith, Olantunji Ruwase, Yuxiong He
[\(yazdani.reza,samyam.zhang,ammar.awan,cheng.li.du.li,elton.zheng,jeff.rasley.shaden.smith.oluwase.yuxiong.he@microsoft.com\)](mailto:(yazdani.reza,samyam.zhang,ammar.awan,cheng.li.du.li,elton.zheng,jeff.rasley.shaden.smith.oluwase.yuxiong.he@microsoft.com)

Abstract
The past several years have witnessed the success of transformer-based models, and their scale and application scenarios continue to grow aggressively. The current landscape of transformer inference is dominated by large-scale models, with the number of parameters increasing dramatically with the largest being of hundred-billion parameters. The main characteristic of inference is the sparse nature of the model weights. Exploiting the sparse structure can be latency-critical or throughput-oriented; the deployment hardware can be latency-critical or throughput-oriented. Maximizing effective memory bandwidth at small batch sizes is challenging due to the high memory access overhead for fully-connected (or linear) layers, which contain the majority of the model weights, while also minimizing kernel launch and data movement overhead of other operators like layernorm and softmax. The GEMM implementations and other kernels required for inference primarily focus on maximizing compute utilization at very large batch sizes and are sub-optimal for latency-critical inference.

In this paper, we present DeepSpeed Inference, a comprehensive system solution for transformer model inference to address the above-mentioned challenges. DeepSpeed Inference consists of (1) a multi-stage inference pipeline for latency efficiency, maximizing the throughput of both dense and sparse transformer models when they fit in aggregate GPU memory, and (2) a heterogeneous deployment strategy that leverages both GPU memory in addition to the GPU memory and compute to enable

DeepSpeed
Jul 2022
Multiple GPUs
GPUs Parallelism

FLASHDECODING++: FASTER LARGE LANGUAGE MODEL INFERENCE ON GPUs

Ke Hong¹,
Tsinghua University
& Infinigence-AI

Qihai Mao²,
Tsinghua University
& Infinigence-AI

Kangdi Chen³,
Infinigence-AI

Xisheng Li⁴,
Peking University

Yuhan Dong⁵,
Tsinghua University

Yu Wang⁶,
Tsinghua University

[daiguohao@infini-ai.com](mailto:daiguohao@sjtu.edu.cn), [yu-wang@tsinghua.edu.cn](mailto:daiguohao@infini-ai.com)

ABSTRACT

As the Large Language Model (LLM) becomes increasingly important in various domains, the performance of LLM inference has become a key concern. There are two major challenges when it comes to accelerating LLM inference: (1) Synchronized partial softmax update. The softmax operation requires a synchronized update operation among each partial softmax result, leading to ~20% overheads for the attention computation in LLMs. (2) Under-utilized computation of GEMM. The shape of matrices performing GEMM in LLM inference is flat, leading to significant computation waste when the computation size is small.

FlashDecoding++
Nov 2023
Single GPU
Computation Efficiency

T1: Flat GEMM Double Buffering

FlashDecoding++, DeepSpeed

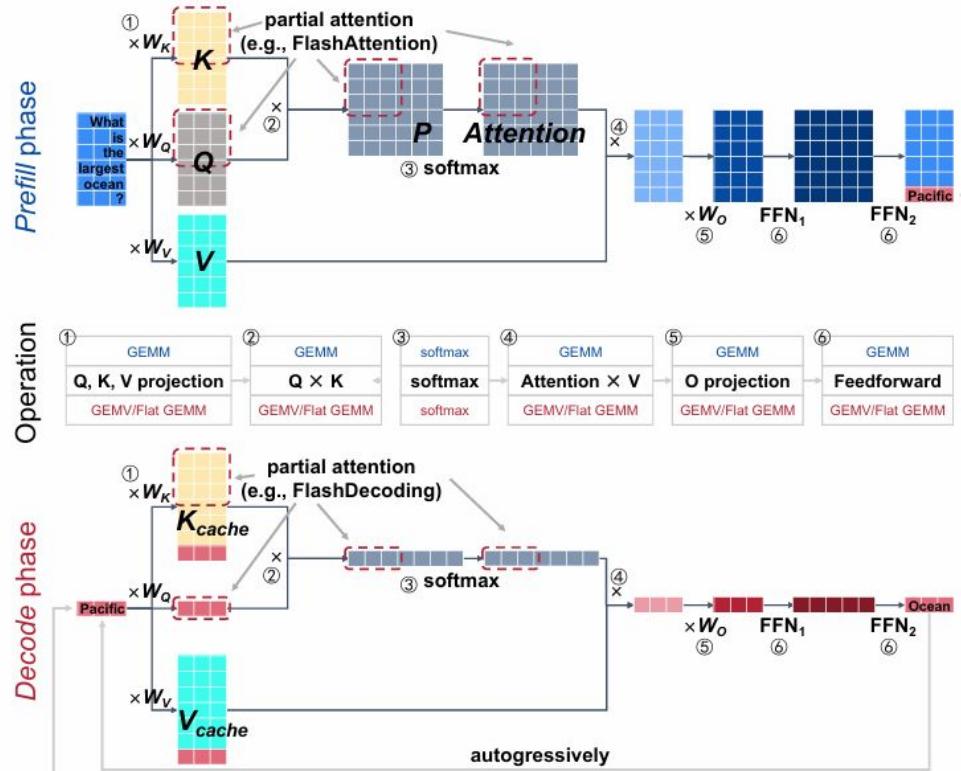
Problem	cuBLAS/cuTCLASS TensorCore libraries badly optimized for skinny matrices (few tokens). Small M
Insights	For small N, GEMM is parallelism-bounded (# SMs) For big N, GEMM is memory-bounded.
Solution	Flat GEMM custom implementation with tiling and double buffering Execution of different GEMM implementations based on data nature.

T1: Flat GEMM Double Buffering

FlashDecoding++, DeepSpeed

PROBLEM

Flat matrices
in decode
phase



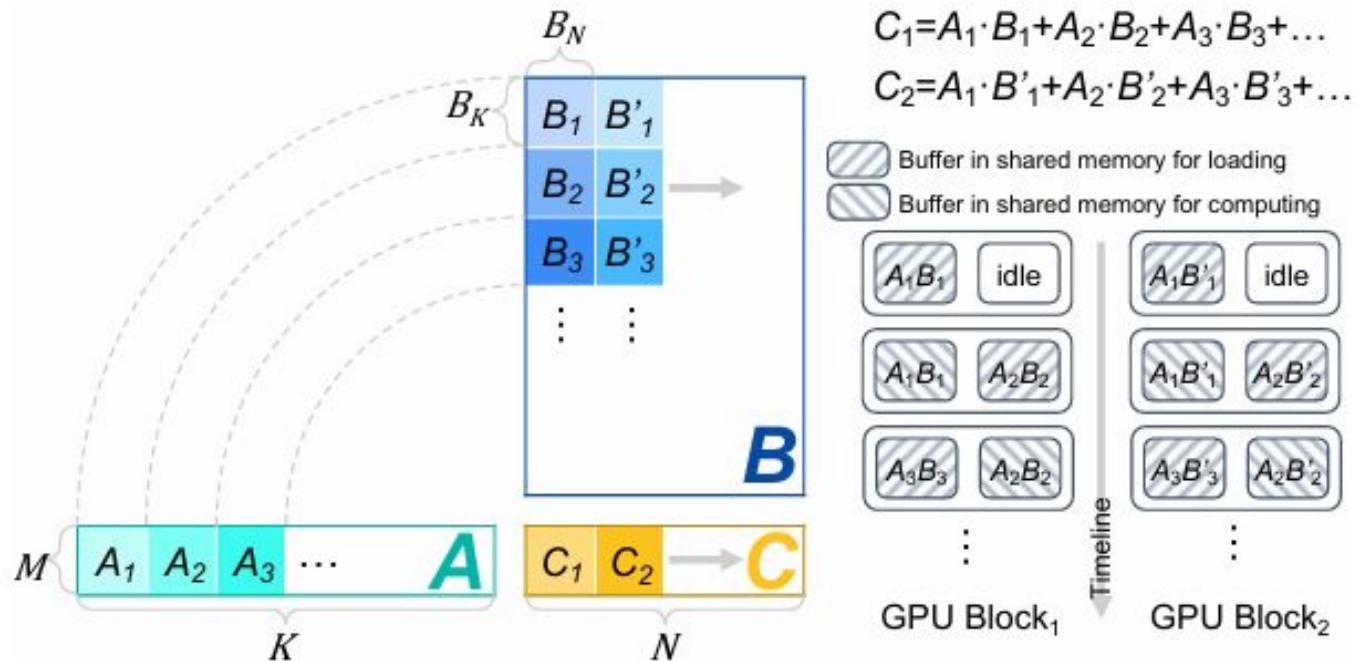
Source: FlashDecoding++: Faster large language model inference on GPU”

T1: Flat GEMM Double Buffering

FlashDecoding++, DeepSpeed

S1: Flat
GEMM

-Tiles
-Double
Buffer



T1: Flat GEMM Double Buffering

FlashDecoding++, DeepSpeed

S2: Different GMMs

- cuBLAS
- FastGEMV
- FlatGEMM

	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

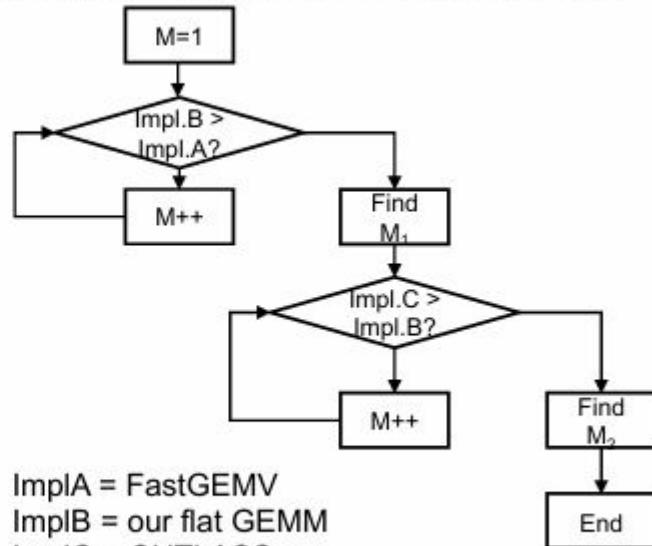
B: Batch size

SeqLen: Input sequence length

Only 4 shapes!

(a) Different shapes of GEMMs in LLM

For a certain LLM, traverse four [N, K] selections



(b) Decision flow

T1: Flat GEMM Double Buffering

FlashDecoding++, DeepSpeed

T2: Memory Allocation

TurboTransformers

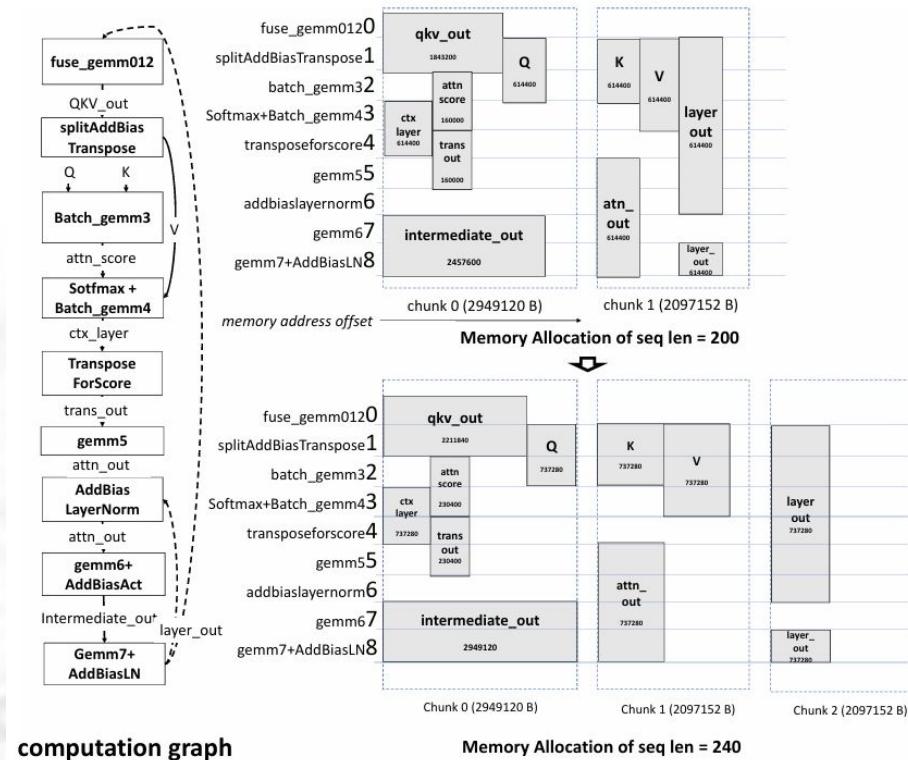
Problem	Few # Allocations → Allocating big memory pool → Big Memory Footprint
Insights	Can allocate and reuse Per-Chunk Computation Graph (CG) → Predict Chunk Life
Solution	Sequence-length aware allocator algorithm Finds gaps in memory pool to fit in chunks Computes offsets based on lifetimes from CG to reuse memory

T2: Memory Allocation

TurboTransformers

S: Seq-Aware Chunk Allocation

- cuBLAS
- FastGEMV
- FlatGEMM



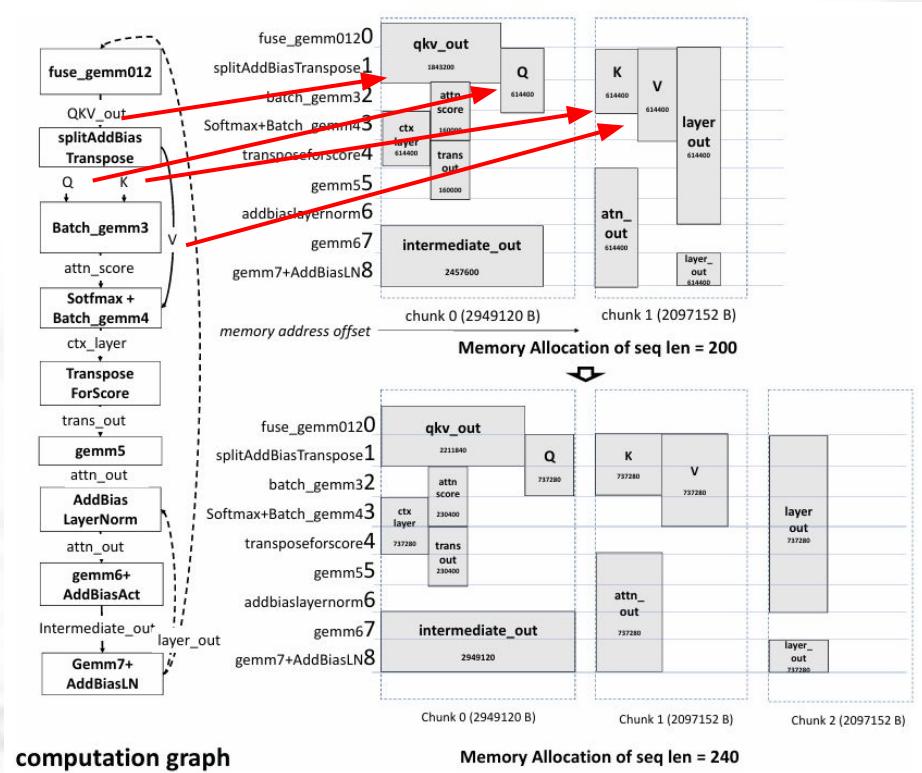
Source: TurboTransformers: an efficient GPU serving system for transformer models

T2: Memory Allocation

TurboTransformers

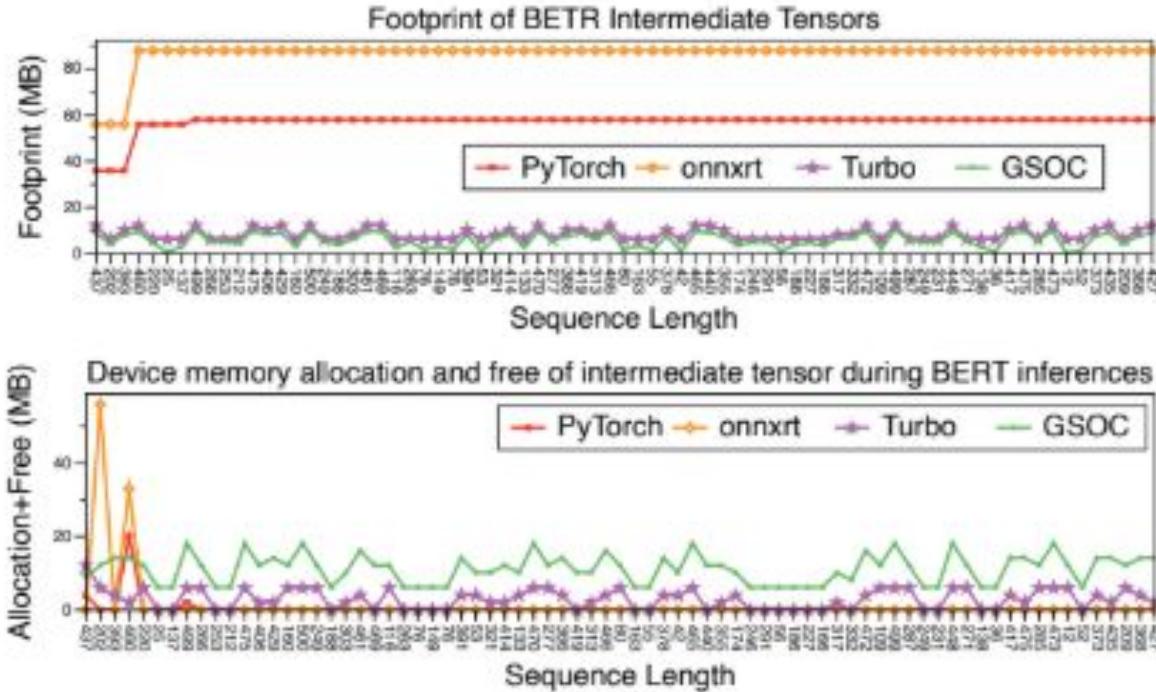
S: Seq-Aware Chunk Allocation

-cuBLAS
-FastGEMV
-FlatGEMM



T2: Memory Allocation

TurboTransformers



T3: Kernel Fussion

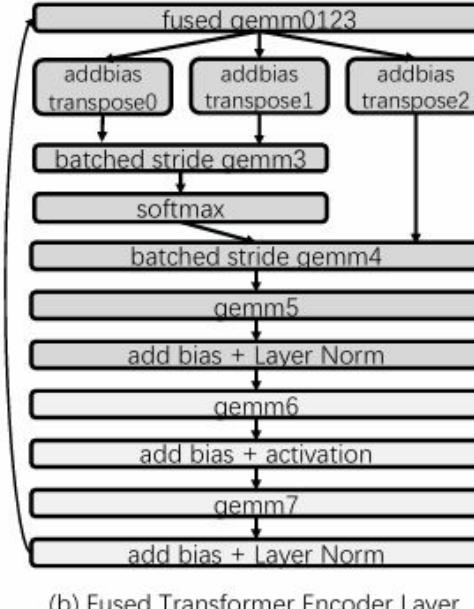
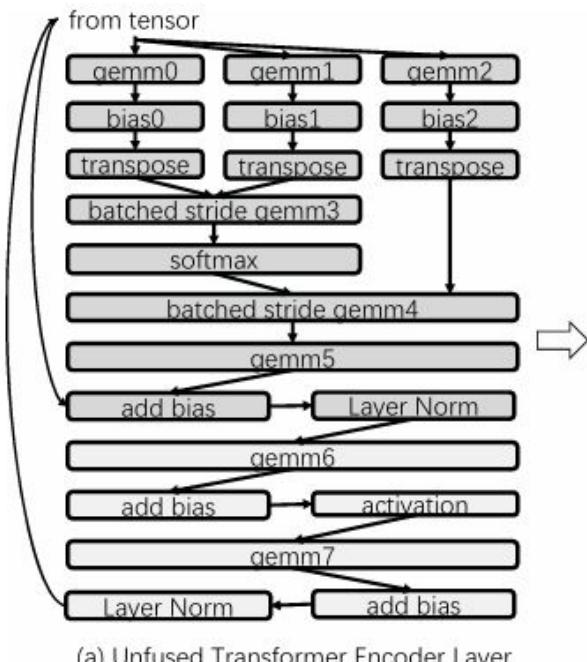
TurboTransformers, DeepSpeed

Problem	Kernel Invocation Adds Latency Overhead Global Memory Data-Transfers Between Kernels
Insights	Model Training Requires Activation Values Model Inference DOES NOT Require Activations
Solution	<p>Fuse many operations (multiple kernels) into same kernel.</p> <p>Deep-Fusion: Tile multiple related kernels into single Tiled Kernel.</p> <p>Use CUDA Graphs</p>

T3: Kernel Fussion

TurboTransformers, DeepSpeed

21 kernels
15 length
compute path



13 kernels
11 length
compute path

T4: Model Parallelism

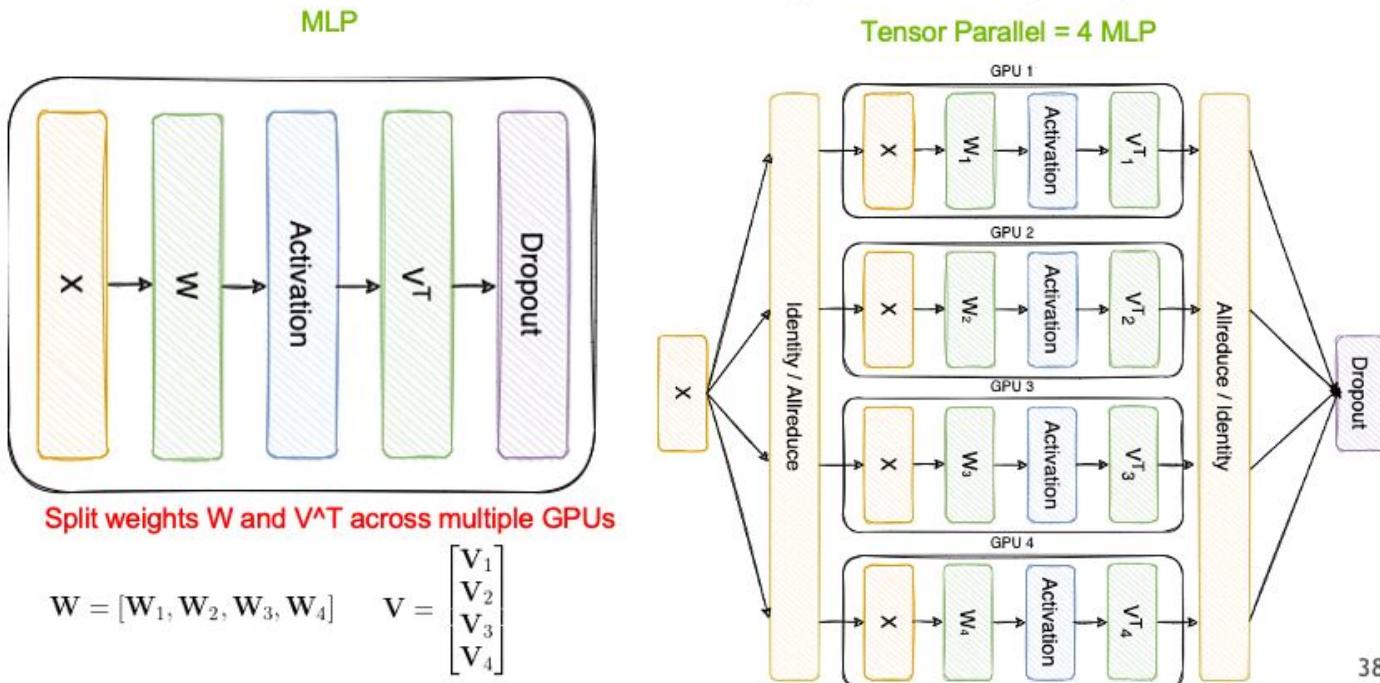
DeepSpeed

Problem	Big LLM Models (+100B params) do not fit in a single or multiple GPUs
Insights	GPUs connected in same node have high BW. Can use more nodes with lower BW.
Solution	Tensor parallelism (layers horizontal slicing) for GPUs same node Pipeline parallelism (layers vertical slicing) for more nodes.

T4: Model Parallelism

DeepSpeed

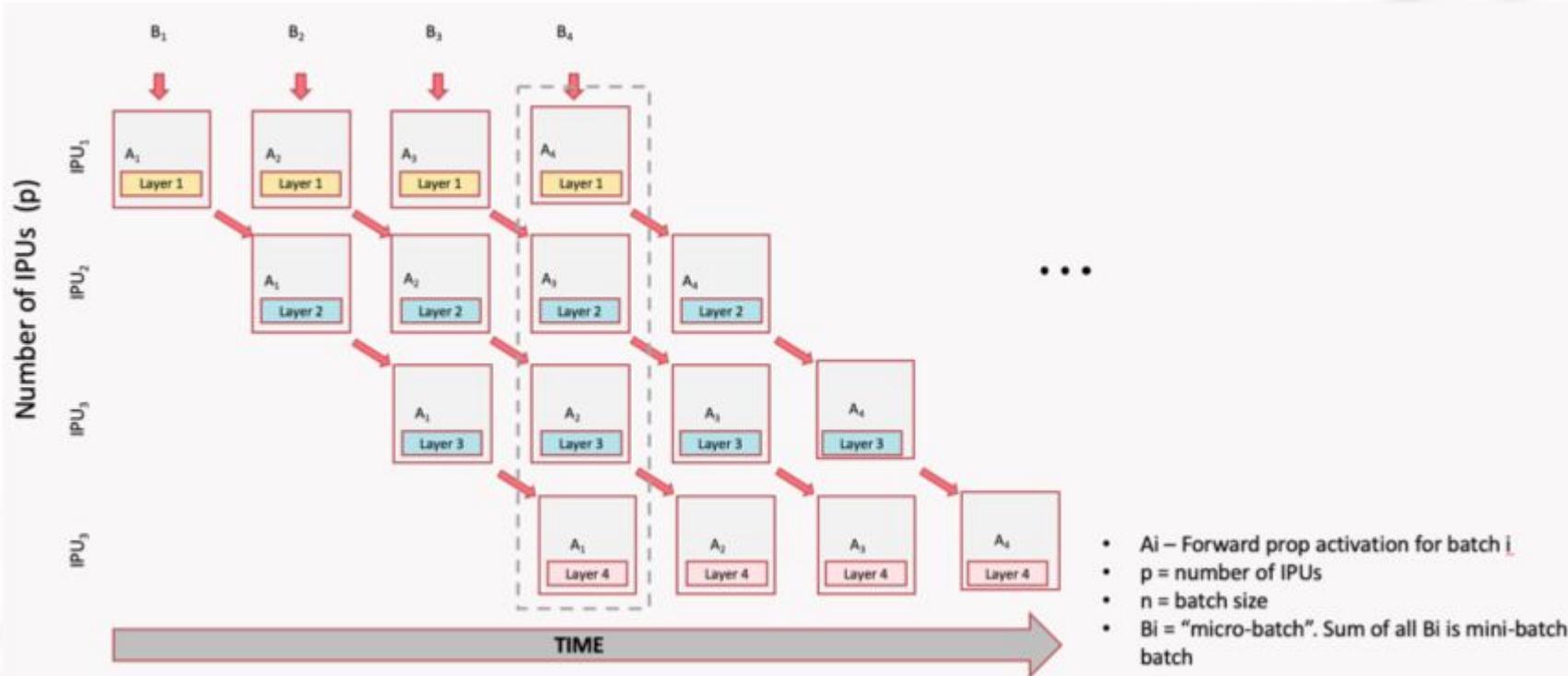
Tensor Parallelism (Intra-Layer)



T4: Model Parallelism

DeepSpeed

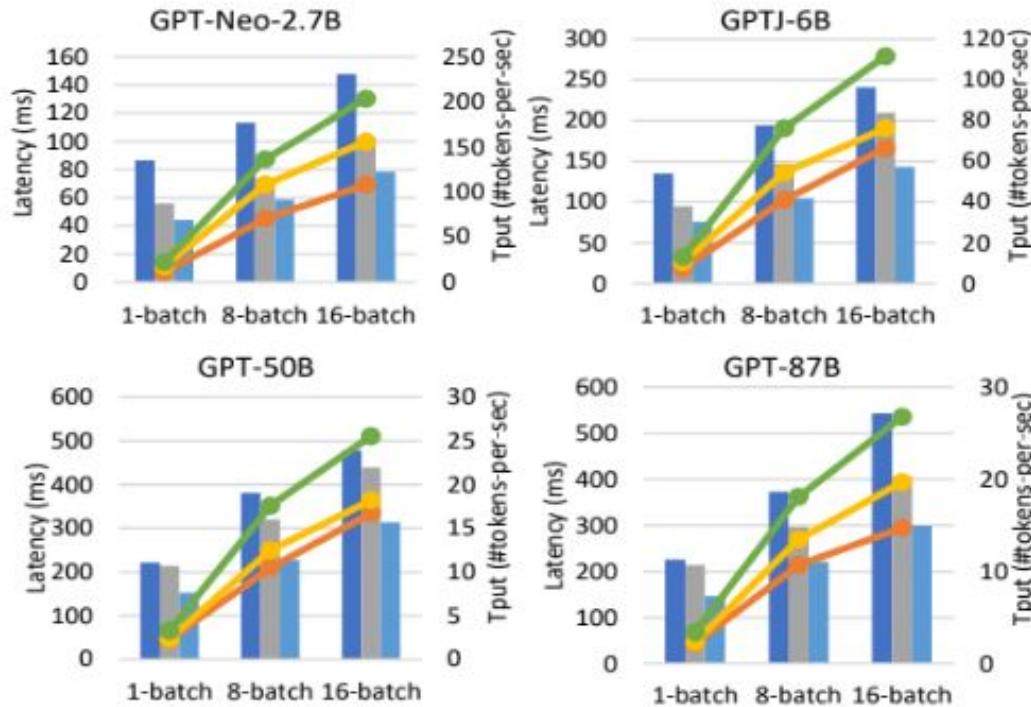
Pipeline Parallelism



T4: Model Parallelism

DeepSpeed

Model Scaling Results



Conclusions

- LLM Inference is a high-impact, compute-heavy task, many constraints → Interesting Problem.
- Multiple Techniques, Different Types.
 - Memory and Compute
 - Single GPU and Multi GPU performance.
- Enormous Effort To Put Everything Together
 - Heterogeneous computing
 - Evolving techniques
 - Variety of models and workflows.
 - Etc

References

- K. Hong et al., “FlashDecoding++: Faster large language model inference on GPUs,” arXiv.org, Nov. 02, 2023. <https://arxiv.org/abs/2311.01282>
- R. Y. Aminabadi et al., “DeepSpeed Inference: Enabling Efficient Inference of Transformer Models at Unprecedented Scale,” arXiv.org, Jun. 30, 2022. <https://arxiv.org/abs/2207.00032>
- J. Fang, Y. Yu, C. Zhao, and J. Zhou, “TurboTransformers: an efficient GPU serving system for transformer models,” arXiv.org, Oct. 09, 2020. <https://arxiv.org/abs/2010.05680>
- J.J.Olivera (2025) “LLM Model Size Impact on Application Performance”. UPC MIRI PPTM Presentation.

THANK YOU
FOR NOT
SLEEPING!

QUESTIONS??

