Characterizing Communication Patterns in Distributed Large Language Model Inference

Presented at Hot Interconnects '25

Lang Xu, Kaushik Kandadi Suresh, Quentin Anthony,

Nawras Alnaasan, and Dhabaleswar K (DK) Panda

Department of Computer Science and Engineering,

The Ohio State University, Columbus, Ohio, USA

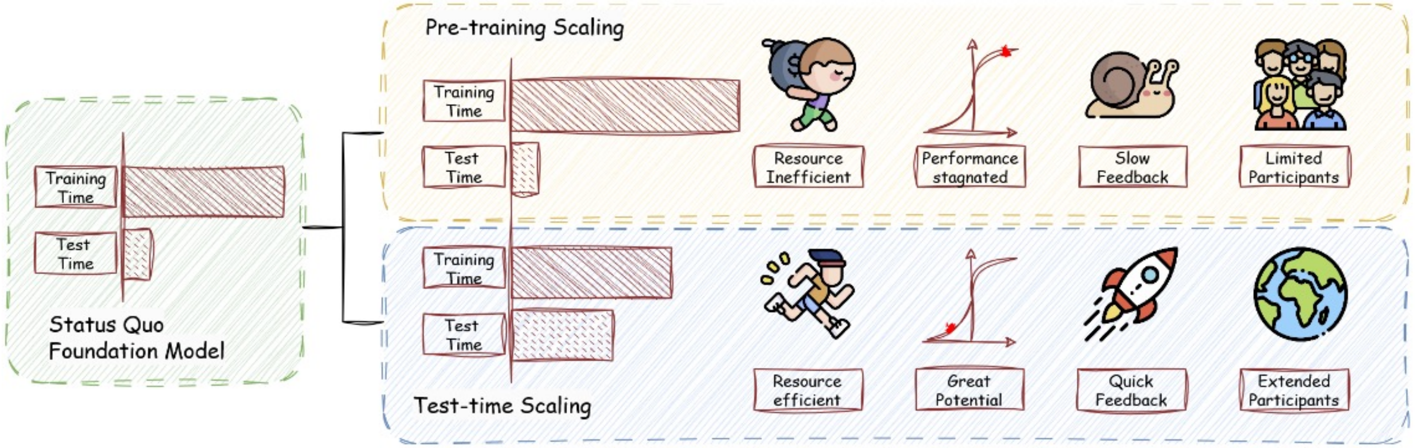
Presentation Outline

- **Introduction and Motivation**
- Problem statements
- Analytical Model
- Analysis and Performance Characterization
- Conclusion

Large Language Model Inference

- Inference: The process of using a pre-trained Large Language Model to generate text or predict on a given input (prompt)
- Emergent capabilities comes with scaling inference-time compute
  - Reasoning, Decision Making, Coding
  - Reinforcement Learning (GRPO, DPO)
  - Better Models (DeepSeek-R1, Gemini 2.5 Pro, OpenAI-o3)

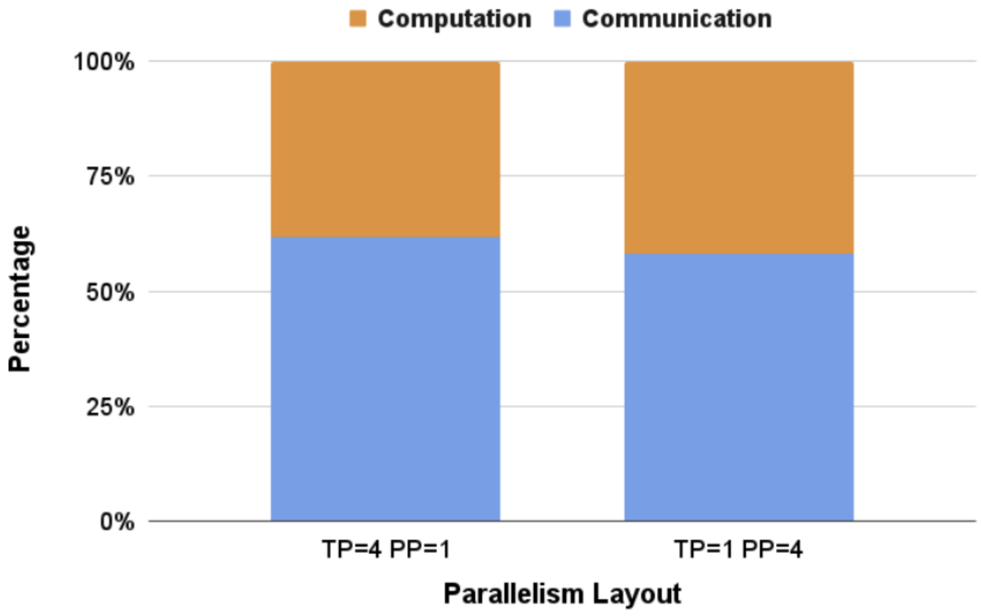
Complex Large Language Model capability emerges with computation resources allocated to **Inference!**

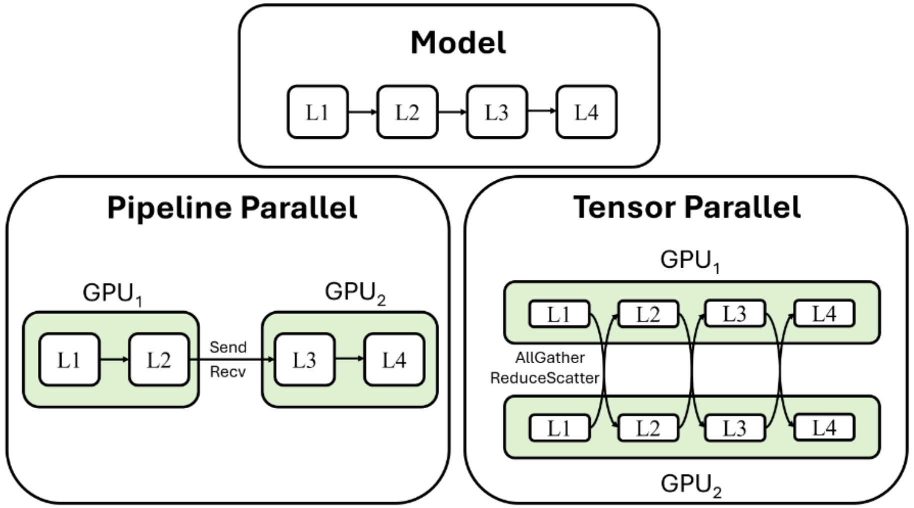


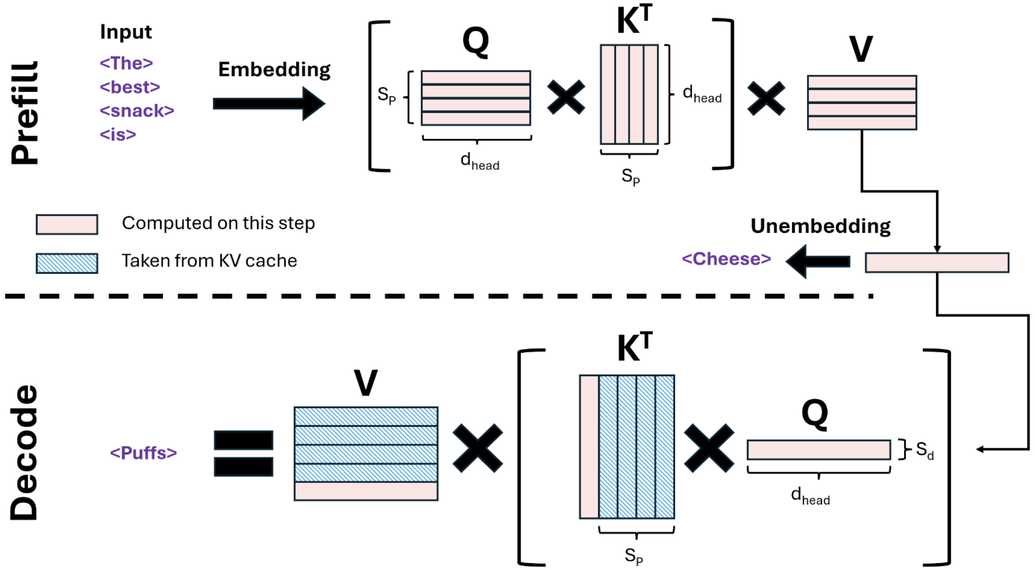
Courtesy: “A Survey on Test-Time Scaling in Large Language Models: What, How, Where, and How Well?“ https://arxiv.org/abs/2503.24235

Large Language Model Inference

- Similar to Pre-Training, Inferencing has similar challenges:
  - Multi-GPU deployment (Tensor/Pipeline Parallelism)
  - Communication overhead
- Prefill-Decode Stages (compute-bound vs memory-bound)
  - Unique communication pattern
- Service-level objectives (SLOs)
  - Latency, time-to-first-token (TTFT), time-per-output-token(TPOT)







Presentation Outline

- Introduction and Motivation
- **Problems statements**
- Analytical Model
- Analysis and Performance Characterization
- Conclusion

Problem Statements

- What are the predominant types, volumes and patterns of communication during multi-GPU inferencing?
- Can we develop analytical models to predict such communication with certain parameters? Parallelism degree, model architecture and such?
- What is the impact of communication patterns when it comes to SLOs?
- Given a set of resources, what is the comparative impact of different parallelism layout?

Presentation Outline

- Introduction and Motivation
- Problem statements
- **Analytical Model**
- Analysis and Performance Characterization
- Conclusion

Analytical Model

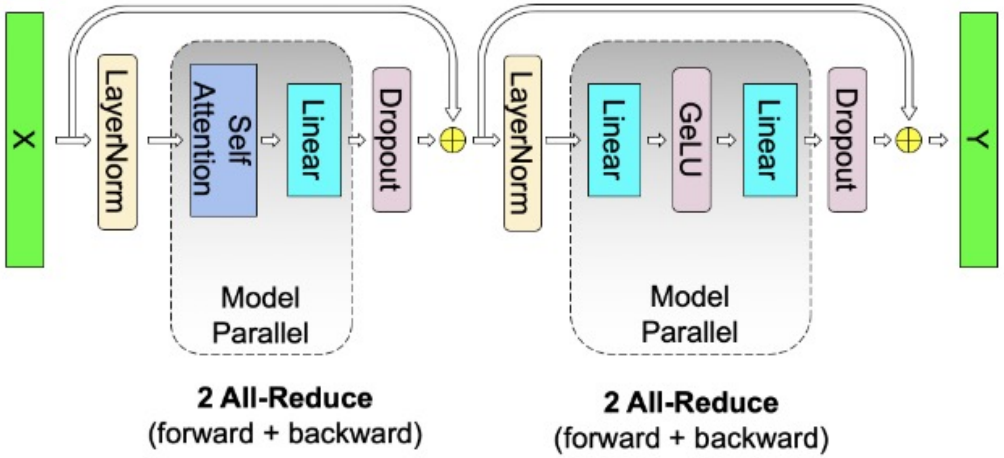
- Modeling communication volume across different parallelism layout.
- Covering **Tensor/Pipeline/Hybrid Parallelism**
- vLLM Framework + Llama-based dense transformer architecture

h	Hidden dimension size	t	Tensor-parallel size
L	Number of transformer layers	p	Pipeline-parallel size
b	Bytes per element	v	Vocabulary size
S_p	Prefill sequence-length	S_d	Decode sequence-length
a	Number of attention heads	d_{head}	Head dimension

Analytical Model – Tensor Parallelism

- **Tensor Parallelism**: Distributed matrix multiplication across GPUs
- Row-Parallel linear layer: input partitioned along 1st dimension, weight along 2nd dimension
- One All-reduce synchronization per layer
- Each Transformer block:
  - MLP down-projection
  - Attention output projection
  - A total of **2 All-reduce** at message size of **h** elements
- **1 All-reduce** at Embedding layer per token
- **1 Gather** at final logit computation per generated token

h	Hidden dimension size	t	Tensor-parallel size
L	Number of transformer layers	p	Pipeline-parallel size
b	Bytes per element	v	Vocabulary size
S_p	Prefill sequence-length	S_d	Decode sequence-length
a	Number of attention heads	d_{head}	Head dimension



Courtesy: "Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism"
http://arxiv.org/abs/1909.08053

$$V_{tp} = (2L+1)\times(S_p+S_d-1)\times h\times b\times 2\left(\frac{t-1}{t}\right)+S_d\times\frac{v}{t}\times b$$

Analytical Model – Pipeline Parallelism

- **Pipeline Parallelism**: Places a subset of transformer layers among GPUs, passing activations using **P2P send & receive**
- Prefill: each pipeline stage forwards $2S_p\, hb$ bytes
- Decode: $2hb$ bytes per generated token
- Number of links: p-1
- 1^{st} pipeline rank receives no input, the last pipeline rank produces no intermediate output

h	Hidden dimension size	t	Tensor-parallel size
L	Number of transformer layers	p	Pipeline-parallel size
b	Bytes per element	v	Vocabulary size
S_p	Prefill sequence-length	S_d	Decode sequence-length
a	Number of attention heads	d_{head}	Head dimension

$$V_{pp} = (p-1) \times 2 \times (S_p + S_d - 1) \times h \times b$$

Analytical Model – Hybrid Parallelism

- **Hybrid Parallelism**: Combining Tensor & Pipeline Parallelism
- Great for Multi-Node setup as we want to minimize inter-node communication overhead
- Additional All-gather to redistribute activations among tensor parallel workers
- For the 1st pipeline rank, we have an additional embedding All-reduce volume of $(S_P+S_d-1)*h*b$ bytes

$$V_{hybrid} = V_{allreduce} + V_{allgather} + V_{gather} + V_{p2p}$$

$$V_{allreduce} = \frac{2L}{p} \times (S_p + S_d - 1) \times h \times b \times 2\left(\frac{t-1}{t}\right)$$

All-reduce volume reduced by p for pipeline parallel

$$V_{allgather} = 2(p-1) \times (S_p + S_d - 1) \times h \times b \times \left(\frac{t-1}{t}\right)$$

$$V_{gather} = S_d \times \frac{v}{t} \times b$$

$$V_{p2p} = (p-1) \times 2 \times (S_p + S_d - 1) \times \frac{h}{t} \times b$$

h	Hidden dimension size	t	Tensor-parallel size
L	Number of transformer layers	p	Pipeline-parallel size
b	Bytes per element	v	Vocabulary size
S_p	Prefill sequence-length	S_d	Decode sequence-length
a	Number of attention heads	d_{head}	Head dimension

Presentation Outline

- Introduction and Motivation
- Problem statements
- Analytical Model
- **Analysis and Performance Characterization**
- Conclusion

Experimental Setup

Hardware:

- OSC Cardinal system
  - Intel Xeon Platinum 8470 (52 cores, 2 GHz)
  - 4 NVIDIA H100 (NVLink, 94 GB HBM2e)
  - InfiniBand NDR400 (4 NICs/Node)

Software packages:

- PyTorch 2.6 (torch.compile off + no custom allreduce)
- vLLM 0.8.5.post1 V0 engine
- NCCL 2.21.5

Models:

- Llama-3.2-3B (h=3072, L=28, v=128256, Dense)
- Llama-3.1-8B (h=4096, L=32, v=128256, Dense)
- Llama-2-13B (h=5120, L=40, v=32000, Dense)

Serving Configuration: Single Request, Batch Size 1

Profiling: PyTorch Profiler + vLLM RESTful observability API

Performance Analysis: Message Size and Frequency

Model	TP Size	Prefill Stage			Decode Stage		
		Collective	Count	Shape	Collective	Count	Shape
Llama-3.1-8B $S_p = 128$ $S_d = 128$	2	Allreduce Gather	65 1	[128,4096] [64128]	Allreduce Gather	8255 127	[1,4096] [64128]
	4	Allreduce Gather	65 1	[128,4096] [32064]	Allreduce Gather	8255 127	[1, 4096] [32064]

TABLE III: Message size and frequency breakdown for intra-node TP using Llama-3.1-8B

	Llama-3.2-3B		Llama-3.1-8B		Llama-2-13B	
Message Size (bytes)	786432	6144	1048576	8192	1310720	10240
Count	57	7239	65	8255	81	10287

TABLE IV: Allreduce message size and count comparison across models for end-to-end inference

- **Tensor Parallelism**
  - All-reduce frequency depends on # Transformer layers and Decoding Steps
  - Message Size depends on sequence length and hidden dimension

Performance Analysis: Message Size and Frequency

Model	PP Size	Prefill Stage			Decode Stage		
		Operation	**Count**	**Shape**	**Operation**	**Count**	**Shape**
Llama-3.1-8B $S_p = 128$ $S_d = 128$	2	Send	2	[128,4096]	Send	254	[1,4096]
		Recv	2	[128,4096]	Recv	254	[1,4096]
	4	Send	6	[128,4096]	Send	762	[1,4096]
		Recv	6	[128,4096]	Recv	762	[1,4096]

TABLE V: Message size and frequency breakdown for pipeline parallelism

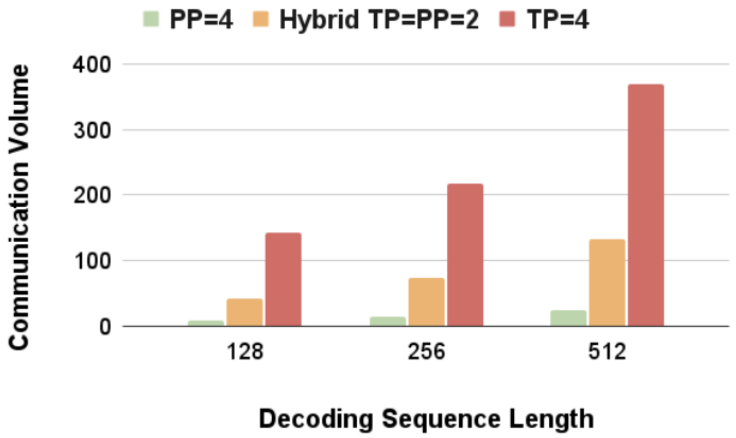
Model	TP×PP	Prefill Stage			Decode Stage		
		Operation	**Count**	**Shape**	**Operation**	**Count**	**Shape**
Llama-3.1-8B $S_p = 128$ $S_d = 128$	2×2	Allreduce	33	[128,4096]	Allreduce	4191	[1,4096]
		Gather	1	[64128]	Gather	127	[64128]
		Allgather	2	[128,4096]	Allgather	254	[1,4096]
		Send/Recv	2	[128,2048]	Send/Recv	254	[1,2048]

TABLE VI: Message size and frequency breakdown for hybrid parallelism (TP×PP) using Llama-3.1-8B

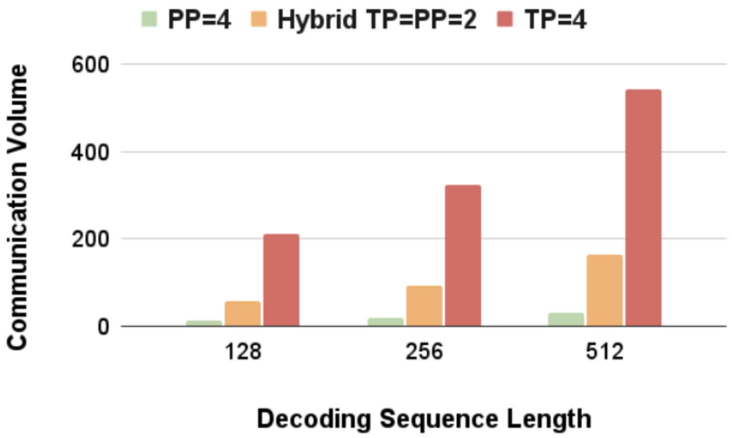
- **Pipeline Parallelism**
  - P2P frequency depends on # pipeline links
  - P2P message size remains small and depends on hidden dimension

- Key Takeaway
  - Moderate Message Size with high Frequency
  - Decode Stage is more communication heavy
  - All-reduce and P2P are the major operations

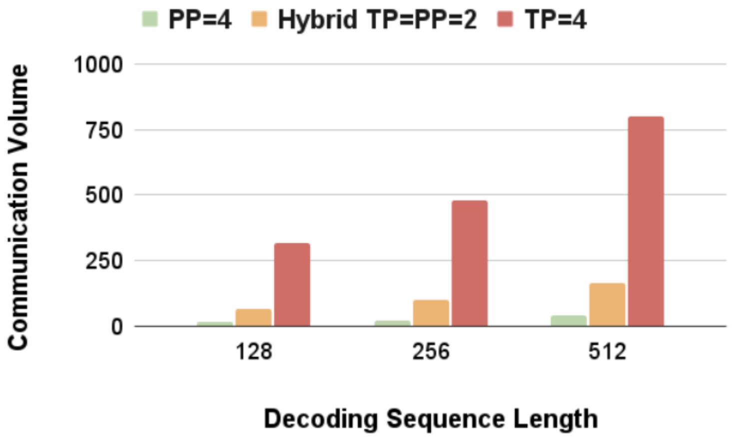
Performance Analysis: Communication Volume



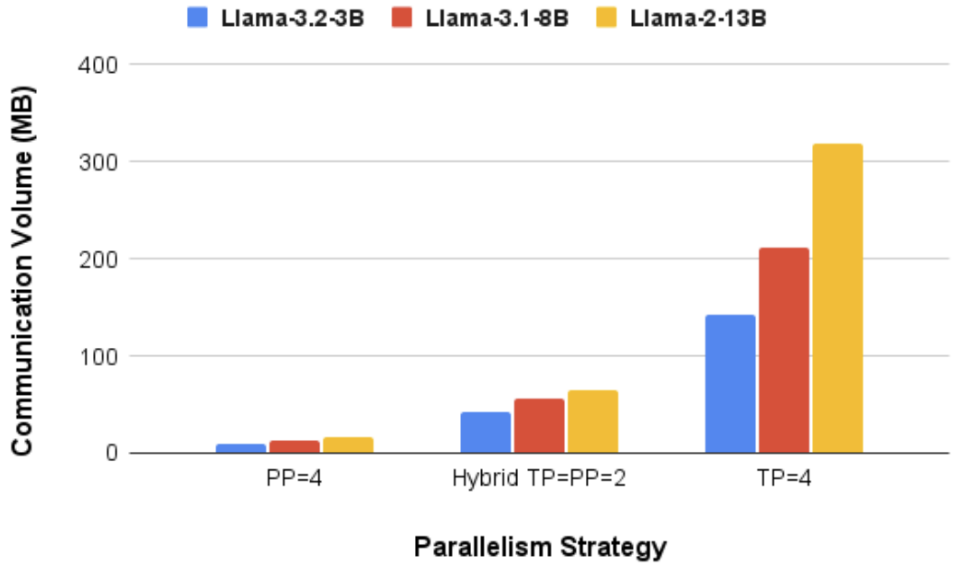
(a) Llama-3.2-3B



(b) Llama-3.1-8B

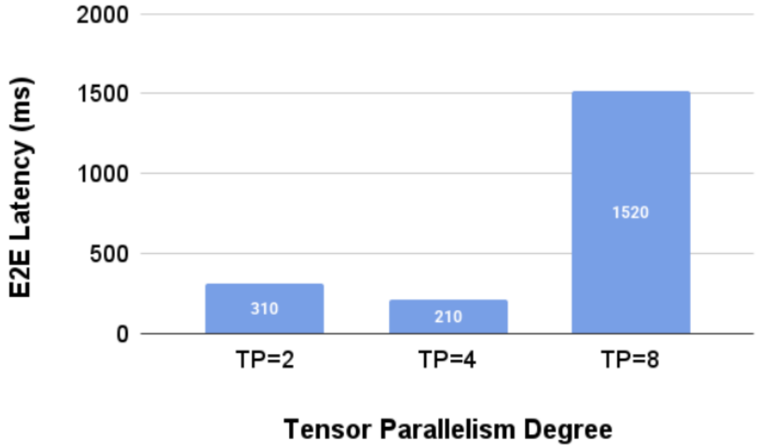


(c) Llama-2-13B

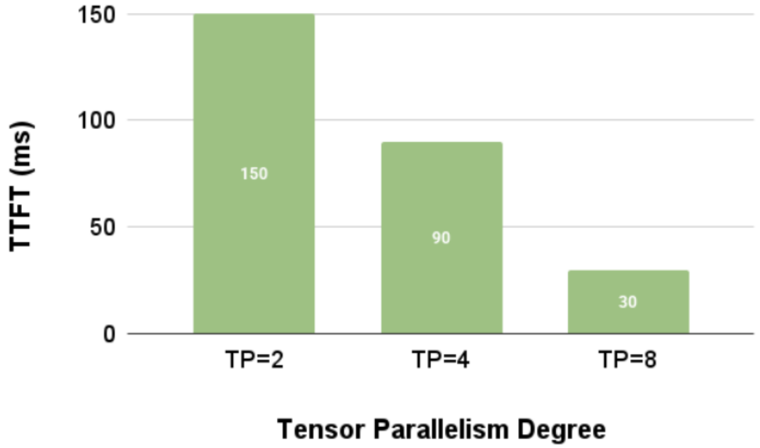


- Key Takeaway
  - Tensor Parallelism has the most communication overhead that scales with model size and sequence length
  - Pipeline Parallelism has minimal pressure on network, good for bandwidth-constrained and long-sequence scenarios. However, it is under-utilizing GPU compute.
  - Hybrid Parallelism strikes a balance between communication overhead and GPU utilization

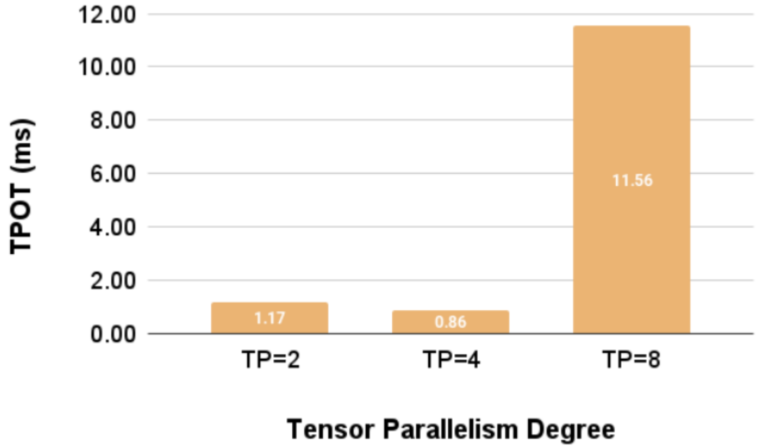
Performance Analysis: SLO Evaluation



(a) End-to-end Latency



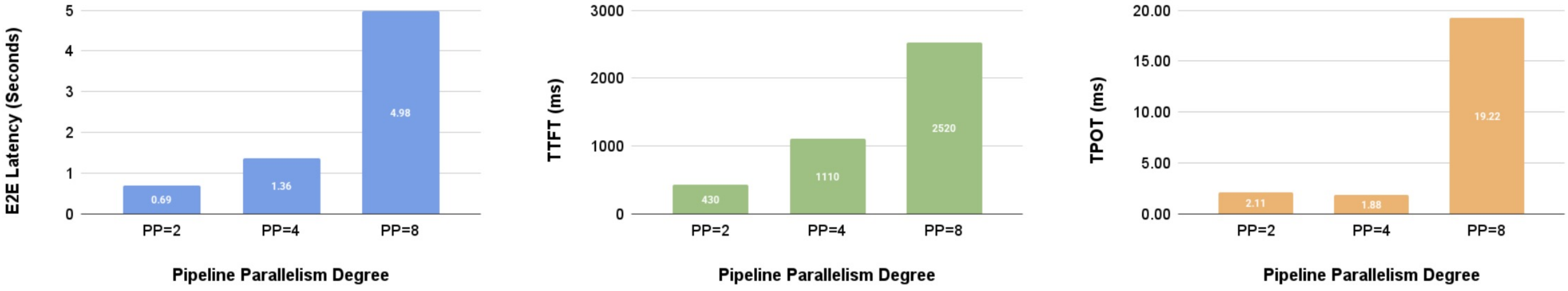
(b) Time-to-first-token



(c) Time-per-output-token

- **Tensor Parallelism (TP)**
  - TTFT: improves as we increase TP degree, since prefill stage is mostly compute-bound
  - TPOT: more memory-bound, TP-8 has crossed inter-node boundary

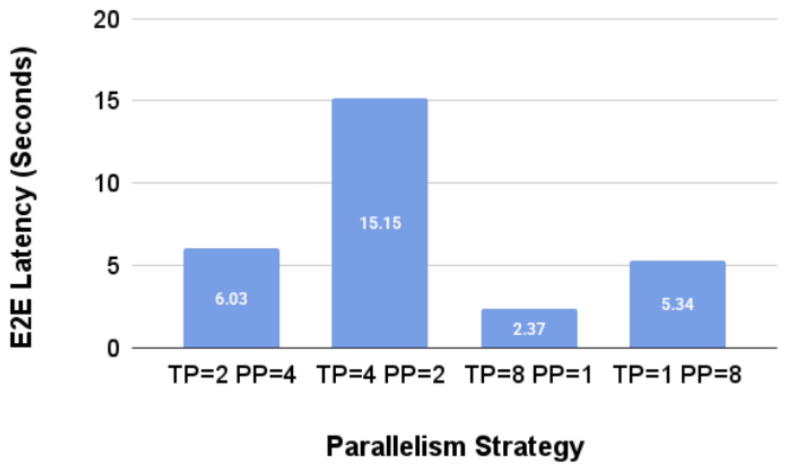
Performance Analysis: SLO Evaluation



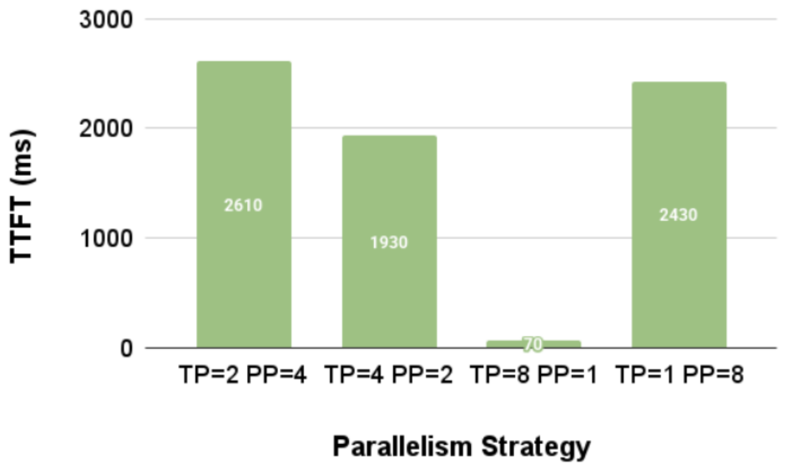
(a) End-to-end Latency (b) Time-to-first-token (c) Time-per-output-token

- **Pipeline Parallelism (PP)**
  - TTFT: Data dependency + latency scales with # links, PP-8 crosses node boundary
  - TPOT: memory-bound, dominated by critical links

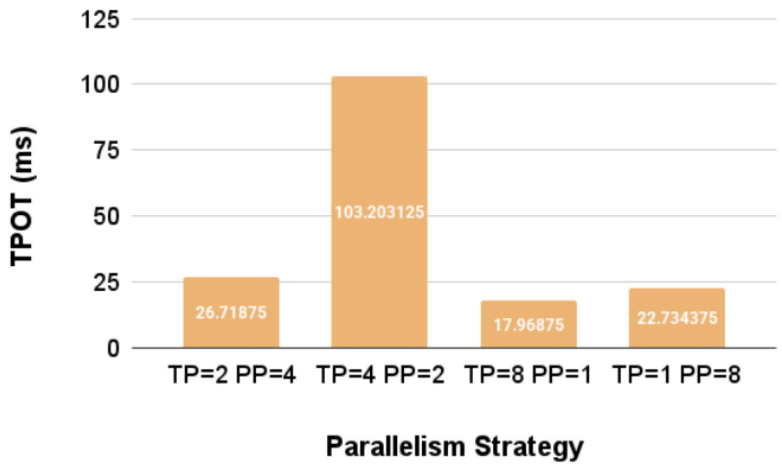
Performance Analysis: SLO Evaluation



(a) End-to-end Latency



(b) Time-to-first-token



(c) Time-per-output-token

- **Hybrid Parallelism (TP + PP)**
  - Pure Tensor Parallelism has the best Latency, TTFT and TPOT (Keeping GPUs busy)
    - Fits low-latency and short generation applications
  - Pure Pipeline Parallelism has acceptable E2E Latency & TPOT
  - TP=4, PP=2 remains mostly unbalanced, small TP collectives + internode link
  - PP = 8 wins with only one inter-node link and much less communication

Conclusions

- Inference workloads impose communications with moderate message size and high frequency.
- **Decode stage** dominates communication frequency.
- **All-reduce** and **P2P** are the two major primitives in Tensor, Pipeline and Hybrid Parallelism.
- Tensor Parallelism offers **better GPU utilization** and computation efficiency but substantial **communication overhead** .
  - Fits latency sensitive and short generation tasks.
- Pipeline Parallelism offers **minimal communication overhead** but low GPU utilization and **data dependency**, which is detrimental to latency.
  - Fits low-bandwidth environments, and long generation tasks.
- **While computational parallelization can overcome communication overhead for short sequences, it diminishes with longer sequences and inter-node deployments.**

Thank You!

{xu.3304, kandadisuresh.1, anthony.301, alnaasan.1}@osu.edu, panda@cse.ohio-state.edu

Network-Based Computing Laboratory
http://nowlab.cse.ohio-state.edu/

Full paper is on Arxiv!
https://arxiv.org/abs/2507.14392

The High-Performance MPI/PGAS Project
http://mvapich.cse.ohio-state.edu/

Follow us on
https://x.com/mvapich

The High-Performance Deep Learning Project
http://hidl.cse.ohio-state.edu/