Optimizing Resource Allocation for Geographically-Distributed Inference by Large Language Models

Tingyang Sun[1], **Ting He[1]**, Bo Ji[2], Parimal Parag[3]

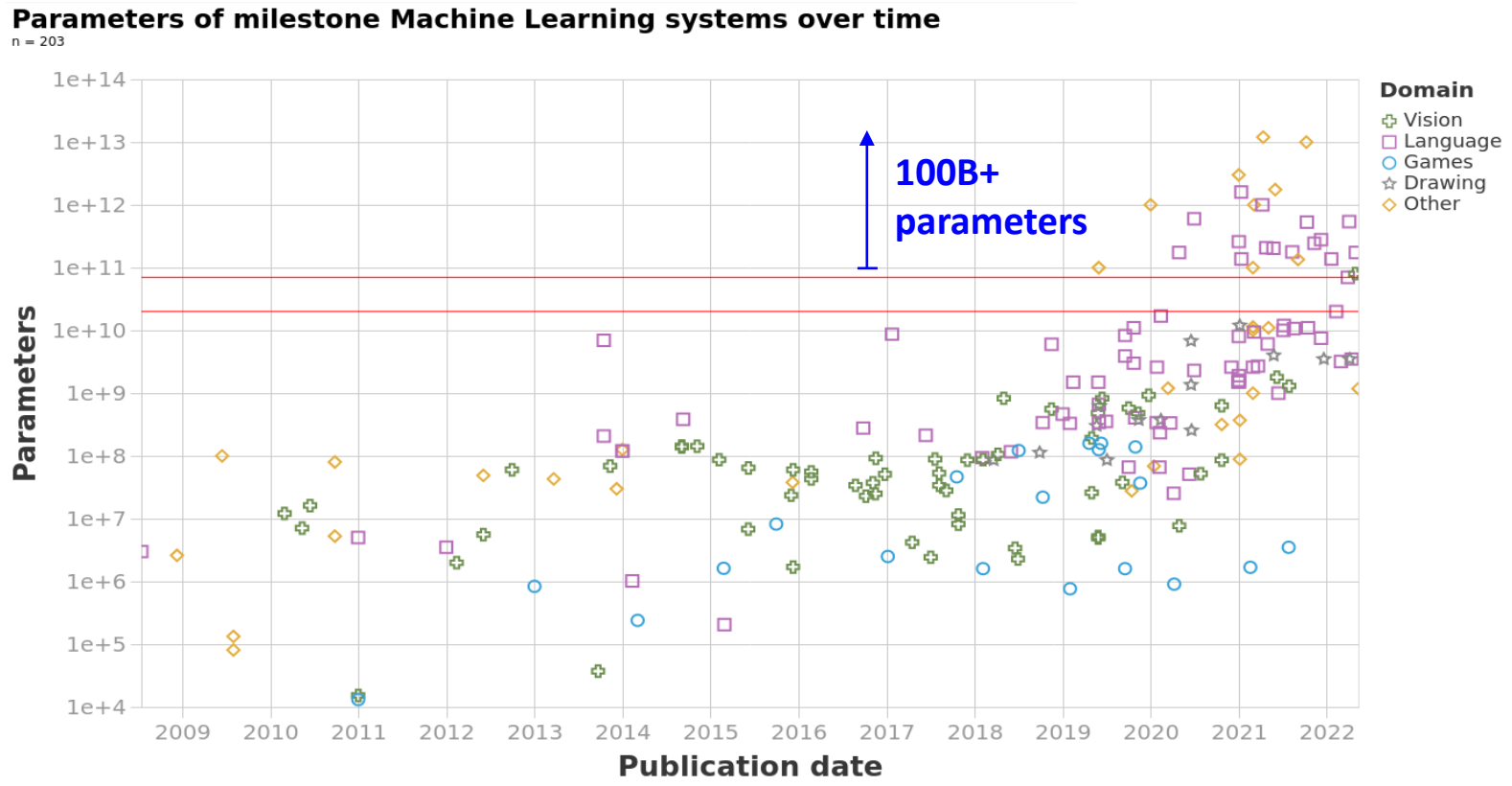
1 CSE@Penn State

2 CS@Virginia Tech

3 ECE@IISc

Deploying Pretrained Large Models at the Edge

- Large models (e.g., LLM) are powerful, but it is challenging to deploy them even after training



E.g.,

- GPT-3: 175B
- GPT-4: 1.76T
- DeepSeek (V3): 671B

Background: Transformer Architecture

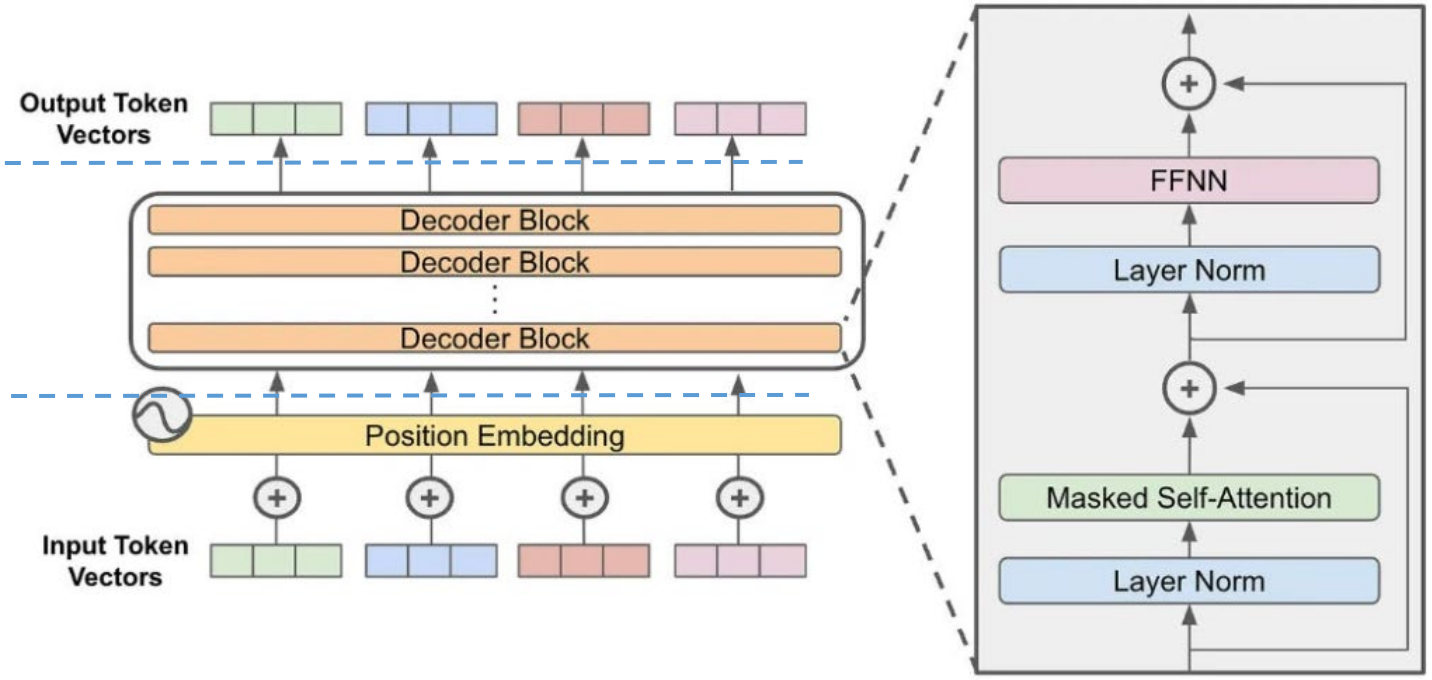
- Decoder-only transformer is the architecture of many state-of-the-art LLMs
  - GPT series, LLaMA, Bard, BLOOM, …

Output layer (token generation)

Identically structured transformer blocks

- **>97% of parameters (BLOOM)**

Input layer (embedding)



Approaches to Handle Large Models

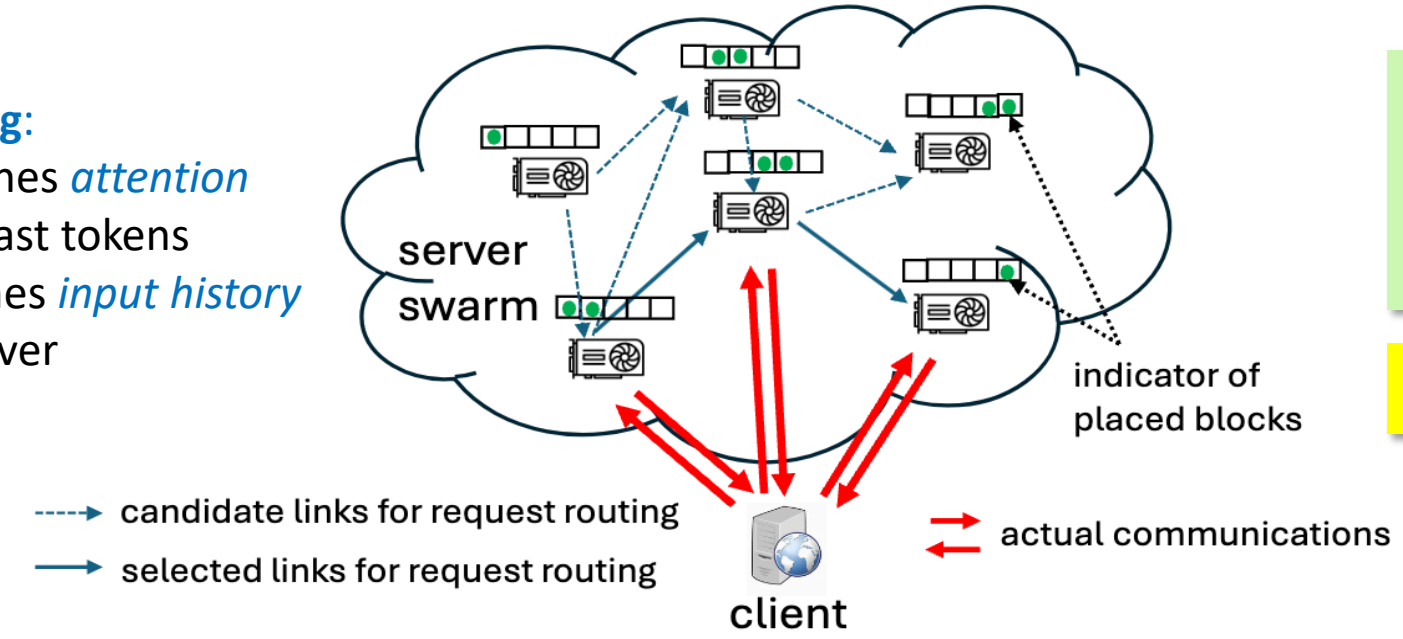
- **Model quantization/pruning**
  - Reduce model size by a constant factor (e.g., 175B parameters → 87.5GB)
  - May lower the inference accuracy
- **Parameter offloading**
  - Significantly slow down the data processing (e.g., >11s/token for 175B model)
- **Model parallelism:** Splitting a pretrained large model across multiple compute nodes
  - Tensor parallelism
  - Pipeline parallelism

SOTA: Geographically-Distributed LLM Inference

- Idea: Pipeline parallelism + Client-server dual caching

PETALS [Borzunov'23]: Open-source system for distributed deployment of LLM by **splitting transformer blocks** across multiple nodes with low-end GPUs.

Dual-caching:
- Server caches *attention values* for past tokens
- Client caches *input history* for each server



Fully preserve the accuracy of the LLM

- via resource pooling

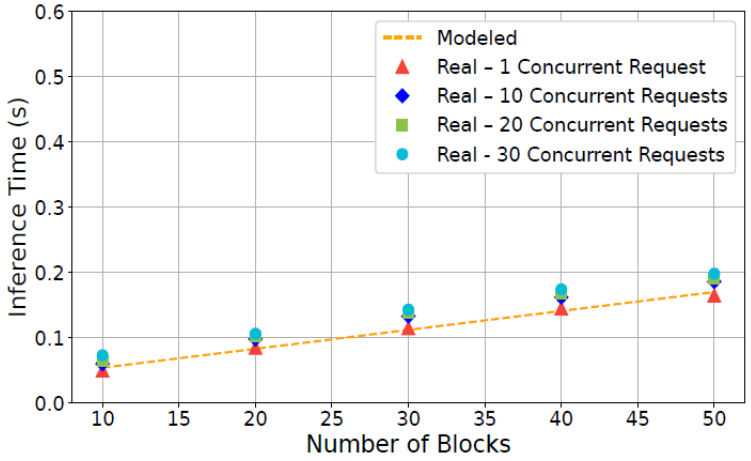
Q: How is the latency?

[Borzunov'23] Distributed Inference and Fine-tuning of Large Language Models over the Internet, NeurIPS '23.

Performance Modeling

- **Per-token inference time:** Assume server j holds m_j blocks starting from block a_j
  - Time per token at server j after server i:

$$t_{ij}^{c} = \underbrace{t_{cj}}_{\text{RTT}} + \underbrace{\tau_j}_{\text{time/block}}(\underbrace{a_j + m_j - a_i - m_i}_{\text{\#processed blocks}})$$



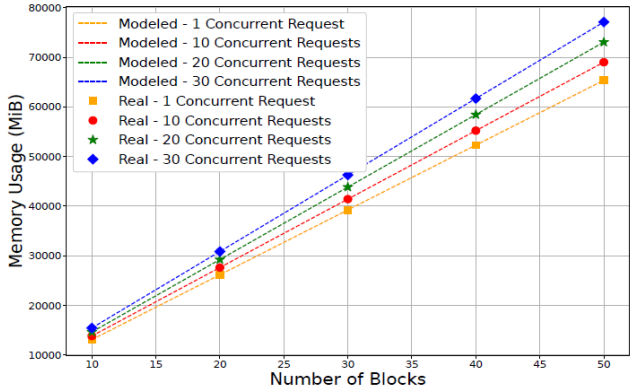
Time for first token:

- $t_{ij}^{c}(l_{max}^{I}) = t_{cj}(l_{max}^{I}) + \tau_j(l_{max}^{I})(a_j + m_j - a_i - m_i)$

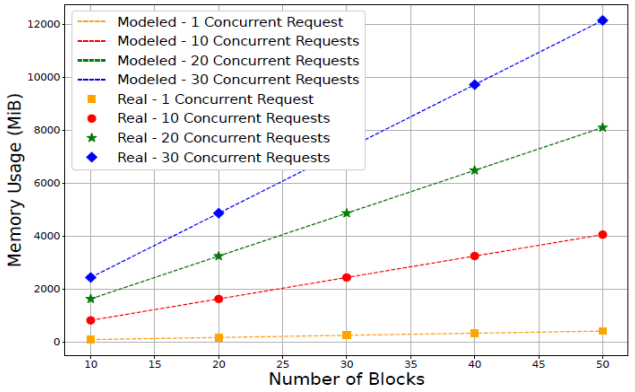
- **Memory consumption:** Assume f_{ij} requests routed over (i, j)
  - Memory consumption at server j:

$$\underbrace{s_m}_{\text{size/block}} m_j + \underbrace{s_c}_{\text{size/cache}} \sum_{i:(i,j)\in E} f_{ij}(a_j + m_j - a_i - m_i)$$

\#processed blocks



Total memory



Memory for caches

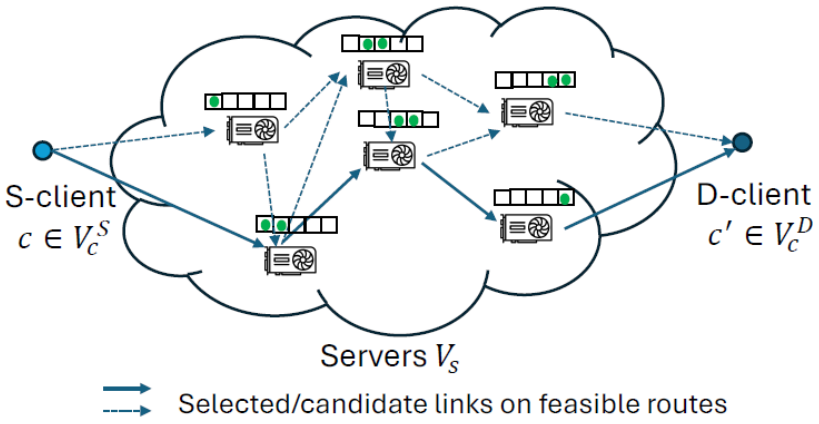
Q: How to **optimally manage** such a system?

Optimizing Distributed LLM Inference

- **Distributed LLM inference has unique characteristics**:
  - Memory-bound
  - Deployment phase and execution phase contend for same resource (GPU memory)
  - Other system-specific features (e.g., only client-server communication, #processed blocks depending on blocks at previous server)

→ New resource allocation problem: **Block Placement and Request Routing (BPRR)**

- MILP formulation, NP-hard



$$\min_{f,a,m} \quad \sum_{c\in V_c}\sum_{r\in\mathcal{R}_c}\sum_{p\in P_c(a,m)} f_p^r \sum_{(i,j)\in p} t_{ij}^c$$

← (total) per-token inference time

$$\text{s.t.} \quad s_m m_j + s_c \sum_{c\in V_c}\sum_{r\in\mathcal{R}_c}\sum_{\substack{p\in P_c(a,m):\\ (i,j)\in p,\exists i}} f_p^r(a_j + m_j - a_i - m_i) \le M_j, \quad \forall j \in V_s,$$

GPU memory constraint

$$\sum_{p\in P_c(a,m)} f_p^r = 1, \quad \forall c \in V_c, r \in \mathcal{R}_c,$$

← route feasibility constraint

$$a_j + m_j - 1 \le L, \quad \forall j \in V_s,$$

← placement feasibility constraint

$$f_p^r \in \{0,1\}, \; a_j, m_j \in [L],$$

Offline BPRR: Algorithm

Algorithm 1: Conservative Greedy BPRR (CG-BPRR)

input : set of clients V_c, set of requests $\bigcup_{c\in V_c} \mathcal{R}_c$, #blocks L, size per block s_m, size per cache s_c, set of servers V_s, parameters for each $j \in V_s$ including GPU memory M_j, processing time τ_j, and per-token RTTs $(t_{cj})_{c\in V_c}$

output: Block placement $(\boldsymbol{a}, \boldsymbol{m})$ and request routing $\boldsymbol{f}$

// conservative assignment of #blocks per server:

$m_j \leftarrow \min(\lfloor M_j/(s_m + s_c|\mathcal{R}|)\rfloor,\ L),\ \forall j \in V_s$, where $|\mathcal{R}| = \sum_{c\in V_c} |\mathcal{R}_c|$;

// greedy block placement:

$C_b \leftarrow 0,\ T_b \leftarrow \widetilde{t}_0|\mathcal{R}|,\ \forall b \in [L]$;

for *each server* $j \in V_s$ *in increasing order of* $\widetilde{t}_j$ **do**

  if $\exists b \in [L]$ *with* $C_b < |\mathcal{R}|$ **then**

    $a_j \leftarrow \arg\max_{a\in[L-m_j+1]:\ C_b<|\mathcal{R}|,\ \exists b\in\{a,\ldots,a+m_j-1\}} \sum_{b'=a}^{a+m_j-1} T_{b'}$;

  else

    $a_j \leftarrow \arg\min_{a\in[L-m_j+1]} (C_a, \ldots, C_{a+m_j-1})$;

  $T_b \leftarrow T_b - (\widetilde{t}_0 - \widetilde{t}_j)\min\left(\max(|\mathcal{R}| - C_b, 0), \overline{f}_j\right),\ \forall b \in \{a_j, \ldots, a_j + m_j - 1\}$;

  $C_b \leftarrow C_b + \overline{f}_j,\ \forall b \in \{a_j, \ldots, a_j + m_j - 1\}$;

// shortest-path request routing:

for *each client* $c \in V_c$ **do**

  $G^c_{a,m} \leftarrow$ the feasible routing topology for client c under block placement $(\boldsymbol{a}, \boldsymbol{m})$, with a node/link set $(V^c, E^c_{a,m})$ and a cost of t^c_{ij} for each $(i, j) \in E^c_{a,m}$;

  $p_c \leftarrow$ shortest path from the S-client to the D-client in $G^c_{a,m}$;

  $f^r_{ij} \leftarrow \mathbb{1}((i, j) \in p_c)$[4], $\forall r \in \mathcal{R}_c,\ (i, j) \in E$;

Step 1: Conservatively set #blocks

GPU memory consumption bounded by $s_m m_j + s_c|R|m_j$

Step 2: Greedy block placement

Amortized inference time: $\widetilde{t}_j := \tau_j + \frac{max_{c\in V_c}\, t_{cj}}{m_j}$

Place blocks at fastest servers first

Step 3: Shortest-path request routing

Route each request to the server chain with minimum communication + processing time

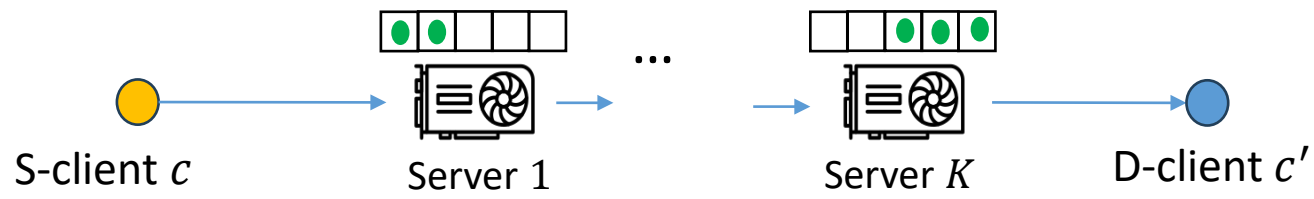
Offline BPRR: Analysis

- Step 1: Conservative #blocks → No waiting (if all blocks are placed)
- Step 2: Greedy block placement → Min avg. per-token inference time under *relaxed routing* (block by block, per-block time $\tilde{t}_j$)
- Step 3: Shortest-path routing with link cost t_{ij}^c → Min avg. per-token inference time under the block placement from Steps 1-2
- **Theorem**: If CG-BPRR gives a feasible solution (all blocks placed), then its **average per-token inference time**:

$$T^g \leq \sum_{j=1}^{K} \tilde{t}_j m_j - \tau_K \left(\sum_{j=1}^{K} m_j - L \right)$$

Amortized inference time:

$$\tilde{t}_j \coloneqq \tau_j + \frac{1}{m_j} \max_{c \in V_c} t_{cj}$$



Top-K servers host all the blocks

Online BPRR

- ***Offline* Conservative Greedy Block Placement (CG-BP):** Steps 1-2 of CG-BPRR for a **target #requests R**
  - Per-token inference time $T^g \leq \sum_{j=1}^{K} \tilde{t}_j m_j - \tau_K\left(\sum_{j=1}^{K} m_j - L\right)$ if #concurrent requests $\leq R$
  - R: parameter controlling "throughput-delay tradeoff"
- ***Online* Waiting-penalized Shortest-path Request Routing (WS-RR):** Step 3 of CG-BPRR with **waiting-penalized link cost $t_{ij}^W(t) + l_{max} t_{ij}^c$**
  - $t_{ij}^W(t)$: waiting time on (i,j) for a request arriving at time t

$$\min_{f, t^W} \quad t^W + l_{\max} \sum_{(i,j)\in E_{a,m}^c} t_{ij}^c f_{ij}$$

$$\text{s.t.} \quad t_{ij}^W(t) f_{ij} \leq t^W, \quad \forall (i,j) \in E_{a,m}^c,$$

$$\sum_{i\in\mathcal{N}^+(j;a,m)} f_{ji} - \sum_{i\in\mathcal{N}^-(j;a,m)} f_{ij} = d_j^c, \quad \forall j \in V^c,$$

$$f_{ij} \in \{0,1\}, \quad \forall (i,j) \in E_{a,m}^c,$$

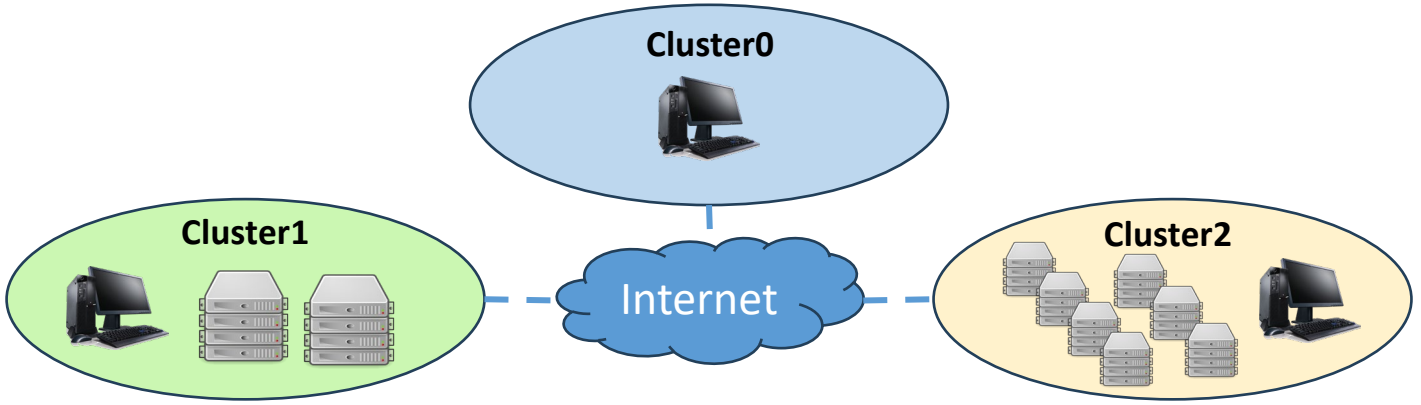
$t^W \leq \Sigma_{(i,j)\in E_{a,m}^c} t_{ij}^W(t) f_{ij}$ ⟹ $\leq \sum_{(i,j)\in E_{a,m}^c} \left(t_{ij}^W(t) + l_{max} t_{ij}^c\right) f_{ij}$

Waiting-penalized cost of link (i,j)

Experiment: Setting 1

- Deploy BLOOM-176B over 9 servers: 2 servers with A100 (80GB), 7 servers with virtual GPU (10GB)
- Simulate network latency/bandwidth by Linux traffic control (tc)

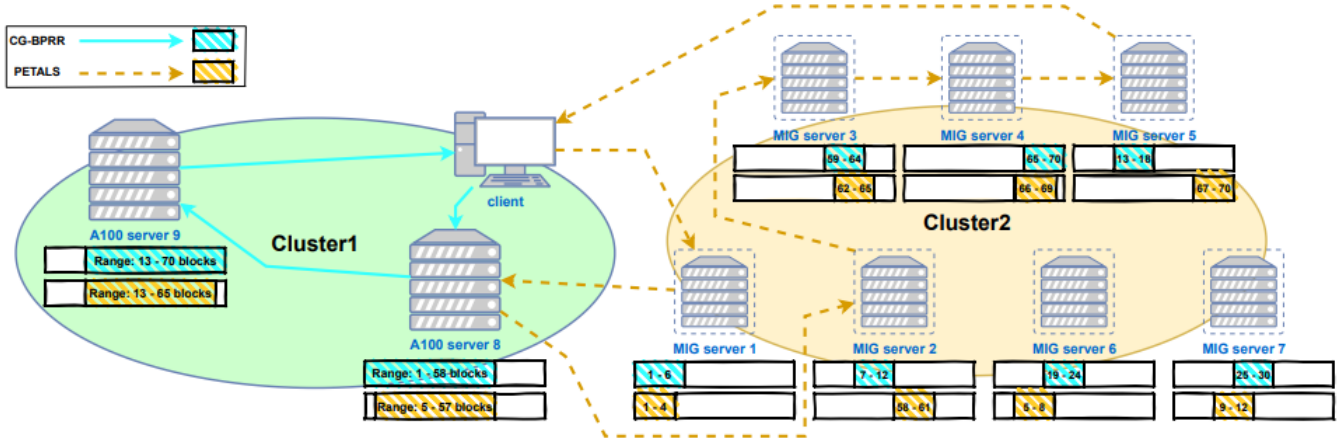
	Cluster0 (CPU)	Cluster1 (2 A100s)	Cluster2 (7 MIGs)
Cluster0 (CPU)	5 ms, 1 Gbit/s	100 ms, 100 Mbit/s	100 ms, 100 Mbit/s
Cluster1 (2 A100s)	100 ms, 100 Mbit/s	5 ms, 1 Gbit/s	100 ms, 100 Mbit/s
Cluster2 (7 MIGs)	100 ms, 100 Mbit/s	100 ms, 100 Mbit/s	5 ms, 1 Gbit/s



Experiment Results: Setting 1

- Inference time (seconds/token):

Client Location	Algorithm	0.1 requests/s		0.5 requests/s	
		$l_{\max} = 64$	$l_{\max} = 128$	$l_{\max} = 64$	$l_{\max} = 128$
Cluster0	PETALS	6.23 (5.33)	4.76 (4.74)	6.28 (5.33)	5.14 (4.74)
	Proposed	1.92 (1.59)	1.43 (0.92)	2.00 (1.59)	1.34 (0.92)
Cluster1	PETALS	5.44 (5.17)	4.60 (4.58)	5.56 (5.17)	4.79 (4.58)
	Proposed	1.78 (1.65)	1.04 (0.83)	1.88 (1.65)	1.11 (0.83)
Cluster2	PETALS	5.30 (4.85)	4.85 (4.07)	5.34 (4.85)	5.25 (4.07)
	Proposed	1.79 (1.59)	1.31 (0.92)	1.94 (1.59)	1.37 (0.92)



- Performance of distributed LLM inference **significantly depends on block placement & request routing**
- Significant performance improvement (**60-70% smaller inference time**) over state of the art

Experiment: Setting 2

- Deploy BLOOM-176B over: (i) 2 A100 + 7 virtual, (ii) 5 A100 + 21 virtual
- Simulate network latency/bandwidth according to ISP topologies:
  - Smaller deployment for AboveNet
  - Larger deployment for BellCanada/GTS-CE

	AboveNet	BellCanada	GTS-CE
#nodes	23	48	149
#links	62	130	386
link capacities (Gbps)	1	1	1
link delays (ms)	[0.100, 13.800]	[0.078, 6.160]	[0.005, 1.081]

Experiment Results: Setting 2

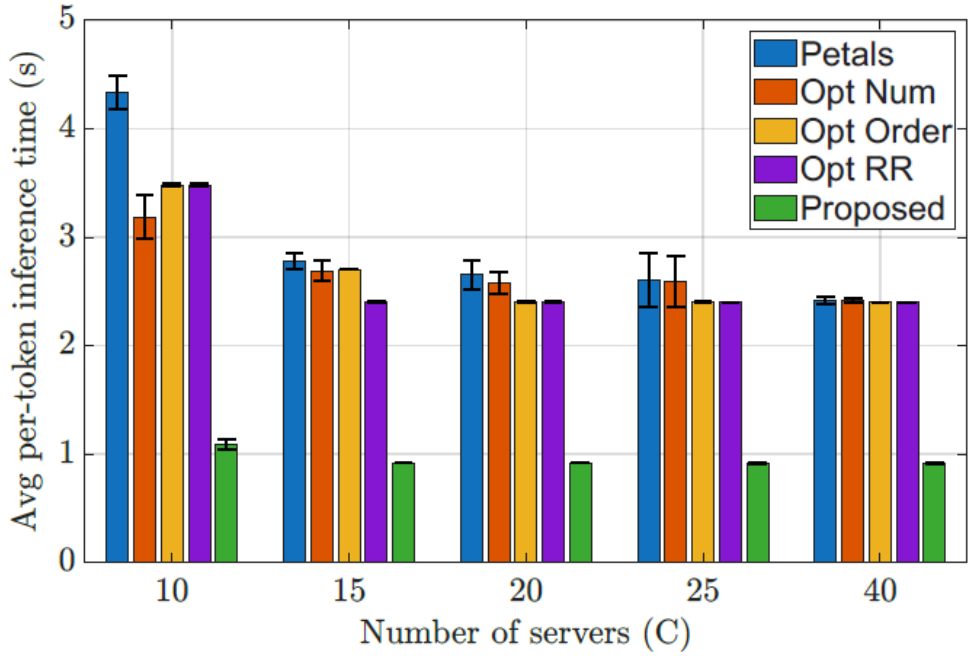
- Inference time (seconds/token):

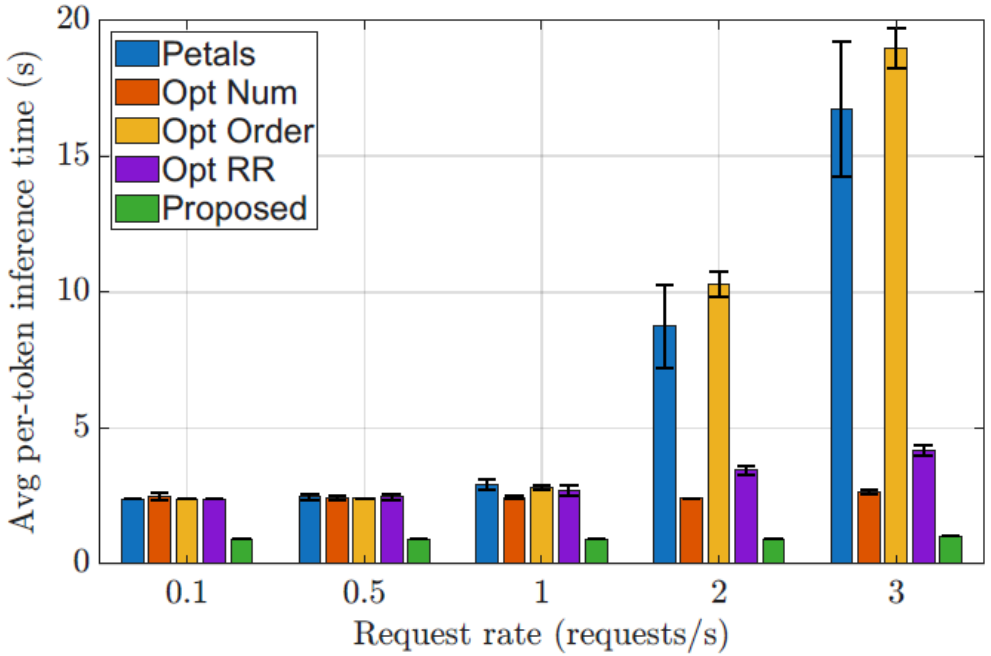
Topology	Algorithm	0.1 requests/s		0.5 requests/s	
		$l_{max} = 64$	$l_{max} = 128$	$l_{max} = 64$	$l_{max} = 128$
AboveNet	PETALS	4.98 (4.75)	4.03 (3.88)	5.26 (5.11)	4.58 (4.10)
	Proposed	1.86 (1.63)	1.44 (1.36)	1.97 (1.83)	1.35 (1.05)
BellCanada	PETALS	6.31 (6.03)	3.82 (3.49)	6.74 (6.19)	4.16 (3.41)
	Proposed	1.33 (1.41)	1.26 (0.92)	1.49 (1.41)	1.11 (0.92)
GTS-CE	PETALS	7.05 (6.12)	4.69 (3.47)	6.89 (5.97)	4.89 (3.37)
	Proposed	1.38 (1.41)	0.95 (0.91)	1.35 (1.40)	1.07 (0.91)

- Significant performance improvement (**60-80% smaller inference time**) over state of the art
- Bigger improvement for larger networks (**60-70% for AboveNet, 80% for GTS-CE**)

Simulation Results

- Experimentally-validated, CPU-only simulator (in Matlab)
  - implements the decision logic & predicts the performance





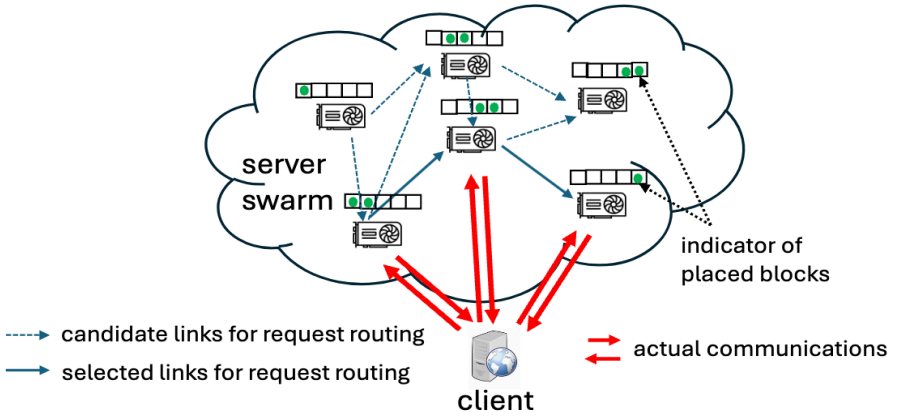
- Optimizing #blocks helps more than optimizing placement order or RR alone
- Optimizing all three (proposed) performs the best

Conclusion and Future Work

- **Distributed LLM inference** requires customized optimizations:
  - Memory-bound
  - Placement phase and routing phase contend for the same resource

  → Novel **Block Placement and Request Routing (BPRR)** problem
    - Offline: **MILP, NP-hard → CG-BPRR**
    - Online: **CG-BP + WS-RR**

- Many **open problems** remains:
  - **Optimality gap**: How close to optimal?
  - **Runtime dynamics:** Unreliable nodes/network?
  - **Generalizability:** Other model serving systems?



Optimizing Resource Allocation for Geographically-Distributed Inference by Large Language Models

Ting He, tinghe@psu.edu

THANK YOU