



IHS-LM: Intra-batch Hybrid Scheduling and Layer Migration for VLM Pipeline Inference Acceleration on Edge Devices

Yun Li¹, Tianfu Pang¹, Zhiyu Cai¹, Zhenxiang Pan¹, Yingchi Mao¹, Jie Wu²

¹ College of Computer Science and Software Engineering, Hohai University, Nanjing, China

²Center for Networked Computing, Temple University, Philadelphia, USA

September 29, 2026

Outline

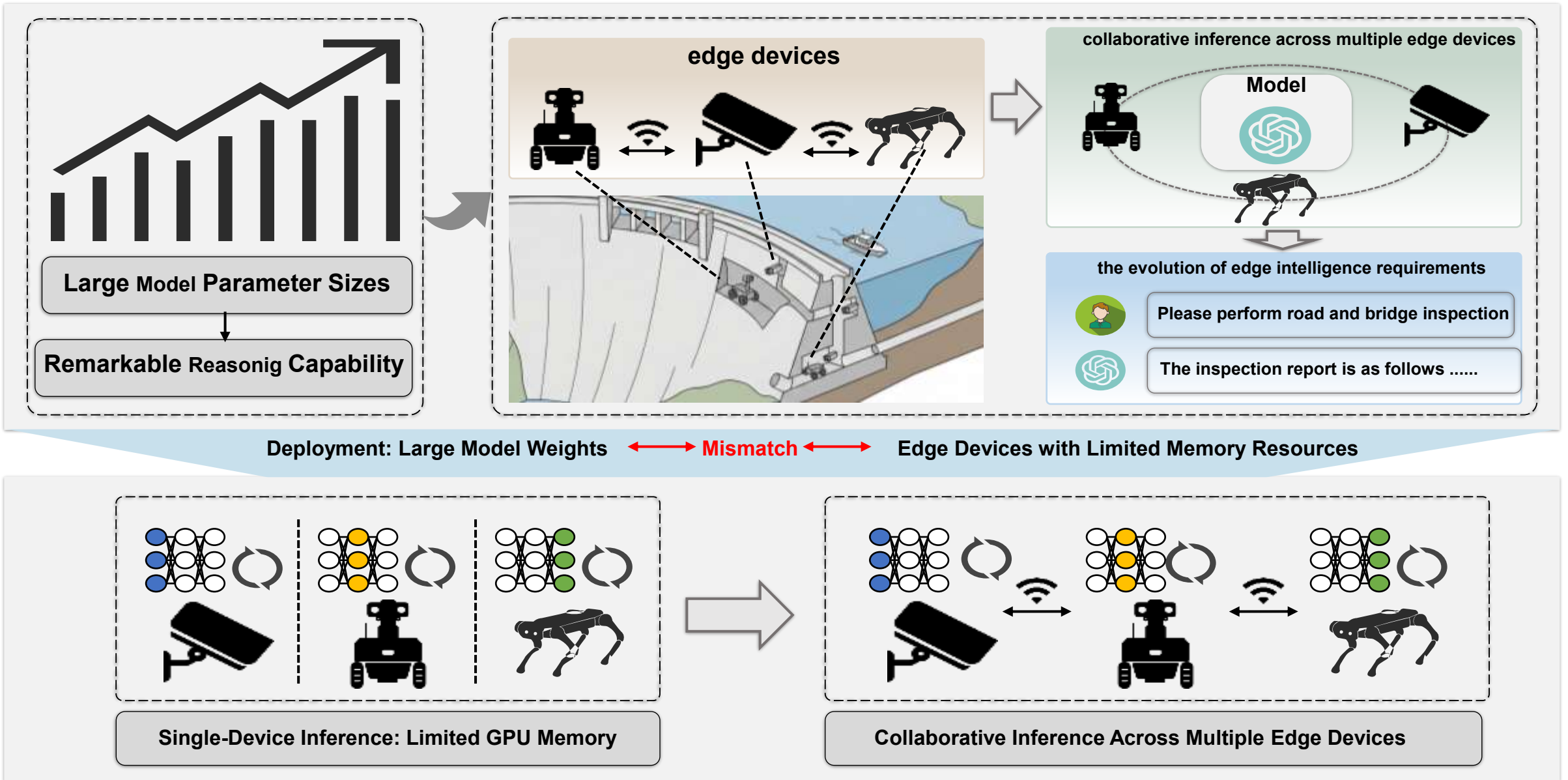
1. Introduction

2. Approach

3. Experimental Evaluation

4. Conclusion

Why Collaborative VLM Inference On Edge Devices?



Why Pipeline Parallelism Fits Edge VLM Inference?



Edge Devices



Limited Memory



Constrained Bandwidth

□ DP & SP

➤ High memory demand

□ TP

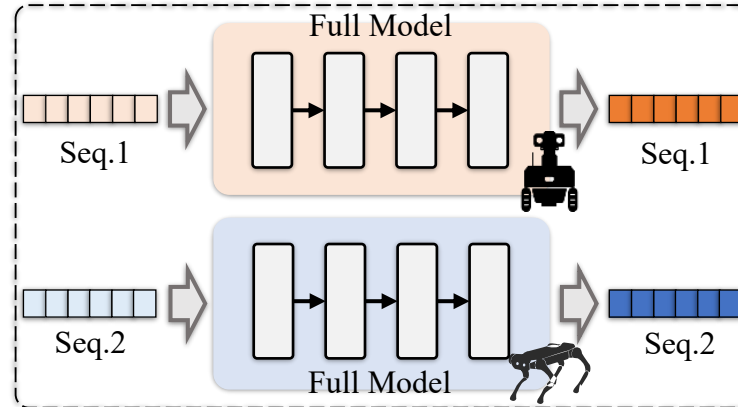
➤ High bandwidth demand

□ PP

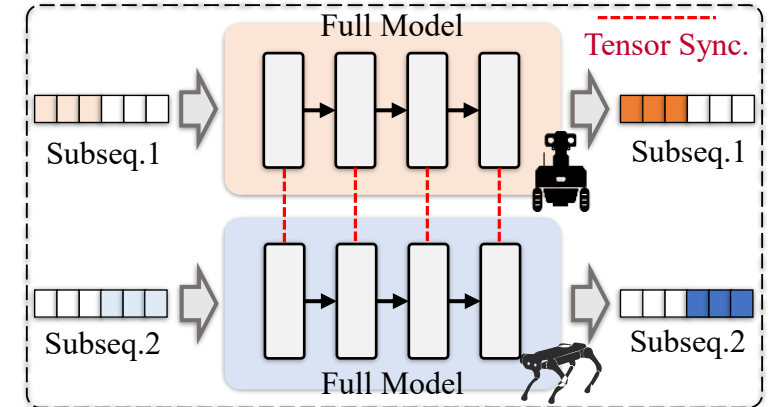
➤ Lower bandwidth demand

➤ Lower memory demand

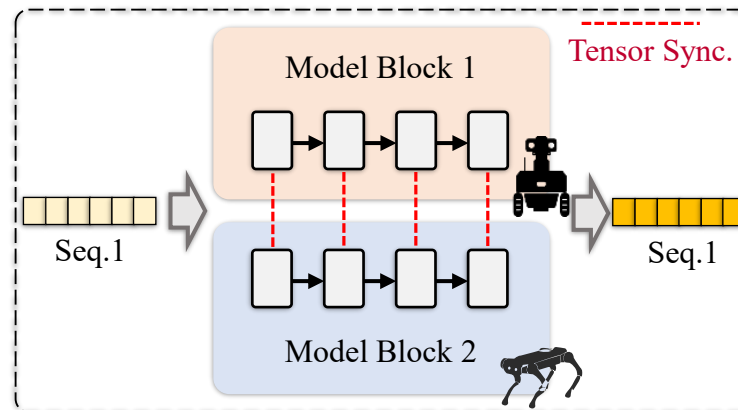
Data Parallelism(DP)



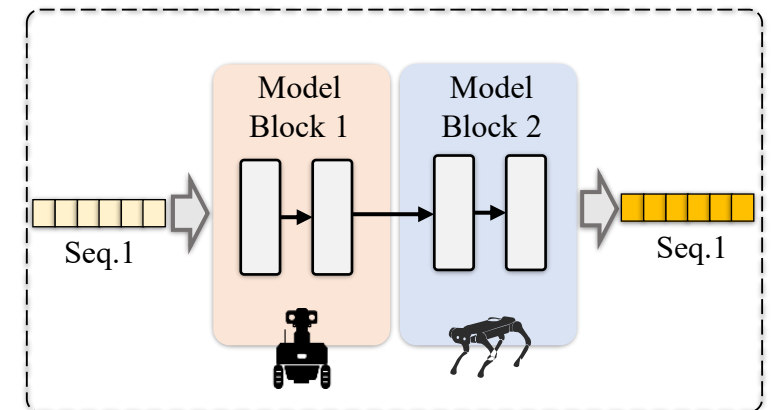
Sequence Parallelism(SP)



Tensor Parallelism(TP)

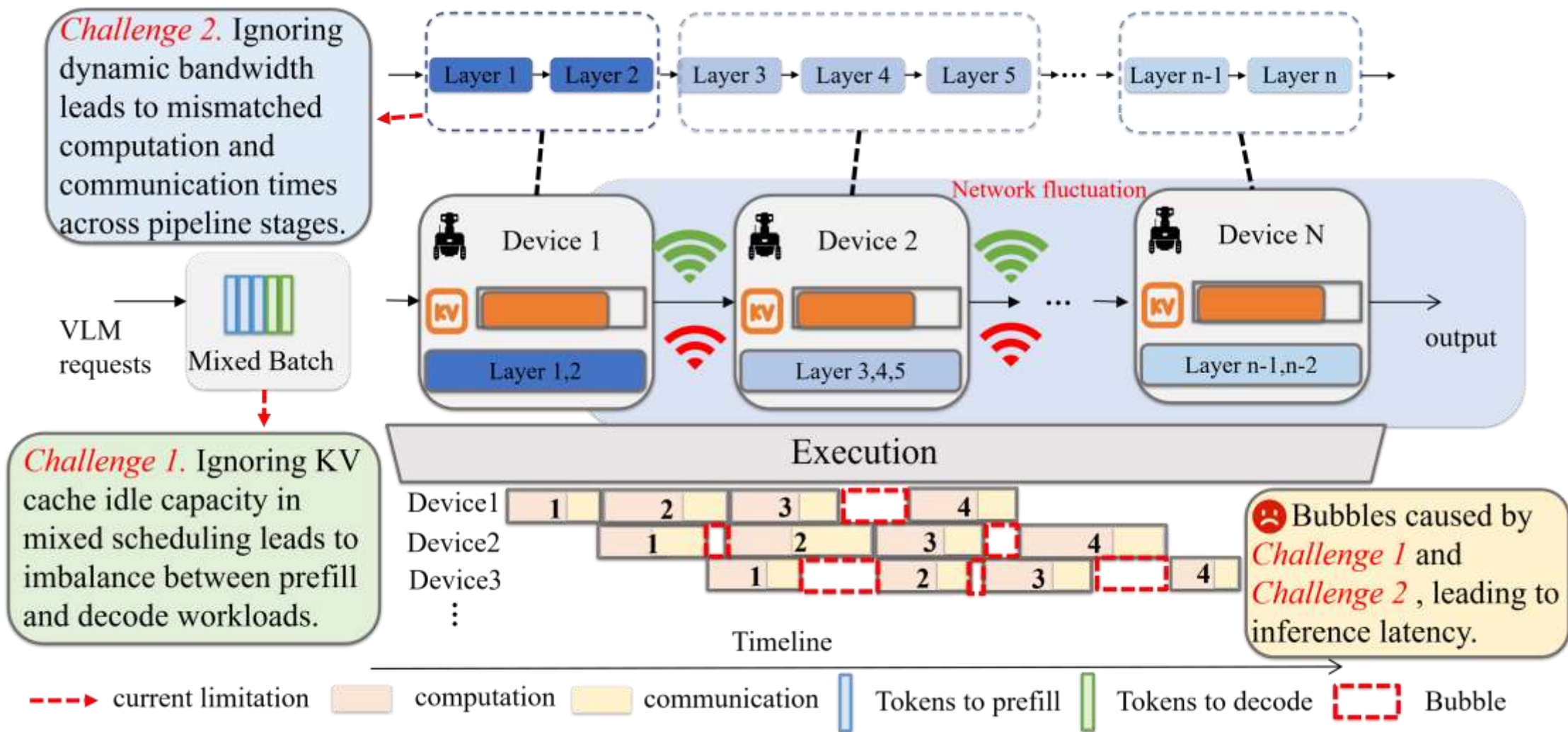


Pipeline Parallelism(PP)



PP is preferred because it reduces per-device model storage and avoids frequent global communication

Challenges in Pipeline-Parallel Inference on Edge Devices



Outline

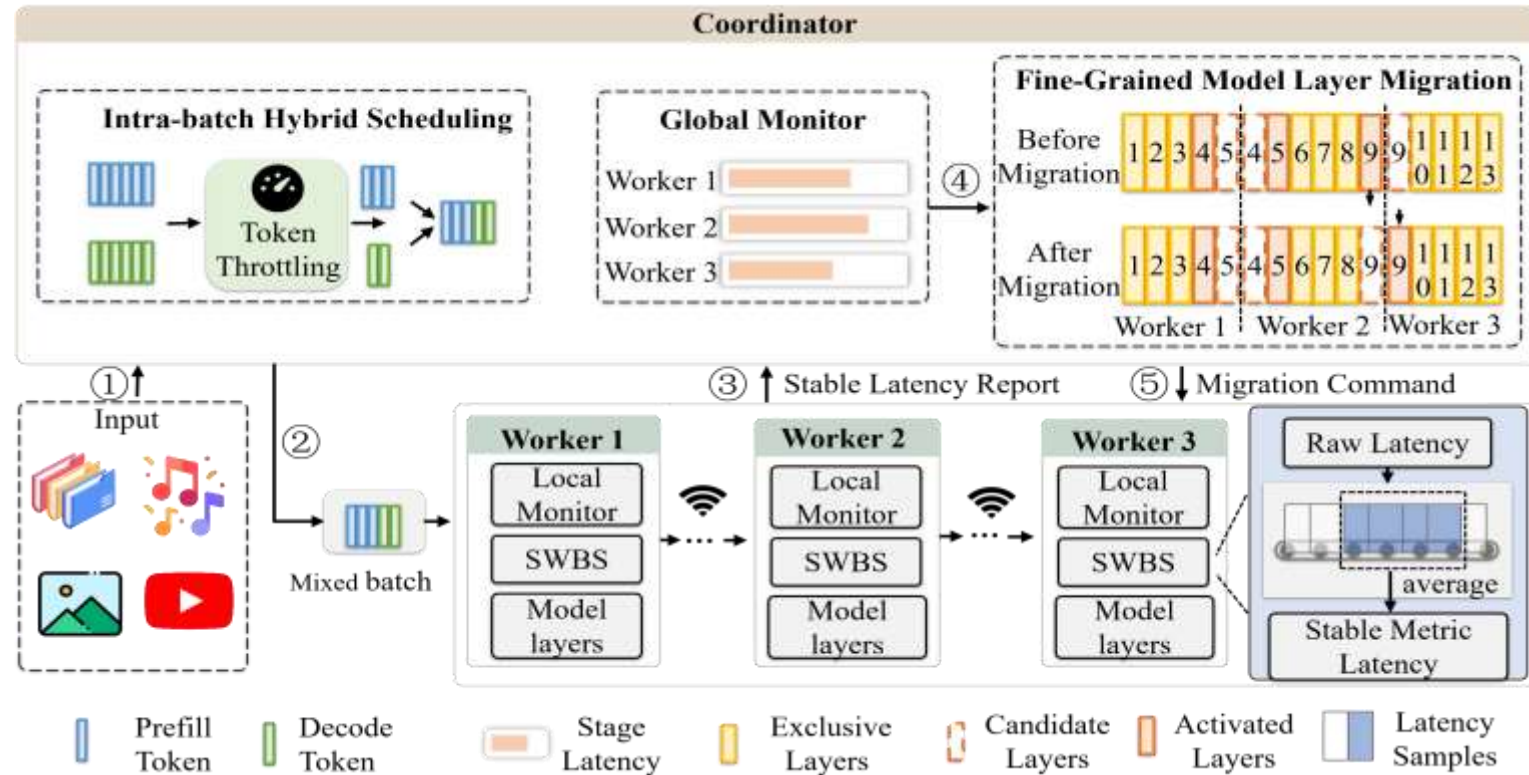
1. Introduction

2. Approach

3. Experimental Evaluation

4. Conclusion

System Overview

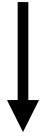


- ① Upload multimodal inputs to the coordinator
- ② IHS balances prefill/decode tokens and dispatches the batch
- ③ Workers report smoothed latency using sliding-window sensing
- ④ The coordinator checks whether stage latency is persistently unbalanced
- ⑤ BAMLM shifts layer boundaries via activate/freeze commands

Intra-batch Hybrid Scheduling

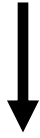
□ Observe State

- pending prefill tokens w_{pre}
- active decode tokens D_{dec}
- available KV-cache free ratio KV_{free}



□ Compute Quotas

- Throttle prefill by KV-cache availability
- *Reserve token budget for decode first*



□ Build Hybrid Micro-batches

- Evenly distribute decode and prefill slices across D_{pp} micro-batches
- balanced execution

Algorithm 1 Intra-batch Hybrid Scheduling Algorithm (IHS)

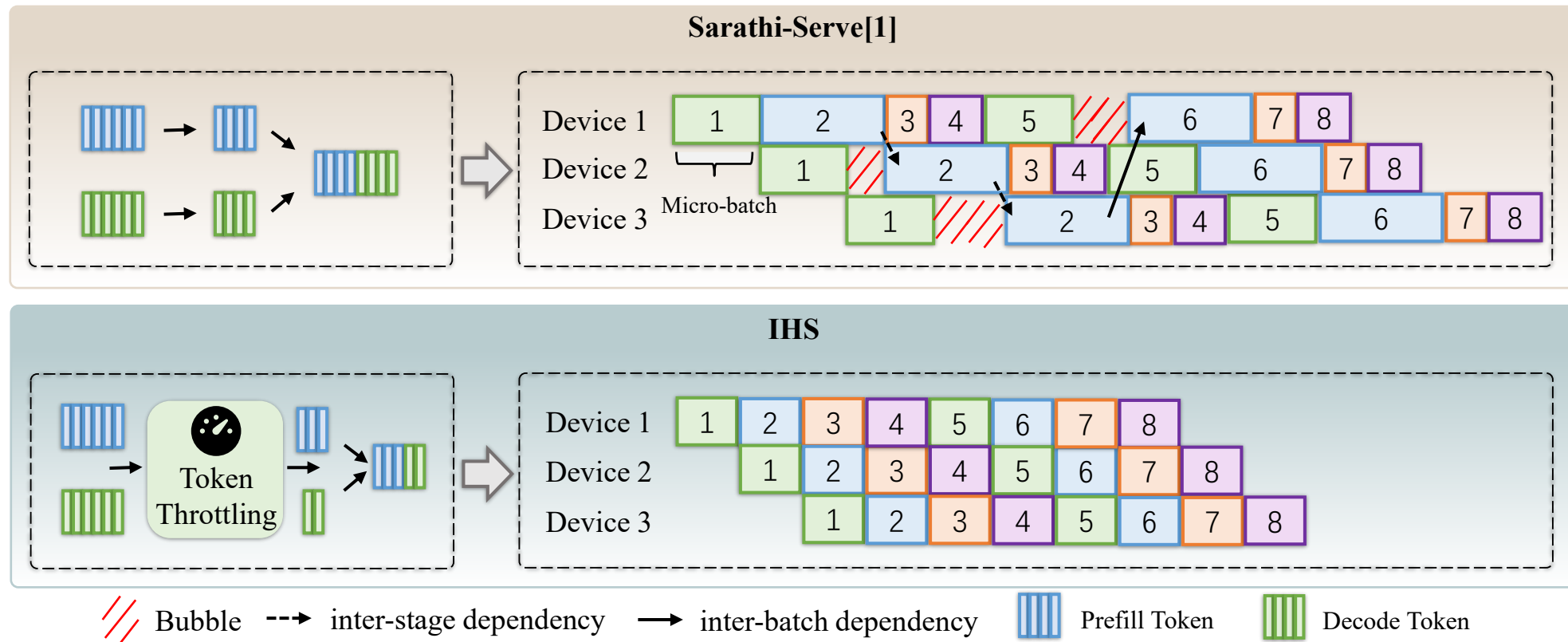
Require: Waiting queue Q_{wait} , running queue Q_{run} , KV cache free ratio KV_{free} , pipeline depth D_{pp} , capacity limits B_{min}, B_{max} , memory threshold KV_{thresh} , target steps K_{step}

Ensure: Micro-batch set $\{T_{micro}^{(1)}, \dots, T_{micro}^{(D_{pp})}\}$

```
1:  $KV_{free} \leftarrow \min_i(KV_{free}^{(i)})$ 
2:  $W_{pre} \leftarrow \text{TokenCount}(Q_{wait})$ 
3:  $W_{dec} \leftarrow \text{TokenCount}(Q_{run})$ 
4:  $\tilde{W}_{dec} \leftarrow \min(W_{dec}, B_{max})$ 
5:  $\eta \leftarrow \max\left(\frac{KV_{free} - KV_{thresh}}{1 - KV_{thresh}}, 0\right)$ 
6:  $B_{pre}^{init} \leftarrow \min\left(W_{pre}, \max\left(\frac{W_{pre}}{K_{step}}, B_{min}\right), B_{max}\eta\right)$ 
7:  $B_{pre} \leftarrow \min\left(B_{pre}^{init}, \max(B_{max} - \tilde{W}_{dec}, 0)\right)$ 
8:  $T_{pre} \leftarrow \text{Fetch}(Q_{wait}, B_{pre})$ 
9:  $T_{dec\_all} \leftarrow \text{Fetch}(Q_{run}, \tilde{W}_{dec})$ 
10: for  $i = 1$  to  $D_{pp}$  do
11:    $n_{dec}^{(i)} \leftarrow \left\lfloor \frac{\tilde{W}_{dec}}{D_{pp}} \right\rfloor$ 
12:    $T_{dec}^{(i)} \leftarrow \text{Pop}\left(T_{dec\_all}, \min\left(\text{TokenCount}(T_{dec\_all}), n_{dec}^{(i)}\right)\right)$ 
13:    $S_{slice} \leftarrow \left\lfloor \frac{B_{pre}}{D_{pp}} \right\rfloor$ 
14:   if  $i = D_{pp}$  then
15:      $S_{slice} \leftarrow \text{TokenCount}(T_{pre})$ 
16:   end if
17:    $T_{micro}^{(i)} \leftarrow \text{Concat}\left(T_{dec}^{(i)}, \text{Pop}(T_{pre}, S_{slice})\right)$ 
18: end for
19: return  $\{T_{micro}^{(1)}, \dots, T_{micro}^{(D_{pp})}\}$ 
```

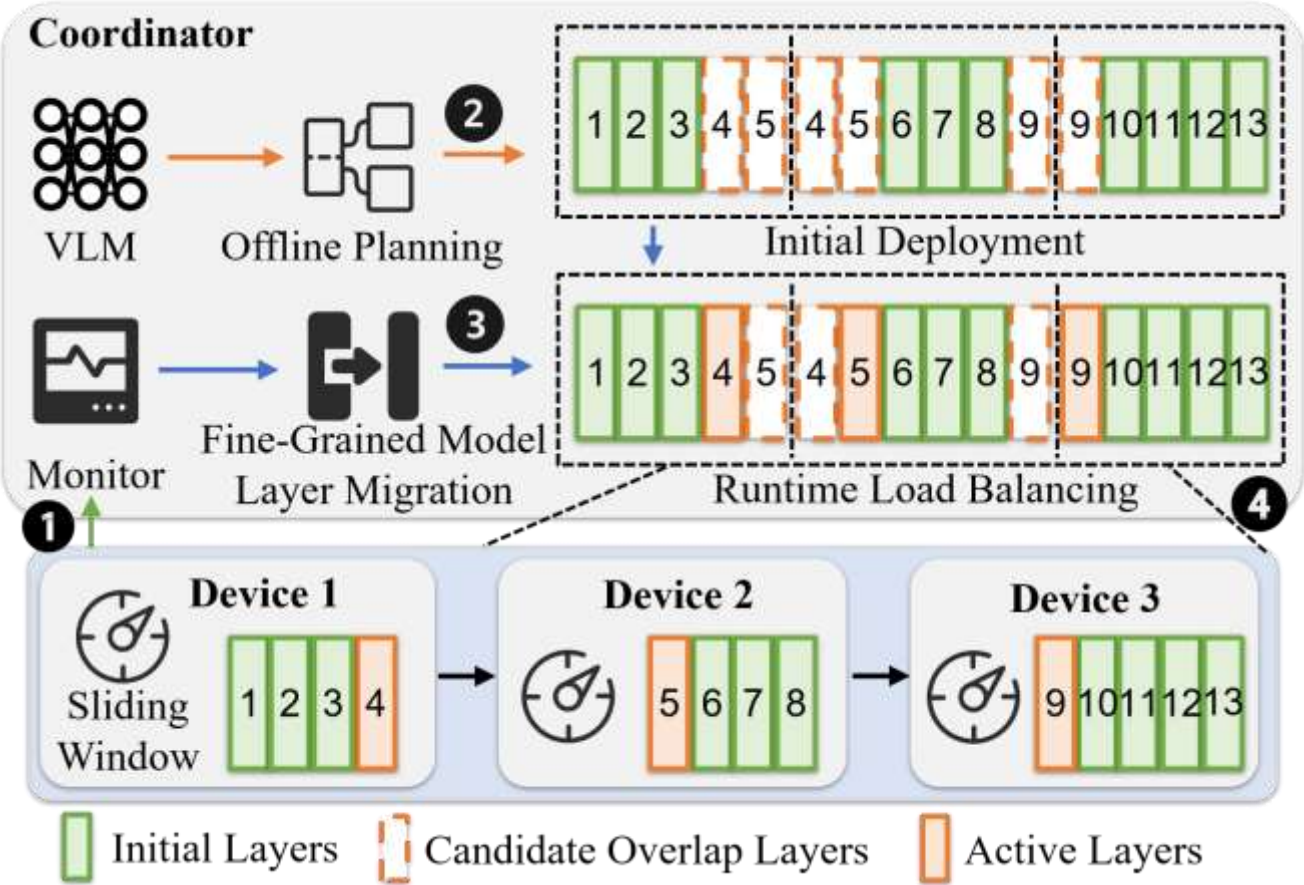
IHS Balances Micro-batches to Reduce Pipeline Bubbles

IHS produces more uniform micro-batches, reducing pipeline idle time



Bandwidth-Aware Layer Migration

- ❑ Monitor Stage Latency
 - Smooth noisy latency with sliding windows
- ❑ Detect Persistent Imbalance
 - Trigger migration when latency gap exceeds threshold
- ❑ Shift Boundary Layers
 - Activate / freeze preloaded candidate layers



Outline

1. Introduction

2. Approach

3. Experimental Evaluation

4. Conclusion

Experimental Settings

Device	Quantity	CPU	Memory Capacity
Jetson Orin Nano	2	6-core Arm Cortex-A78AE	8 GB
Jetson Orin NX	2	8-core Arm Cortex-A78AE	16 GB
Jetson AGX Orin	1	12-core Arm Cortex-A78AE	64 GB



Dataset	Task Type	Scenario Characteristics	Total Samples	Validation Samples
GYU-DET	Classification	Six Typical Surface Defect Categories	11,123	1,112
VizWiz-VQA	Visual Question Answering (VQA)	Open-ended Daily-Life Scenarios	31,173	4,319

Experimental Settings

Metrics

- **Average Inference Latency (AIL):** defined as the average time to generate a single token
- **Inference Accuracy:** representing the proportion of correctly predicted samples
- **Peak GPU Memory Usage:** denoting the maximum consumption across all devices
- **Time to First Token (TTFT):** The elapsed time from request arrival to the generation of the first output token. TTFT mainly reflects queuing, prefill, image encoding, and initial pipeline execution latency
- **Time Per Output Token (TPOT):** The average decoding time per output token after the first token is generated. For a request with L generated tokens, TPOT is computed as $(t_{\text{finish}} - t_{\text{first}})/(L - 1)$

Experimental Settings

- **FasterTransformer[1]**: static batching
- **Orca[2]**: iteration-level scheduling
- **Sarathi-Serve[3]**: chunked prefill + hybrid batching
- **LinguaLinked[4]**: offline overlap-based edge partitioning
- **DynaPipe[5]**: computation-aware dynamic layer assignment

[1] NVIDIA Corporation. 2019. FasterTransformer. <https://github.com/NVIDIA/FasterTransformer>.

[2] Yu, Gyeong-In, et al. "Orca: A distributed serving system for {Transformer-Based} generative models." 16th USENIX symposium on operating systems design and implementation (OSDI 22). 2022.

[3] Agrawal, Amey, et al. "Taming {Throughput-Latency} tradeoff in {LLM} inference with {Sarathi-Serve}." 18th USENIX symposium on operating systems design and implementation (OSDI 24). 2024.

[4] Zhao, Junchen, et al. "LinguaLinked: Distributed large language model inference on mobile devices." Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations). 2024.

[5] Jiang, Chenyu, et al. "DynaPipe: Optimizing multi-task training through dynamic pipelines." Proceedings of the Nineteenth European Conference on Computer Systems. 2024.

Analysis of Intra-Batch Scheduling (IHS) Performance

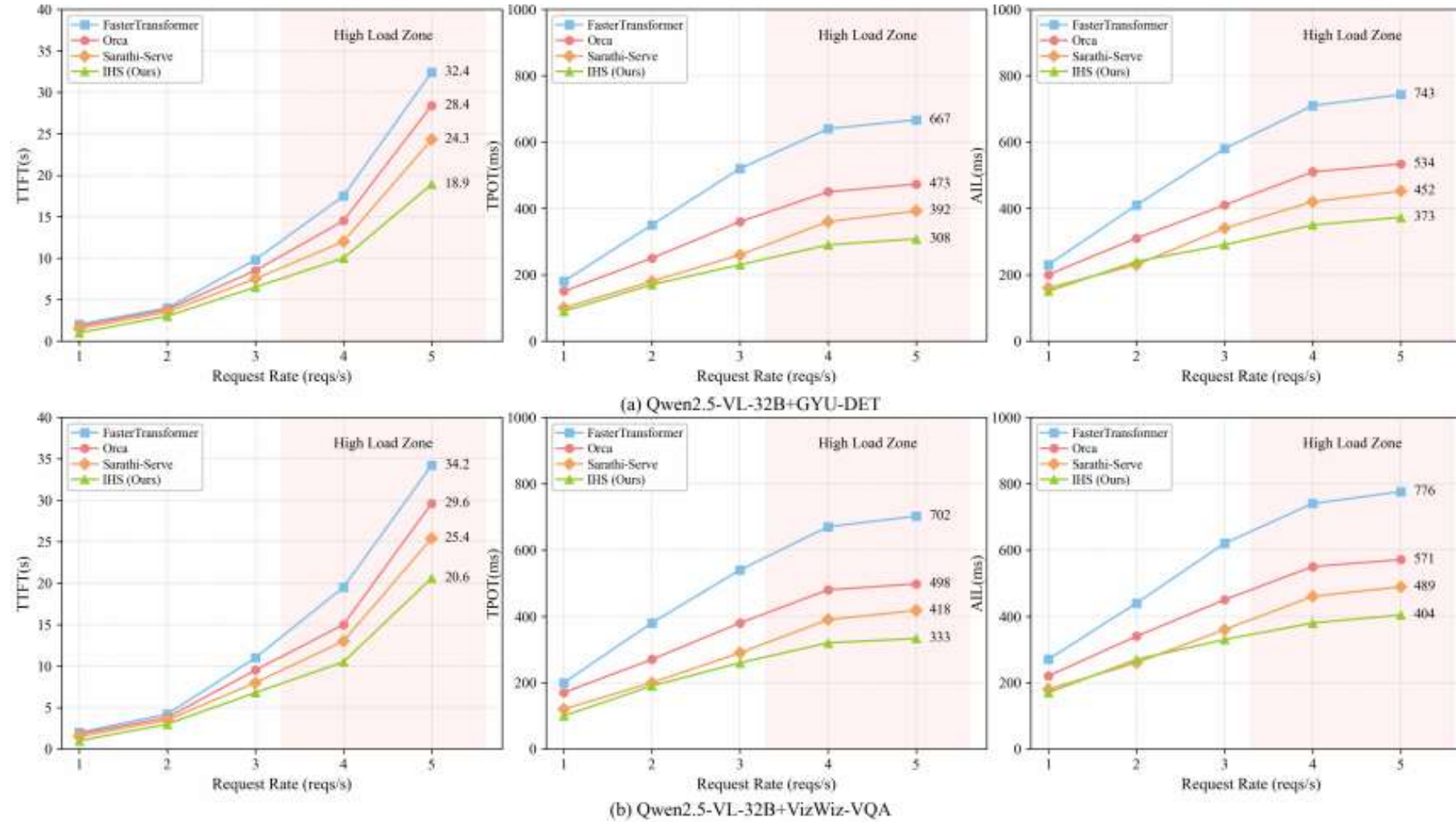


Figure 1: Performance Variation of Different Scheduling Methods Under Dynamic Load

Sensitivity Analysis of Migration Threshold γ

Table 1: Performance Comparison of Imbalance Tolerance Coefficients Under Different Models and Bandwidth Interference Levels.

Model	Bandwidth	Average Inference Latency (ms/token)			
		$\gamma = 0.05$	$\gamma = 0.10$	$\gamma = 0.15$	$\gamma = 0.20$
Qwen2.5-VL-32B	85 Mbps	328	303	306	314
	70 Mbps	366	338	324	386
	50 Mbps	387	354	341	404
Yi-VL-34B	85 Mbps	342	325	318	331
	70 Mbps	371	351	340	394
	50 Mbps	401	379	367	418

We set $\gamma = 0.15$ to balance responsiveness and stability

Analysis of BAMLM Performance

Table 2: Inference Performance Comparison of Different Methods under Heterogeneous Bandwidth Scenarios.

Model	Method	Avg. Latency (ms/token)		Accuracy (%)		Memory (GB)	
		GYU	VizWiz	GYU	VizWiz	GYU	VizWiz
Qwen2.5-VL-32B	DynaPipe	376	392	74.8	75.4	84.8	84.6
	LinguaLinked	365	378	75.2	75.9	87.2	87.1
	BAMLM	341	356	75.2	76.0	89.4	89.2
Yi-VL-34B	DynaPipe	427	434	76.8	77.4	87.1	87.2
	LinguaLinked	396	401	77.6	78.2	88.5	88.6
	BAMLM	367	374	77.4	78.1	89.8	89.7

BAMLM achieves the lowest latency while preserving accuracy with acceptable memory overhead

Overall Performance Evaluation of IHS-LM

Table 3: Overall Inference Performance Comparison of IHS-LM and Baselines.

Model	Method	Avg. Latency (ms/token)		Accuracy (%)		Memory (GB)	
		GYU	VizWiz	GYU	VizWiz	GYU	VizWiz
Qwen2.5-VL-32B	Baseline	376	392	74.8	75.4	84.8	84.6
	w/o BAMLM	358	371	75.1	75.8	89.3	89.1
	w/o IHS	341	356	75.2	76.0	89.4	89.2
	IHS-LM	328	342	75.2	76.1	89.5	89.3
Yi-VL-34B	Baseline	427	434	76.8	77.4	87.1	87.2
	w/o BAMLM	389	398	77.3	77.9	89.7	89.6
	w/o IHS	367	374	77.4	78.1	89.8	89.7
	IHS-LM	352	361	77.5	78.3	89.9	89.8

Sarathi-Serve and DynaPipe

IHS with Sarathi-Serve

BAMLM with DynaPipe

IHS-LM achieves the lowest latency while preserving accuracy, with moderate memory overhead

Outline

1. Introduction

2. Approach

3. Experimental Evaluation

4. Conclusion

Conclusion

- IHS balances prefill/decode workloads with KV-aware token scheduling
- BAMLM mitigates bandwidth-induced stage imbalance via dynamic layer migration
- Real Jetson-cluster experiments show lower latency with preserved accuracy

Thank You!

Email: 3113679296@qq.com