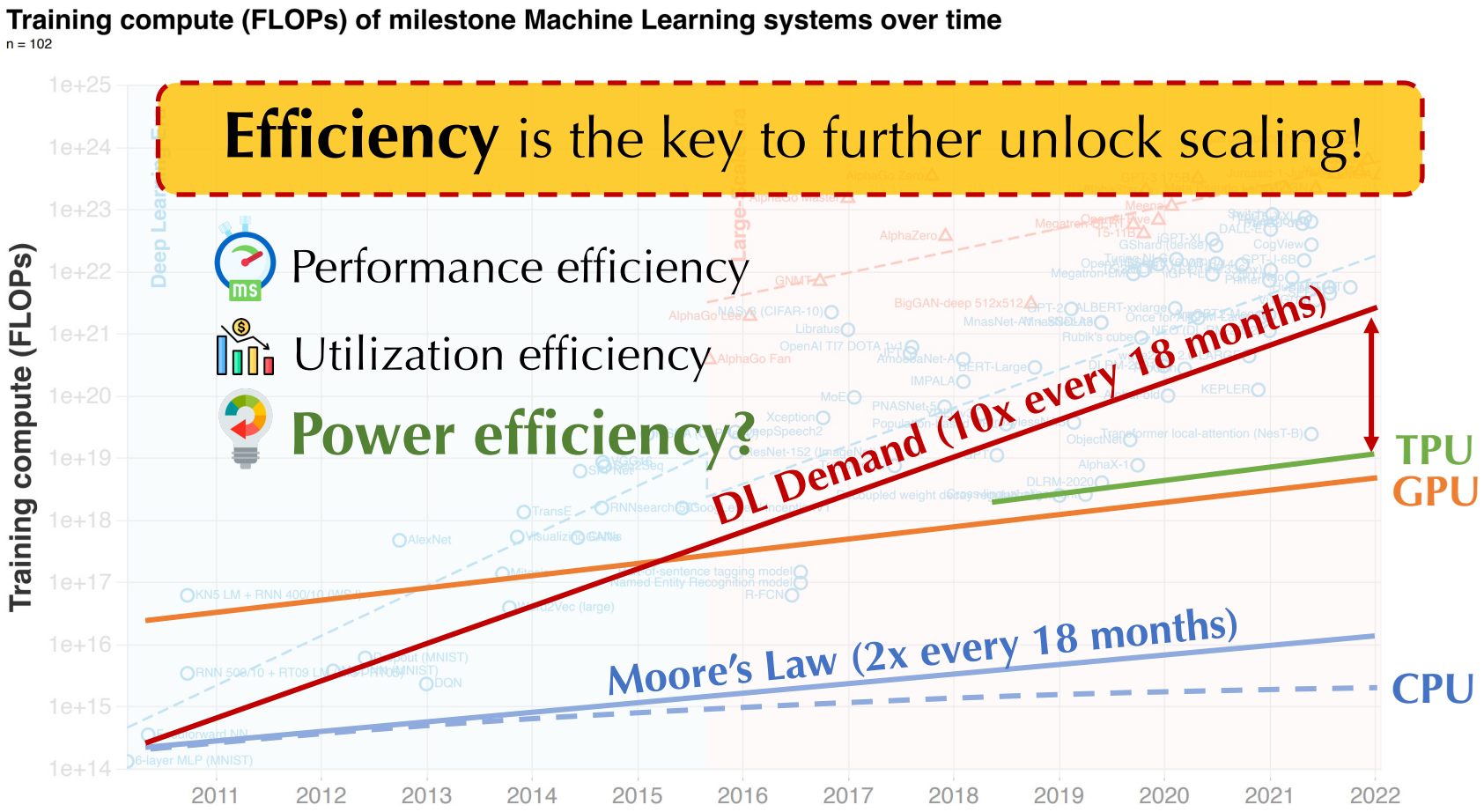
ILLINOIS IBM Research NSF

Power-aware Deep Learning Model Serving with *μ-Serve*

Haoran Qiu[1], Weichao Mao[1], Archit Patke[1], Shengkun Cui[1], Saurabh Jha[2]
Chen Wang[2], Hubertus Franke[2], Zbigniew T. Kalbarczyk[1], Tamer Basar[1], Ravishankar K. Iyer[1]
[1]UIUC [2]IBM Research

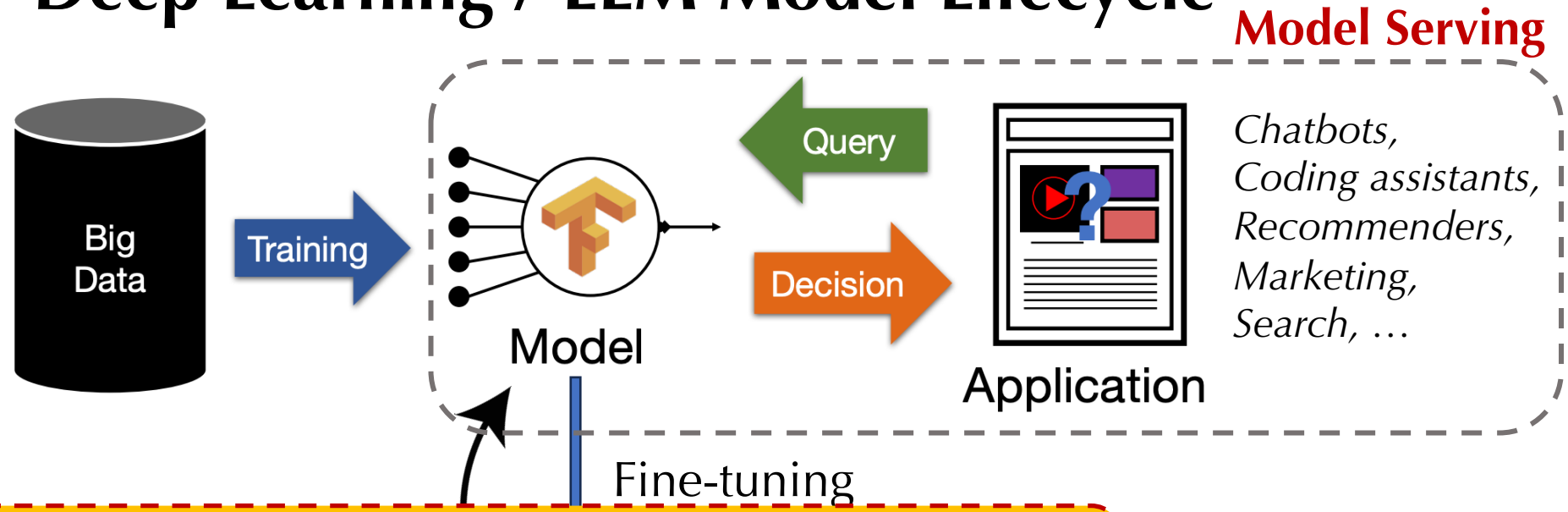
USENIX ATC 2024

Trends in Deep Learning and LLM Era



[1] *Compute Trends across Three Eras of Machine Learning*. J. Sevilla, L. Heim, et al. https://arxiv.org/abs/2202.05924

Deep Learning / LLM Model Lifecycle



Google estimate that, in 2020-22, **40%** of carbon footprint goes to **model training** while **60%** goes to **model serving**. [1]

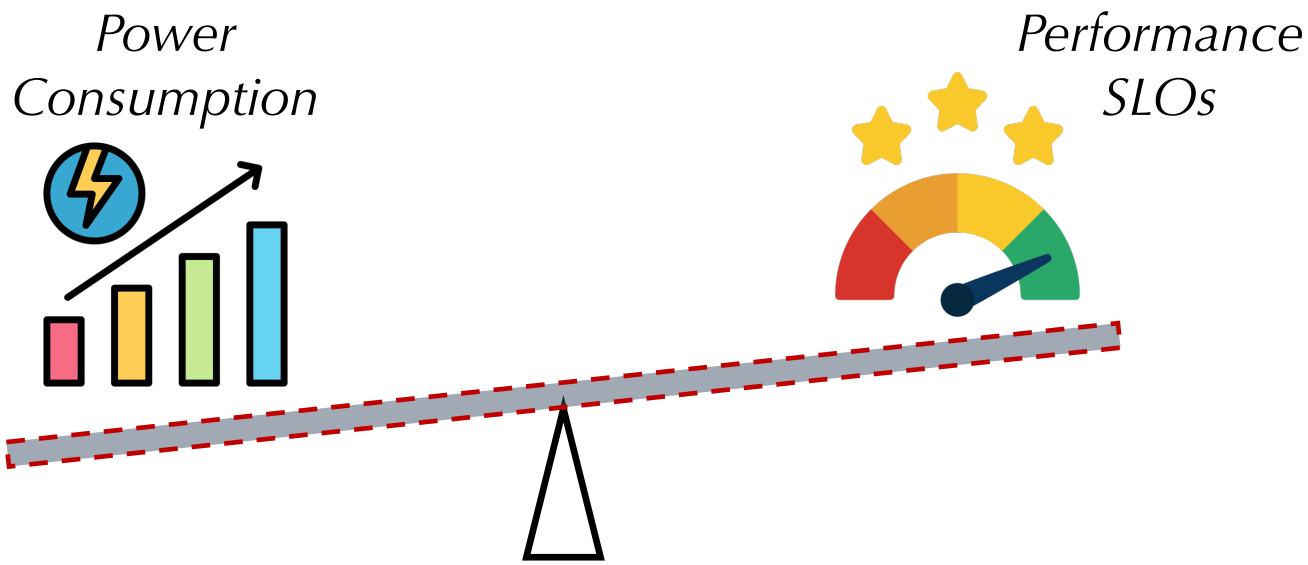
At Meta, power capacity breakdown is **10:20:70** for AI infrastructures used for **Experimentation**, **Training**, and **Inference**. [2]

[1] *Good News About the Carbon Footprint of Machine Learning Training*. https://blog.research.google/2022/02/good-news-about-carbon-footprint-of.html
[2] Sustainable AI: Environmental Implications, Challenges, and Opportunities. https://arxiv.org/abs/2111.00364

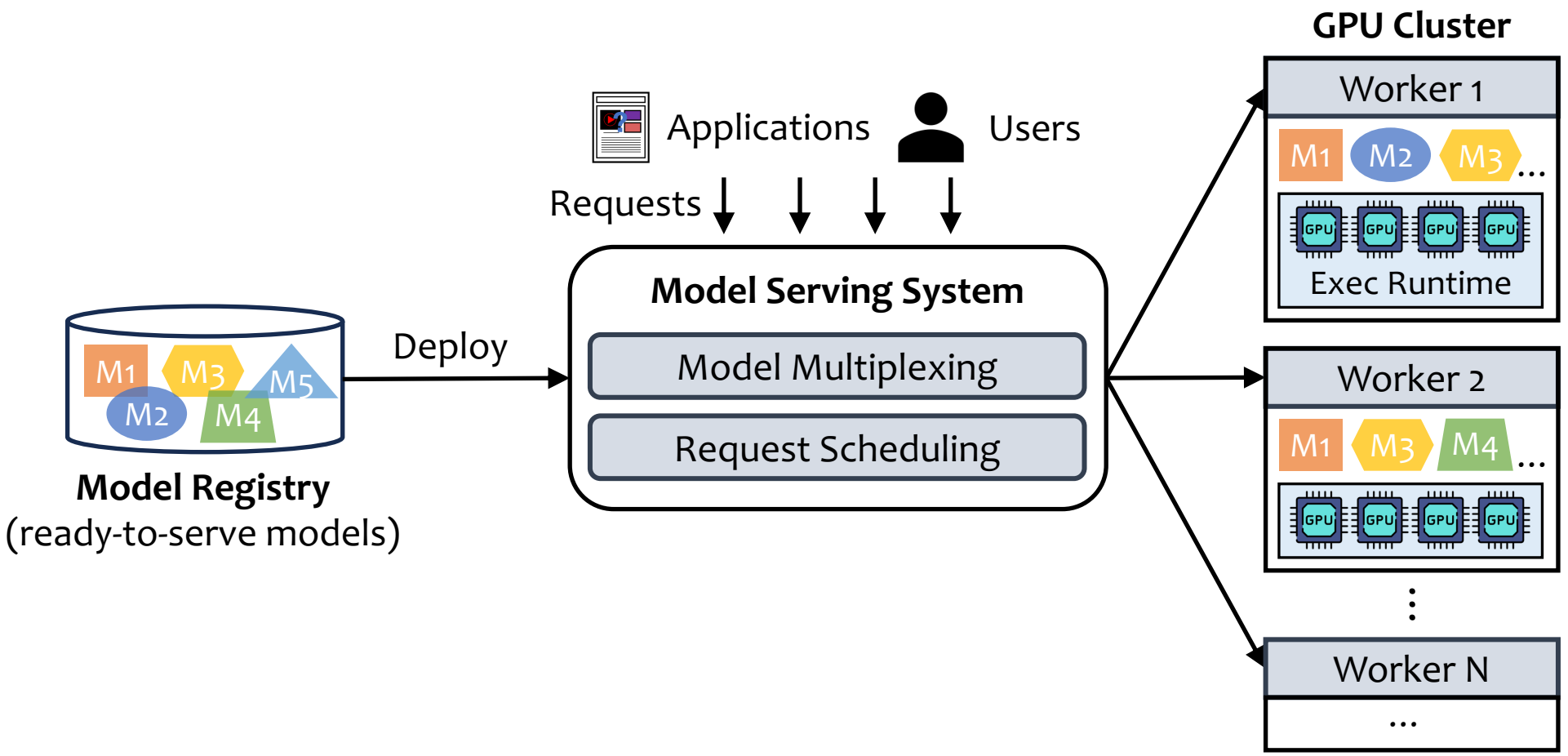
Contribution

μ-Serve is the first power-aware deep learning model serving system that achieves **1.2-2.6x power saving** while **preserving SLOs**

- Open-sourced at: https://gitlab.engr.illinois.edu/DEPEND/power-aware-model-serving

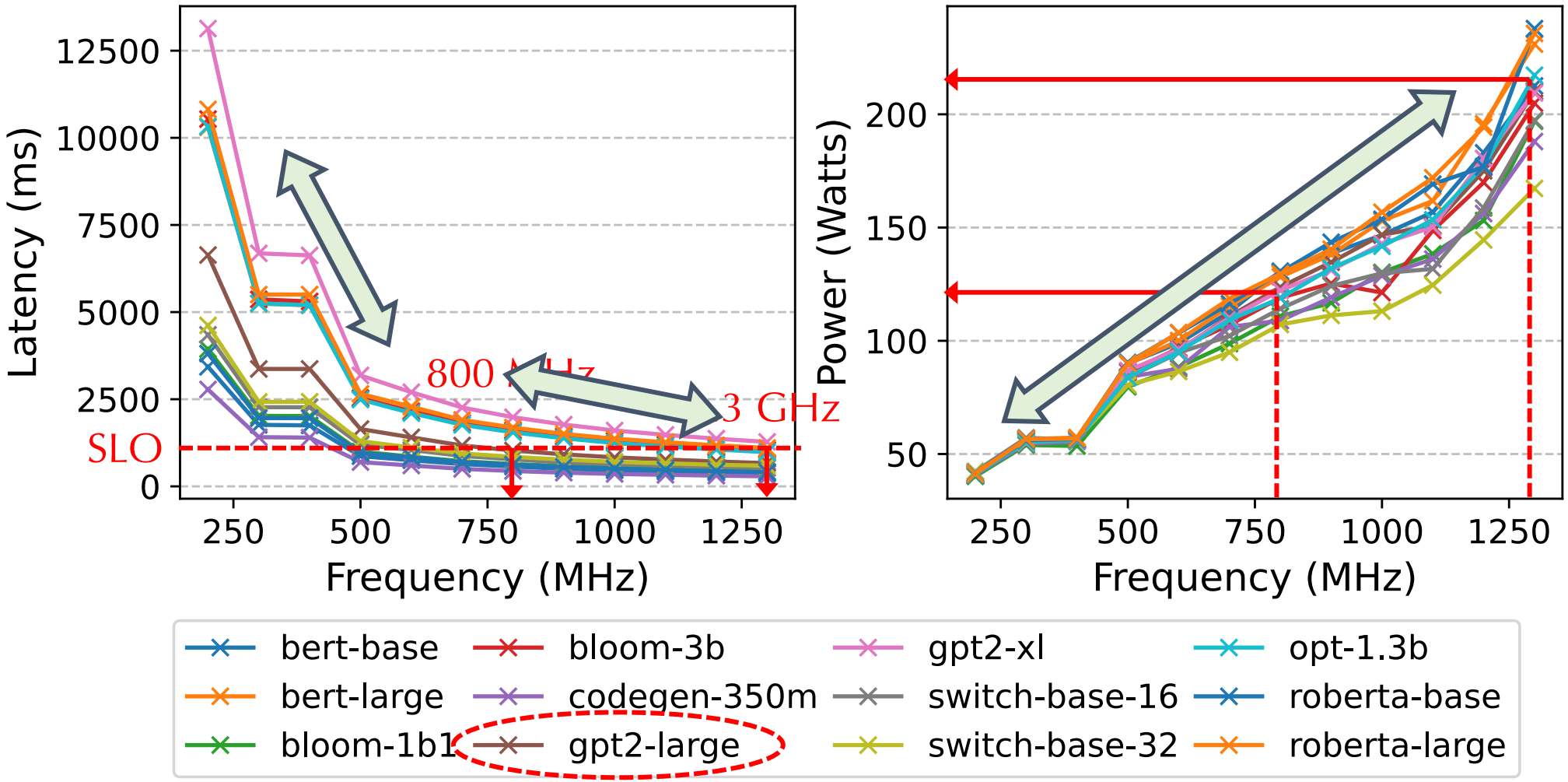


Model Serving Systems



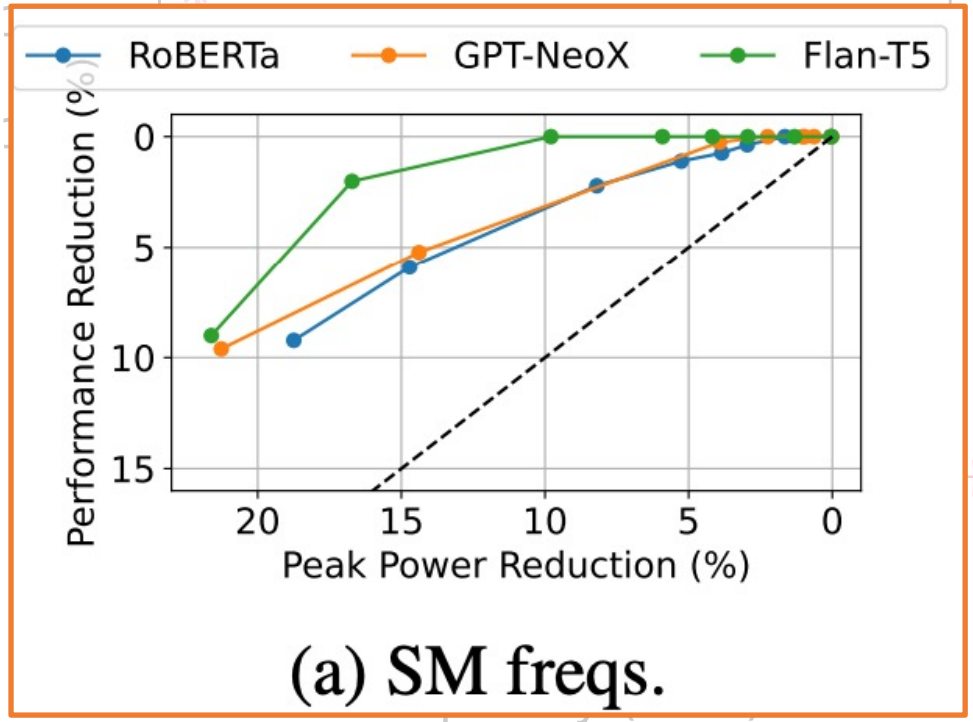
Power Saving Opportunities

214w -> 120w: **44% reduction**

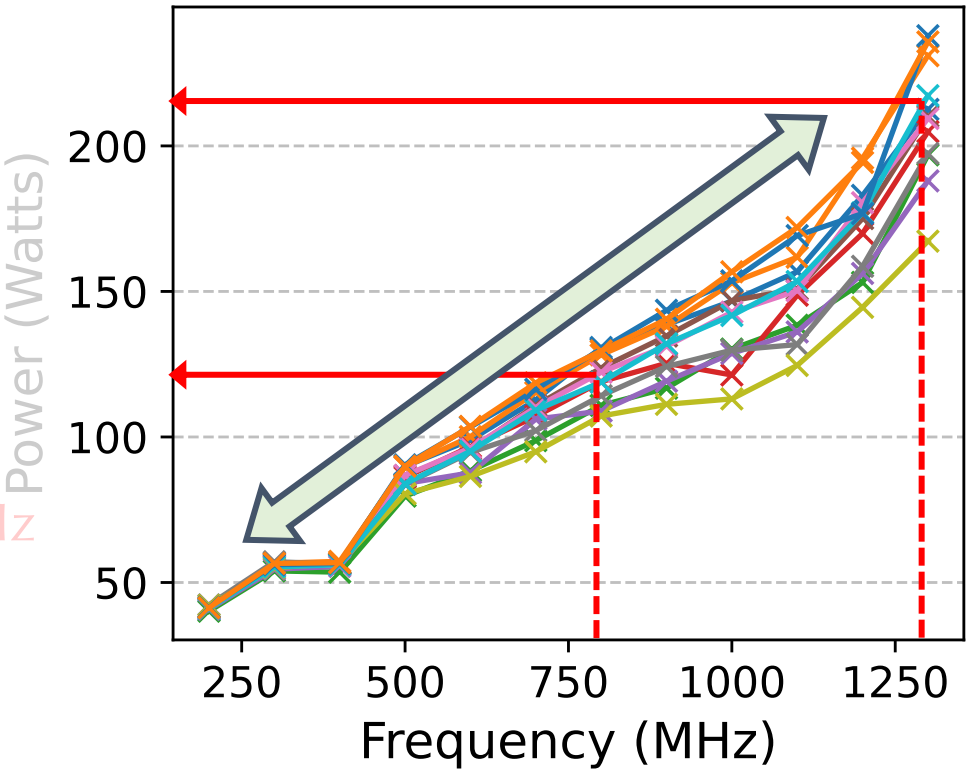


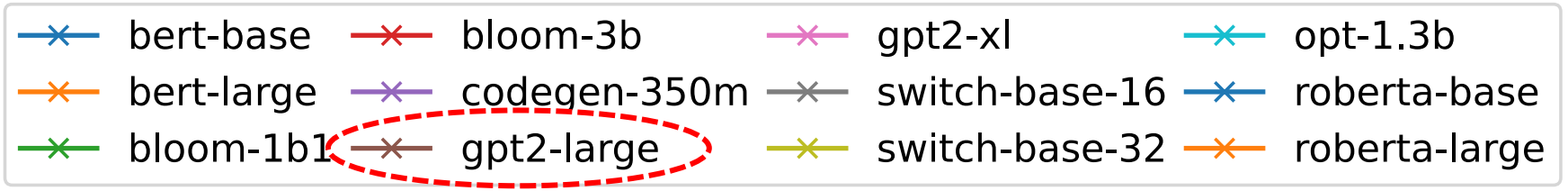
Power Saving Opportunities

Similarly observed in production POLCA (Microsoft, ASPLOS24)

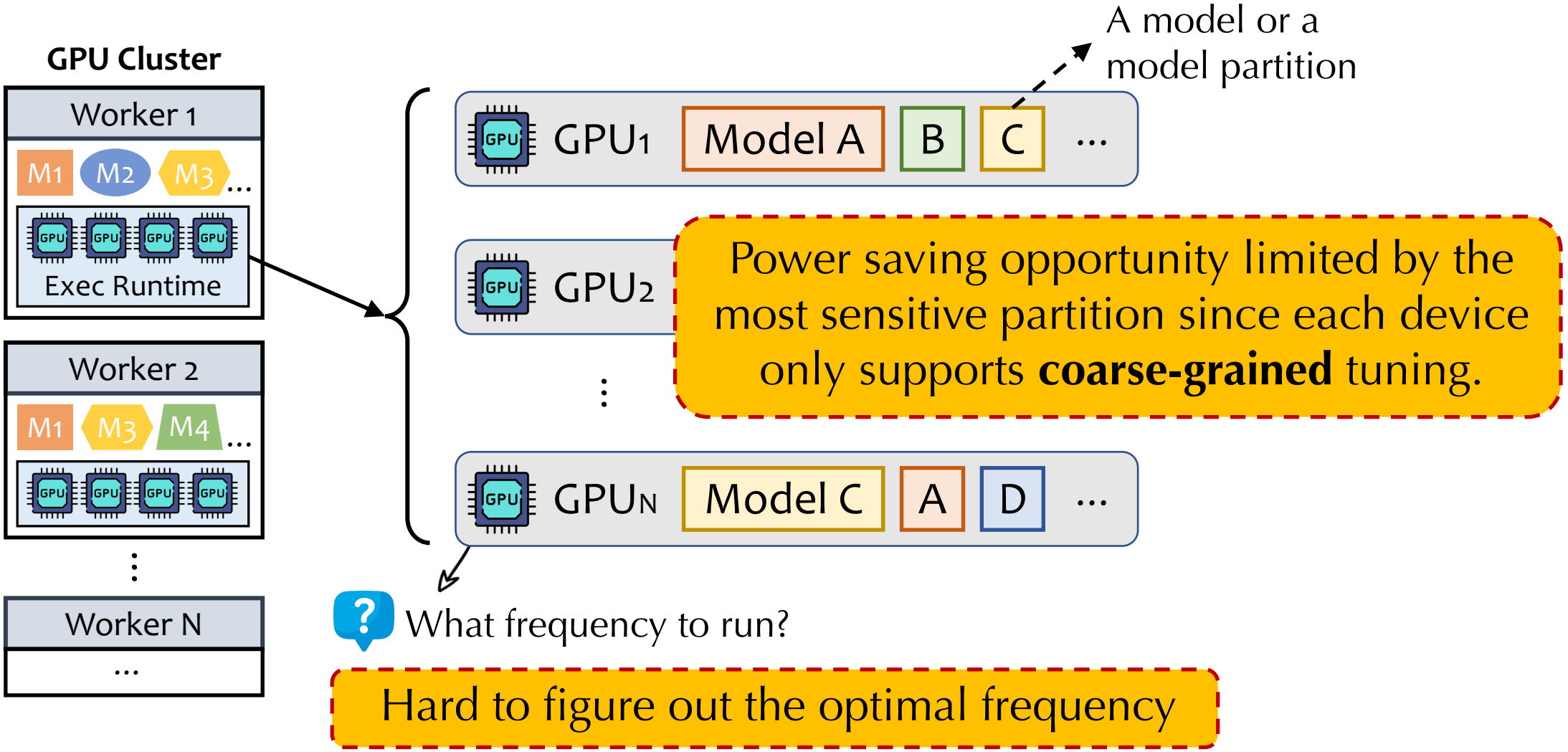


214w -> 120w: **44% reduction**





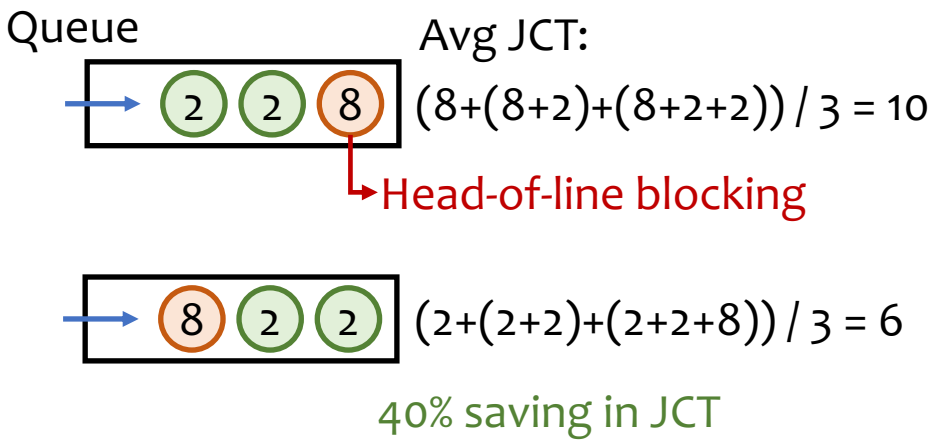
Challenge #1: Coarse-grained GPU Frequency Tuning

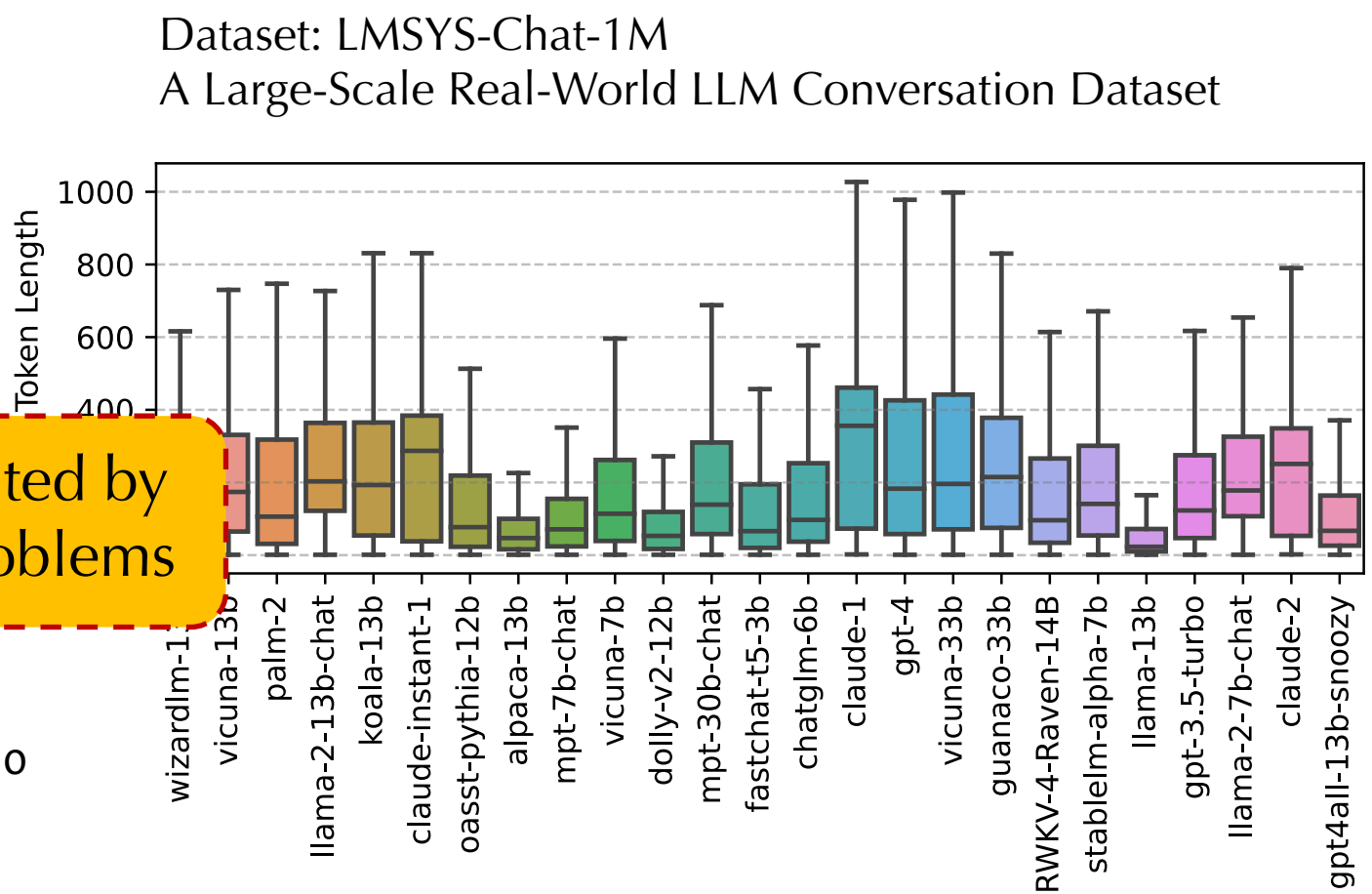


Challenge #2: Non-deterministic LLM Executions

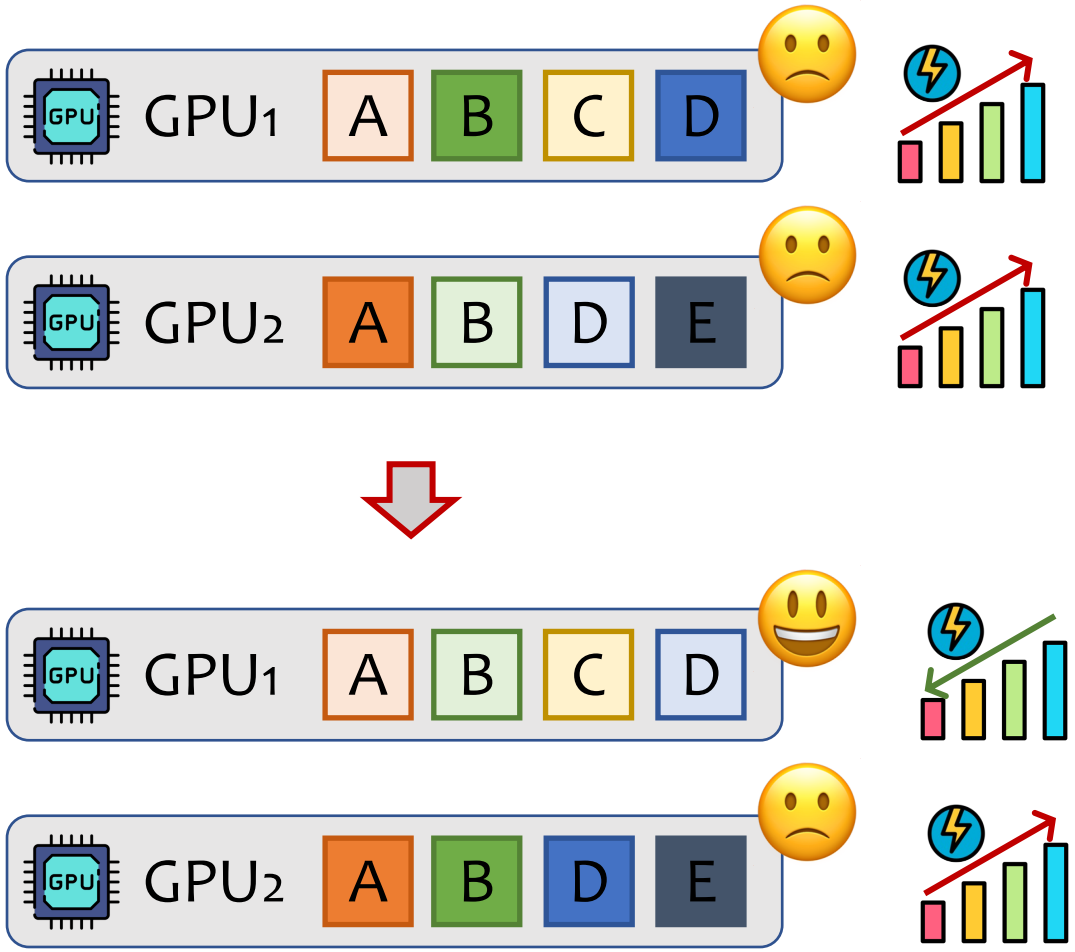
- **Autoregressive** nature of LLMs
- Can lead to head-of-line (**HoL**) blocking in FCFS
- Likely SLO violations on job completion times (JCT)

Power saving opportunity limited by **non-determinism** and HoL problems





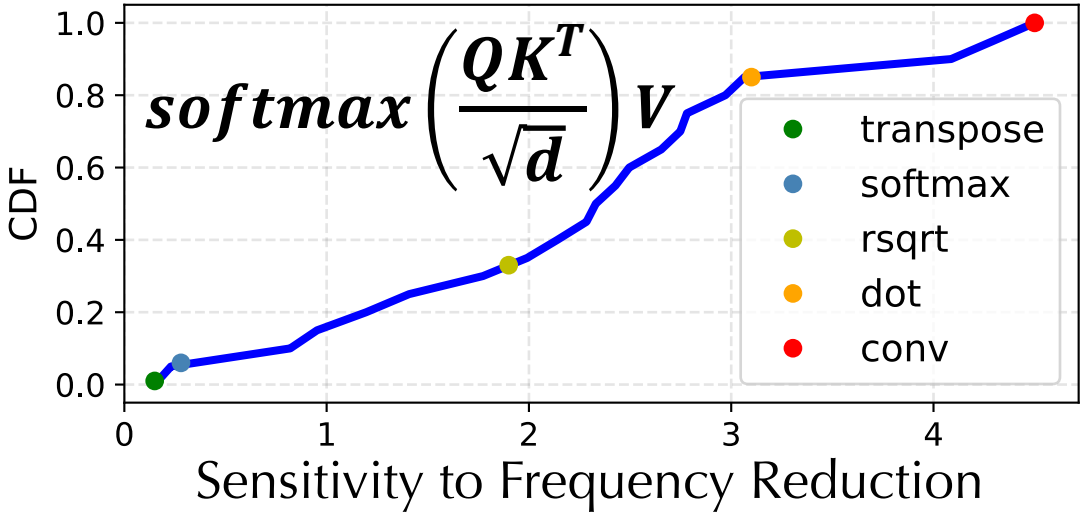
Observation #1: Model Partitions Have Diverse Sensitivities



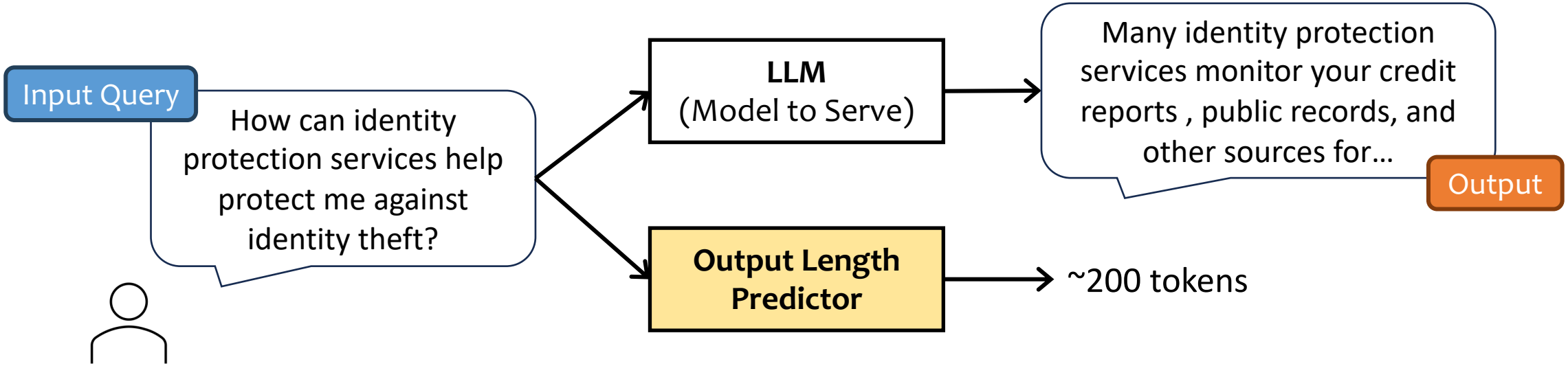
A A model or a model partition

A B C D **Less sensitive** to frequency reduction

A B D E **More sensitive** to frequency reduction



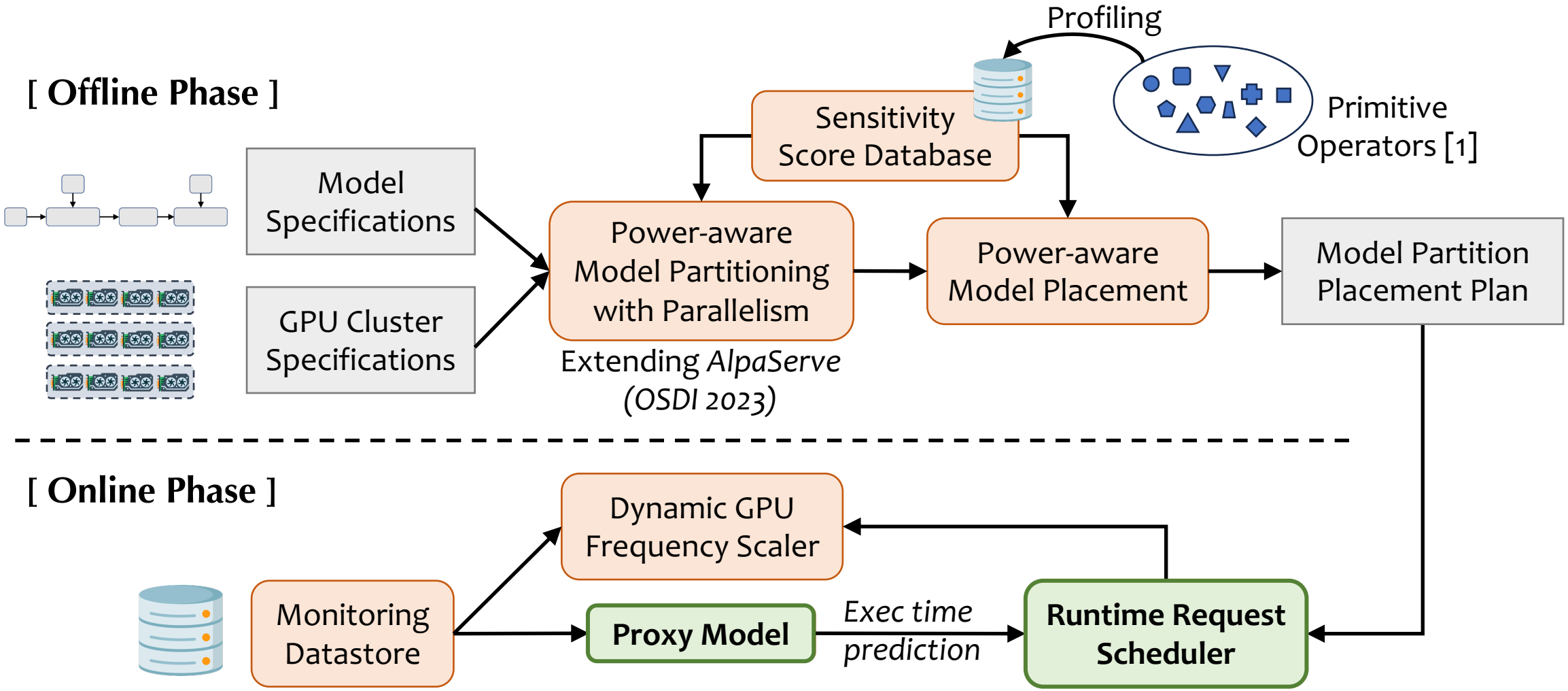
Observation #2: A Small Proxy Model Knows LLMs' Verbosity



- **A small proxy model (e.g., BERT-base/tiny) can predict well**
- Intuition: Hints on the output length (number of tokens) of LLM responses
  - "**Translate**…" -> Response length approximate to the prompt length
  - "Write an **article** about…" -> Long respo
  - "**Summarize**…" -> Shorter response than

Proxy models can indicate LLM verbosity to avoid HoL and potentially increase power-saving "**opportunities**"

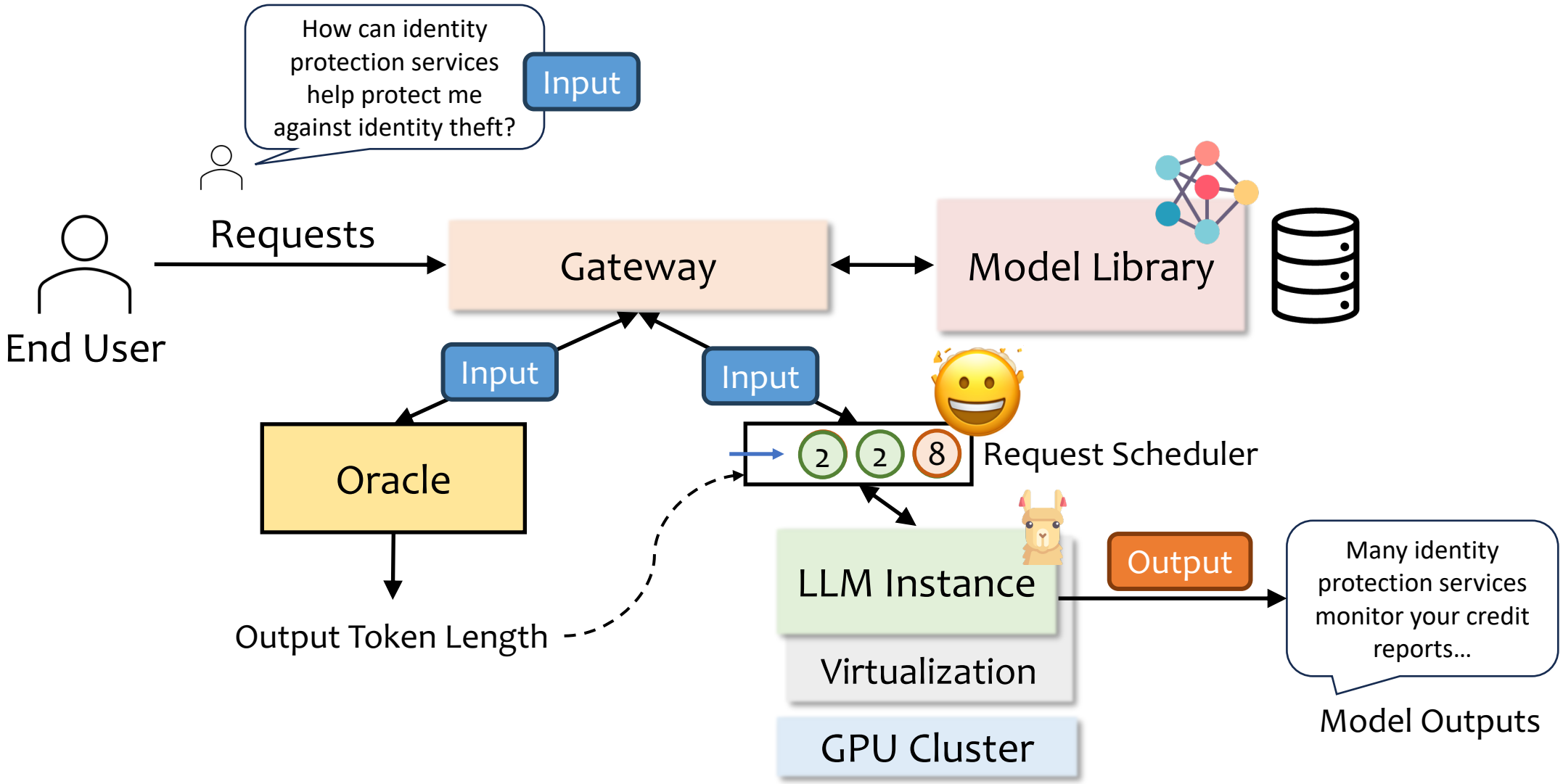
Design of μ-Serve: Power-aware DL/LLM Model Serving



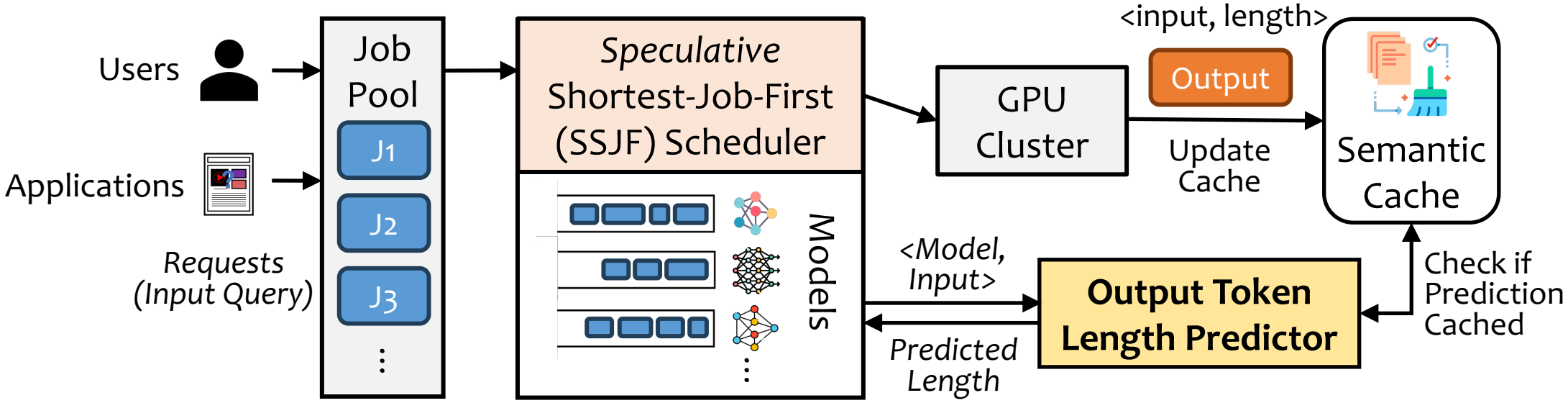
[1] XLA's HLO Representation, https://github.com/openxla/stablehlo/blob/main/docs/spec.md#ops

How to design and train a **lightweight** predictor that can *understand the behavior* of an LLM and *estimate the output token length* ***before*** *serving the request* on the LLM?

Workflow


How can identity protection services help protect me against identity theft?
Input
Requests
End User
Gateway
Model Library
Input
Input
Oracle
2
2
8
Request Scheduler
Output Token Length
LLM Instance
Virtualization
GPU Cluster
Output
Many identity protection services monitor your credit reports...
Model Outputs

SSJF: Prediction-based Shortest Job First Scheduling



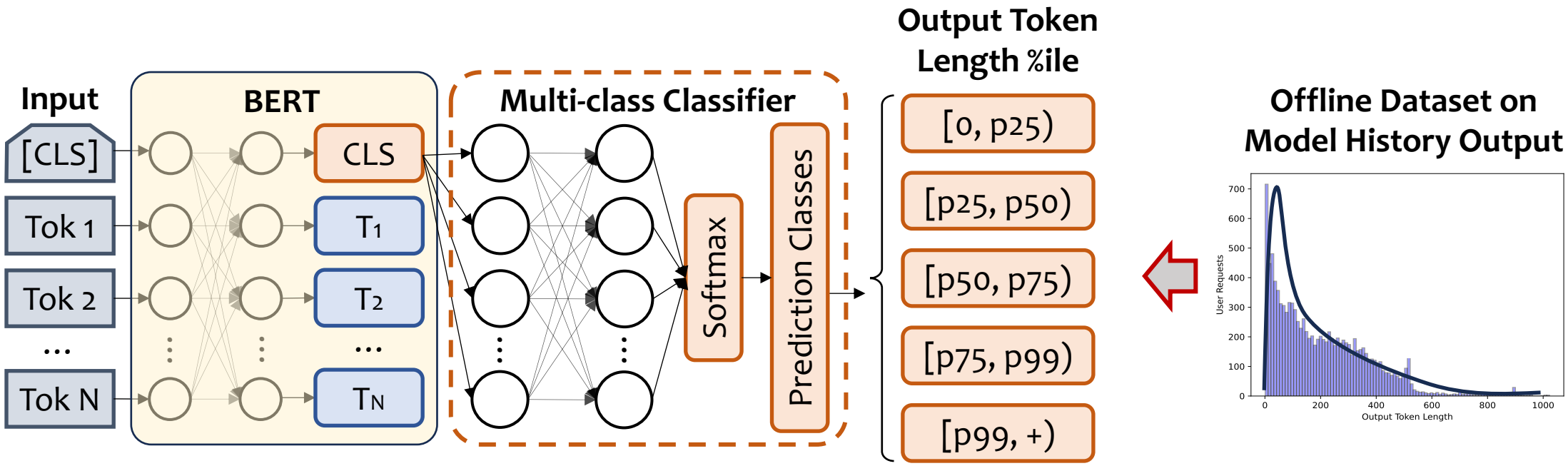
- **SSJF: Using output token length prediction as the exec time estimation**
  - Exec time = ***Const*** + ***K*** * Output token length

Model query overhead:
- E.g., input token processing

Prediction overhead:
- Deterministic inference time

K: Per-token generation latency (constant for same instance)
- GPT-3.5: 35ms
- GPT-4: 94ms
- Llama-2-7B: 19ms
- Llama-2-70B: 46ms

Proxy-model-based Predictor



How to decide X-class classification? Dependent on proxy model and LLM to serve

More number of classes leads to **low accuracy** (regression is the hardest)

Less number of classes leads to worse scheduling (too **coarse-grained**)

***Evaluation*:**
Are the predictors **lightweight**?
Are the predictors **useful in scheduling**?
How much more **power-saving opportunities** does it gain?

System and Models Setup

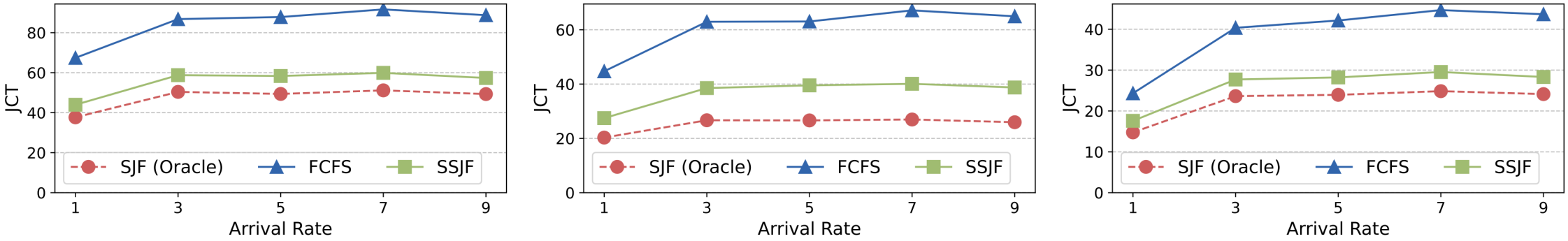
- **Platform**: AlpaServe and Ray
- **VM on IBM Cloud**: 16 vCPU 128 GiB RAM with 2x NVIDIA Tesla V100 16 GB
- Open-source LLMs and non-autoregressive models
- Model input from **LMSYS-Chat-1M** and workload patterns from **Azure LLM Traces**

Model	# of Params	Size	Latency	AR?
ResNet-50	25M	0.2 GB	51 ms	No
BERT-base	110 M	0.5 GB	123 ms	No
BERT-large	340 M	1.4 GB	365 ms	No
RoBERTa-base	125 M	0.5 GB	135 ms	No
RoBERTa-large	355 M	1.4 GB	382 ms	No
OPT-1.3b	1.3 B	5.0 GB	1243 ms	Yes
OPT-2.7b	2.7 B	10.4 GB	2351 ms	Yes
GPT2-large	774 M	3.3 GB	832 ms	Yes
GPT2-xl	1.5 B	6.4 GB	1602 ms	Yes
CodeGen-350m	350 M	1.3 GB	357 ms	Yes
CodeGen-2b	2.0 B	8.0 GB	2507 ms	Yes
Bloom-1b1	1.1 B	4.0 GB	523 ms	Yes
Bloom-3b	3.0 B	11.0 GB	1293 ms	Yes
Switch-base-16	920 M	2.4 GB	348 ms	Yes
Switch-base-32	1.8 B	4.8 GB	402 ms	Yes

Results (1): Scheduling Performance - JCT

Reduce JCT by 34.5% / 39.6% / 33.2%
Oracle by 43.7% / 58.2% / 43.0%

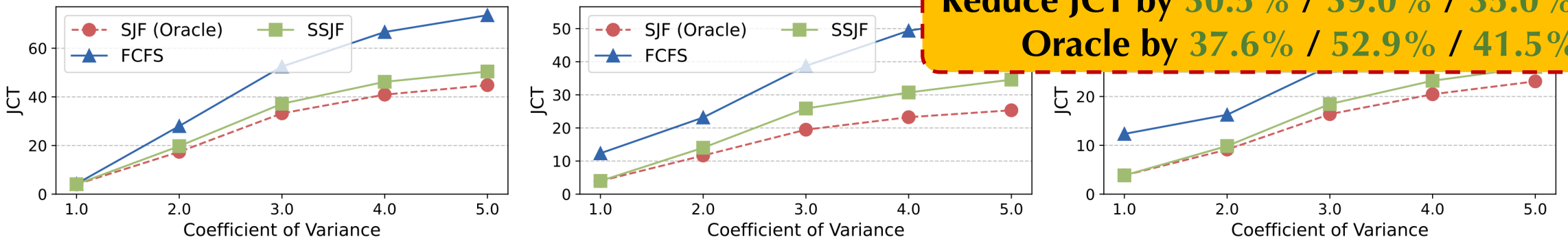
At varying rates



(a) No batching. (b) Dynamic batching. (c) Continuous batching.

At varying variations

Reduce JCT by 30.5% / 39.0% / 35.0%
Oracle by 37.6% / 52.9% / 41.5%

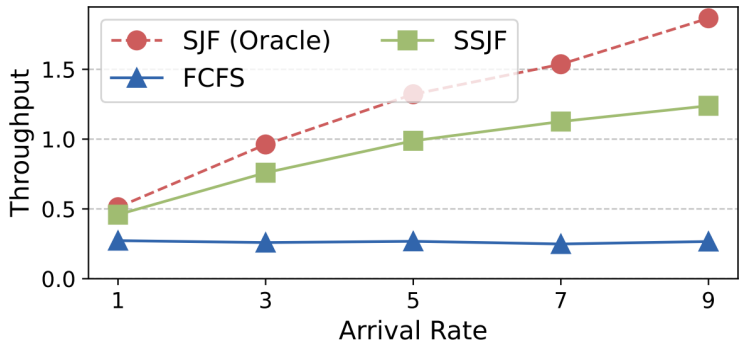


(a) No batching. (b) Dynamic batching. (c) Continuous batching.

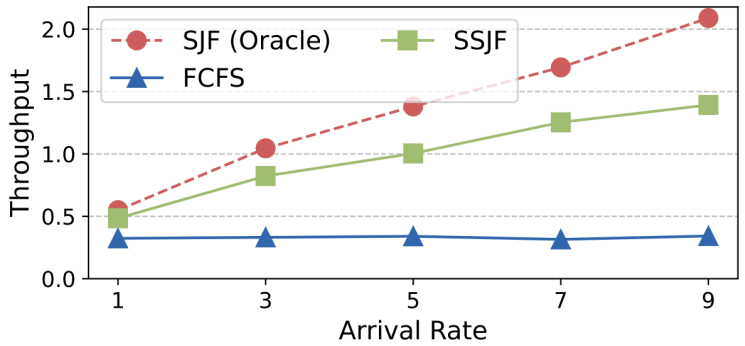
Results (2): Scheduling Performance - Throughput

↑ Throughput by 3.6x / 3.0x / 2.8x
Oracle by 4.7x / 4.1x / 3.2x

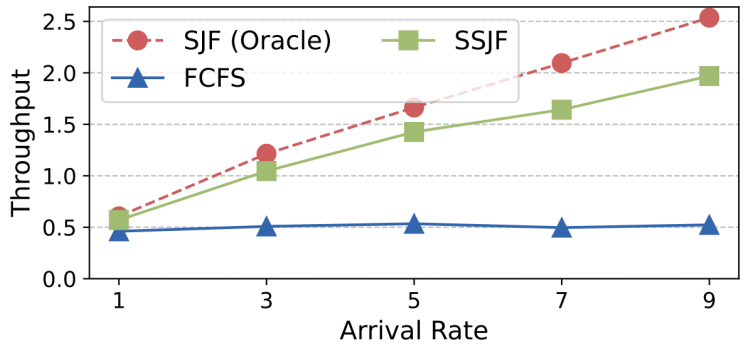
At varying rates



(a) No batching.



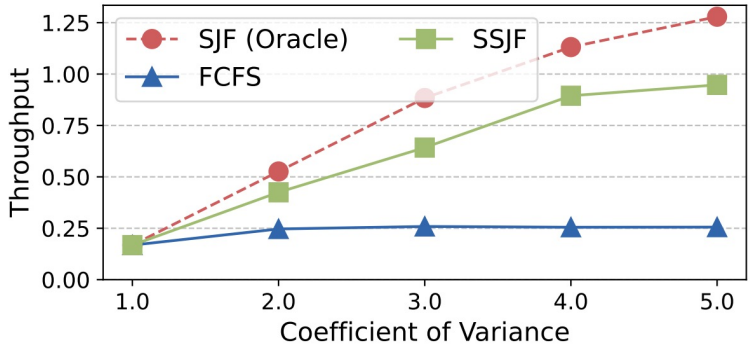
(b) Dynamic batching.



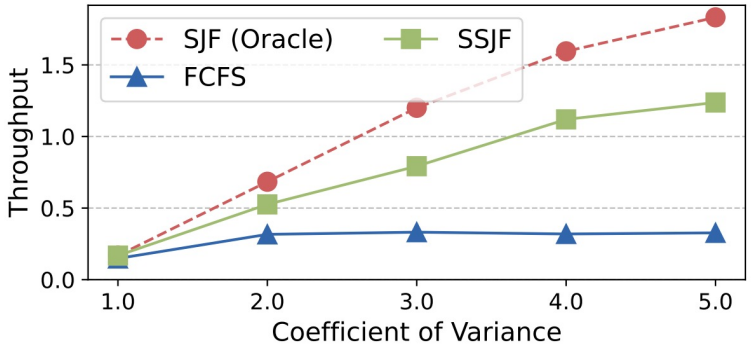
(c) Continuous batching.

↑ Throughput by 2.6x / 2.6x / 2.2x
Oracle by 3.4x / 3.8x / 2.7x

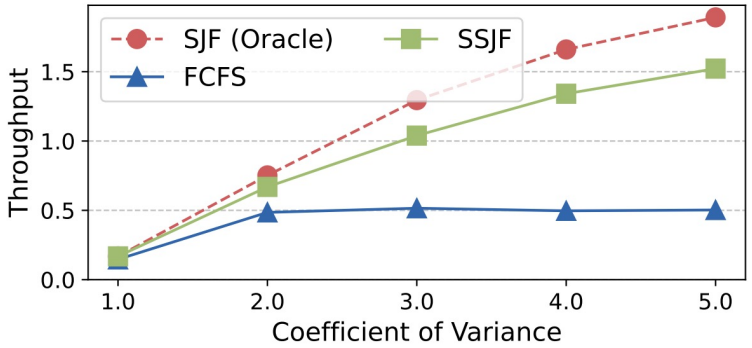
At varying variations



(a) No batching.

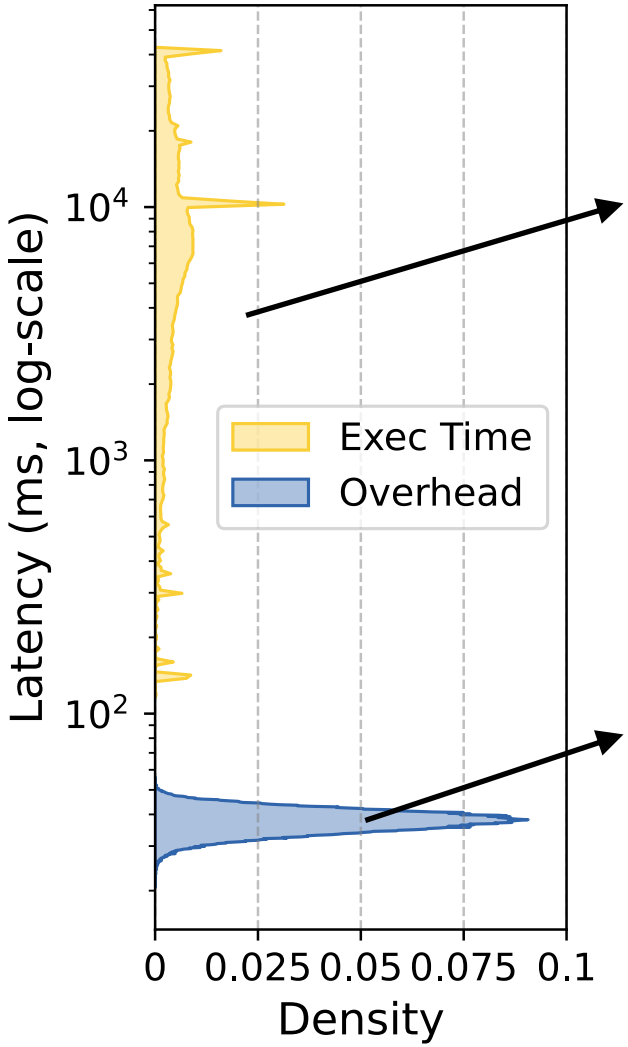


(b) Dynamic batching.



(c) Continuous batching.

Results (3): Scheduling Performance – Proxy Model Overhead



Model-serving Duration

- P5 = 360ms
- P1 = 140ms
- P0.1 = 140ms
- Min = **120ms**

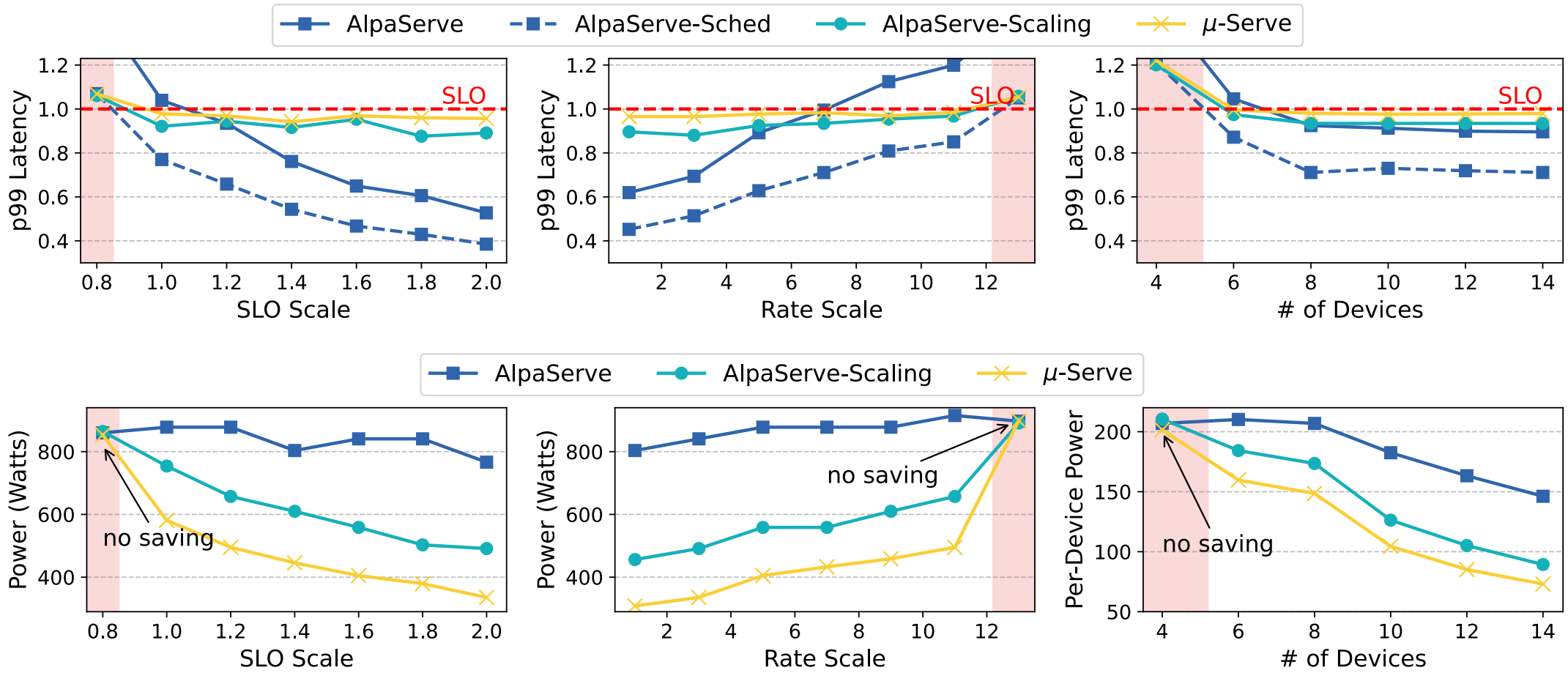
***μ*-Serve** improves JCT by **30-40%** and throughput by **2.2-3.6x** with **negligible** runtime overhead.

BERT-base Prediction Overhead

Avg Inference Latency = **7.6ms**

- Median = 7.6ms
- P99 = 8.0ms
- Max = 20.2ms

Results (4): Power Saving



Compared to AlpaServe, ***μ*-Serve** achieves **1.2–2.6x higher power saving** by dynamic frequency scaling **without** SLO attainment violations.

Summary

μ-Serve is the first power-aware deep learning model serving system that achieves **1.2-2.6x power saving** while **preserving request-serving SLOs**

- Sensitivity-score based model multiplexing and placement
- Proxy-model based model serving request scheduling
- Dynamic GPU frequency scaling to exploit power-saving opportunities

Thank you!

Haoran Qiu[1], Weichao Mao[1], Archit Patke[1], Shengkun Cui[1], Saurabh Jha[2]

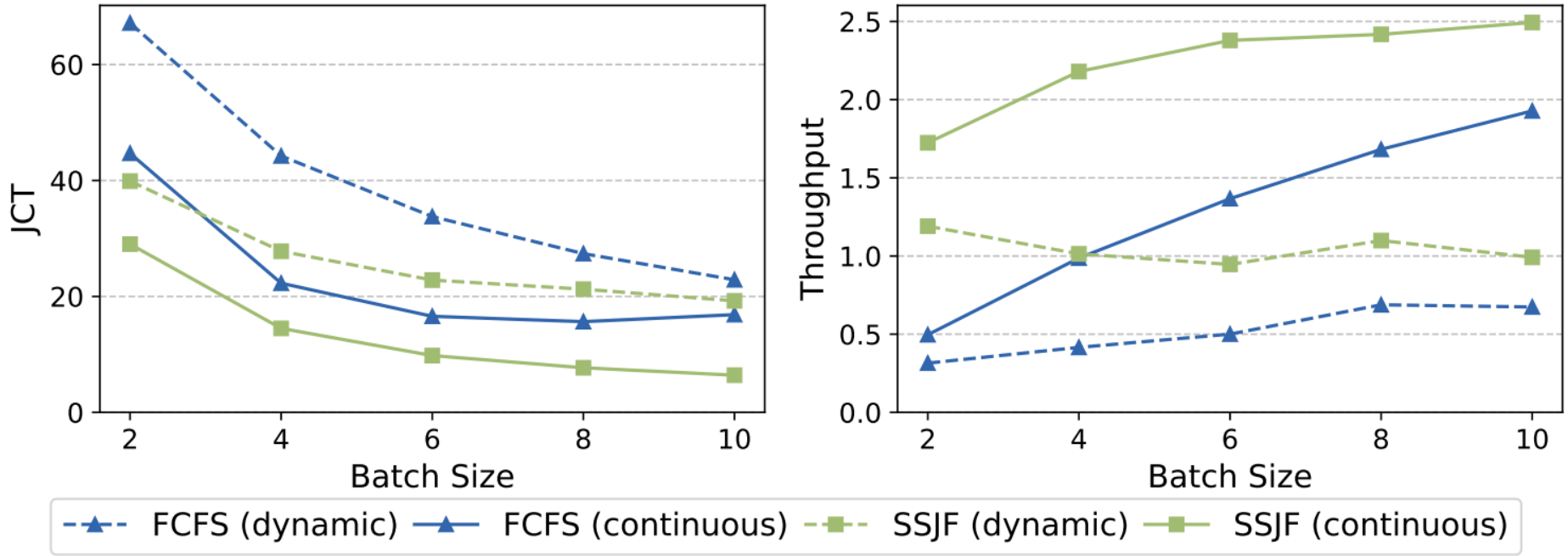
Chen Wang[2], Hubertus Franke[2], Zbigniew T. Kalbarczyk[1], Tamer Basar[1], Ravishankar K. Iyer[1]

ILLINOIS IBM Research NSF

Backup Slides

Results (5): At Varying Batch Sizes

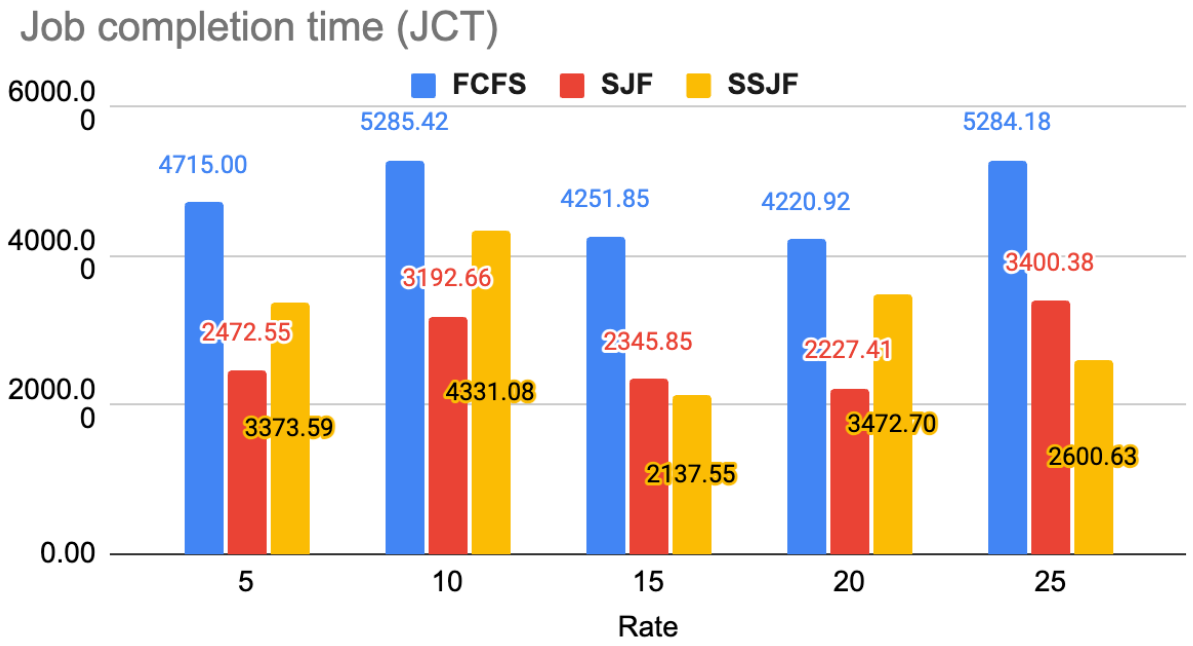
***μ*-Serve** continues to provide **improvement in JCT and throughput** under **various batch sizes** with a diminishing return.

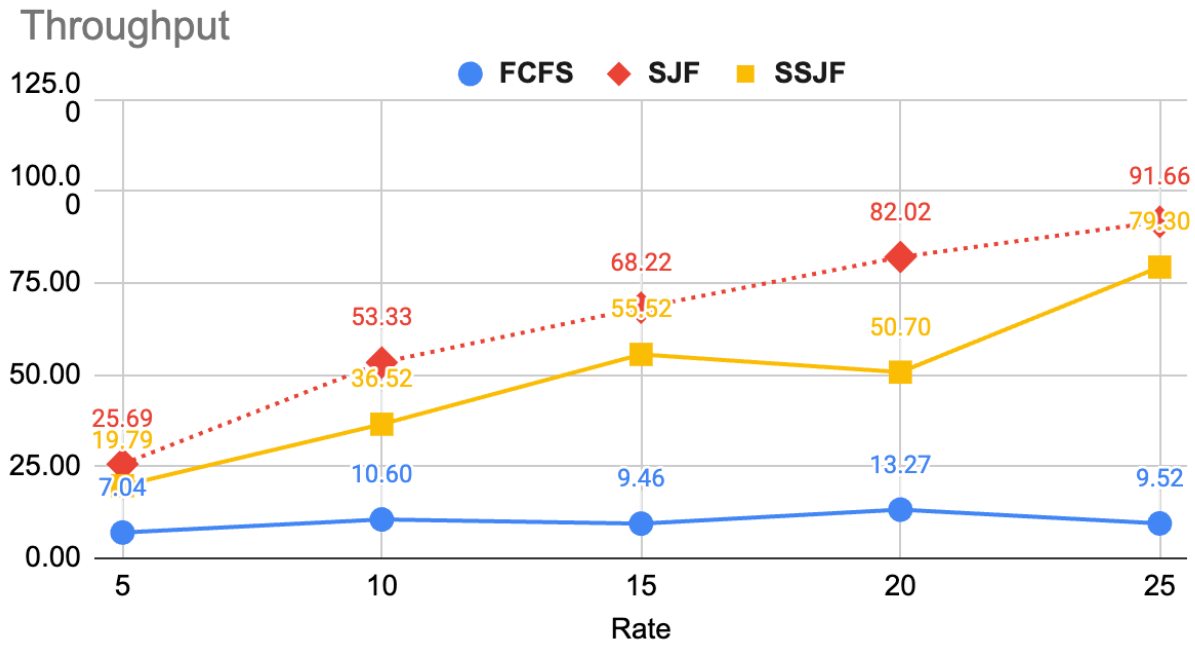


Continuous (iterative) batching > dynamic batching (same observation as in Orca, OSDI 22)

Results (6): Integration with vLLM

- Model: facebook/opt-350m, max memory usage: 23.6 GB, 75-85% SM utilization



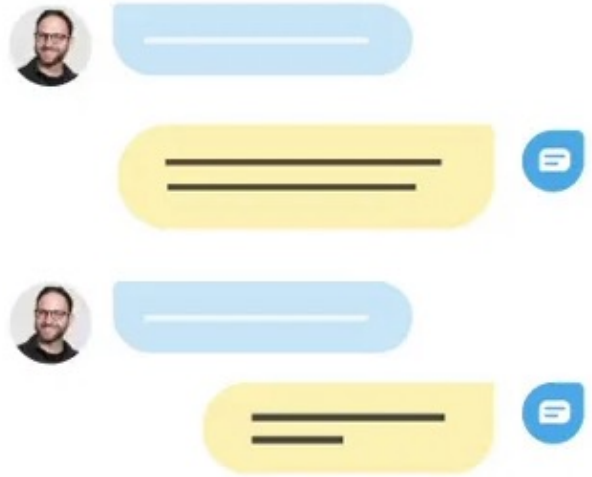


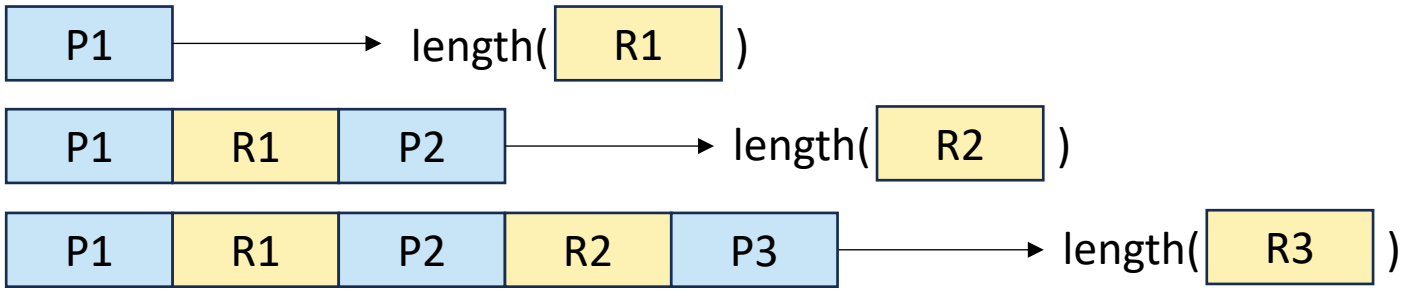
SJF (oracle) achieves **43%** and **6.3x** improvement in **JCT** and **throughput** than FCFS.

***μ*-Serve** (SSJF) achieves **33%** and **4.9x** improvement in **JCT** and **throughput** than FCFS.

Supporting Multi-round Conversations

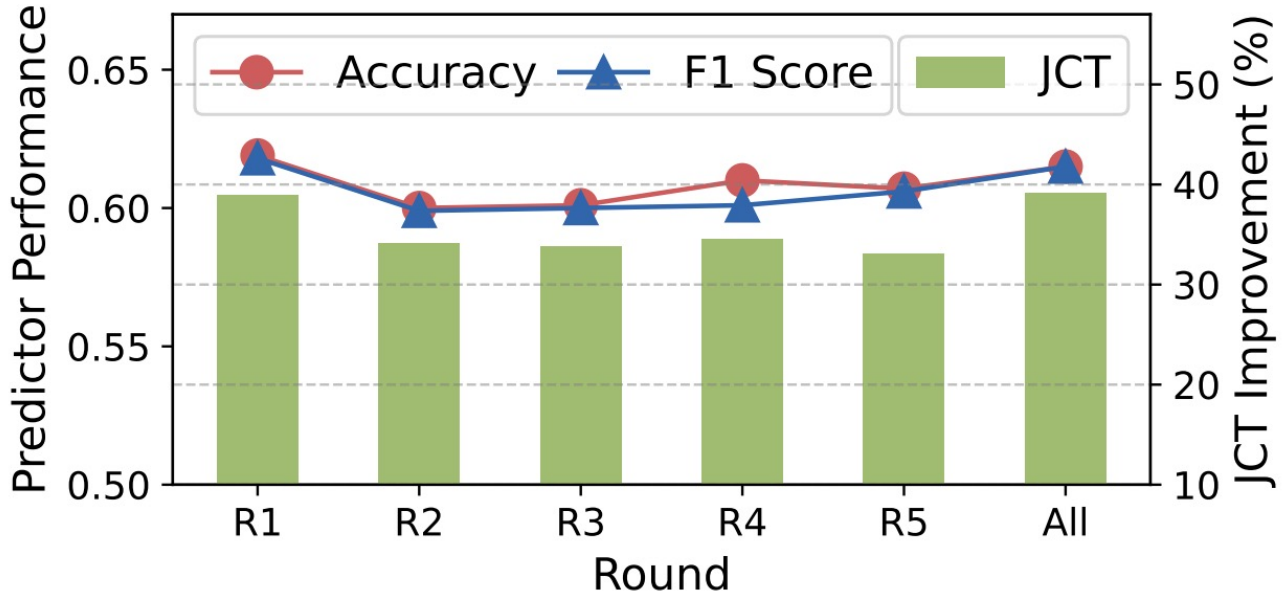
- Interactions with LLMs usually take multiple rounds:
  - Follow up questions in the same context
  - Previously: Output token prediction only for the first round
- **Simple strategy: Concatenation + truncation**
  - Concatenate all history **user prompts** and **LLM responses**





  - Truncate context longer than 512 tokens (BERT's limit)
  - **Head (128) + tail (382)** [Sun et al., 2019]:

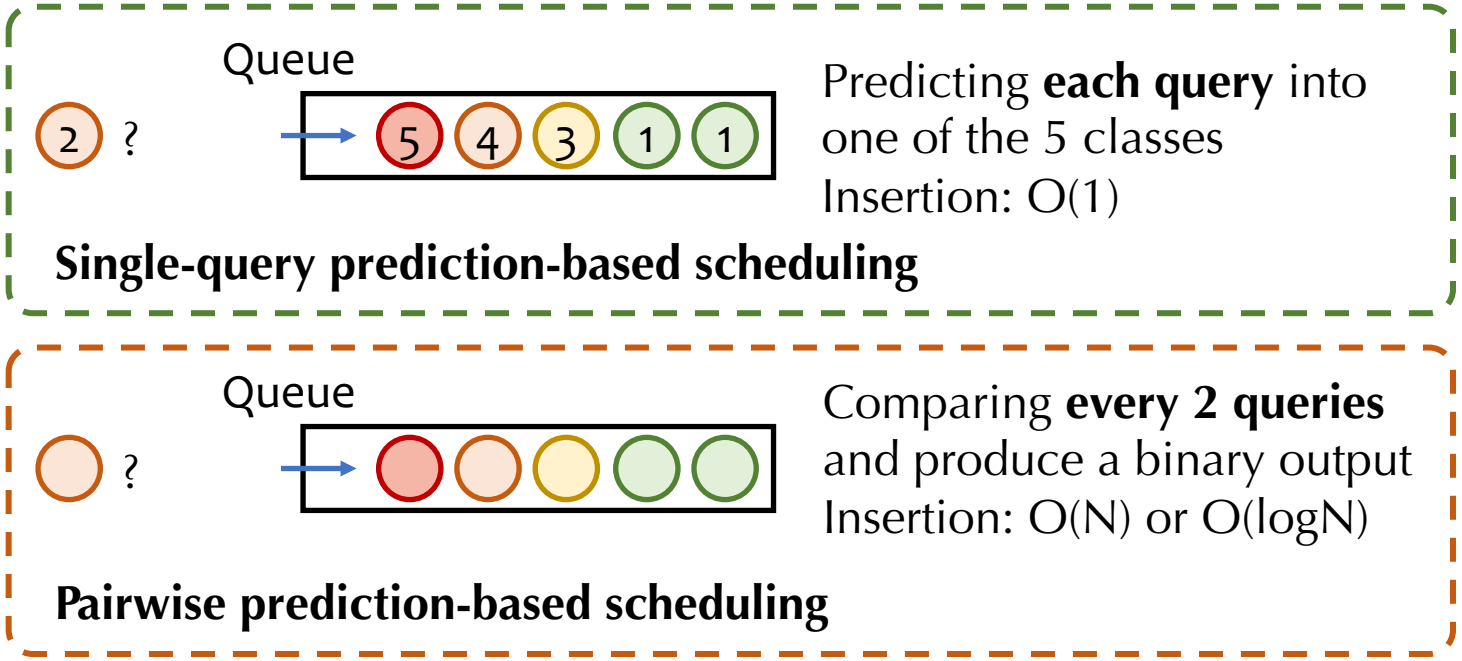
Results(7): Evaluation across conversation rounds



SSJF performance is consistent across rounds in interactive, multi-round conversation settings.

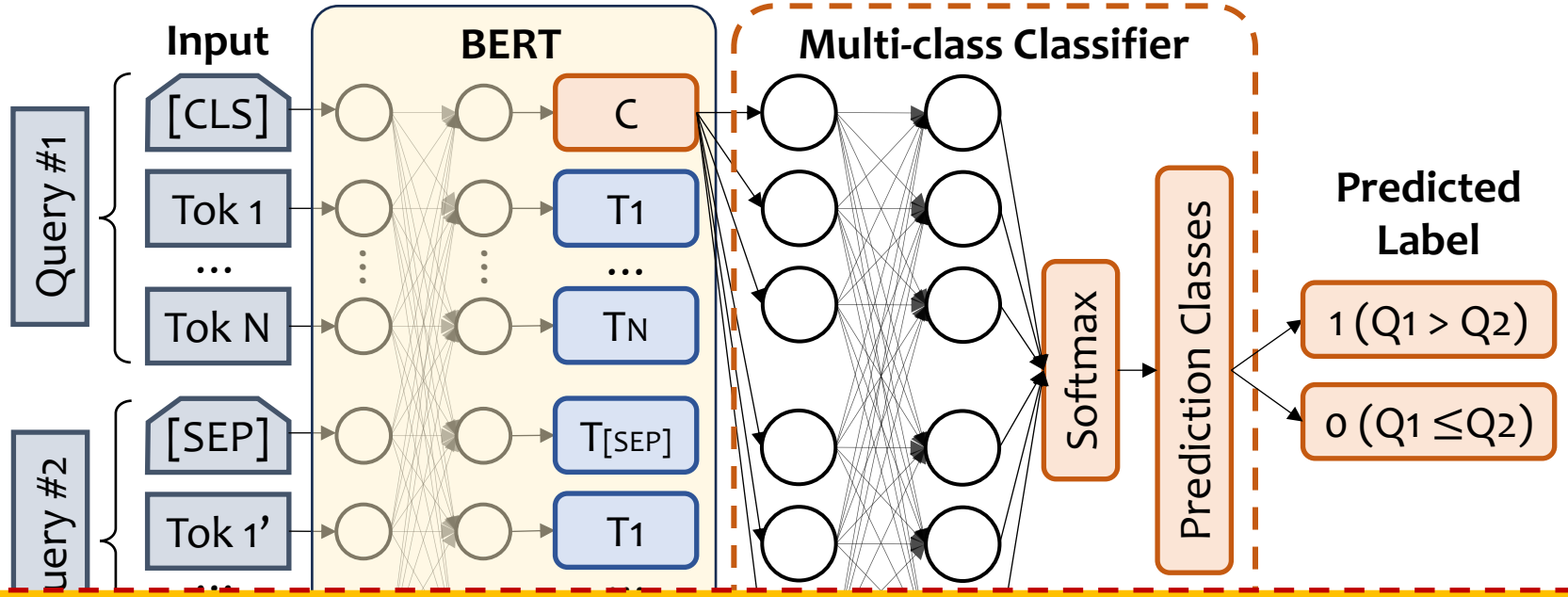
(Alternative) Pairwise Prediction: Which input leads to longer output?

- Can we have a pairwise predictor?
  - Give input query A, B, it can predict which one leads to a longer output
  - Rationale: We don't need the exact prediction for every single query, but only want to know the ranking between every pair of queries in the queue



(Alternative) Pairwise Prediction: Which input leads to longer output?

- We slightly change the model architecture here to support two input queries as input, using the [SEP] token in between to separate the two.
- Binary output: 1 representing Q1's output > Q2' and 0 otherwise



Pairwise predictor **does not help with scheduling** (because pairwise prediction is neither commutative nor transitive) and thus we do not proceed with this design choice.

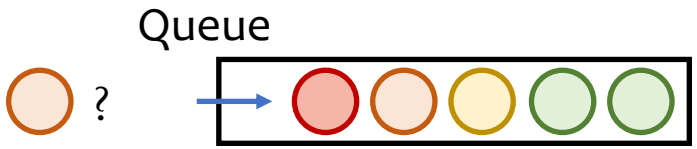
Pairwise Prediction Accuracy

- Prediction performance:
  - Accuracy: 0.69
  - Precision: 0.69
  - Recall: 0.71
  - F1: 0.69
- Compared to Binary Classification (single-query prediction)
  - Accuracy: 0.72
  - Precision: 0.74
  - Recall: 0.69
  - F1: 0.71

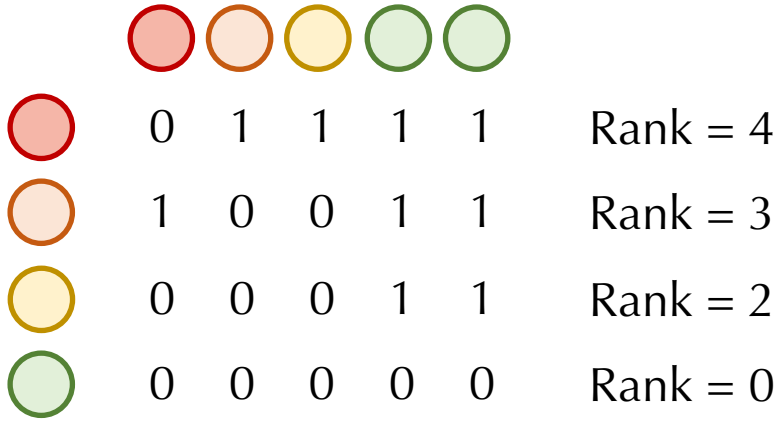
The accuracy of the pairwise predictor **does not improve (slightly worse)** compared to single-query predictor with binary classification (0.69 < 0.72)

Does Pairwise Prediction Help with Scheduling?

- FCFS: Avg JCT: 185.16 Avg Throughput: 0.1
- SJF (Oracle): Avg JCT: 92.45 Avg Throughput: 0.31
- **SJFP: Avg JCT: 179.2 Avg Throughput: 0.1 (using pairwise predictions to insert)**
  - **Only 3.2% Improvement on top of FCFS**
- SJFP: Avg JCT: 179.9 Avg Throughput: 0.11 (using *ranks* within the wait queue)
- SJFP: Avg JCT: 183.42 Avg Throughput: 0.1 (using *ranks* globally in a batch)
- **The pairwise comparison is not commutative and not transitive**



Comparing **every 2 queries** and give binary output
Insertion: O(N) or O(logN)



Pairwise predictor **does not help with scheduling** and thus we do not proceed with this design choice.

Data Cleaning

- A considerable number of samples (~4%) have “empty” responses

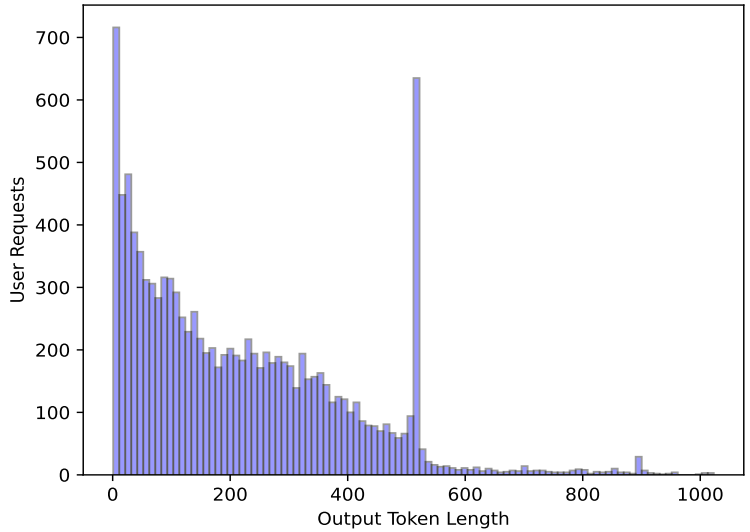
```
 {'content': 'tienes algún enlace con un tutorial de como analizar documentos?', 'role': 'user'}
 {'content': '', 'role': 'assistant'}
 {'content': 'estás ahí?', 'role': 'user'}
 {'content': '', 'role': 'assistant'}]
```

```
[{'content': "You are the text completion model and you must complete the assistant answer below, only s
end the completion based on the system instructions.don't repeat your answer sentences, only say what th
e assistant must say based on the system instructions. repeating same thing in same answer not allowed.\
nuser: Who are you?\nassistant: ", 'role': 'user'}
 {'content': 'I', 'role': 'assistant'}]
```

- Predictor accuracy significantly improves after removing all such samples

Data Cleaning

- Further observation: Some sample responses were **truncated** to 512 tokens in lmsys-chat-1M, while others were not, even for the same LLM model




```
ism.\n\nAnother security concern with IEEE 802.11 WLANs is the vulnerability of the access points themse
lves. Because APs are connected to wired networks, they can be used as a point of entry for attackers to
 gain access to the wired network. To address this issue, IEEE 802.11 includes security measures such as
 the 80', 'role': 'assistant'}]
```

- **Data cleaning: Remove all samples whose response length is <= 1 or >=512**

Power Saving Evaluation (w/o SSJF Scheduler)