The Grainger College of Engineering
Coordinated Science Laboratory

The Grainger College of Engineering
IBM-Illinois Discovery Accelerator Institute

Unique Cybertwin to Model and Design Sustainable Robust Clouds

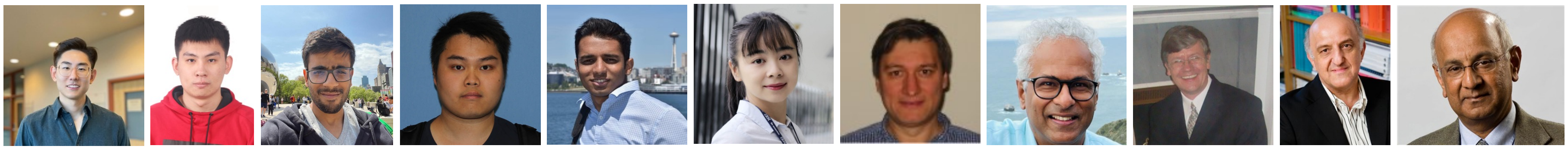
Ravishankar (Ravi) K. Iyer

Electrical and Computer Engineering, Computer Science and

The Coordinated Science Laboratory

University of Illinois at Urbana-Champaign

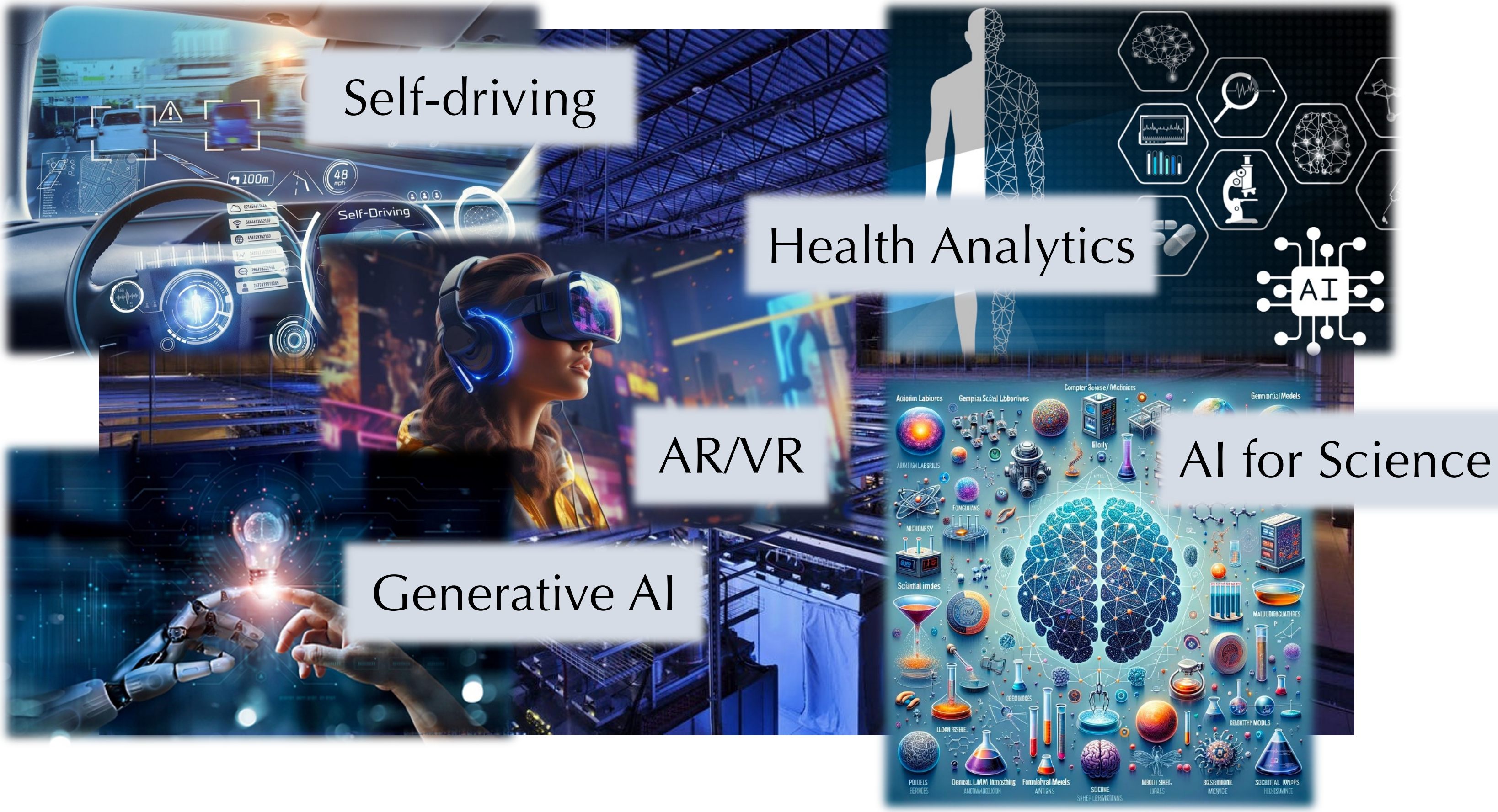
Our Team

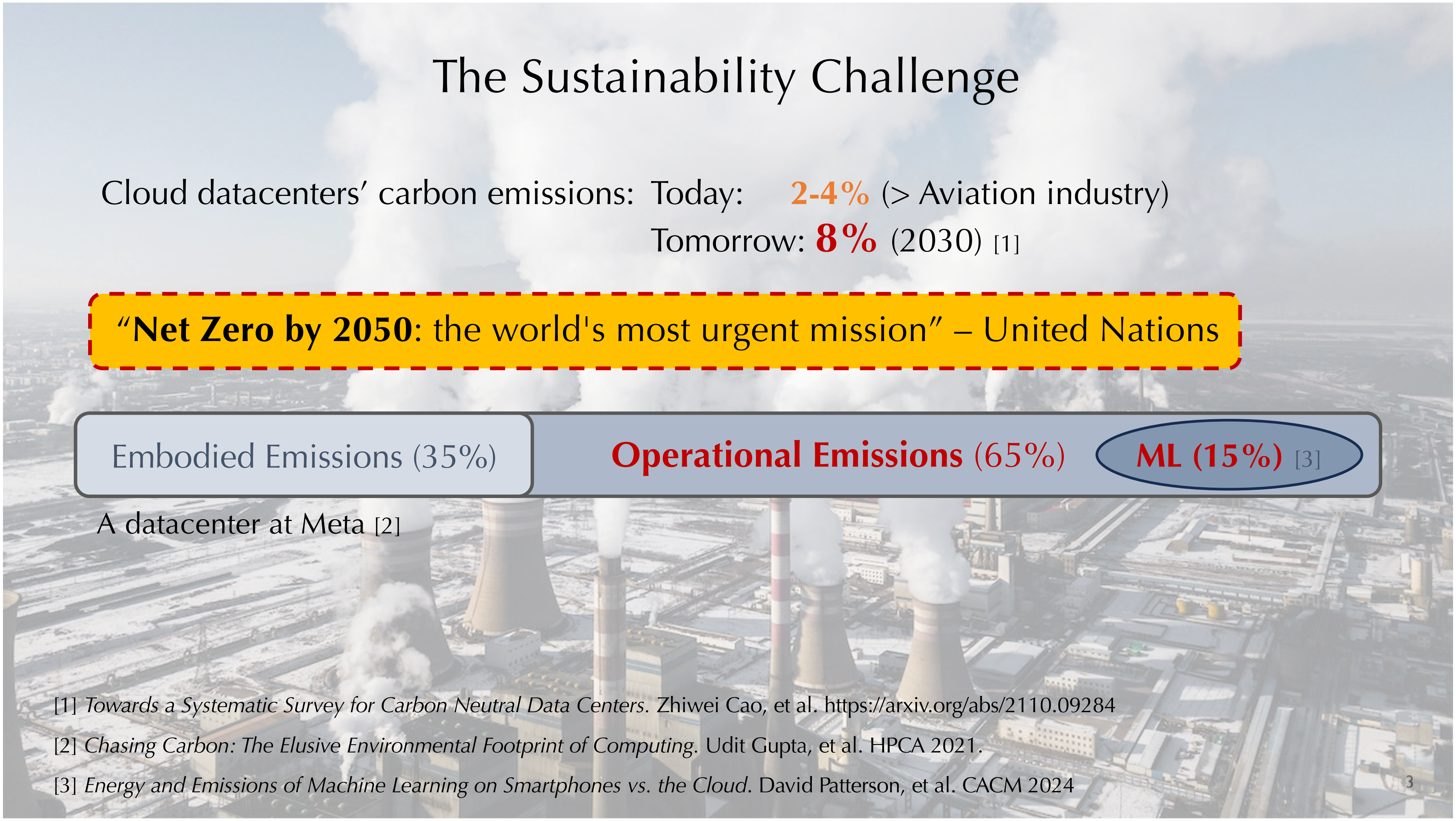


Haoran Qiu[1], Weichao Mao[1], Archit Patke[1], Shengkun Cui[1], Saurabh Jha[2]
Chen Wang[2], Hubertus Franke[2], Chandra Narayanaswami[2], Zbigniew T. Kalbarczyk[1], Tamer Basar[1]
Ravishankar K. Iyer[1]

ILLINOIS IBM Research

Clouds Increasingly the Backbone for Energy Hungry ML-driven Applications





The Sustainability Challenge

Cloud datacenters' carbon emissions: Today: **2-4%** (> Aviation industry)
Tomorrow: **8%** (2030) [1]

"**Net Zero by 2050**: the world's most urgent mission" – United Nations

Embodied Emissions (35%) | **Operational Emissions** (65%) | **ML (15%)** [3]

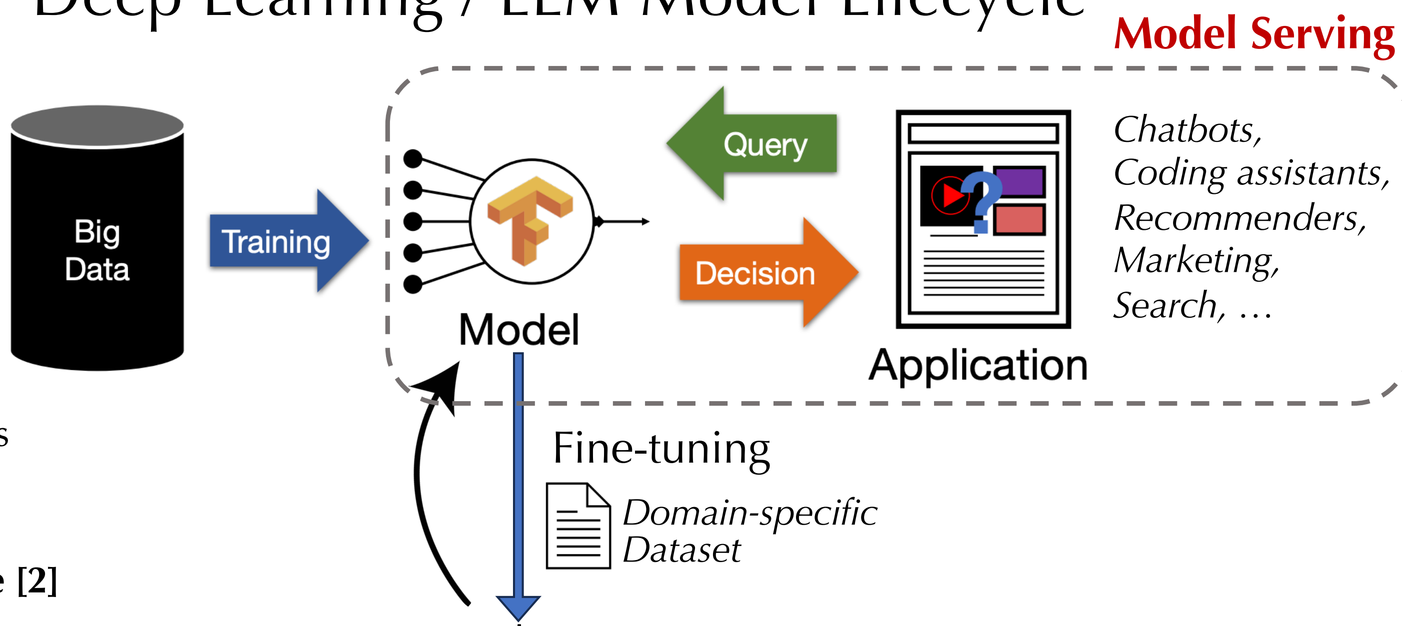
A datacenter at Meta [2]

[1] *Towards a Systematic Survey for Carbon Neutral Data Centers.* Zhiwei Cao, et al. https://arxiv.org/abs/2110.09284

[2] *Chasing Carbon: The Elusive Environmental Footprint of Computing.* Udit Gupta, et al. HPCA 2021.

[3] *Energy and Emissions of Machine Learning on Smartphones vs. the Cloud.* David Patterson, et al. CACM 2024

Deep Learning / LLM Model Lifecycle



Training of GPT-4:
- 25k Nvidia A100s
- ~90-100 days
- $100 million
- 50-60k MWh

Inference: 10x more [2]

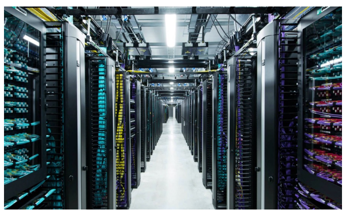
Google estimate that [1], in 2021, **40%** of carbon footprint goes to **model training** while **60%** goes to **model serving**.

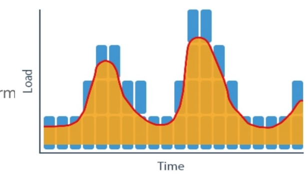
[1] *Carbon Footprint of Machine Learning Training*. Google. https://blog.research.google/2022/02/good-news-about-carbon-footprint-of.html

[2] *AI's Staggering Energy Cost*. https://www.numenta.com/blog/2023/08/10/ai-is-harming-our-planet-2023/

ML in Systems and Cloud

ML has been increasingly used in systems for optimizing efficiency / energy while adapting to dynamic cloud environments… *assessment plus action*





Datacenter management and cluster orchestration

- Power management
- Job scheduling
- Container placement
- Capacity scaling
- Resource config
- Autoscaling

$$\sum_i \boldsymbol{Power} * \boldsymbol{\Delta T_i} * \boldsymbol{CarbonIntensity}$$

Less Energy

Greener Energy

Use of green energy?

How to optimize the use of green energy while meeting **cloud SLOs** and ensuring resilience against both **classic system failures** and potential new **vulnerabilities introduced by ML**?

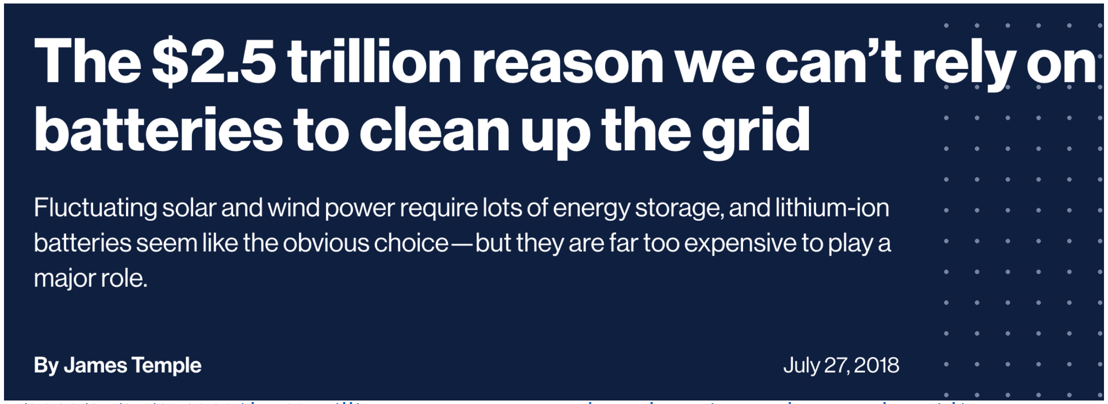
Why Worry about SLOs and System Failures? (Impact on Carbon Footprint Optimization)

- Carbon footprint optimization can lead to **SLA/SLO violations** due to:
  - Processor throttling, load shaping, power capping, etc.
  - SLO violations lead to large financial losses in mission-critical systems

- Continuous fault management is needed to meet SLOs (e.g., in availability and performance) and deliver quality of service
  - By data redundancy (e.g., replication), compute redundancy, coding, storage
  - ML introduces different redundancy requirements and uncertainties especially in critical societal applications
- Fault management adds substantially to the energy consumption
  - 40-60% of the total performance cost is due to fault management overhead
- Not enough done to manage ***Out of Distribution (OOD)*** situations

Can We Rely on Batteries? No Free Lunch for Pure Green Energy

- Today, all green energy (e.g., solar, wind) has fossil fuel component!!
- Cost of resilience; Requires substantial cloud management efforts
  - Any instability can affect the resilience lead to high compute costs
- Power storage cost can be very high, estimated to be trillions of dollars
  - Storage (batteries, other?), unreliable and polluting
  - Currently only used in mission-critical situations
- Requires significant new research including in **SysML & resilience communities**



https://www.technologyreview.com/2018/07/27/141282/the-25-trillion-reason-we-cant-rely-on-batteries-to-clean-up-the-grid/

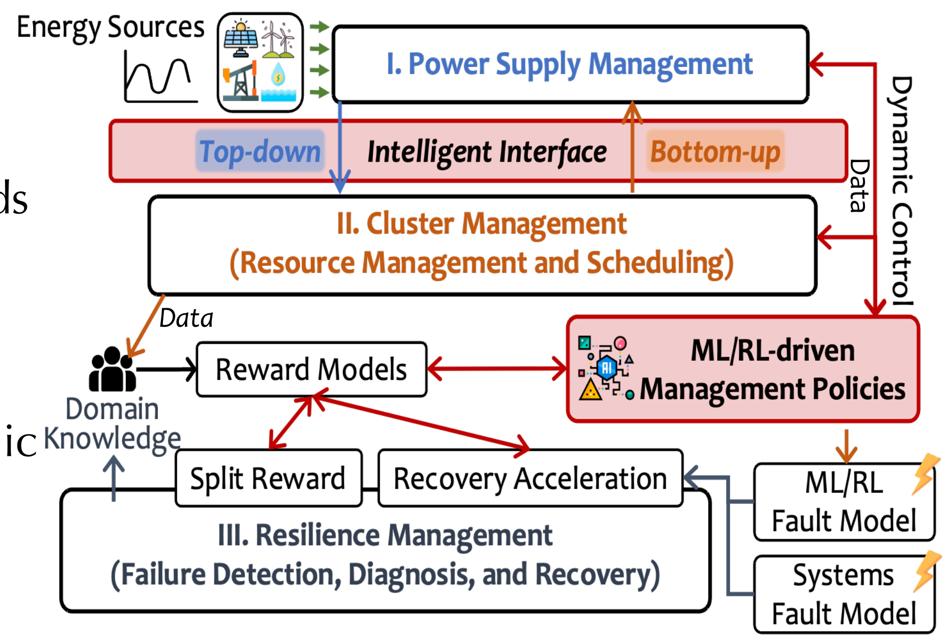
Transition to Green Computing: A Game-theoretic Perspective

Two-fold meaning of "sustainability":

- **Sustainable Energy/Carbon Cost**: Minimize carbon footprint
- **Sustainable Performance**: Multi-tenant clouds need to deliver consistent SLA/SLOs

Managing future large-scale systems:

- How to achieve resilient, **SLO-driven** dynamic optimization towards **green energy**
- How to address **system + ML resilience** management?

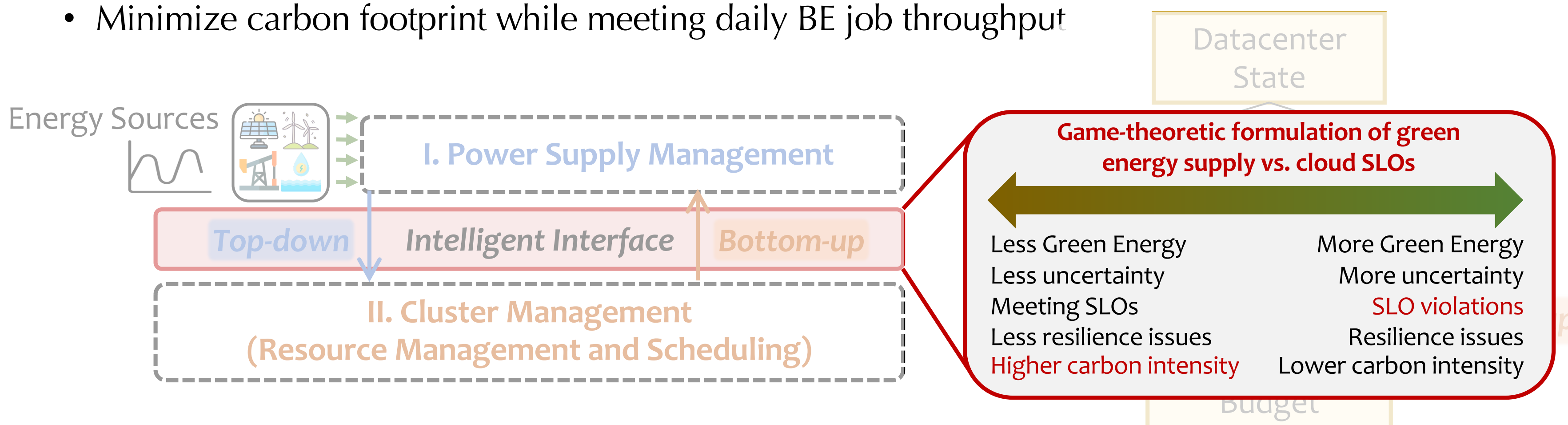


Joint Model Overview: Carbon Footprint-SLO-Resilience Cross-stack Optimization

When Green Computing Meets Performance and Resilience SLOs. Haoran Qiu, Weichao Mao, Chen Wang, Saurabh Jha, Hubertus Franke, Chandra Narayanaswami, Zbigniew T. Kalbarczyk, Tamer Başar, Ravishankar K. Iyer. DSN 2024 Distrupt Track.

Top-down vs. Bottom-up

- *Top-down* approach (MLSys Workshop @NeurIPS23)
  - Get the power cap based on carbon footprint optimization or power limits/budget
  - Resource manager adjust resource allocation accordingly to compensate reduced core frequency
  - Extra buffer added by bringing green energy; relaxing power cap
- *Bottom-up* approach
  - Get the power demand based on the resource + frequency required to meet SLOs
  - Aggregate to get the power demand distribution across servers/racks
  - Minimize carbon footprint while meeting daily BE job throughput

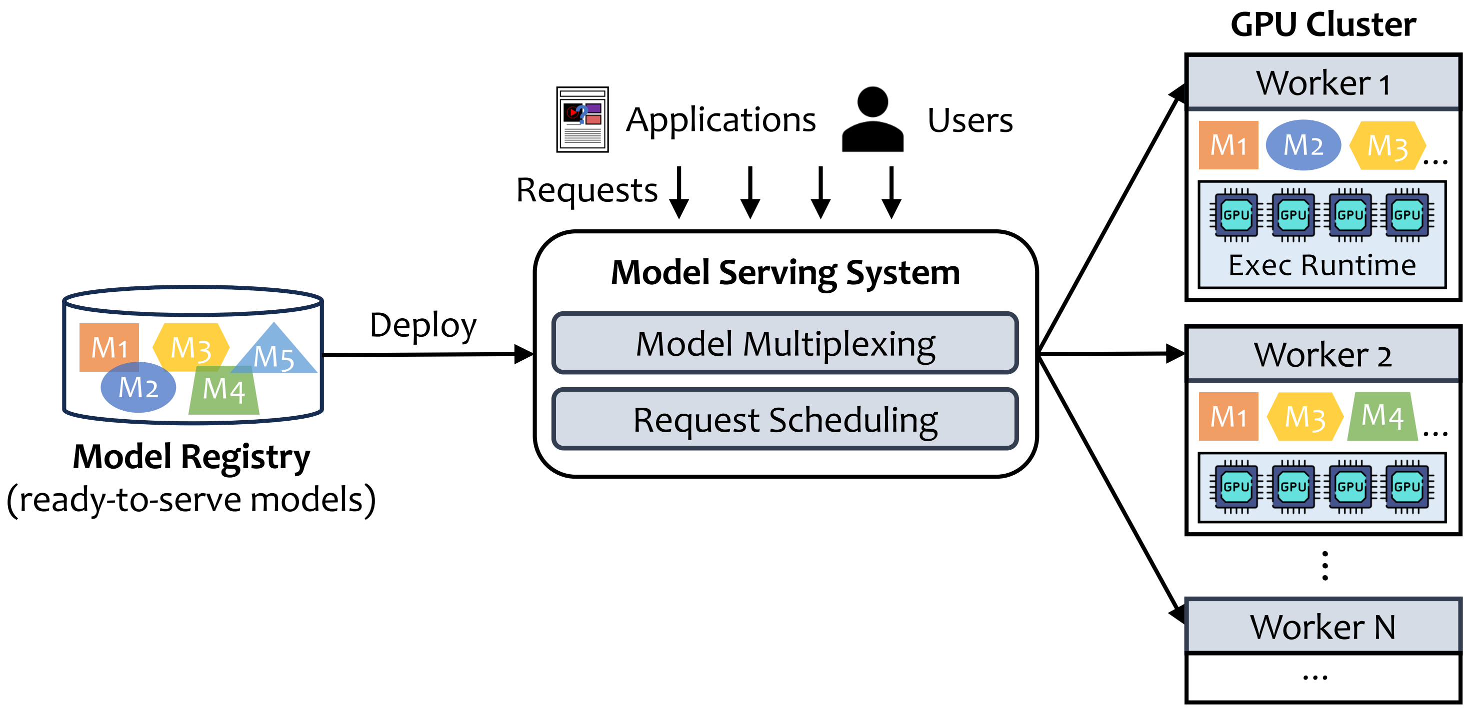


Bottom-up Approach with ML for Carbon Footprint Optimization
A Cyber-Twin for Continuous Green Transition

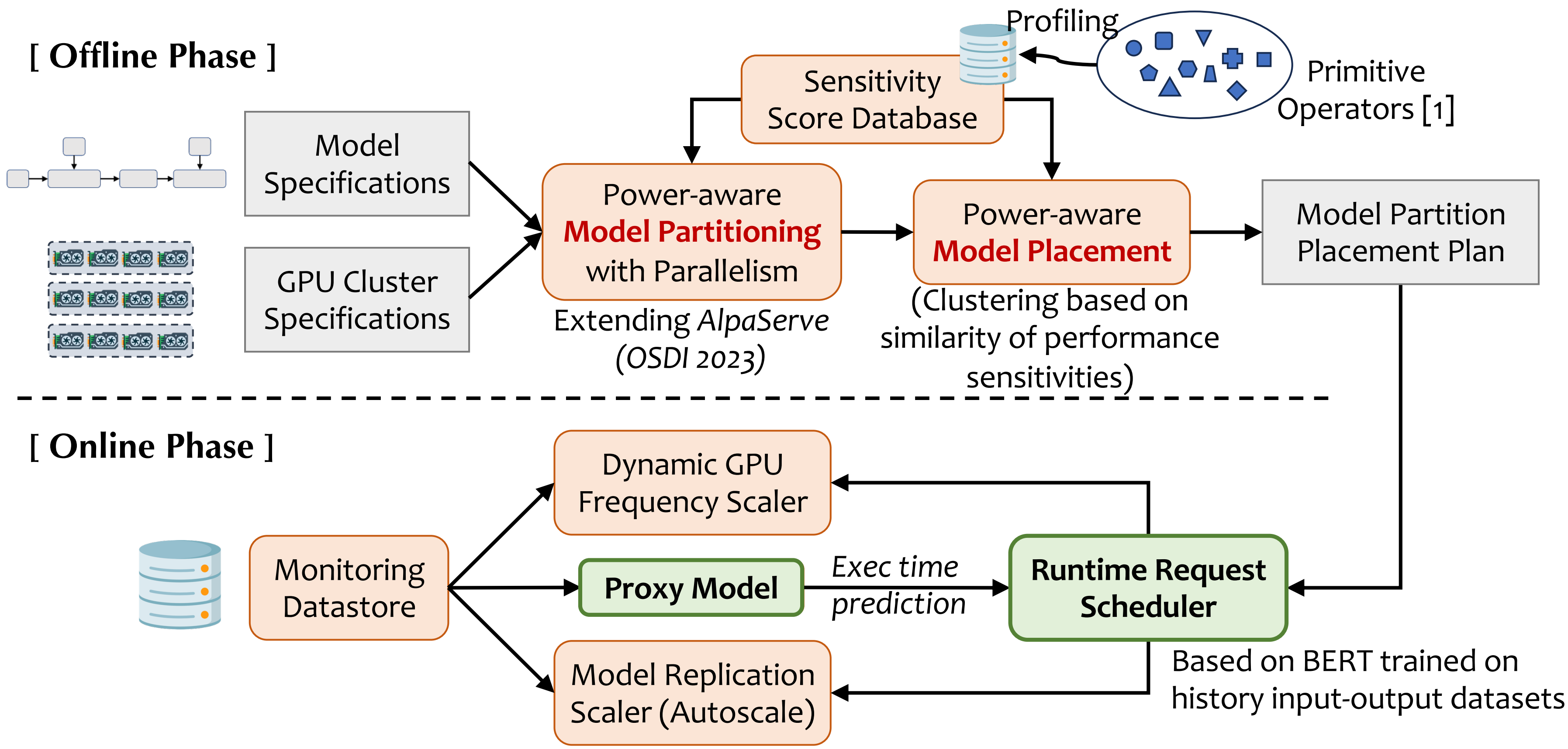
- **Time Window:** We assume that the total period [0, T] is partitioned into sub-periods, say $[t_k, t_{k+1})$, which could be one hour or a half-hour, i.e., "**time interval t**"
- **Carbon Intensity Forecasting:** $CI(t)$ → **ML-driven time series**
- **Applications:** Latency-critical (LC) jobs $l \in LC(t)$, Best-effort (BE) jobs $b \in BE(t)$
- **Servers:** $s \in S(t)$
- **Binary Decision Variable** for Delaying BE job: $delay_b(t)$ → **ML Scheduler**
- **Binary Decision Variable** for Job Placement: $place_{b,s}(t)$, $place_{l,s}(t)$
- **Power Consumption:** server $P_s(t)$, LC job $P_l(t)$, BE job $P_b(t)$ → **ML Autoscaler, e.g., FIRM (OSDI20)**
  - $P_s(t) = \sum_l P_l(t) \cdot place_{l,s}(t) + \sum_b P_b(t) \cdot place_{b,s}(t)$
- **Constraints:**
  - $\sum_s place_{b,s}(t) \leq 1, \forall b, t;$ $\sum_s place_{b,s}(t) + delay_b(t) = 1, \forall b, t;$ $\sum_s place_{l,s}(t) = 1, \forall l, t$
  - $\sum_t \sum_{b,s} place_{b,s}(t) > Daily\_Throughput\_Threshold$
- **Penalties:** e.g., carbon intensity, SLO violations, resilience breakdowns
- **Minimize** Total Carbon Footprint: $\sum_{s,t} P_s(t) \cdot CI(t)$

How to achieve continuous, fast re-optimization (recovery) under system + ML failures?

Model Serving Systems



μ-Serve ***Model Serving Example***: for Power-aware DL/LLM



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

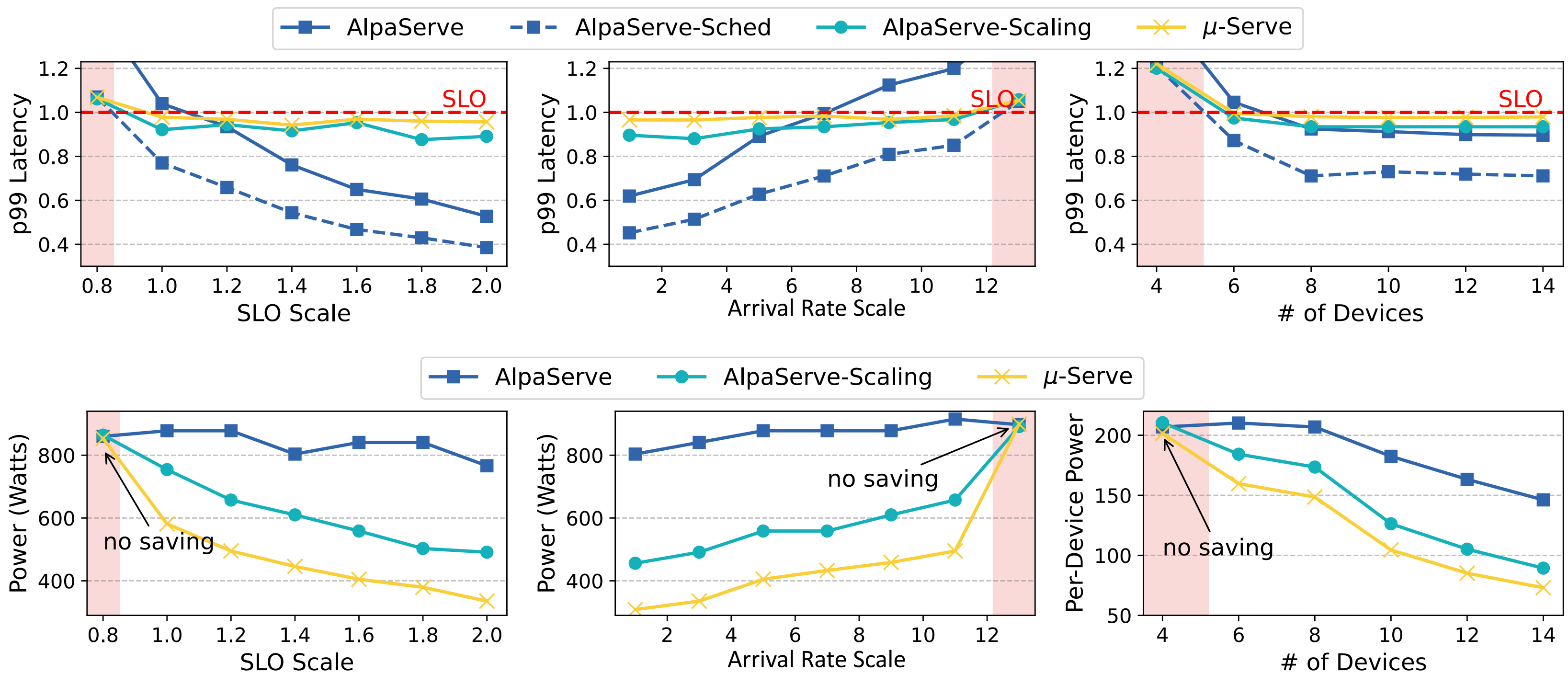
Power-aware Deep Learning Model Serving with μ-Serve. Haoran Qiu, Weichao Mao, Archit Patke, Shengkun Cui, Saurabh Jha, Chen Wang, Hubertus Franke, Zbigniew T. Kalbarczyk, Tamer Başar, Ravishankar K. Iyer. USENIX ATC 2024

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** (largest open-source dataset available) and workload patterns from **Azure Function 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: Power Saving



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

A Disruptive Systems Approach to Sustainable Computing with Efficient and Robust ML

CPU Cloud Efficiency with ML

Microservices

FIRM
- OSDI 2020

Serverless Computing

SIMPPO
- SoCC 2022, NeurIPS 2022
- MLSys @NeurIPS 2023

GPU Cloud Efficiency with ML

DL Model Serving

µ-Serve
- ATC 2024a
- NSDI 2025*

Disaggregated Memory

INDIGO
- ASPLOS*, COMPSYS 2022

Robust ML for Systems

Cloud Heterogeneity

FLASH
- MLSys 2024

Reliable RL Exploration

AWARE
- ATC 2023, NeurIPS 2023

Cloud Multi-tenancy

MAPPO
- EuroMLSys 2022, WoSC 2021

Holistic (Green) Energy Optimization Jointly with Cloud Systems-ML Resilience

- NSF WSCS 2024, DSN 2024

DEPEND Group Contributions: A Disruptive Approach to Sustainable Computing with Efficient and Robust ML

CPU Cloud Efficiency with ML

Microservices | Serverless Computing

FIRM *SIMPPO*

Robust ML for Systems

Cloud Heterogeneity

FLASH

- Continuously manage resources and scheduling optimally
- Optimally manage robustness including system and ML reliability

GPU Cloud Efficiency with ML

- ATC 2023, NeurIPS 2023

DL Model Serving | Disaggregated Memory

μ-Serve
- ATC 2024a
- NSDI 2025*

INDIGO
- ASPLOS*, COMPSYS 2022

Cloud Multi-tenancy

MAPPO
- EuroMLSys 2022, WoSC 2021

Holistic (Green) Energy Optimization Jointly with Cloud Systems-ML Resilience

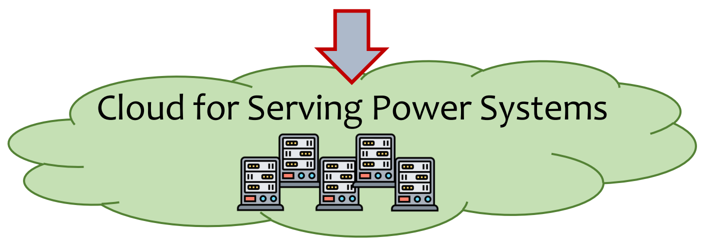
- NSF WSCS 2024, DSN 2024

Are batteries the future to sustainable computing?
AI/ML Cloud for Power Storage Serving

Robustness?

Cost of Resilience?

Power systems management



Instability?

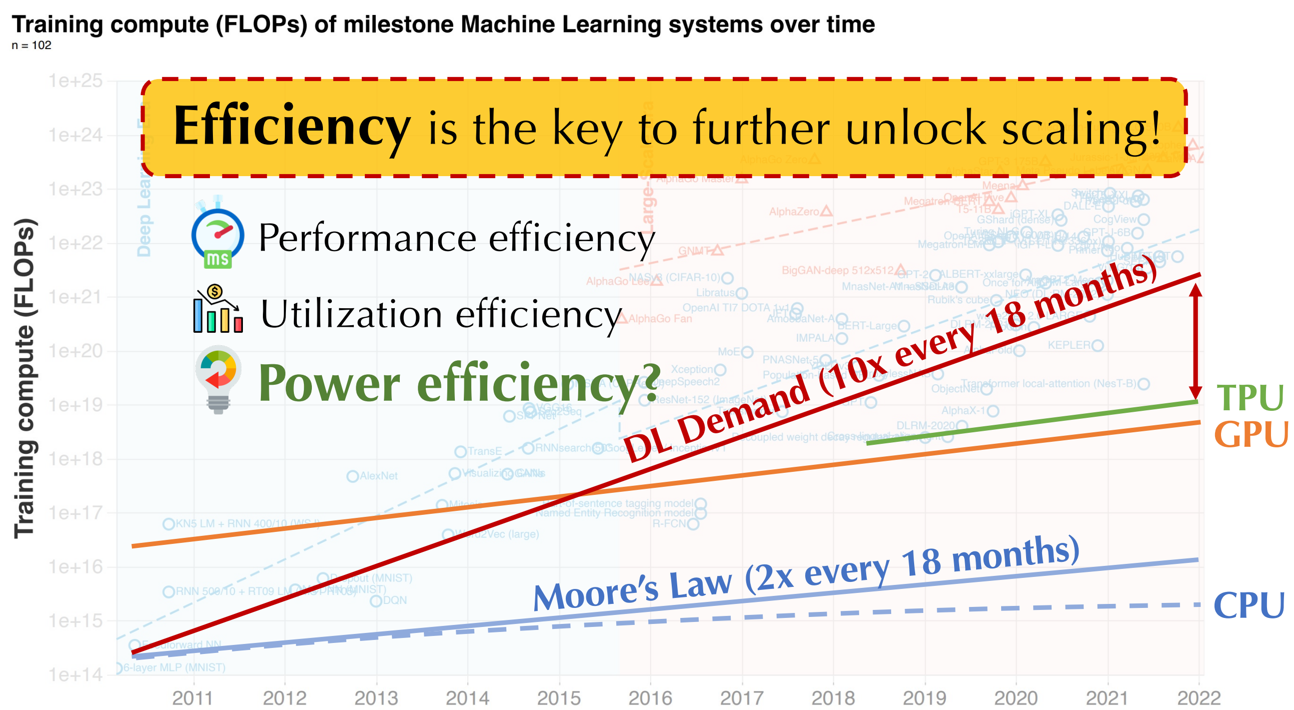
Heterogeneity?

Good estimate of the battery storage costs essential

An intelligent, resilience-aware cloud is needed for power storage serving

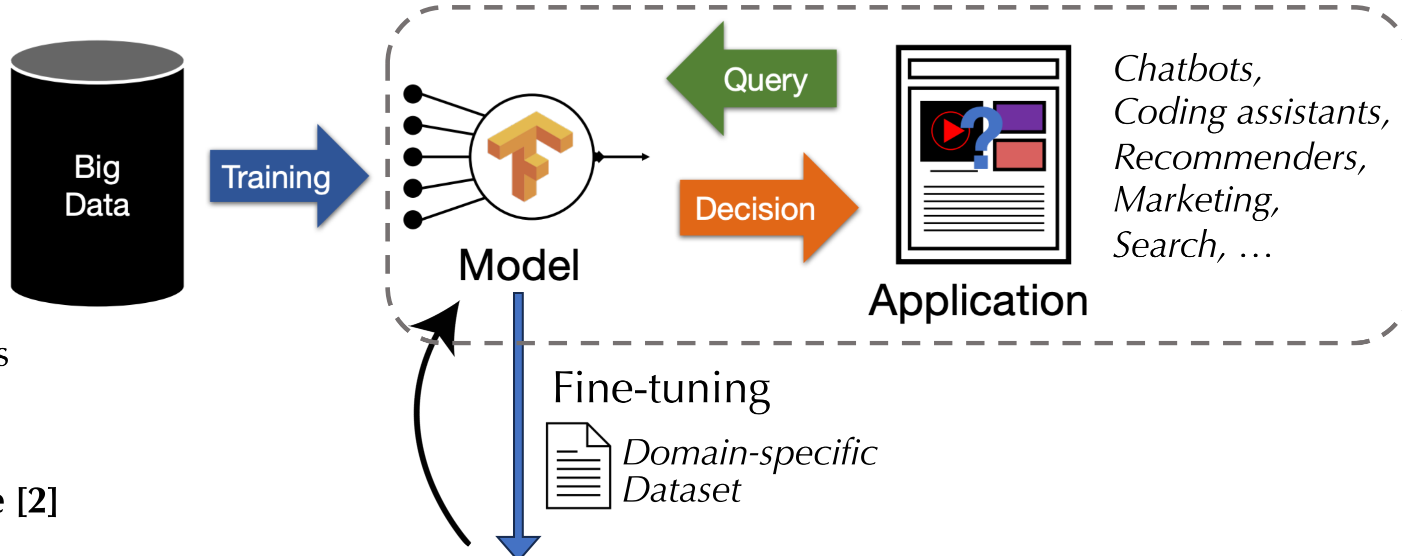
Back up slides

Deep Learning and Foundation Model 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



Training of GPT-4:
- 25k Nvidia A100s
- ~90-100 days
- $100 million
- 50-60k MWh

Inference: 10x more [2]

Google estimate that, in 2021, **40%** of carbon footprint goes to **model training** while **60%** goes to **model serving**.

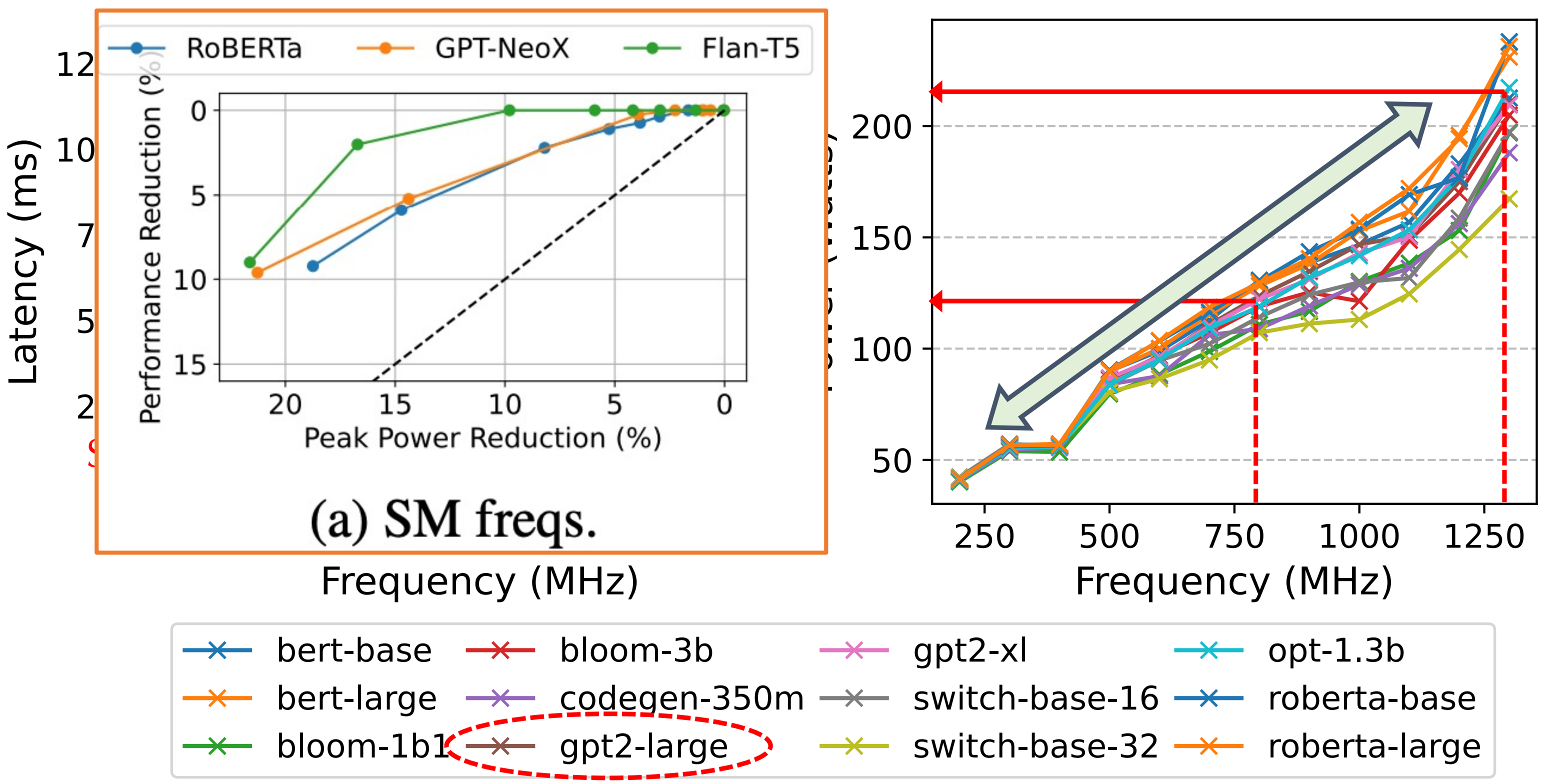
[1] *Carbon Footprint of Machine Learning Training*. Google. https://blog.research.google/2022/02/good-news-about-carbon-footprint-of.html

[2] *AI's Staggering Energy Cost*. https://www.numenta.com/blog/2023/08/10/ai-is-harming-our-planet-2023/

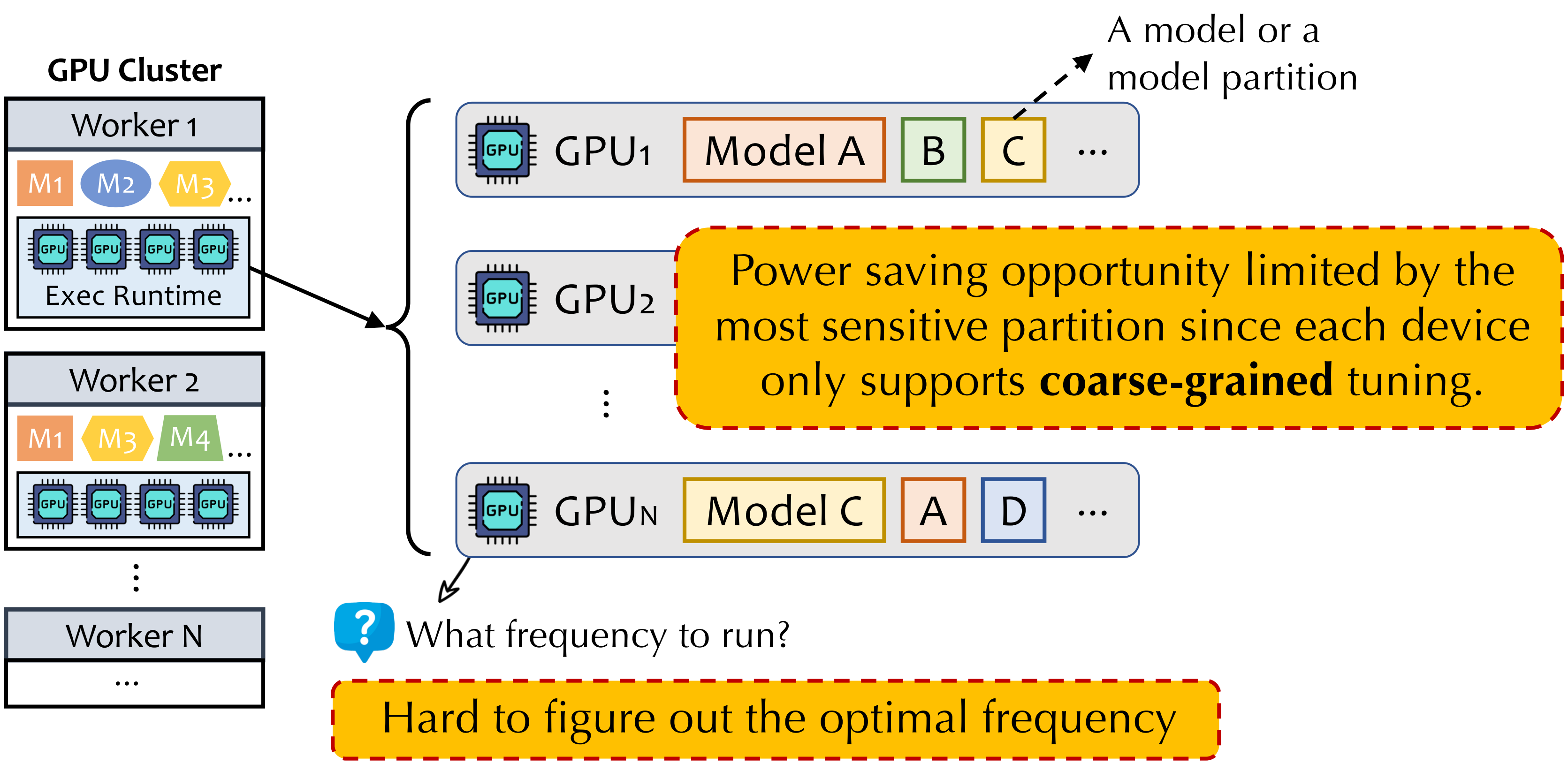
Power Saving Opportunities

POLCA (Microsoft, ASPLOS24)

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



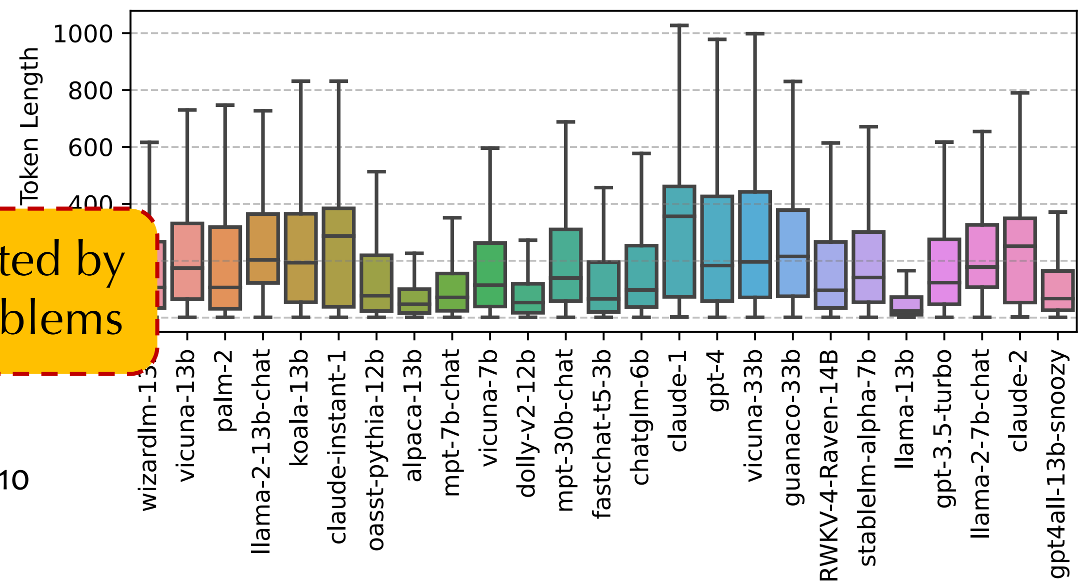
Challenge #1: Coarse-grained GPU Frequency Tuning



Challenge #2: Non-deterministic LLM Executions

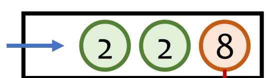
- **Autoregressive** nature of LLMs
- Can lead to head-of-line (**HoL**) blocking in FCFS
- Likely SLO violations on job completion times (JCT)

Dataset: LMSYS-Chat-1M
A Large-Scale Real-World LLM Conversation Dataset



Power saving opportunity limited by **nondeterminism** and HoL problems

Queue



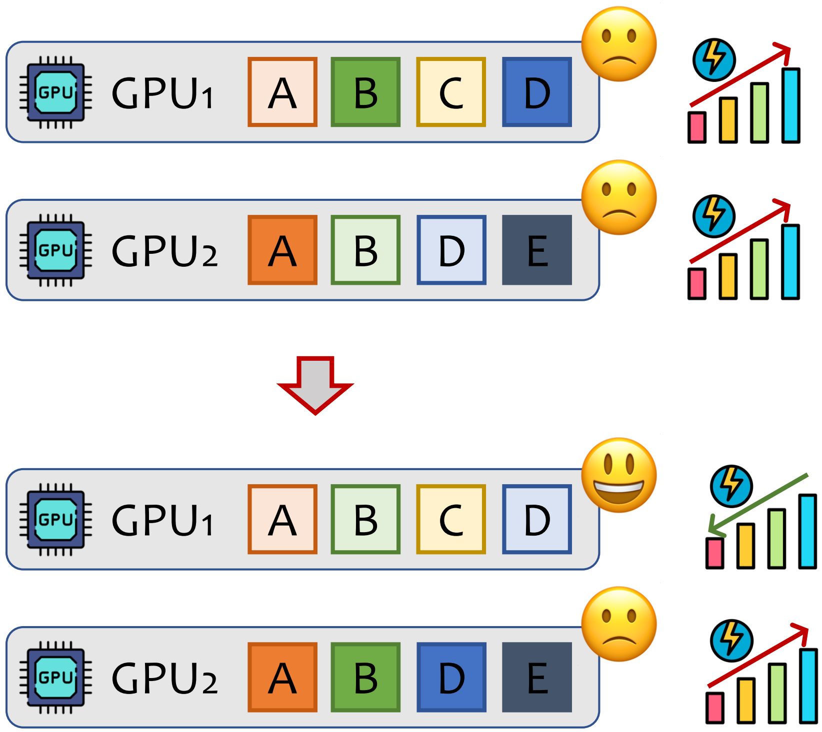
Avg JCT:
(8+(8+2)+(8+2+2)) / 3 = 10

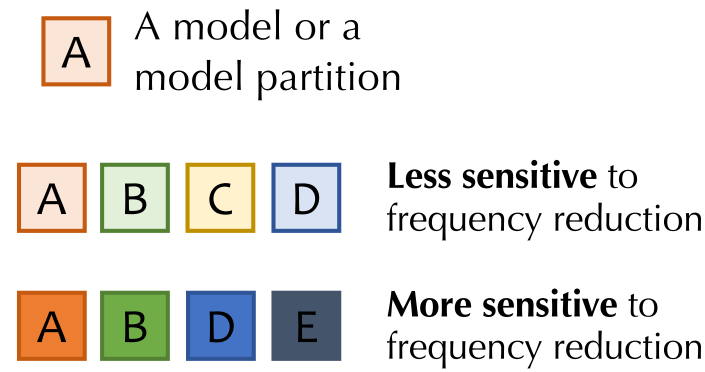
Head-of-line blocking

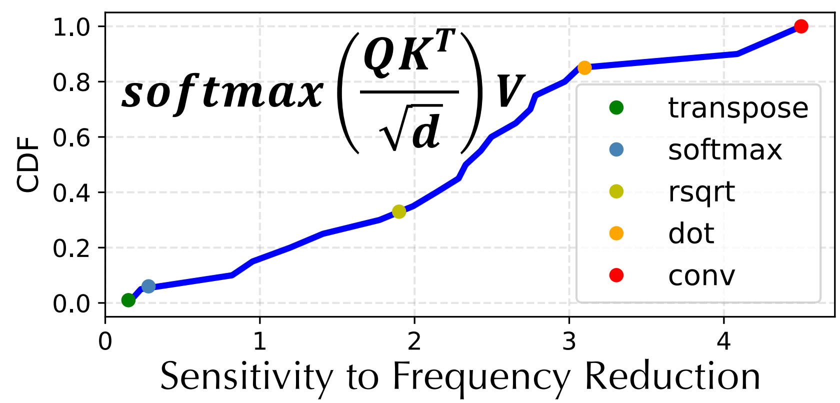
(2+(2+2)+(2+2+8)) / 3 = 6

40% saving in JCT

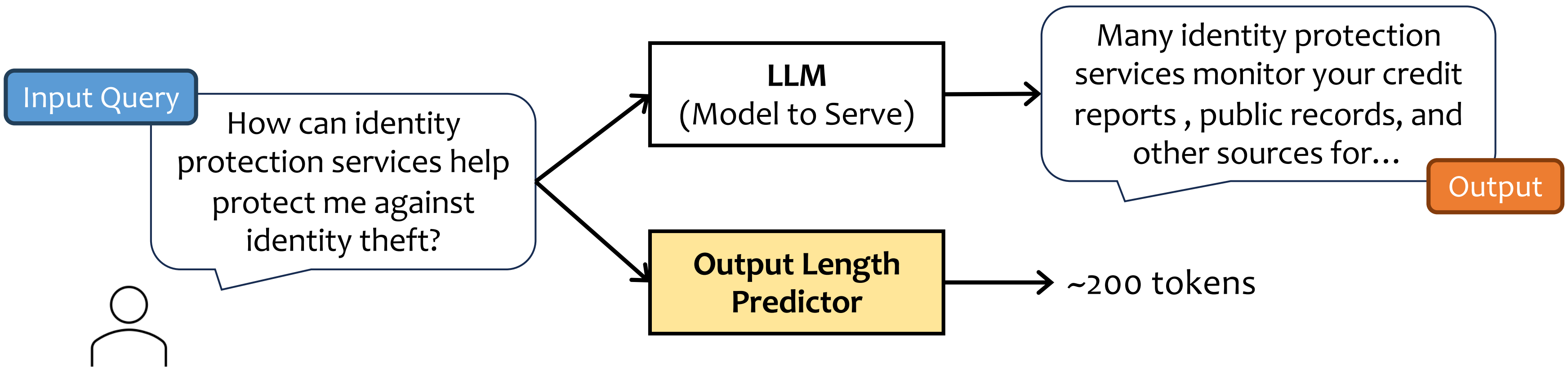
Observation #1: Model Partitions Have Diverse Sensitivities







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

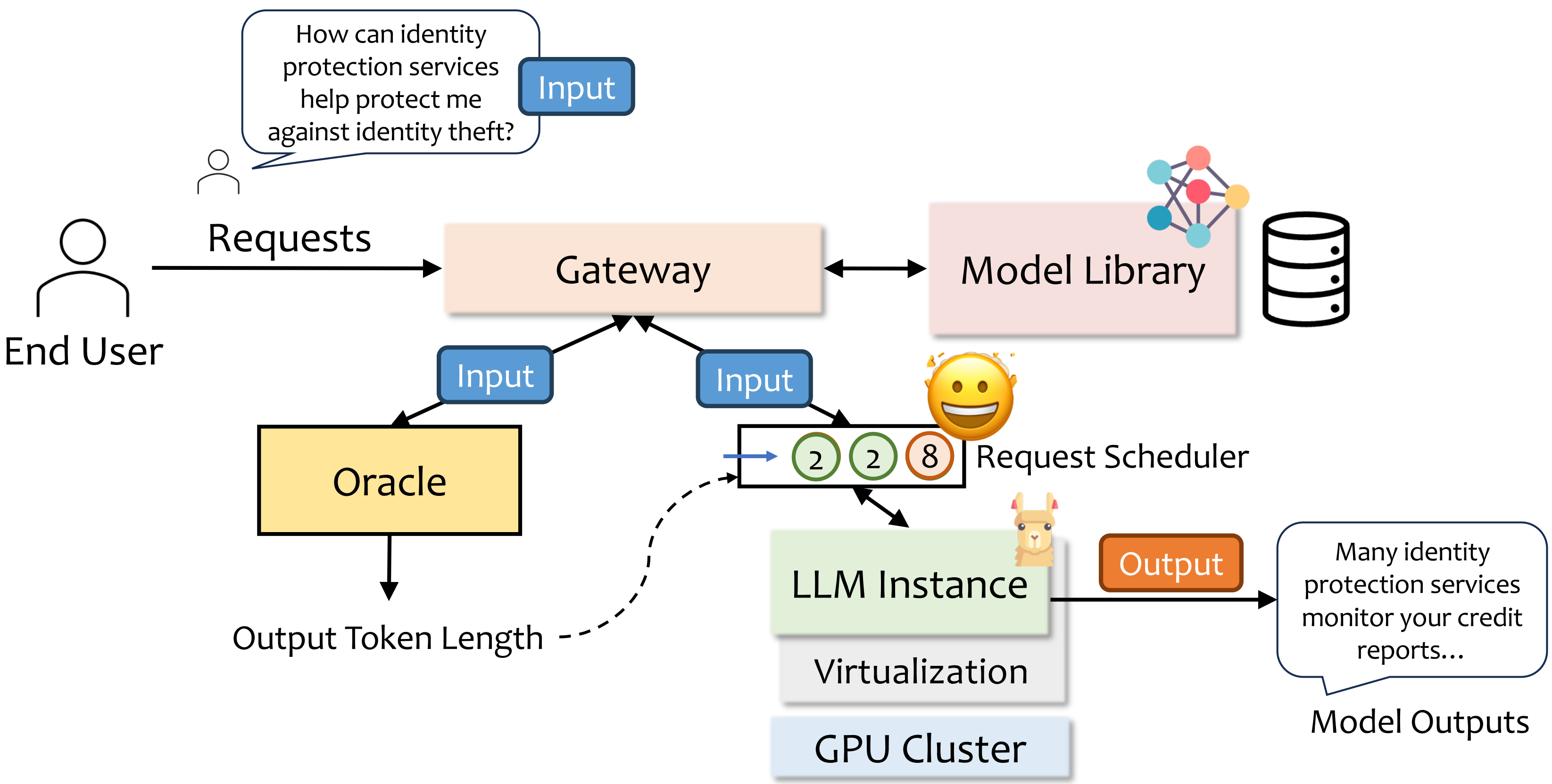


- **A small proxy model (e.g., BERT-base/tiny) can predict well**
- Intuition: Hints on the output length (number of tokens) of LLM responses
  - "**Translate**…" -> Response length approximate to the prompt length
  - "Write an **article** about…" -> Long respo
  - "**Summarize**…" -> Shorter response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

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

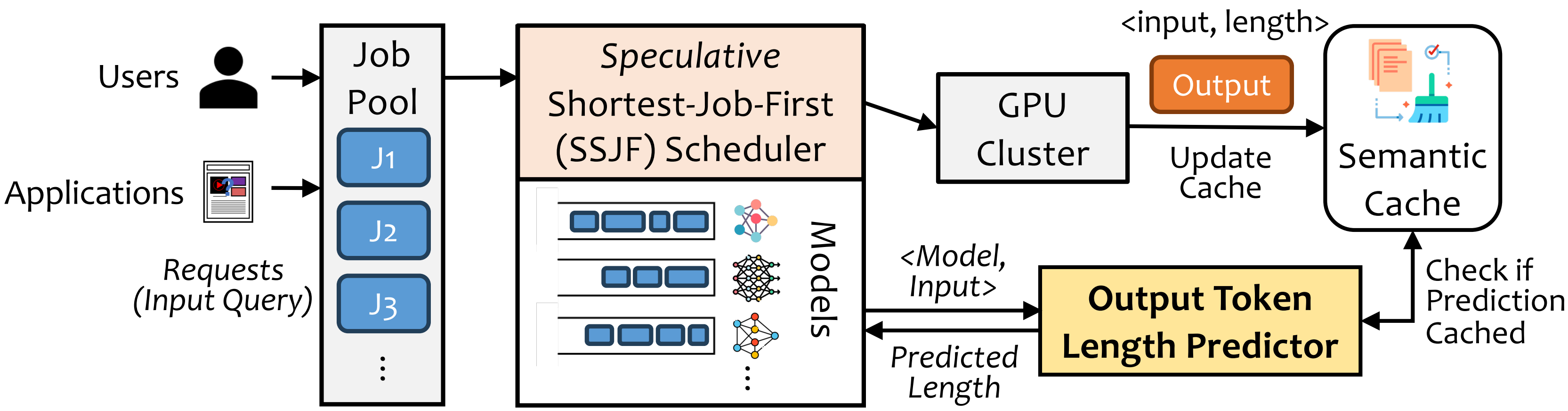
How to design and train a lightweight predictor that can *understand the behavior* of an LLM and *estimate the output token length before serving the request* on the LLM?

Workflow



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

SSJF: Prediction-based Shortest Job First Scheduling



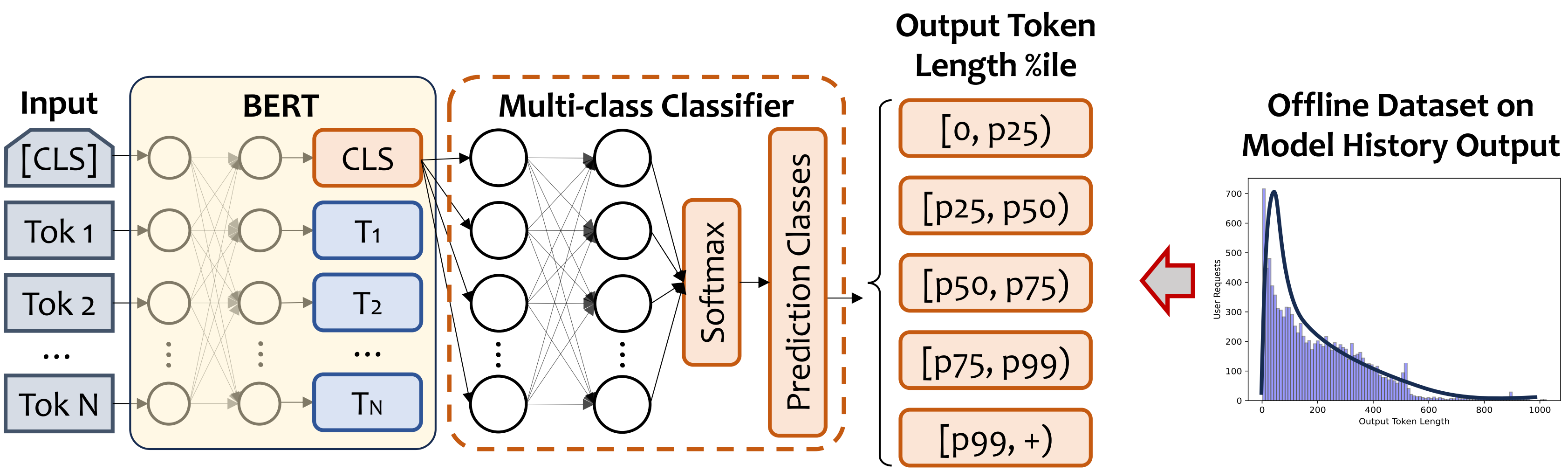
- **SSJF: Using output token length prediction as the exec time estimation**
  - Exec time = ***Const*** + ***K*** * Output token length

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?

System and Models Setup

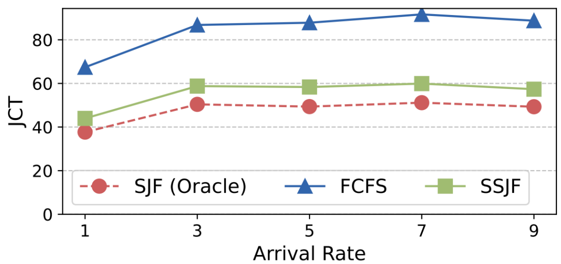
- **Platform**: AlpaServe and Ray
- **VM on IBM Cloud**: 16 vCPU 128 GiB RAM with 2x NVIDIA Tesla V100 16 GB
- Open-source LLMs and non-autoregressive models
- Model input from **LMSYS-Chat-1M** and workload patterns from **Azure Function Traces**

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

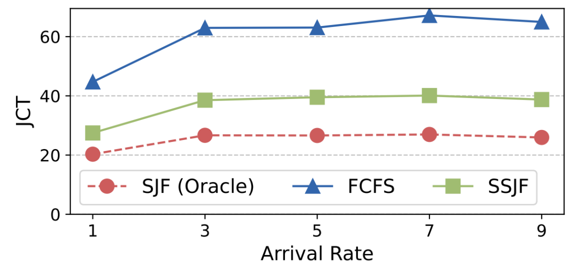
Results (1): Scheduling Performance - JCT

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

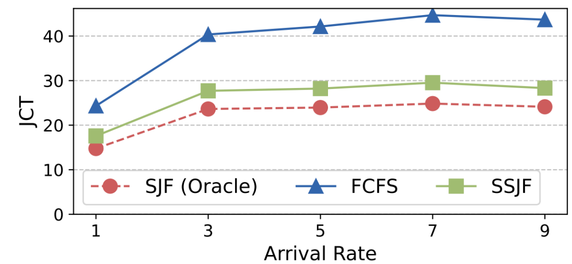
At varying rates



(a) No batching.



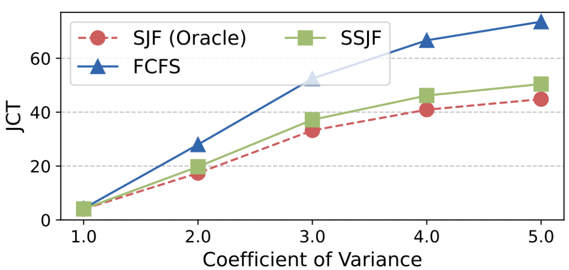
(b) Dynamic batching.



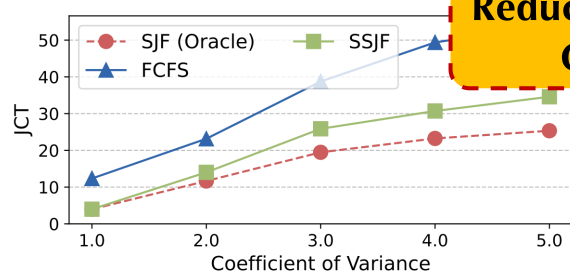
(c) Continuous batching.

At varying variations

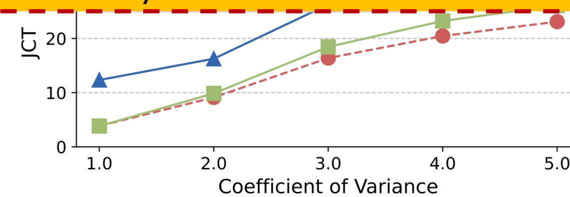
Reduce JCT by 30.5% / 39.0% / 35.0%
Oracle by 37.6% / 52.9% / 41.5%



(a) No batching.



(b) Dynamic batching.

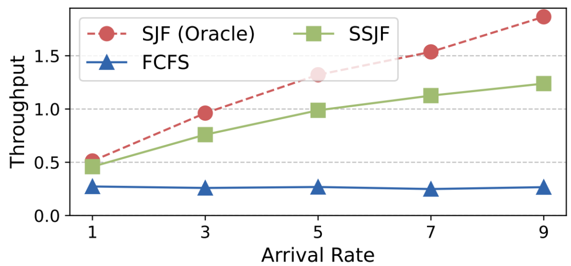


(c) Continuous batching.

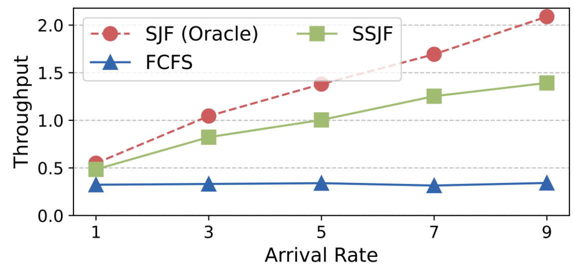
Results (2): Scheduling Performance - Throughput

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

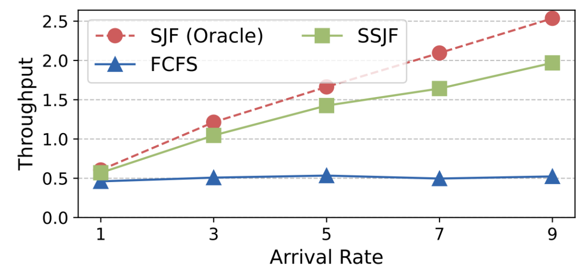
At varying rates



(a) No batching.



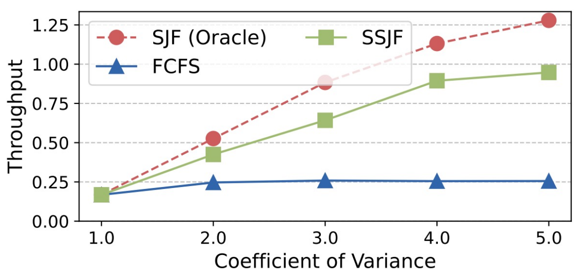
(b) Dynamic batching.



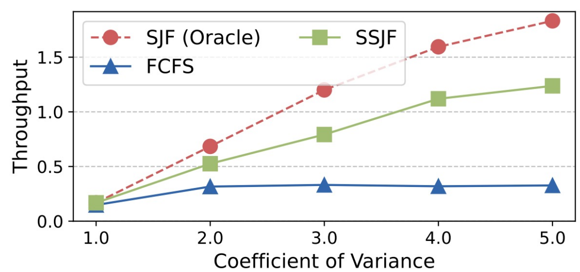
(c) Continuous batching.

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

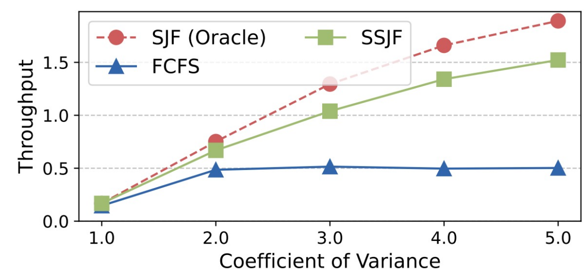
At varying variations



(a) No batching.

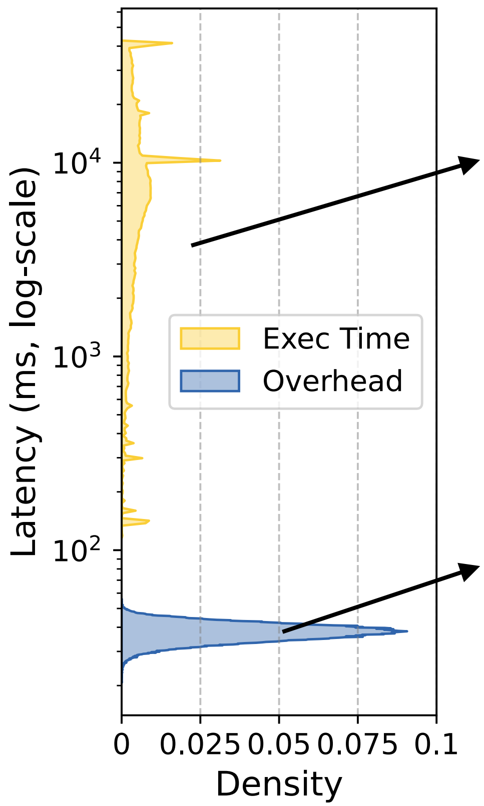


(b) Dynamic batching.



(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.

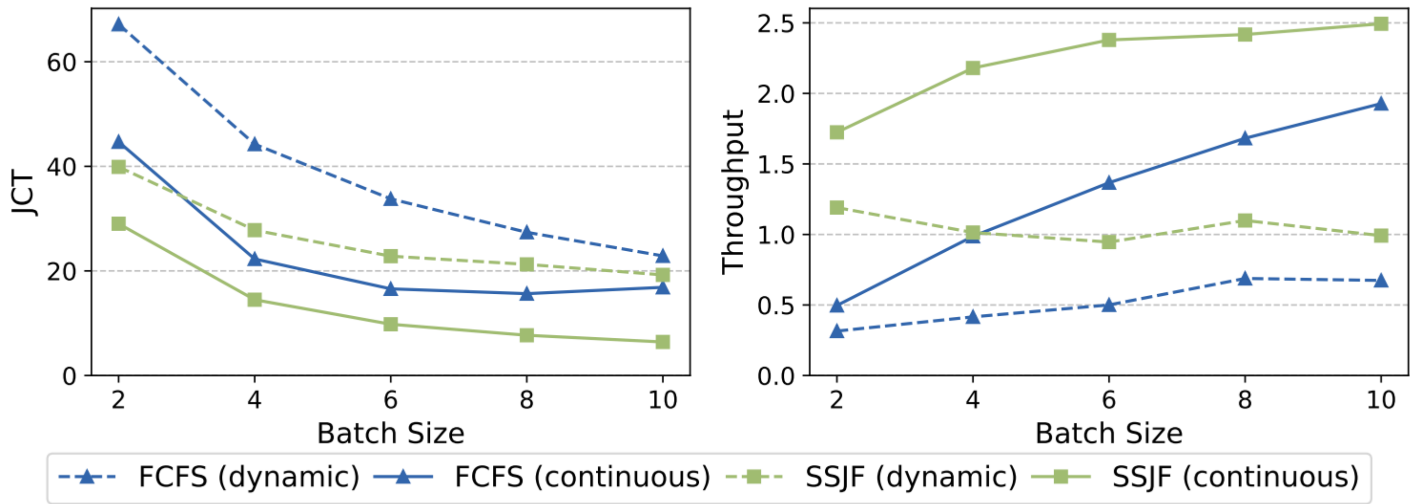
BERT-base Prediction Overhead

Avg Inference Latency = **7.6ms**

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

Results (4): At Varying Batch Sizes

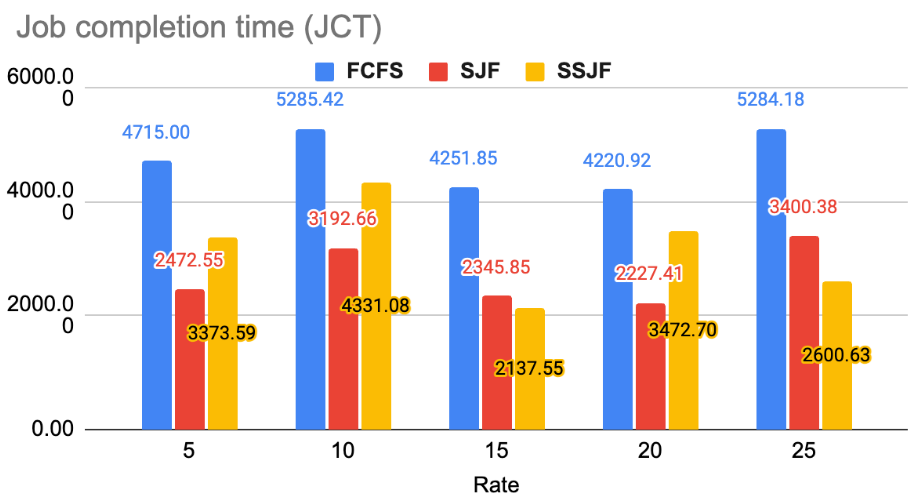
***μ*-Serve** continues to provide **improvement in JCT and throughput** under **various batch sizes** with a diminishing return.

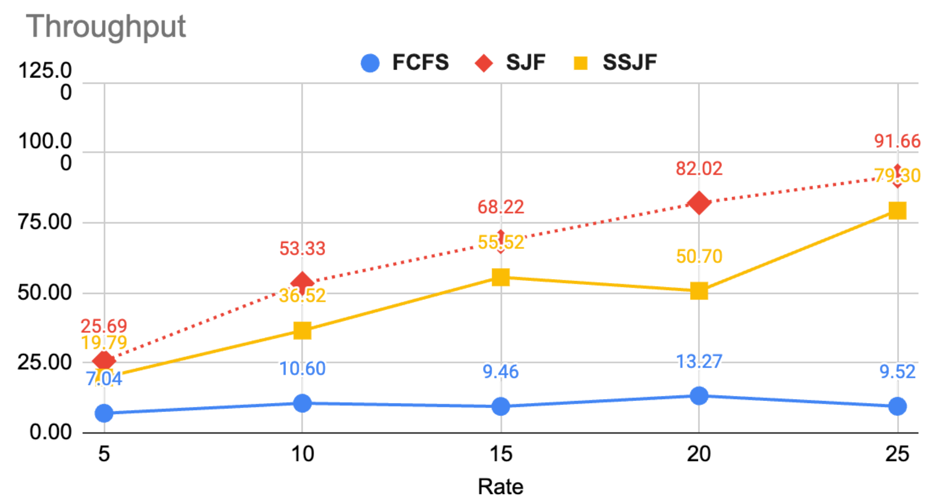


Continuous (iterative) batching > dynamic batching
(same observation as in Orca, OSDI 22)

Results (5): Integration with vLLM

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

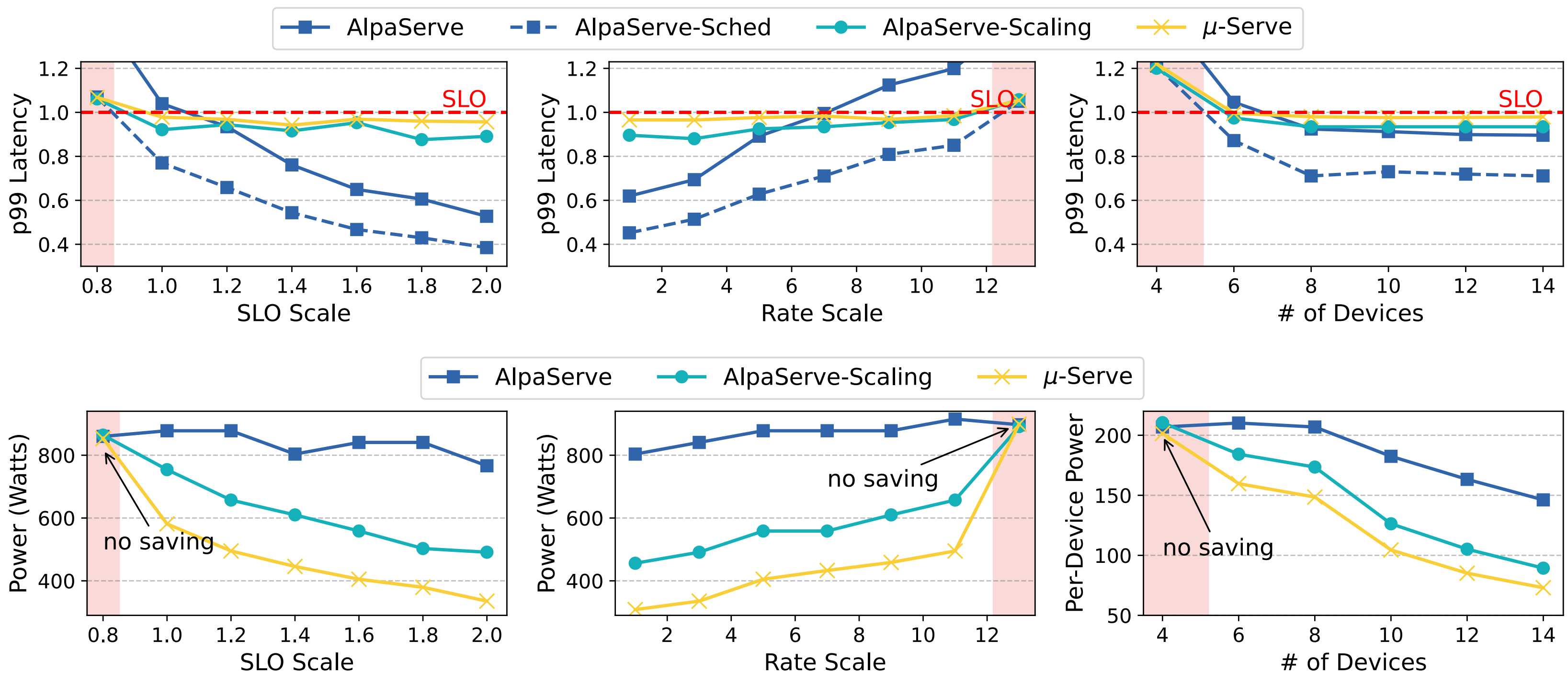




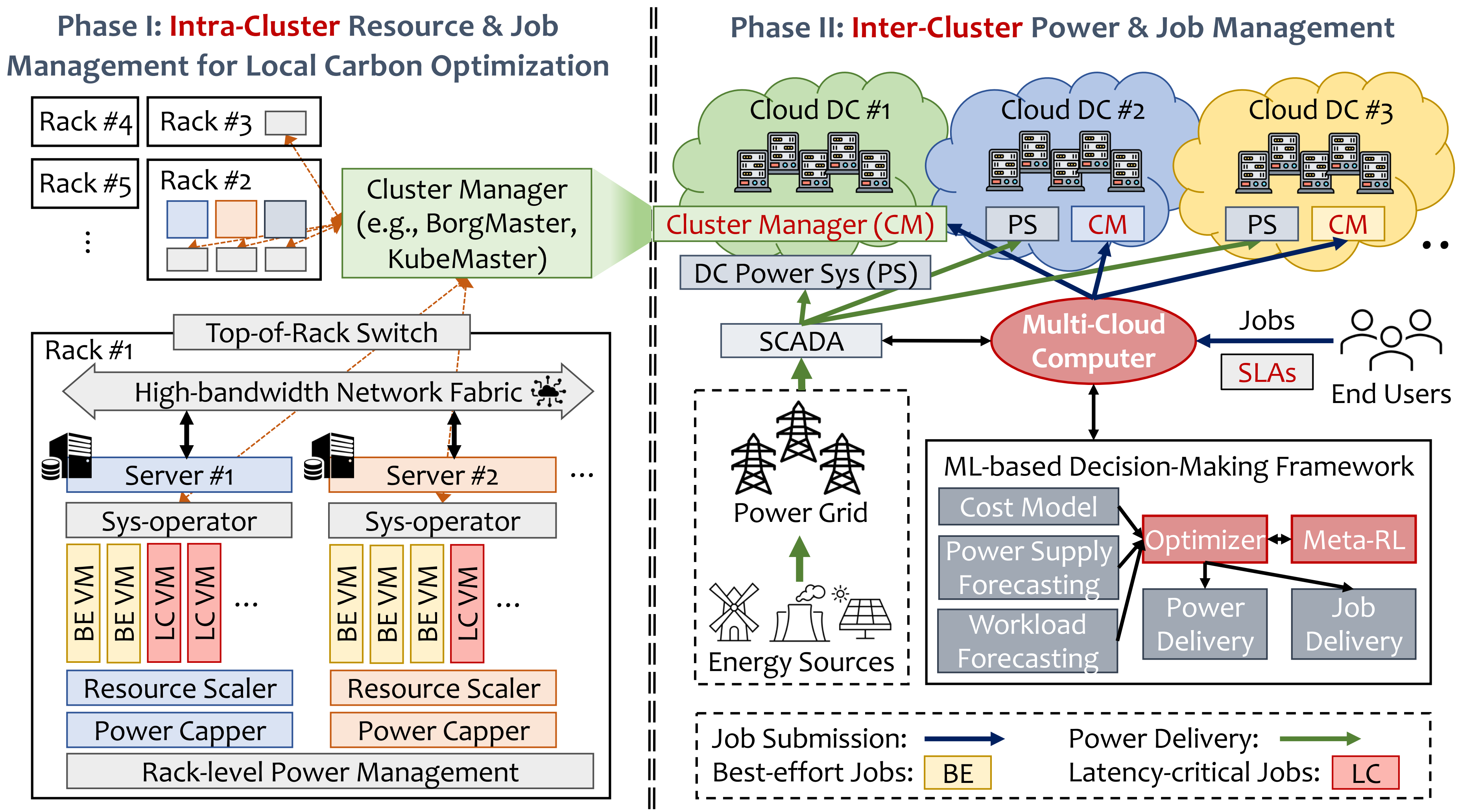
SJF (oracle) achieves **43%** and **6.3x** improvement in **JCT** and **throughput** than FCFS.

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

Results (6): Power Saving



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


Phase I: Intra-Cluster Resource & Job Management for Local Carbon Optimization
Rack #4
Rack #3
Rack #5
Rack #2
Cluster Manager (e.g., BorgMaster, KubeMaster)
Rack #1
Top-of-Rack Switch
High-bandwidth Network Fabric
Server #1
Server #2
Sys-operator
Sys-operator
BE VM
BE VM
LC VM
LC VM
BE VM
BE VM
BE VM
LC VM
Resource Scaler
Resource Scaler
Power Capper
Power Capper
Rack-level Power Management
Phase II: Inter-Cluster Power & Job Management
Cloud DC #1
Cloud DC #2
Cloud DC #3
Cluster Manager (CM)
PS
CM
PS
CM
DC Power Sys (PS)
SCADA
Multi-Cloud Computer
Jobs
SLAs
End Users
Power Grid
Energy Sources
ML-based Decision-Making Framework
Cost Model
Power Supply Forecasting
Workload Forecasting
Optimizer
Meta-RL
Power Delivery
Job Delivery
Job Submission:
Power Delivery:
Best-effort Jobs: BE
Latency-critical Jobs: LC

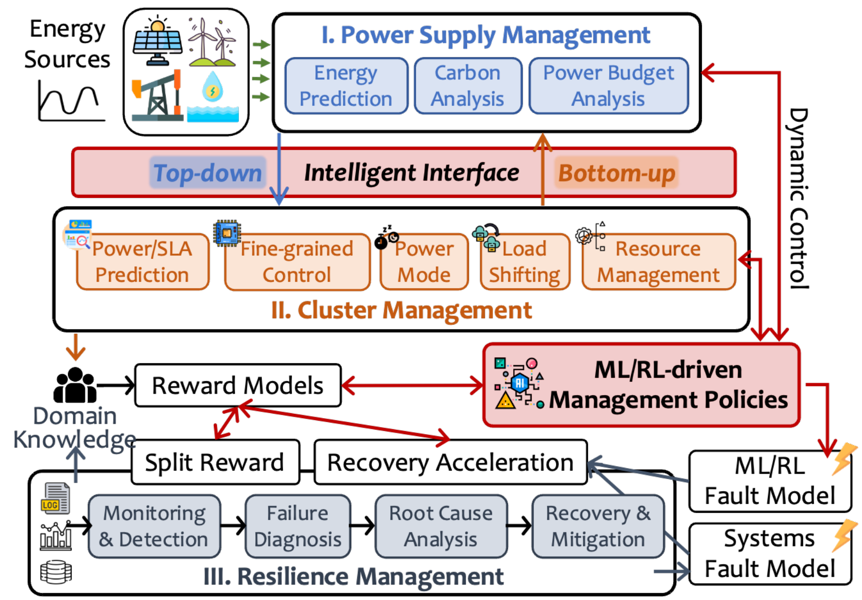
Dependable Transition to Green Computing

Two-fold meaning of sustainability:

- **Sustainable Energy/Carbon Cost**: Minimize carbon footprint
- **Sustainable Performance**: Multi-tenant clouds need to deliver consistent SLA/SLOs

Key Research Questions:

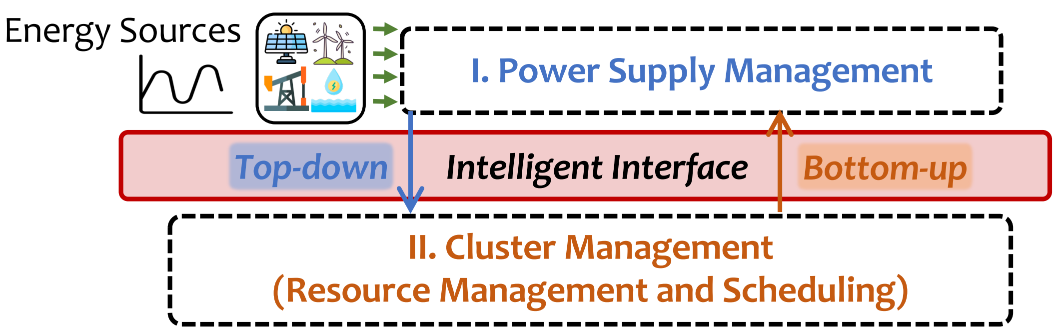
- How to achieve resilient, **SLO-driven** dynamic optimization of **green energy** usage
- How to address **System + ML resilience management**?

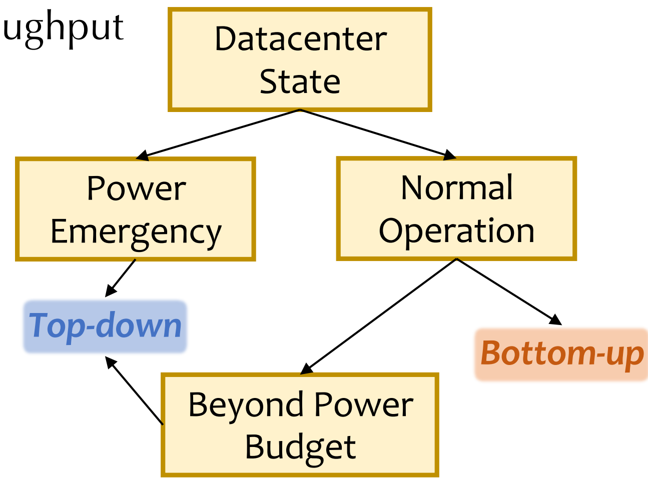


Overview of Proposal

Top-down vs. Bottom-up

- *Top-down* approach (MLSys Workshop @NeurIPS23)
  - Get the power cap based on carbon footprint optimization or power limits/budget
  - Resource manager adjust resource allocation accordingly to compensate reduced core frequency
- *Bottom-up* approach
  - Get the power demand based on the resource + frequency required to meet SLOs
  - Aggregate to get the power demand distribution across servers/racks
  - Minimize carbon footprint while meeting daily BE job throughput





Bottom-up Approach with ML for Carbon Footprint Optimization

- **Time Window:** We assume that the total period [0, T] is partitioned into sub-periods, say $[t_k, t_{k+1})$, which could be one hour or a half-hour, i.e., "**time interval t**"
- **Carbon Intensity Forecasting:** $CI(t)$
- **Applications:** LC jobs $l \in LC(t)$, BE jobs $b \in BE(t)$
- **Servers:** $s \in S(t)$
- **Binary Decision Variable** for Delaying BE job: $delay_b(t)$

ML Scheduler

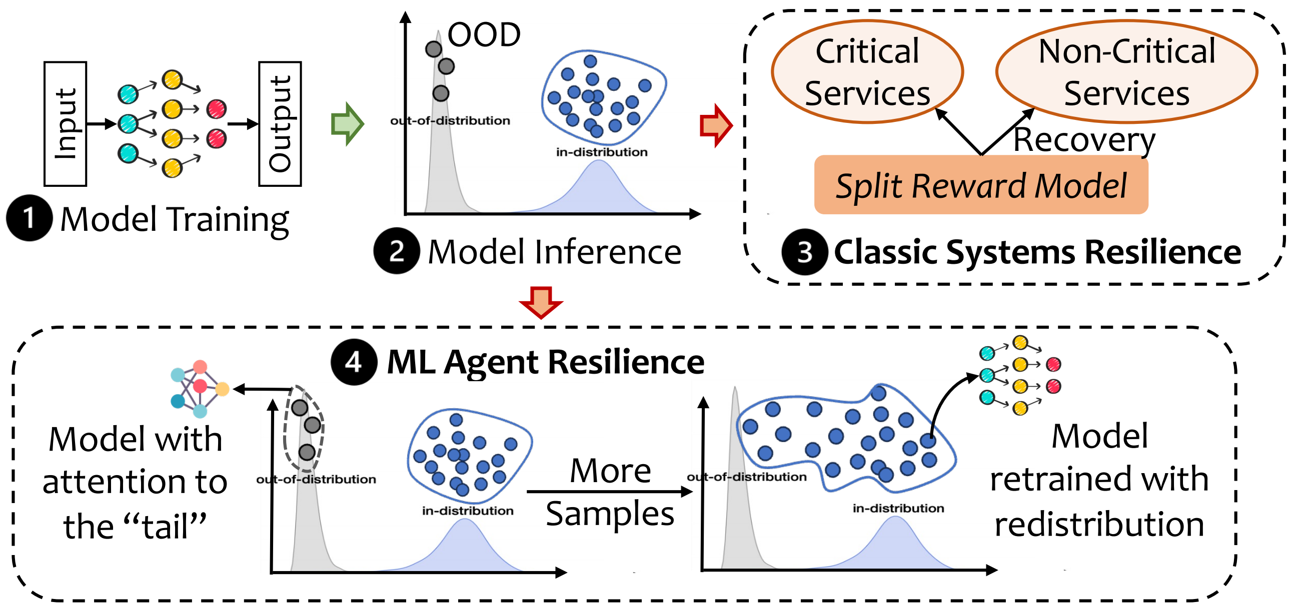
- **Binary Decision Variable** for Job Placement: $place_{b,s}(t)$, $place_{l,s}(t)$
- **Power Consumption:** server $P_s(t)$, LC job $P_l(t)$, BE job $P_b(t)$
  - $P_s(t) = \sum_l P_l(t) \cdot place_{l,s}(t) + \sum_b P_b(t) \cdot place_{b,s}(t)$

RL Autoscaler, e.g., FIRM (OSDI20)

- **Constraints:**
  - $\sum_s place_{b,s}(t) \le 1, \forall b, t;$ $\sum_s place_{b,s}(t) + delay_b(t) = 1, \forall b, t;$ $\sum_s place_{l,s}(t) = 1, \forall l, t$
  - $\sum_t \sum_{b,s} place_{b,s}(t) > Daily\_Threshold$
- **Minimize** Total Carbon Footprint: $\sum_{s,t} P_s(t) \cdot CI(t)$

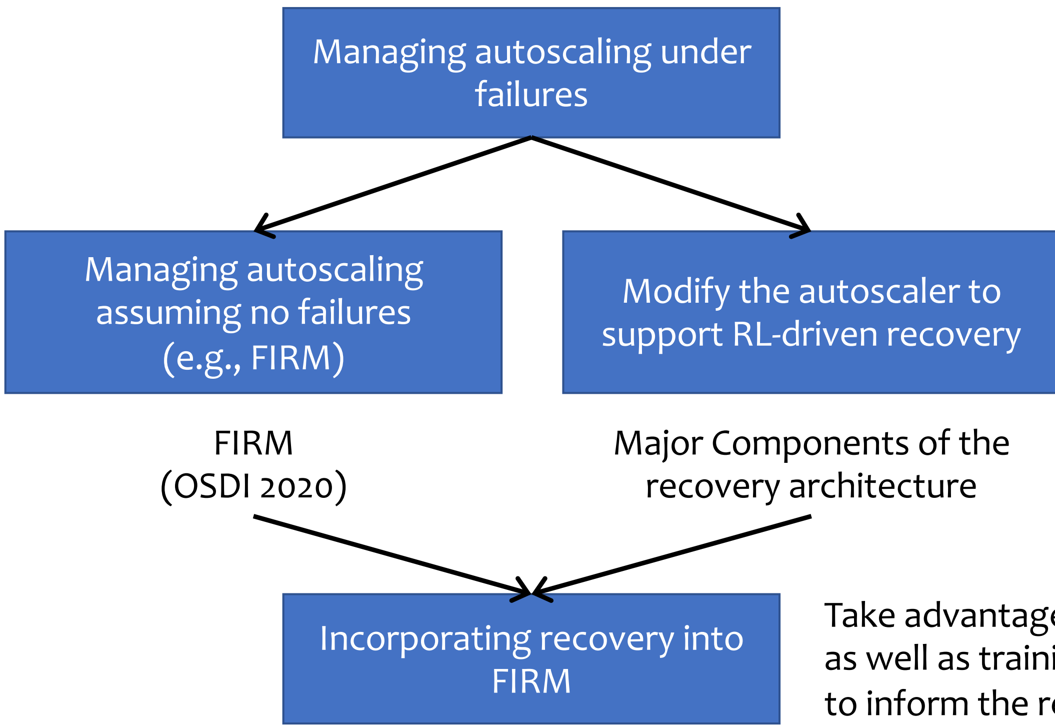
How to achieve continuous, fast re-optimization (recovery) under system + ML failures?

Fast Recovery from Systems-ML Failure Domains



Fast detection of OOD and differential service recovery are critical

Robust and Reliable ML for Sustainable Computing – Autoscaling as an Example



Take advantage of FIRM's key architecture as well as training/inference process (RL) to inform the recovery

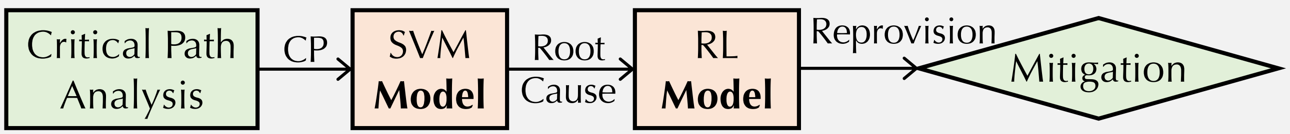
FIRM: An Intelligent Fine-Grained Resource Management Framework for SLO-Oriented Cloud Microservices

OSDI 2020

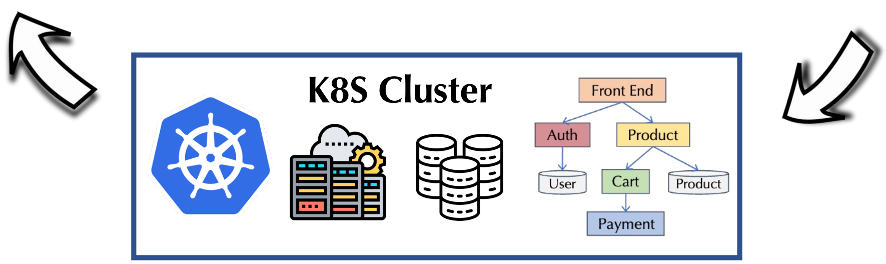
What FIRM Does in SLO Mitigation

A Two-tier ML+RL Framework

- Integrating ML/RL in SLO-oriented resource management
  - Reduces SLO violation mitigation time by up to 9×
  - Reduces the average tail latencies by up to 11×
  - Reduces the overall average requested CPU limit by up to 62%
- Decoupling with SVM-based root cause analysis to reduce RL state-action-space
  - Interpretability & Less training
- RL to generate workload-specific SLO violation mitigation policies
- Operationalized on IBM Cloud



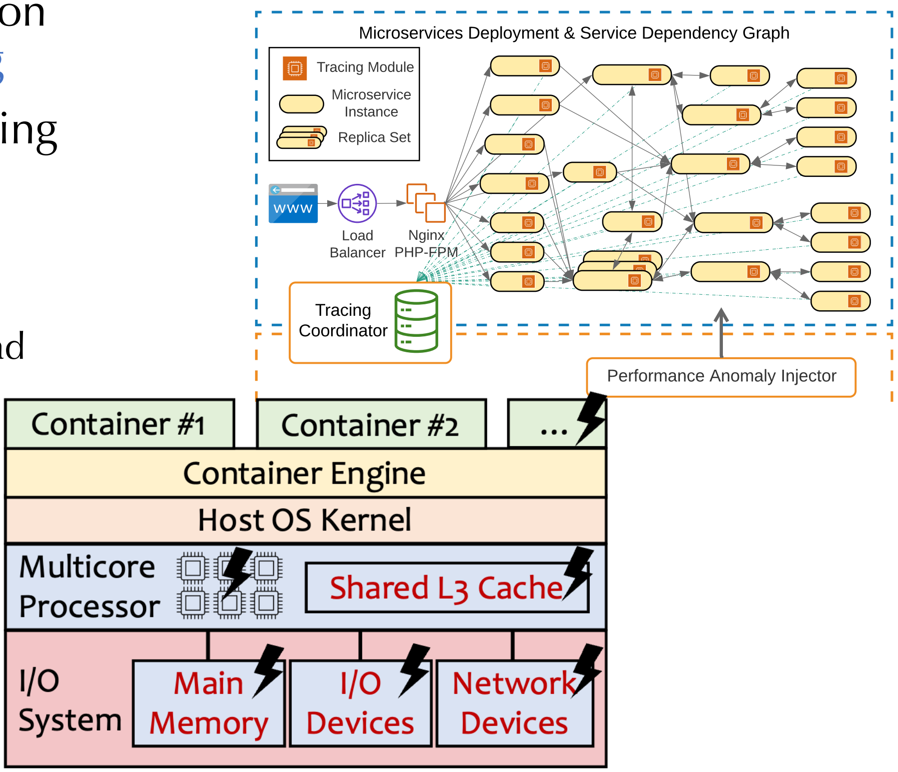
Tracing Data (App + Systems)



Resource Reprovisioning Actions

Data for State Inference

- Real-time observability on request execution provided by end-to-end distributed tracing
- Recreate the anomaly and auto-label training data driven with performance anomaly injection
- States (assume that such info is available):
  - Application-level: latency, request rates, payload
  - OS-level: CPU/memory utilization, network bandwidth, I/O usage, cache hit/misses

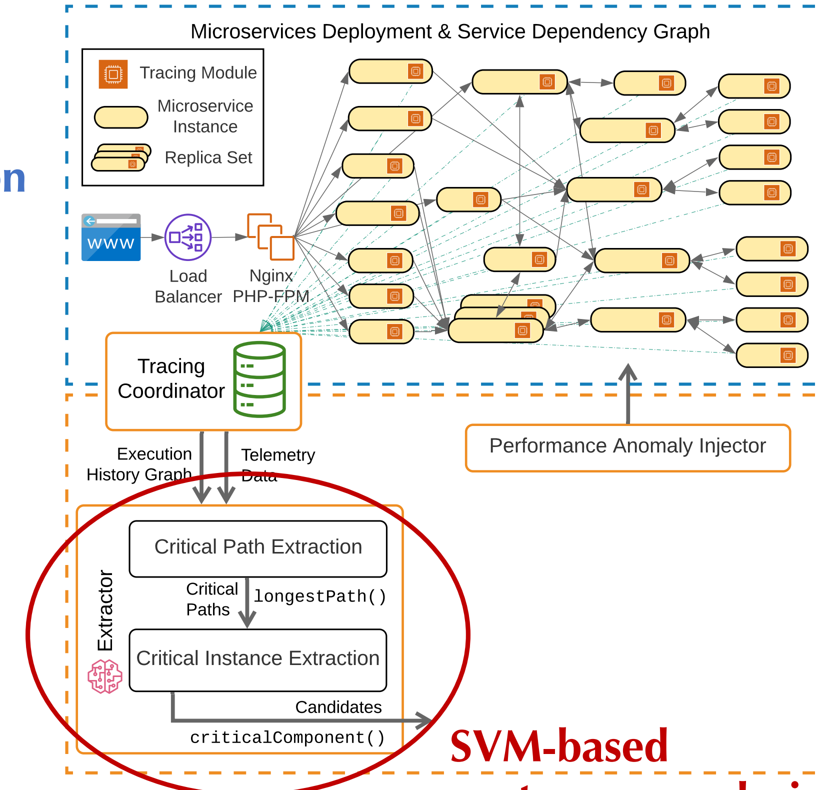


Anomaly Model

Step #1: Identifying Critical Components with SVM-based Root Cause Analysis

Which microservice instance should we focus on?

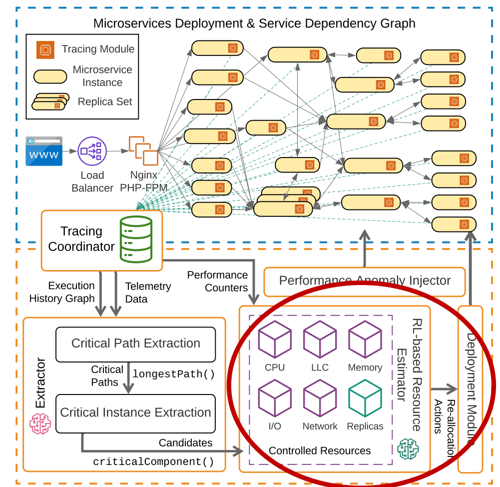
- **SVM-based critical component localization**
  - Given individual latency vector T_i, and end-to-end latency vector T_{CP}
  - **Relative importance (RI)**: Pearson correlation coefficient between T_i and T_{CP} -> Variance explained
  - **Congestion intensity (CI)**: 99-th percentile value divided by various percentiles (e.g., median) of T_i -> Chance of improvement
  - **SVM(RI, CI) -> binary output: Y or N** microservice candidate for SLO violations



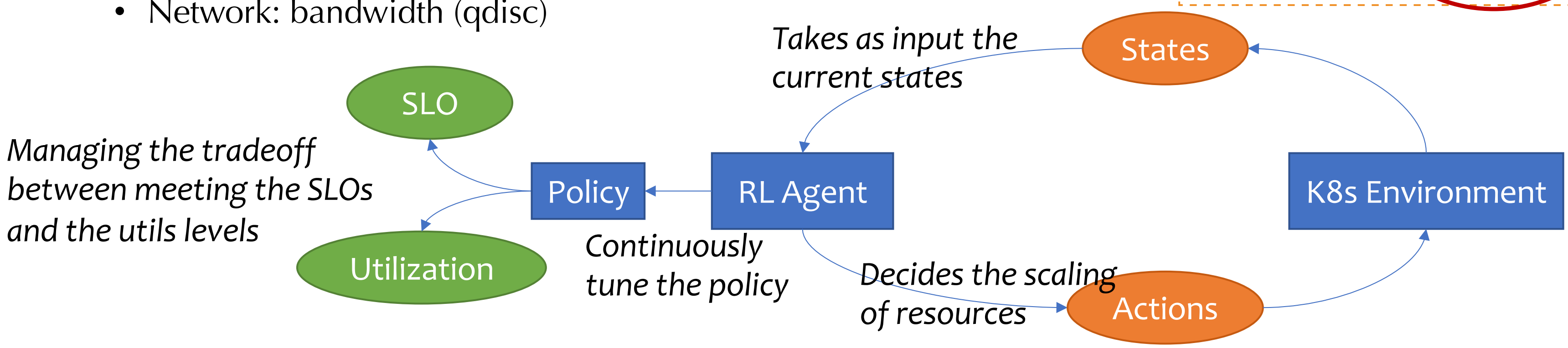
SVM-based root cause analysis

Step #2: SLO Violation Mitigation with RL

- **SLO violation mitigation action generation based on RL**
  - Identifies low-level resource in contention (state approximation)
  - Estimates reprovisioning resources to mitigate the SLO violation (action inference)
  - **Action model:**
    - CPU: CPU limits
    - Memory: capacity + bandwidth
    - LLC: capacity (intel-cat)
    - I/O: bandwidth (blkio)
    - Network: bandwidth (qdisc)

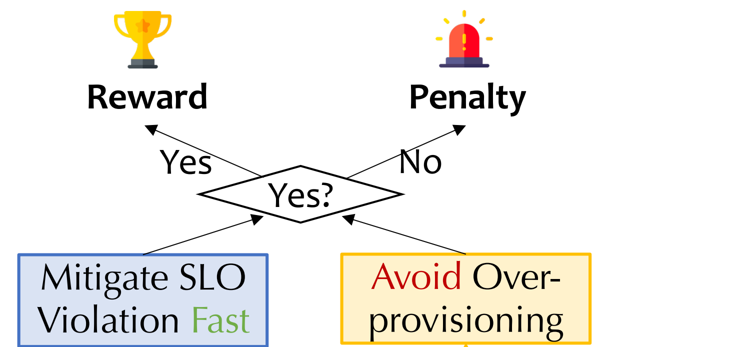


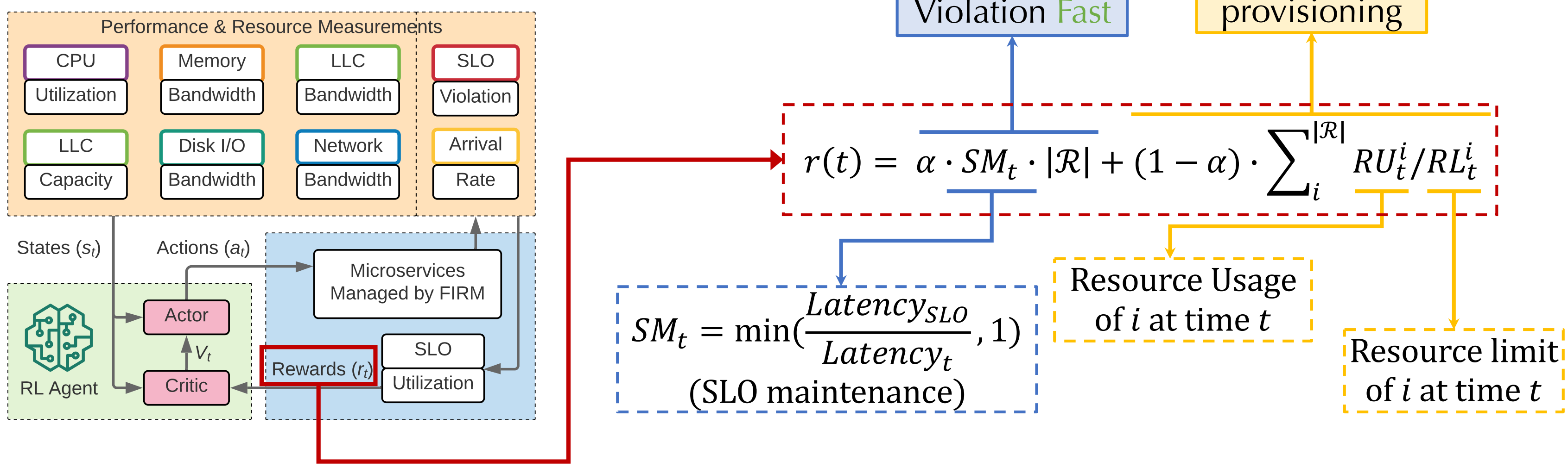
RL provides a feedback control-based dynamic environment



RL Formulation and Reward Function

- Optimizes end-to-end objectives:
  - Maximize resource utilization efficiency
  - Minimize SLO violation
  - (assuming no failures in the systems)

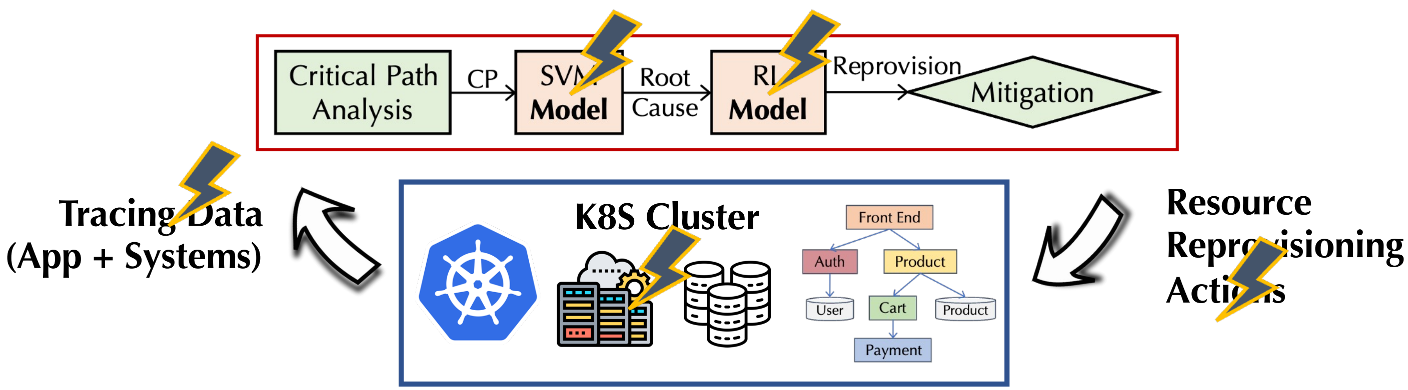




FIRM in the Process of Handling Cloud Failures and Recovery

Case Study: Handling Failures in Cloud Systems

- **FIRM** represents a category of ***learning-based systems management*** solutions
  - Application-centric for sustainable computing
  - Learned model is from the traces/dataset generated from the application running on the cloud environment
- However, when deploying such ML/RL agents in production cloud systems, it is critical to ensure the **robustness** and **reliability** of the learned models in:
  - Handling failures in the systems (maintain some of the critical services as the bottom line) without violating any SLAs/SLOs, especially for those *mission-critical* applications.



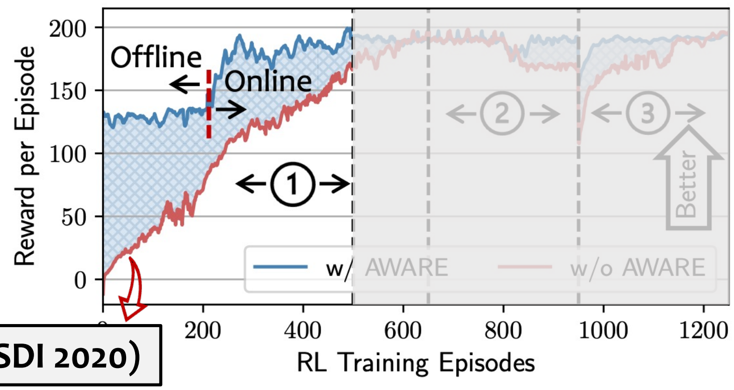
Problem Statement

- Take FIRM as the basis, which will function well if there's NO failure
- Now, your cloud is hit by a series of failures that significantly impact the normal operations (latency/availability SLOs) of your managed services
- Your goal is to design a mitigation strategy by re-engineering the RL solution to maintain the SLOs for critical applications (hospitals, financial sectors) while tolerating a lower SLOs for non-critical applications
- In doing so, you need to re-engineer the RL solution (e.g., the reward function) to bring back the system to its normal functionality

Failure Example #1

In the early training stages, RL agents tend to generate poor autoscaling decisions (due to RL exploration)

- Lower than baseline rewards (i.e., worse agent performance) and more SLO violations



FIRM (OSDI 2020)

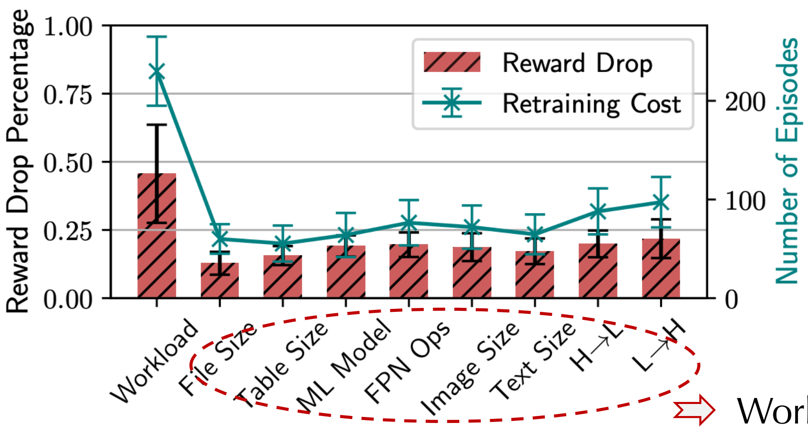
RL Episodes	EP #1-100	EP #101-200	EP #201-300	EP #301-400
CPU Util	-32.3% ± 14%	-42.9% ± 15%	-22.1% ± 12%	-10.0% ± 6%
Memory Util	-28.8% ± 11%	-30.5% ± 10%	-26.5% ± 8%	-7.8 % ± 2%
SLO Violations	56.1 ± 14x	22.2 ± 7x	12.7 ± 5x	10.1 ± 3x

Overprovisioning -> CPU & memory utils deficit compared w/ baseline

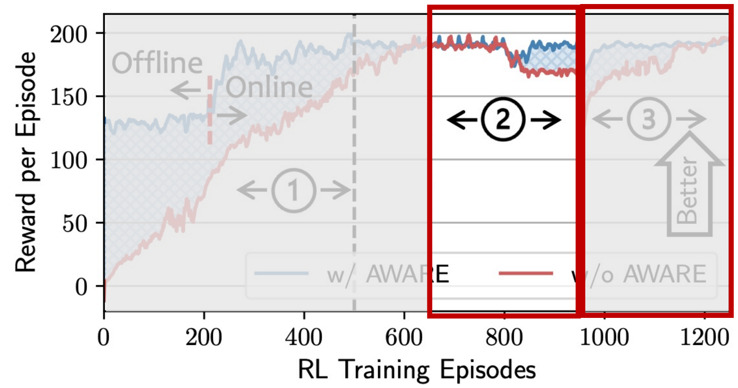
Unable to re-scale properly for workloads changes -> SLO violations

Failure Example #2

During policy-serving stage, RL agent performance degrades when workloads are updated



Workload changes leads to 21.8% reward drops



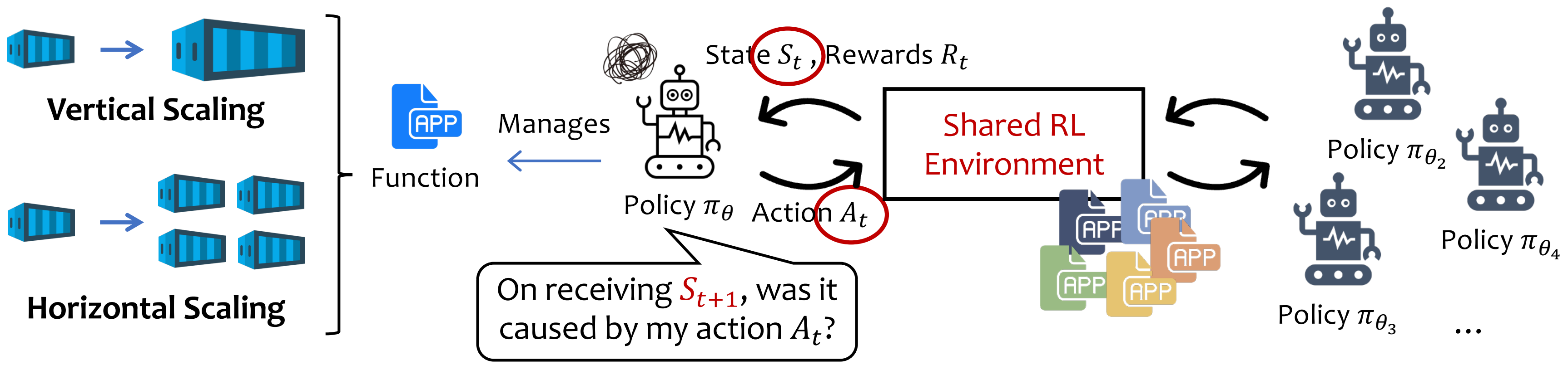
Trained policies are application-specific, costly to adapt to new applications

- 45.6% reward degradation (~230 eps retraining)

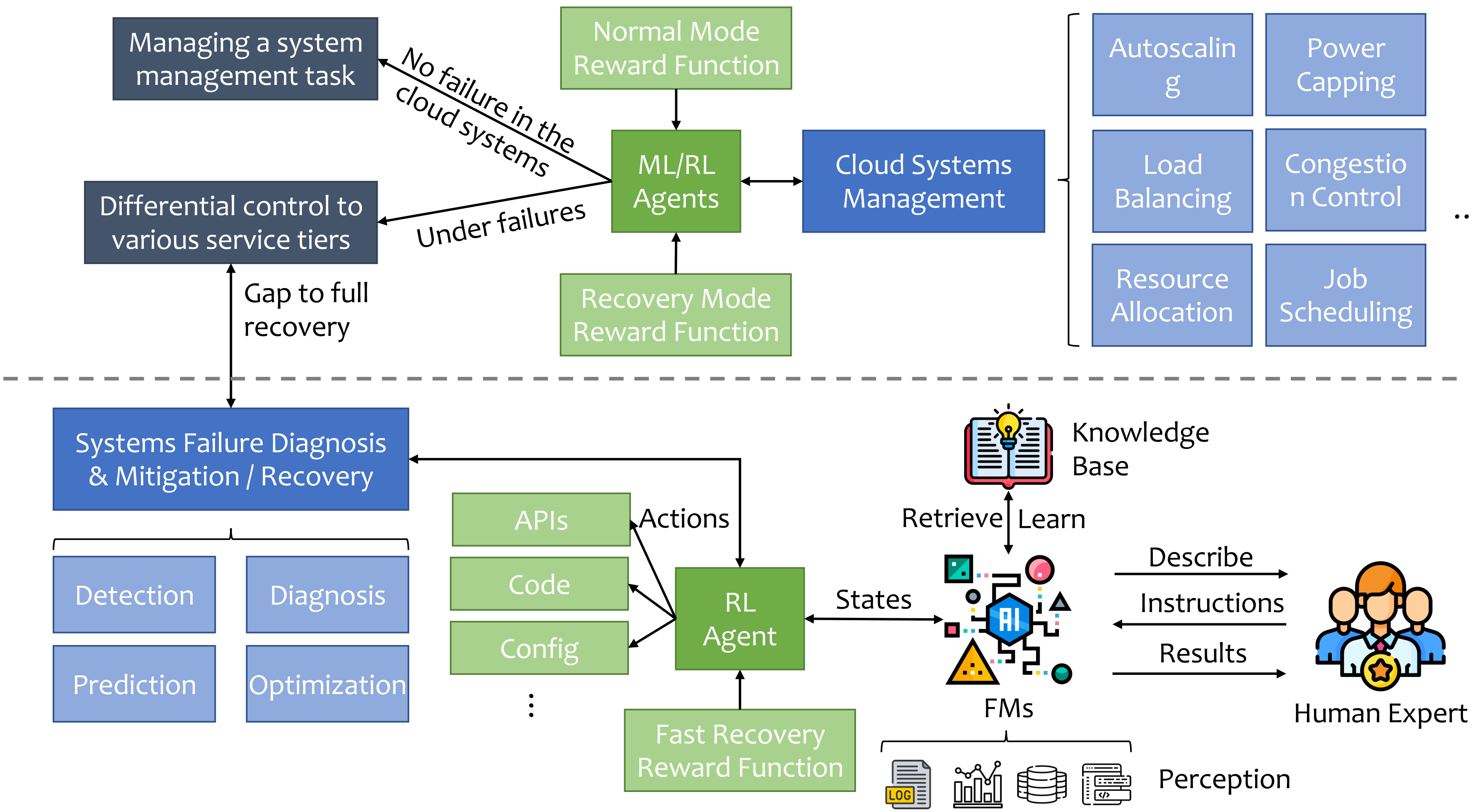
Enabling built-in intelligence in cloud systems with less manual intervention while achieving high robustness and self-adaptation (in both training/inference)

Failure Example #3

- **Challenges due to Scalability and Multi-tenancy**
  - **RL-based solutions** for resource management / autoscaling: e.g., FIRM
  - A single RL agent in an isolated environment – which we call "**single-agent RL**"
- RL assumes that the underlying environment is **stationary** (state transitions)
  - **Not true anymore!** from each agent's perspective when multiple self-interested RL agents are added to manage diverse function workloads (single-agent RL not aware of the others)



An Example Solution



Managing a system management task
No failure in the cloud systems
Normal Mode Reward Function
ML/RL Agents
Cloud Systems Management
Autoscaling
Power Capping
Load Balancing
Congestion Control
Resource Allocation
Job Scheduling
...
Differential control to various service tiers
Under failures
Recovery Mode Reward Function
Gap to full recovery
Systems Failure Diagnosis & Mitigation / Recovery
Detection
Diagnosis
Prediction
Optimization
APIs
Code
Config
Actions
RL Agent
Fast Recovery Reward Function
States
Knowledge Base
Retrieve
Learn
FMs
Describe
Instructions
Results
Human Expert
LOG
Perception

Discussion

- **Systems + ML Resilience**
  - New fault model that combines the intricate relation between system and ML failures is needed
  - Fast recovery under the new fault model
- **Scalability**: How to make the optimization framework scalable to the large number of applications and servers in a datacenter cluster
  - Introducing hierarchy -> How to deal with out of capacity and job migration
- **Feasibility** of optimization solution: How to assess the feasibility?
  - E.g., cluster capacity is enough for all LC job to meet SLOs
  - Especially when there are failures or capacity loss in the cluster, feasibility is affected
- **Time granularity**
  - Energy optimization and power management in the level of minutes or hours
  - Resource management and ML/RL agents are in the level of seconds
  - When to trigger the optimizer to run (i.e., frequency)

CPU Cloud Efficiency with ML

Microservices

FIRM

- OSDI 2020

Serverless Computing

SIMPPO

- SoCC 2022, NeurIPS 2022
- MLSys @NeurIPS 2023

GPU Cloud Efficiency with ML

DL Model Serving

µ-Serve

- ATC 2024a
- AIOps 2024

Disaggregated Memory

INDIGO

- ASPLOS*, COMPSYS 2022
- NSDI 2025*

Robust ML for Systems

Cloud Heterogeneity

FLASH

- MLSys 2024

Reliable RL Exploration

AWARE

- ATC 2023, NeurIPS 2023

Cloud Multi-tenancy

MAPPO

- EuroMLSys 2022
- WoSC 2021

Holistic Optimization with Renewable Energy & Embodied Carbon Emission

- NSF WSCS 2024, DSN 2024

Back up slides

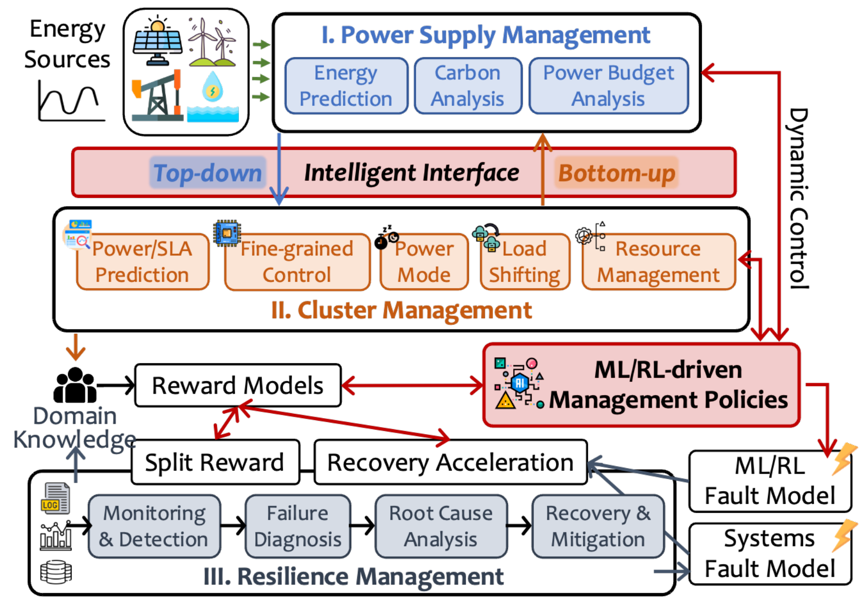
Dependable Transition to Green Computing

Two-fold meaning of sustainability:

- **Sustainable Energy/Carbon Cost**: Minimize carbon footprint
- **Sustainable Performance**: Multi-tenant clouds need to deliver consistent SLA/SLOs

Key Research Questions:

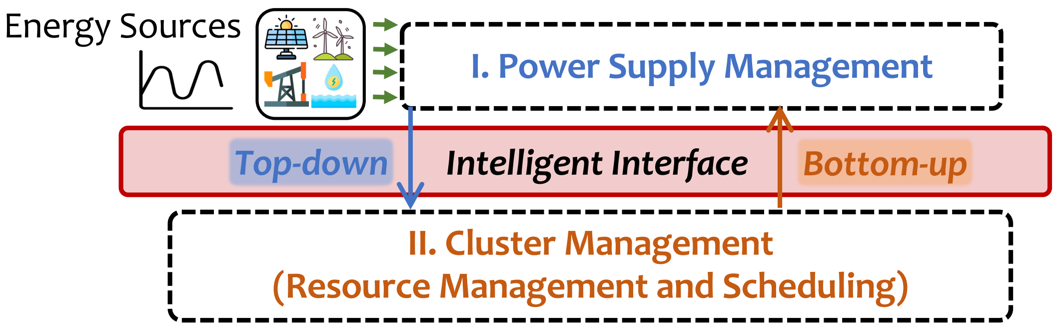
- How to achieve resilient, **SLO-driven** dynamic optimization of **green energy** usage
- How to address **System + ML resilience management**?

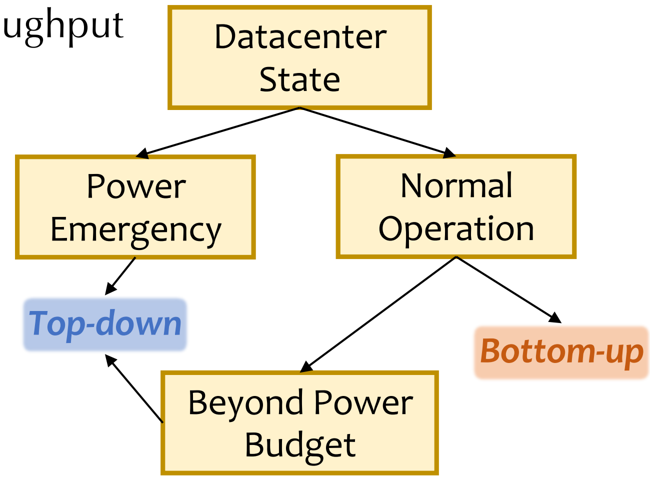


Overview of Proposal

Top-down vs. Bottom-up

- *Top-down* approach (MLSys Workshop @NeurIPS23)
  - Get the power cap based on carbon footprint optimization or power limits/budget
  - Resource manager adjust resource allocation accordingly to compensate reduced core frequency
- *Bottom-up* approach
  - Get the power demand based on the resource + frequency required to meet SLOs
  - Aggregate to get the power demand distribution across servers/racks
  - Minimize carbon footprint while meeting daily BE job throughput



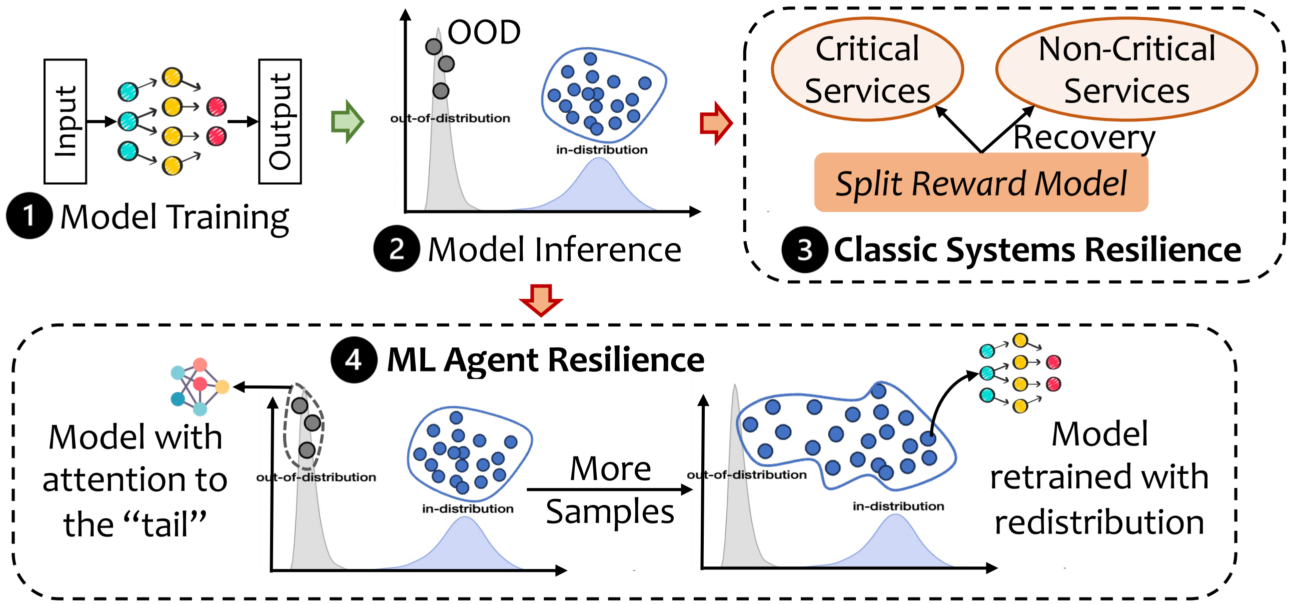


Bottom-up Approach with ML for Carbon Footprint Optimization

- **Time Window:** We assume that the total period [0, T] is partitioned into sub-periods, say $[t_k, t_{k+1})$, which could be one hour or a half-hour, i.e., "**time interval t**"
- **Carbon Intensity Forecasting:** $CI(t)$
- **Applications:** LC jobs $l \in LC(t)$, BE jobs $b \in BE(t)$
- **Servers:** $s \in S(t)$
- **Binary Decision Variable** for Delaying BE job: $delay_b(t)$ **ML Scheduler**
- **Binary Decision Variable** for Job Placement: $place_{b,s}(t)$, $place_{l,s}(t)$
- **Power Consumption:** server $P_s(t)$, LC job $P_l(t)$, BE job $P_b(t)$ **RL Autoscaler, e.g., FIRM (OSDI20)**
  - $P_s(t) = \sum_l P_l(t) \cdot place_{l,s}(t) + \sum_b P_b(t) \cdot place_{b,s}(t)$
- **Constraints:**
  - $\sum_s place_{b,s}(t) \leq 1, \forall b, t;\quad \sum_s place_{b,s}(t) + delay_b(t) = 1, \forall b, t;\quad \sum_s place_{l,s}(t) = 1, \forall l, t$
  - $\sum_t \sum_{b,s} place_{b,s}(t) > Daily\_Threshold$
- **Minimize** Total Carbon Footprint: $\sum_{s,t} P_s(t) \cdot CI(t)$

How to achieve continuous, fast re-optimization (recovery) under system + ML failures?

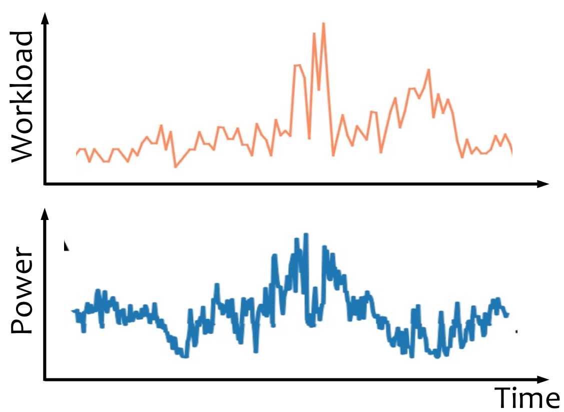
Fast Recovery from Systems-ML Failure Domains

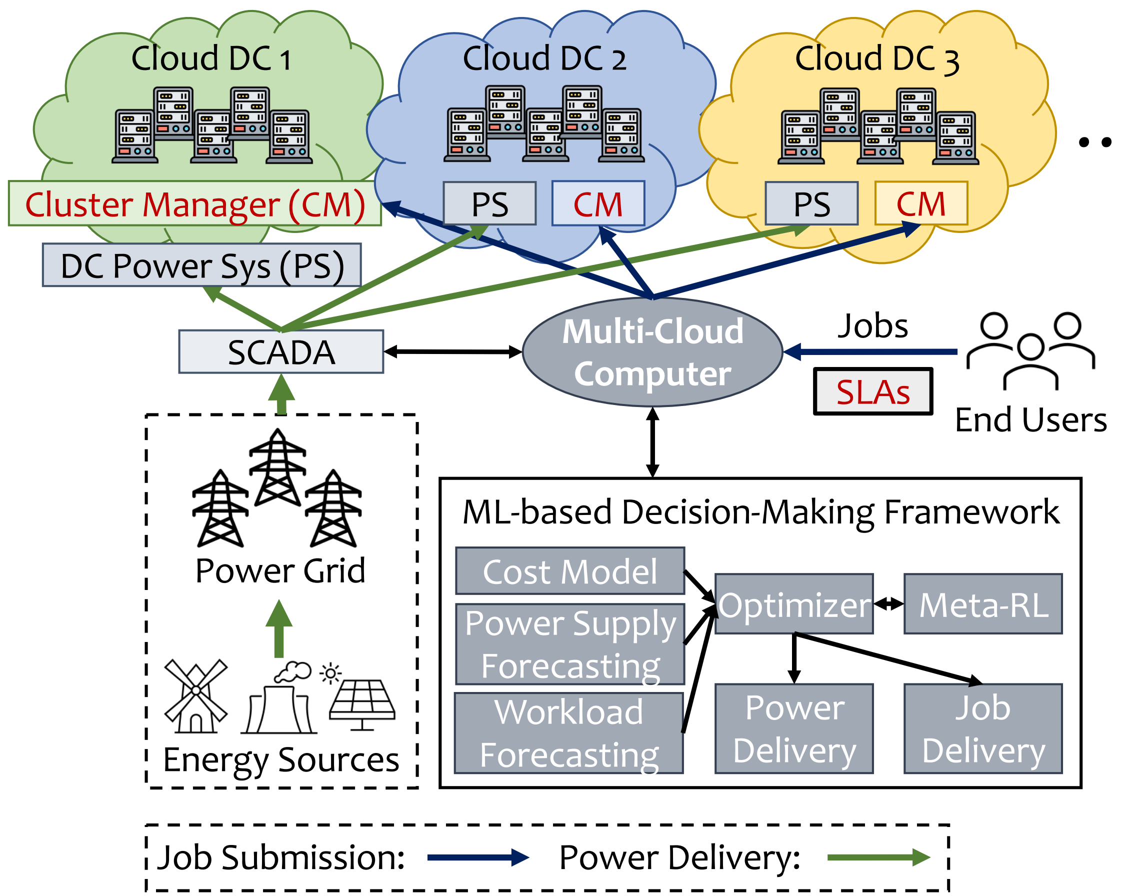


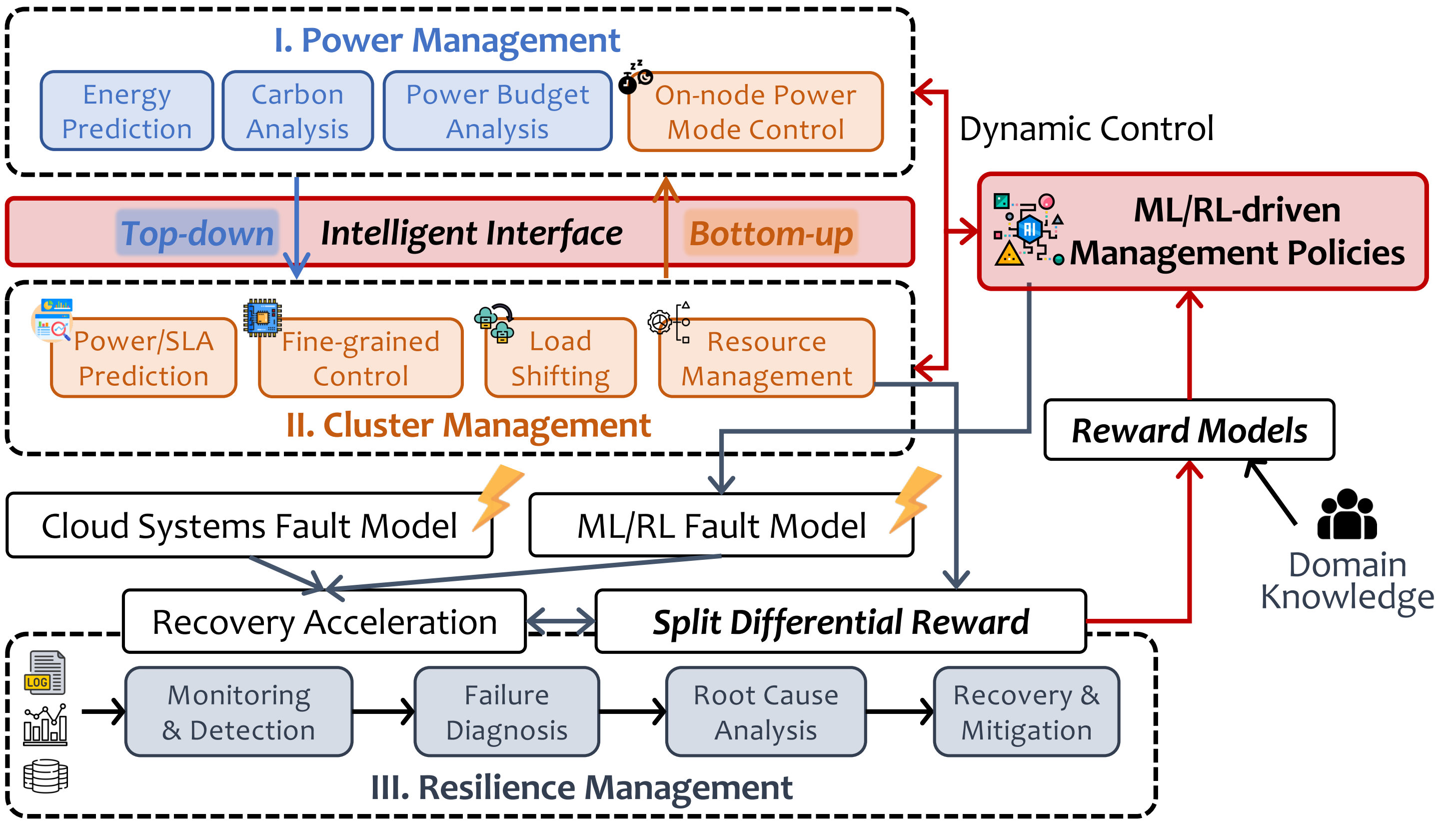
Fast detection of OOD and differential service recovery are critical

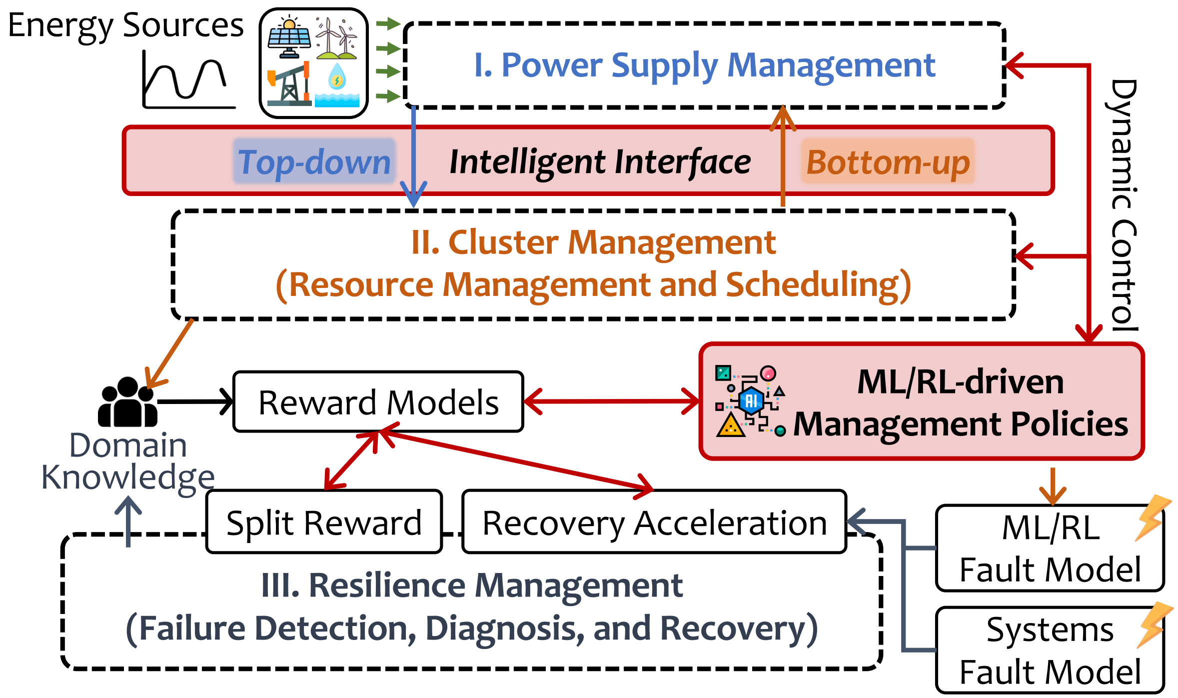
Multi-tier ML-driven Framework

- Power distribution
- Workload & power supply forecasting
  - Job characteristics
  - Load prediction
  - Power generation condition (e.g., weather) prediction

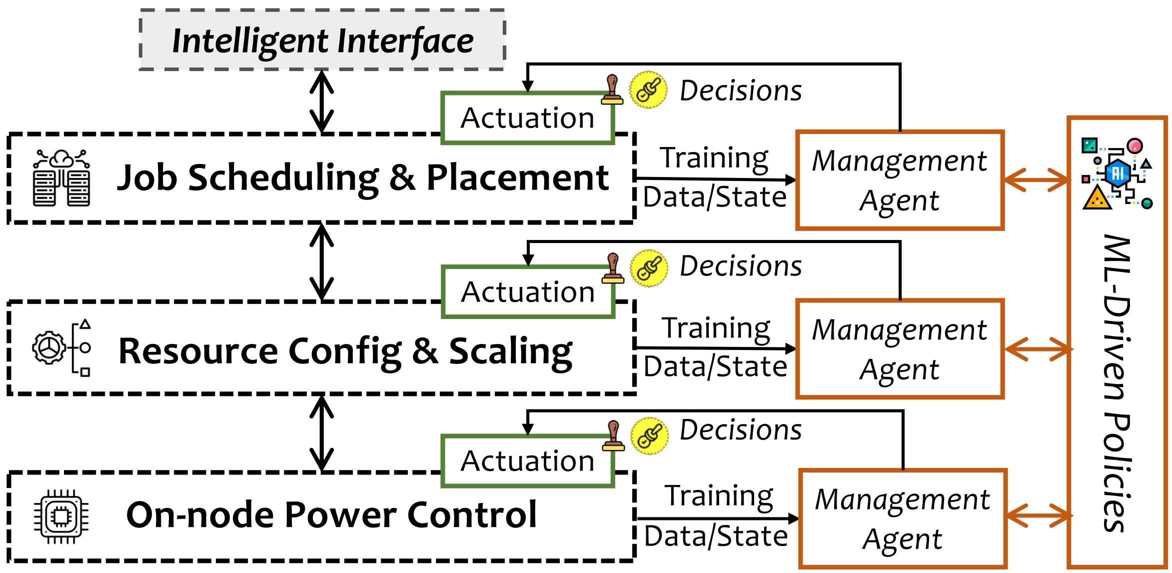





I. Power Management
Energy Prediction
Carbon Analysis
Power Budget Analysis
On-node Power Mode Control
Dynamic Control
Top-down
Intelligent Interface
Bottom-up
ML/RL-driven Management Policies
Power/SLA Prediction
Fine-grained Control
Load Shifting
Resource Management
II. Cluster Management
Reward Models
Cloud Systems Fault Model
ML/RL Fault Model
Domain Knowledge
Recovery Acceleration
Split Differential Reward
LOG
Monitoring & Detection
Failure Diagnosis
Root Cause Analysis
Recovery & Mitigation
III. Resilience Management



Energy Sources
I. Power Supply Management
Top-down
Intelligent Interface
Bottom-up
Dynamic Control
II. Cluster Management
(Resource Management and Scheduling)
Reward Models
ML/RL-driven Management Policies
Domain Knowledge
Split Reward
Recovery Acceleration
ML/RL Fault Model
Systems Fault Model
III. Resilience Management
(Failure Detection, Diagnosis, and Recovery)



Intelligent Interface
Actuation
Decisions
Job Scheduling & Placement
Training
Data/State
Management
Agent
Actuation
Decisions
Resource Config & Scaling
Training
Data/State
Management
Agent
Actuation
Decisions
On-node Power Control
Training
Data/State
Management
Agent
ML-Driven Policies