IRIS: Interference and Resource Aware Predictive Inference Serving on Cloud Infrastructures

IEEE International Conference on Cloud Computing 2023

A. Ferikoglou, P. Chrysomeris, A. Tzenetopoulos, E. Katsaragakis, D. Masouros, and D. Soudris

Microprocessors and Digital Systems Laboratory, ECE, National Technical University of Athens (NTUA), Greece

{**aferikoglou**, pchrysomeris, atzenetopoulos, mkatsaragakis, demo.masouros, dsoudris}@microlab.ntua.gr

4/7/2023

European Commission | Horizon 2020 European Union funding for Research & Innovation

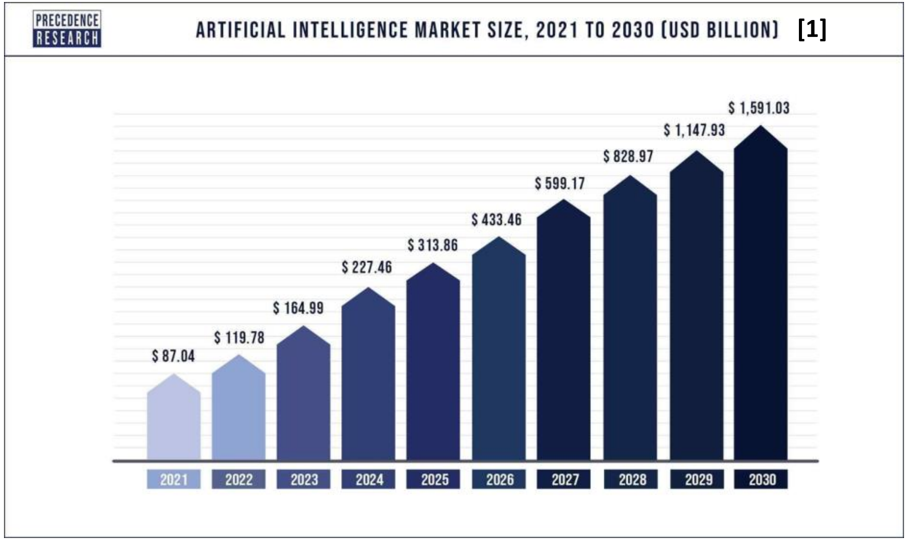
nephele

Rise of AI and SoA Model Complexity

- Ever-increased use of **Artificial Intelligence**

- Ever-increased **complexity of deployed models** in terms of:
    - **Computation**
    - **Memory**
    - **Storage**

- E.g., **GPT-3 ≈175B** parameters



Prohibitive for end-user to support Inference

[1] https://www.precedenceresearch.com/artificial-intelligence-market

MLaaS for DL Inference

- **Cloud** "as a solution" to the **resource wall challenge of DL inference**

  Inference takes **90% of total infrastructure cost [1]**

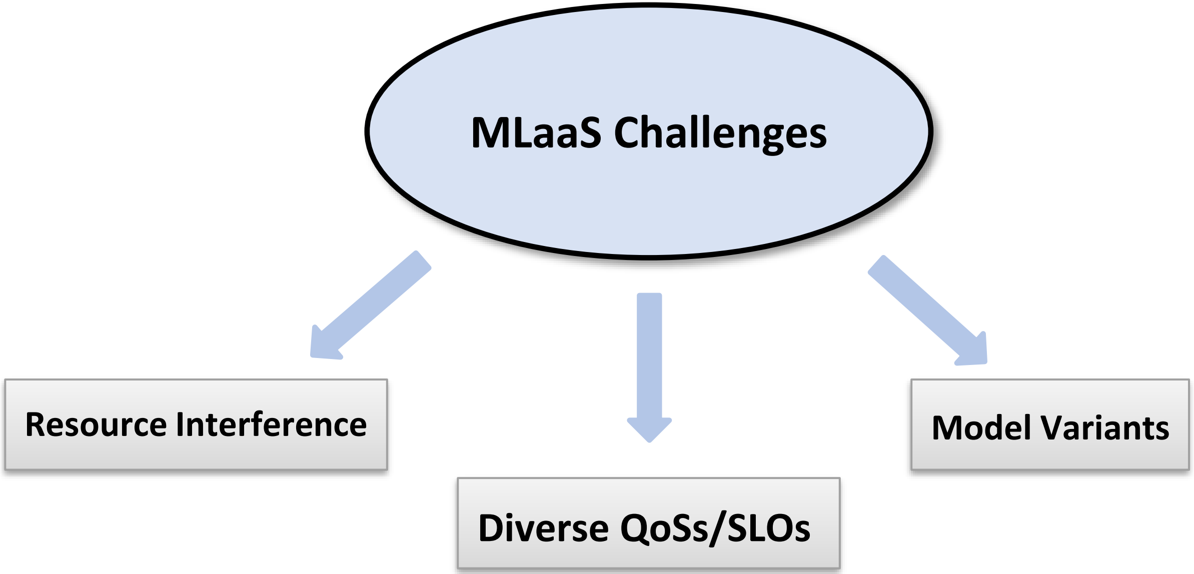
  Serves **tens-of trillions inference tasks** a day **[2]**

- To further exploit the trend **MLaaS** was introduced
  - End-user provide **pre-trained model** along with **throughput-/latency- QoS**

Amazon SageMaker   Azure Machine Learning   IBM Watson   vertex.ai

[1] "Deliver high performance ML inference with AWS Inferentia." https://d1.awsstatic.com/events/reinvent/2019/REPEAT 1 Deliver high performance ML inference with AWS Inferentia CMP324-R1.pdf. Accessed: 04-03-2023.
[2] K. Hazelwood, S. Bird, D. Brooks, S. Chintala, U. Diril, D. Dzhulgakov, M. Fawzy, B. Jia, Y. Jia, A. Kalro, et al., "Applied machine learning at Facebook: A datacenter infrastructure perspective," in 2018 IEEE International Symposium on High Performance Computer Architecture (HPCA), pp. 620–629, IEEE,2018.

MLaaS Challenges



Guarantee the user-defined QoS/SLO requirements and **maximize resource efficiency**

Inference Serving Testbed

- HW/SW Infrastructure
  - 2 VMs serving as **master** and **worker** of a **Kubernetes** cluster

VM	vCPUs	RAM
Master	4	8 GB
Worker	8	16 GB

- Inference Engine Workloads
  - **Image classification** + **Object detection** inference engines from **MLPerf [1]**
- Synthetic Interference
  - Microbenchmarks that stress **CPU**, **L2/L3 cache** and **memory bandwidth/capacity** from **iBench [2]**

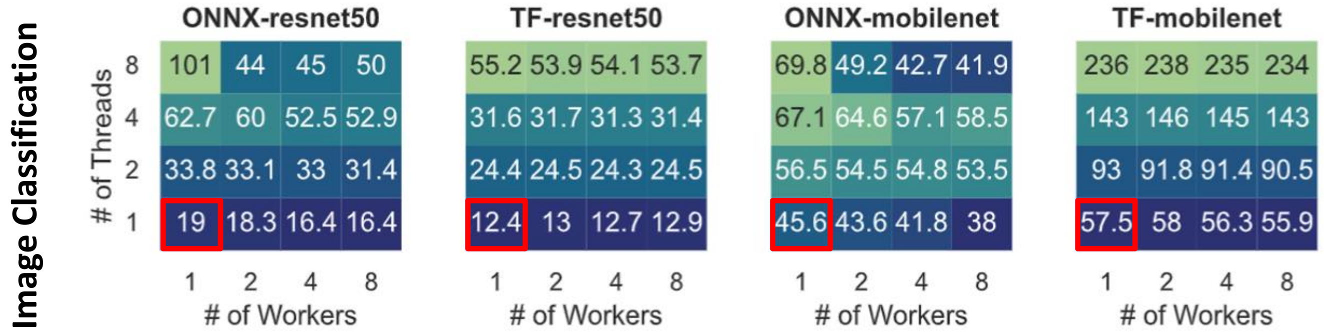
[1] V. J. Reddi et al., "MLPerf Inference Benchmark," 2019
[2] C. Delimitrou and C. Kozyrakis, "ibench: Quantifying interference for datacenter applications," in 2013 IEEE international symposium on workload characterization (IISWC), pp. 23–33, IEEE, 2013.

Characterizing Inference Serving

- Quantify the impact of the following to inference engine performance
  - **Different model representation backends** i.e., TensorFlow, ONNX Runtime
  - **Vertical/Horizontal scaling**
  - **Interference**

Isolated Execution

Q1: How do different model variants for the same task behave in terms of performance?

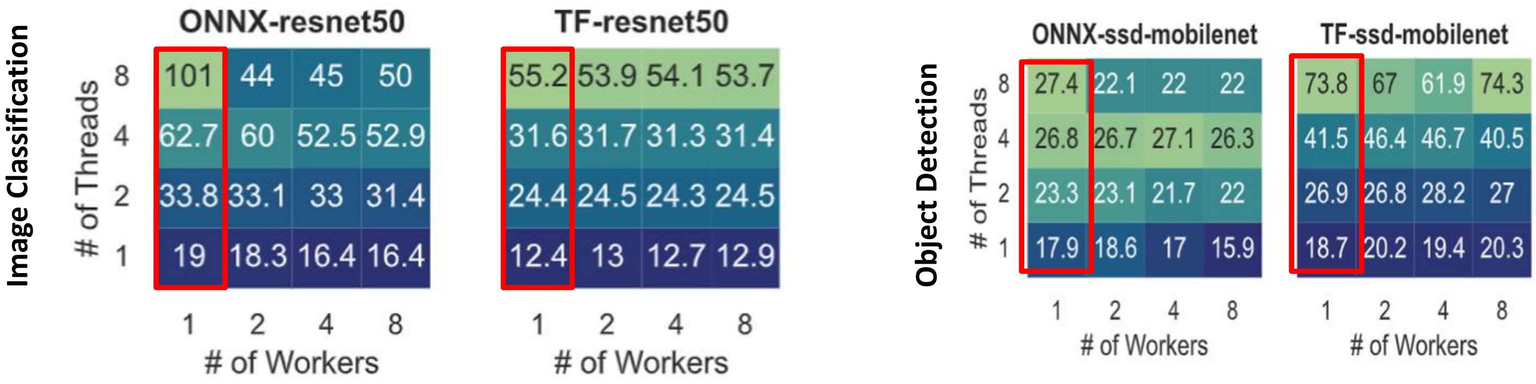


TF-MobileNet 4.6x higher QPS than TF-ResNet50

No clear dominance of TensorFlow over ONNX Runtime and vice versa

Isolated Execution

Q2: How does vertical scaling (i.e., #Threads) of resources affect performance?



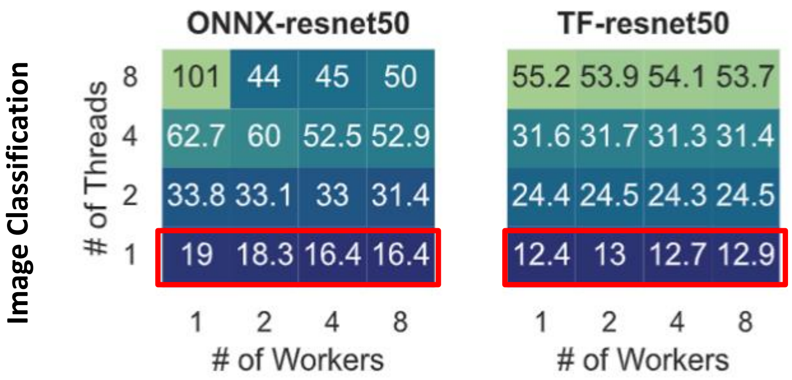
#Threads = 8 compared to #Threads = 1

2.8x higher QPS on average for ONNX Runtime

3.8x higher QPS on average for TensorFlow

Isolated Execution

Q3: How does horizontal scaling (i.e., #Workers) of resources affect performance?

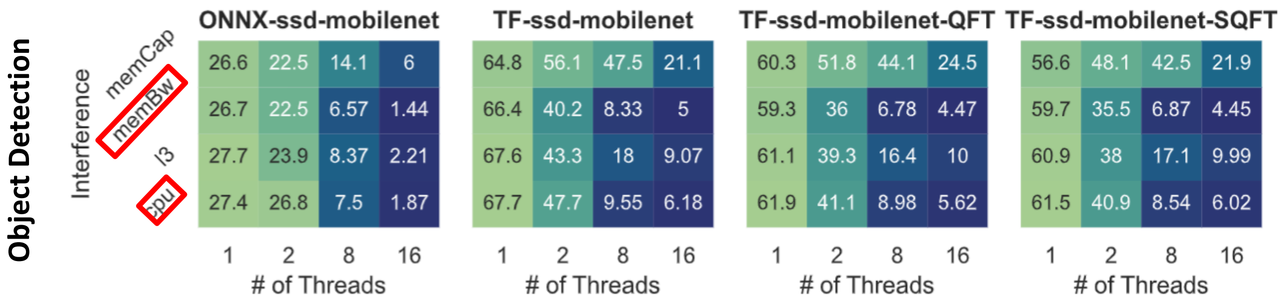


TensorFlow not affected due to CPython's GIL

ONNX Runtime presents a 14% QPS drop on average

Execution under Interference

Q4: How does resource interference affect the performance of the inference engines?



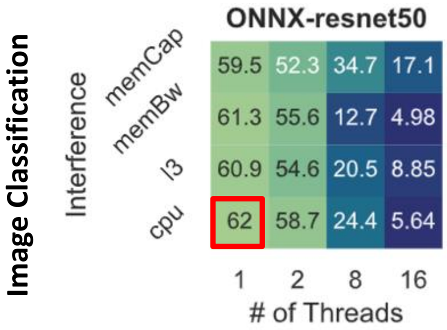
#iBench = 16 on Object Detection

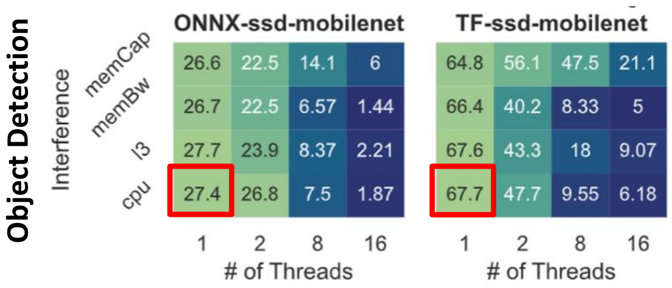
11.7x lower QPS on average stressing CPU

12.5x lower QPS on average stressing Memory BW

Execution under Interference

Q5: Do different backends reveal different performance sensitivity w.r.t resource interference ?





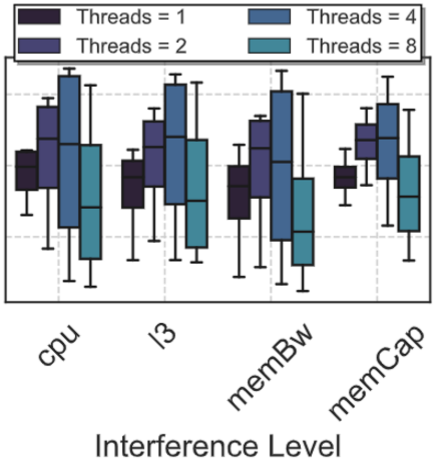
ONNX-ResNet50 1.5x higher QPS than TF-ResNet50

TF-SSDMobileNet 2.5x higher QPS than ONNX-SSDMobileNet

Execution under Interference

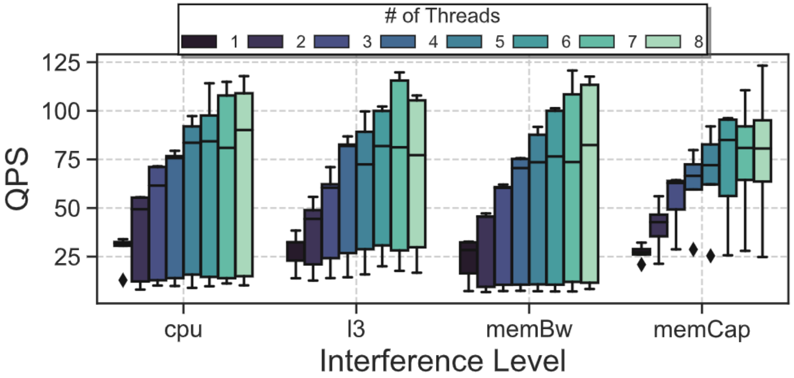
Q6: How do different resource allocations affect the performance of the inference engines under the presence of interference?

ONNX Runtime



#Threads = 2 or 4

TensorFlow



#Threads > 5

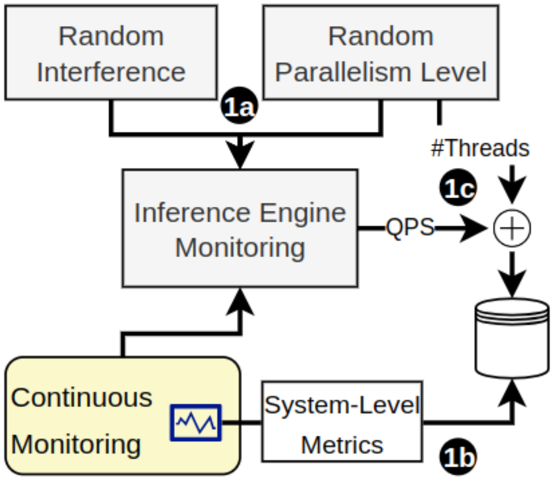
IRIS Design

- Based on our observations we design **IRIS**, an **interference**- and **resource**-**aware predictive** orchestration methodology
  - **Identifies interference effects** by exploiting **low-level performance events**
  - **Provides accurate performance predictions**
  - **Automatically applies horizontal/vertical scaling** policies and **chooses** the appropriate **model variant**

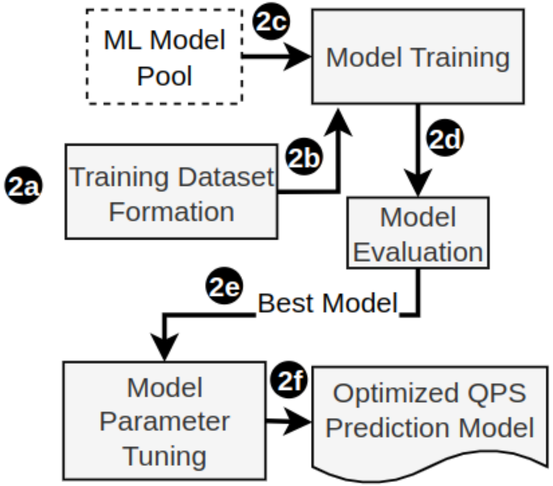
IRIS Design – Offline Phase

Inference Engine

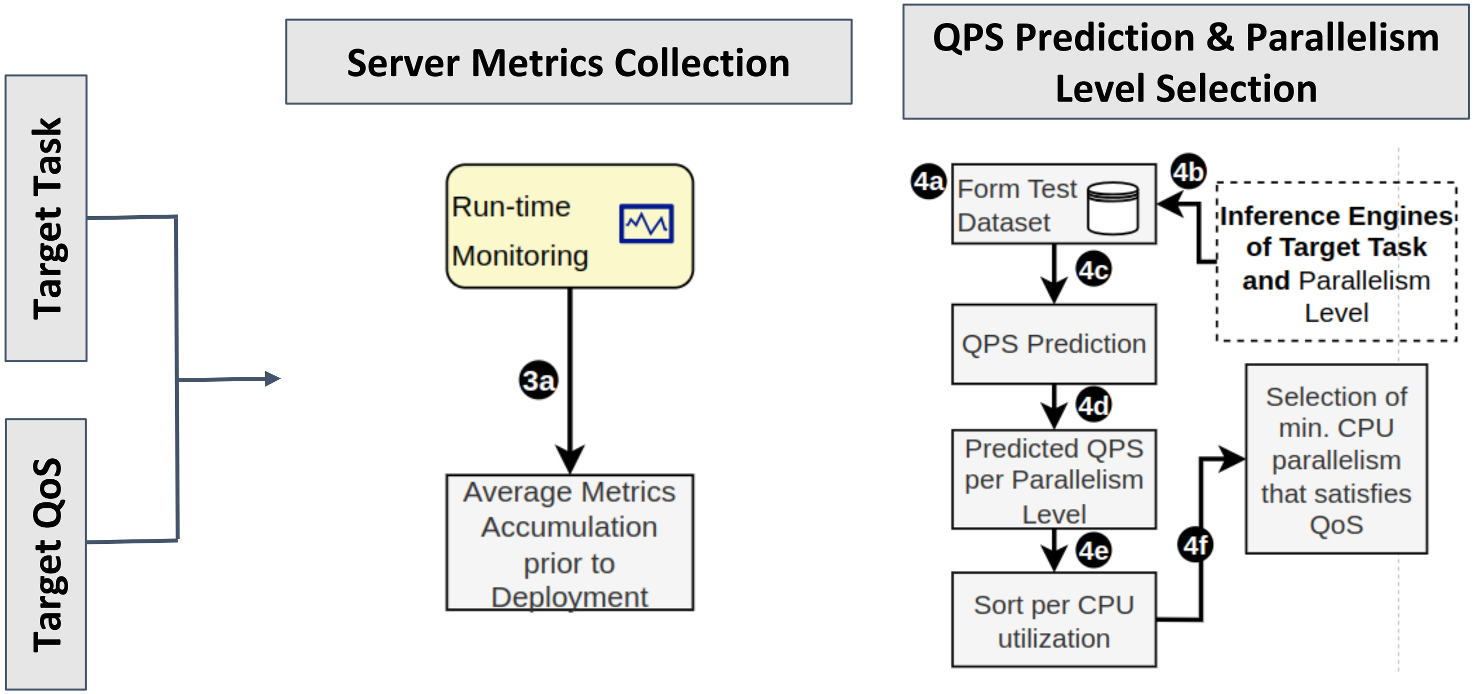
Random Scenario Execution



Training Dataset Formation, ML Model Selection and Tuning



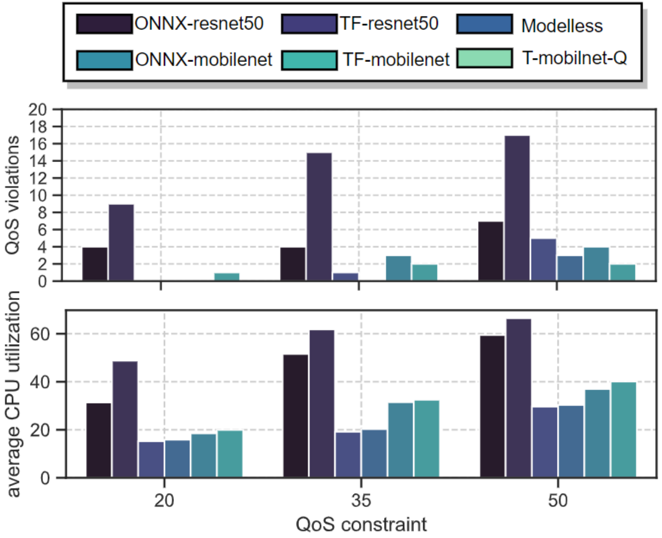
IRIS Design – Online Phase

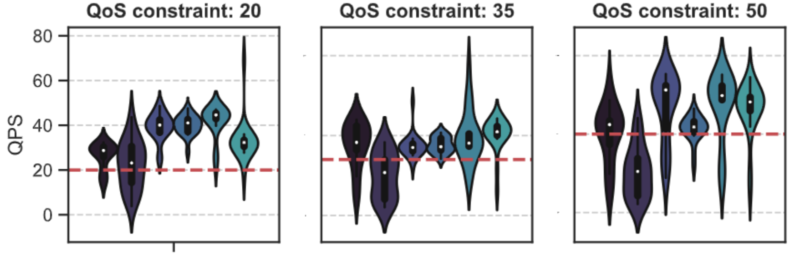


Evaluation

- We evaluate **IRIS** approach using:
  - **different interference scenarios** of varying intensity
  - **different QoS constraints** per inference engine (**Low**, **Medium**, and **High**)
- The **model-less** version of **IRIS** is compared with:
  - **All** the **interference-aware model-specific IRIS** schedulers

Evaluation – Model-less (Image Classification)



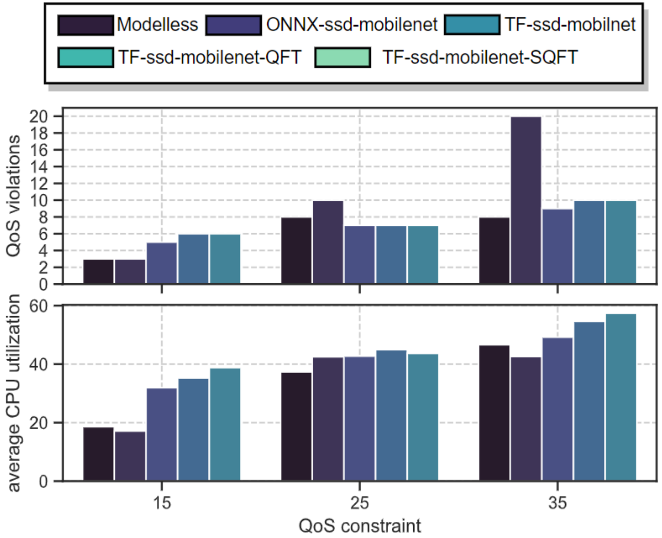


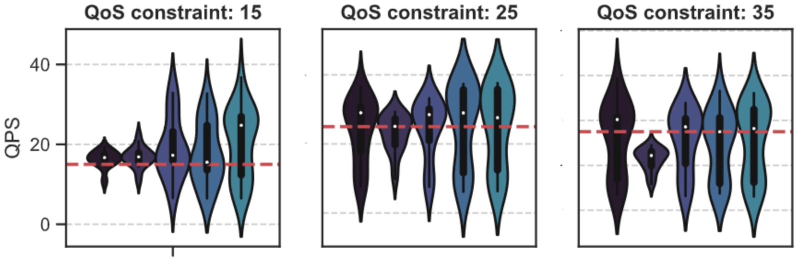
57.6% less QoS violations on average compared to **model-specific schedulers**

21.3% CPU utilization on average

2x, **1.2x**, and **1.6x** **higher QPS** for **Low**, **Medium**, and **High QoS constraint** on average

Evaluation – Model-less (Object Detection)





23.9% less QoS violations on average compared to **model-specific schedulers**

34.2% CPU utilization on average

1.4x, **1.8x**, and **1.5x higher QPS** for **Low**, **Medium**, and **High QoS constraint** on average

Conclusion

- We presented **IRIS** an **interference-** and **resource-aware predictive** scheduling framework for **ML inference serving**
  - Guarantees the **application-specific QoS constraints** while **minimizing resource utilization**
- The **model-less** feature achieves:
  - **1.5x fewer violations** on average compared to **model-specific**
  - **≈30% less CPU utilization** on average compared to **model-specific**

Thank You ☺