Serving and Optimizing Machine Learning Workflows on Heterogeneous Infrastructures

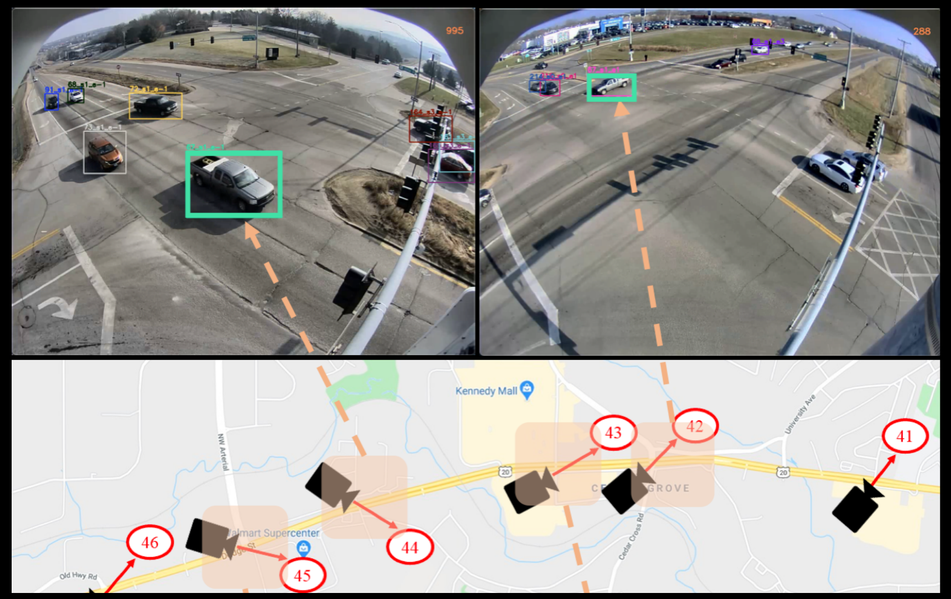
Yongji Wu, Matthew Lentz, Danyang Zhuo, Yao Lu

Duke UNIVERSITY

Microsoft

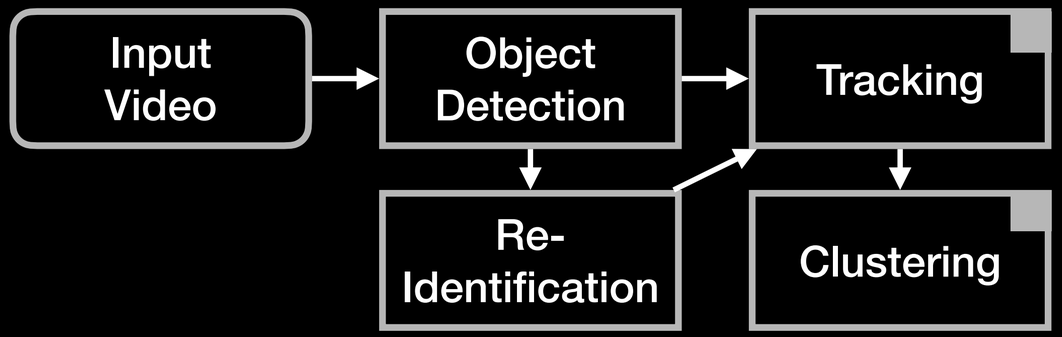
ML Workflows are Widely Deployed

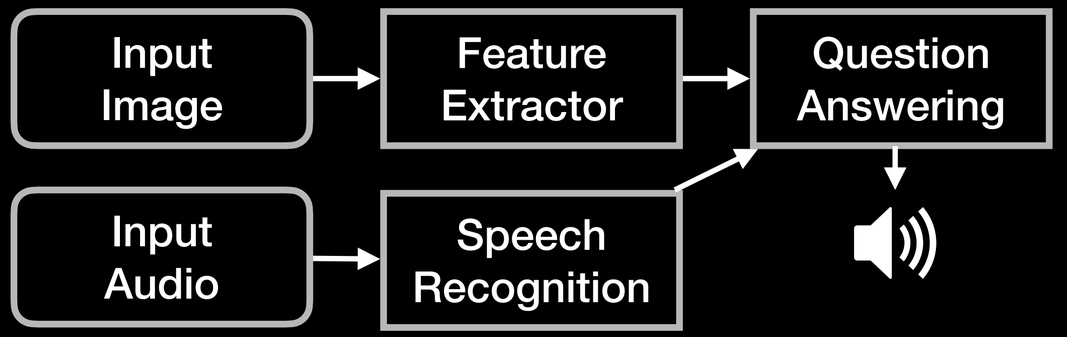
Inputs
ML Operators (Models)
Non-ML Operators



Multi-Camera Vehicle Tracking

Visual Question Answering





Growth of Heterogeneous Infrastructure

- **Inputs derived from sensors at the edge**
  - Cameras, IoT devices, etc
  - Some even have support for on-board compute

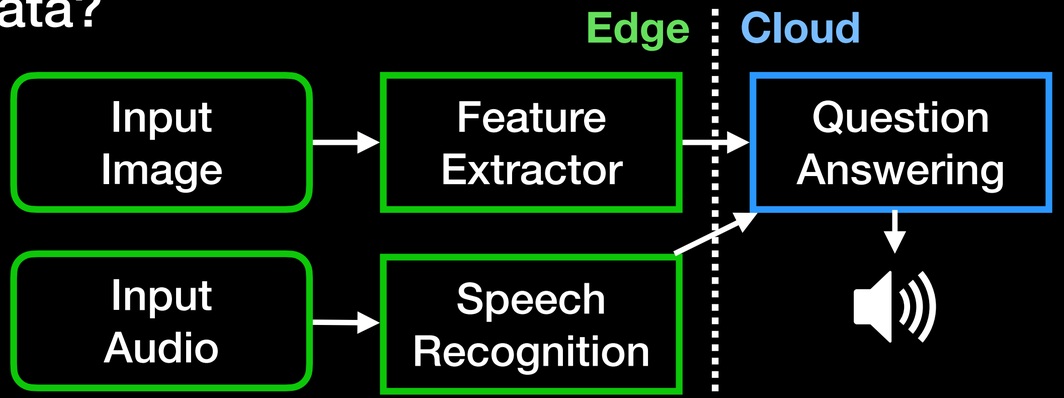
Boulder AI
DNN Cam

- **Connected to more powerful compute over network**
  - Edge-local hubs and cloud computing resources

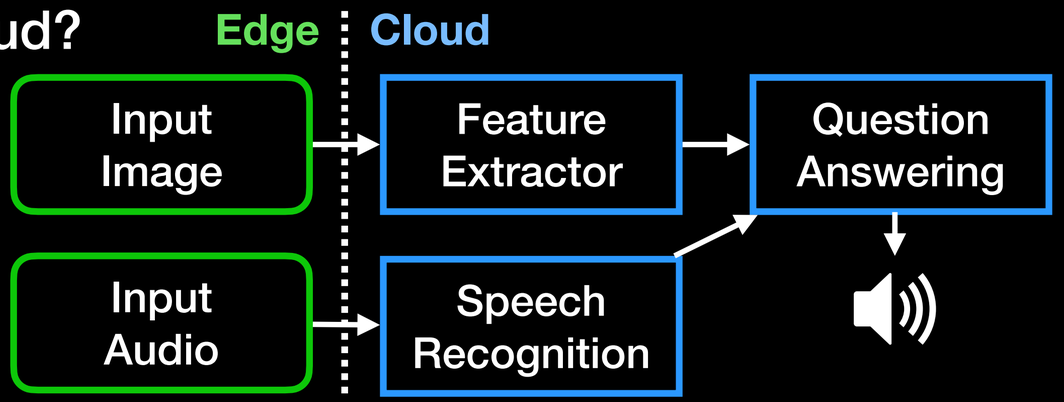
Many Choices for Serving Workflows

Worker Assignment (WA)

Near data?



On cloud?



Model Selection (MS)

Speech Recognition

#	Model	Accuracy	Size	Cost
A	wav2vec2-base	70%	94 M	Low
B	wav2vec2-large	75%	315 M	High
...				

Different architectures
Model compression techniques

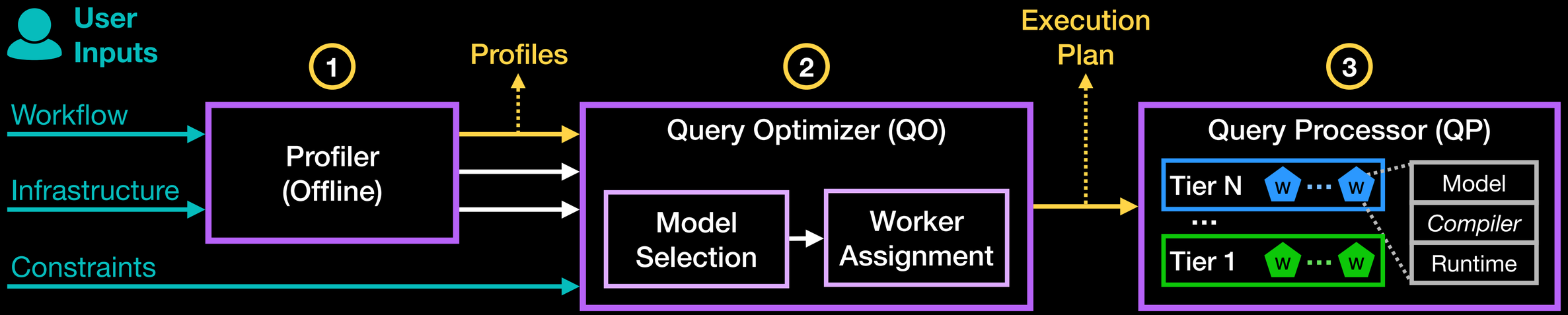
JellyBean: Optimizing Execution Plans

Goal	Minimize serving cost subject to throughput and accuracy constraints

- **Insight**: Jointly optimize worker assignment and model selection
  - Each decision is affected by the other
  - Should yield more efficient execution plans
    - However, search space becomes much larger!
- Prior work only leveraged WA →

System	Parallelism	QO	
		MS	WA
PyTorch [52]	Data	×	×
TF [12]	Data	×	×
Spark [65]	Data	×	×
Clipper [21]	Data	×	×
Ray [47]	Data, Model	×	×
Optasia [42]	Data, Op	×	✓
Pathways [14]	Data, Model	×	×
Llama [58]	Data, Op	×	✓
Scrooge [27]	Data, Op	×	✓
JellyBean (Ours)	Data, Op	✓	✓

JellyBean: System Overview

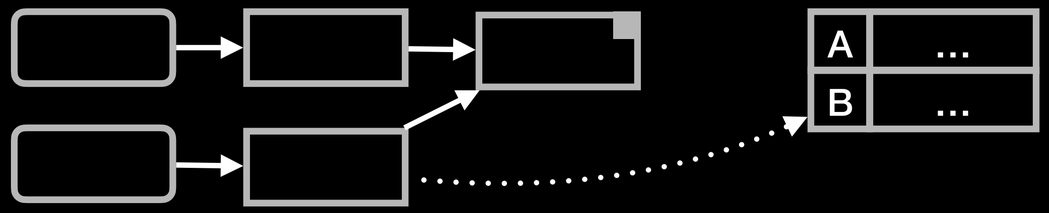


1. Profile models across workers to understand impact of model selection
2. Use profiles (and constraints) to compute an optimized execution plan
3. Deploy and run execution plan across heterogeneous infrastructure

Overview of User Inputs

Workflow

A directed acyclic graph (DAG) with nodes as inputs or operators (ML / relational)
Each ML operator may have one or more model choices
Provides a validation set for each ML operator for automated profiling



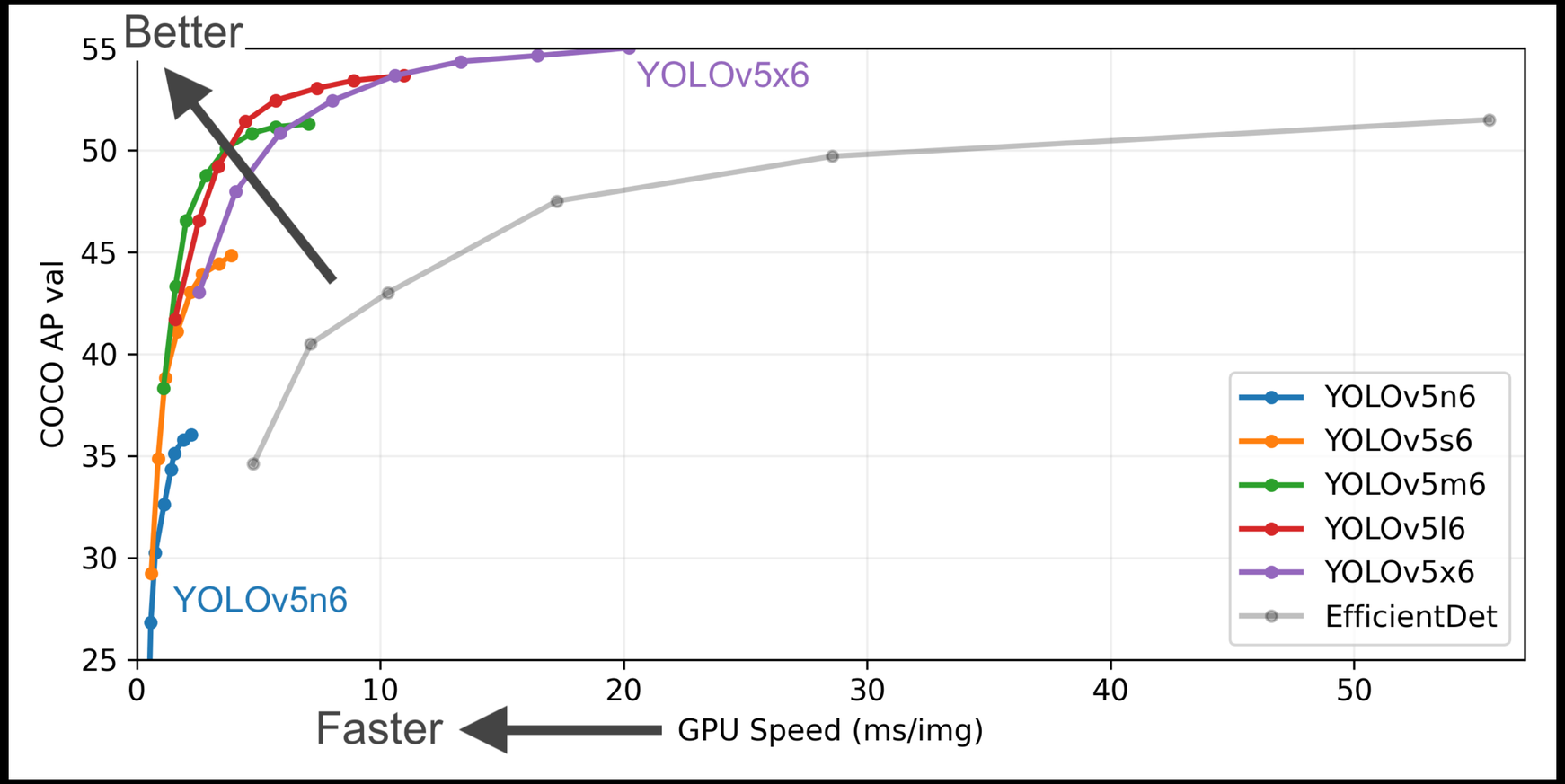
Infrastructure

Properties of the available compute and network resources:
Type (e.g., CPU, V100 GPU), Count, Bandwidth, Cost

Constraints

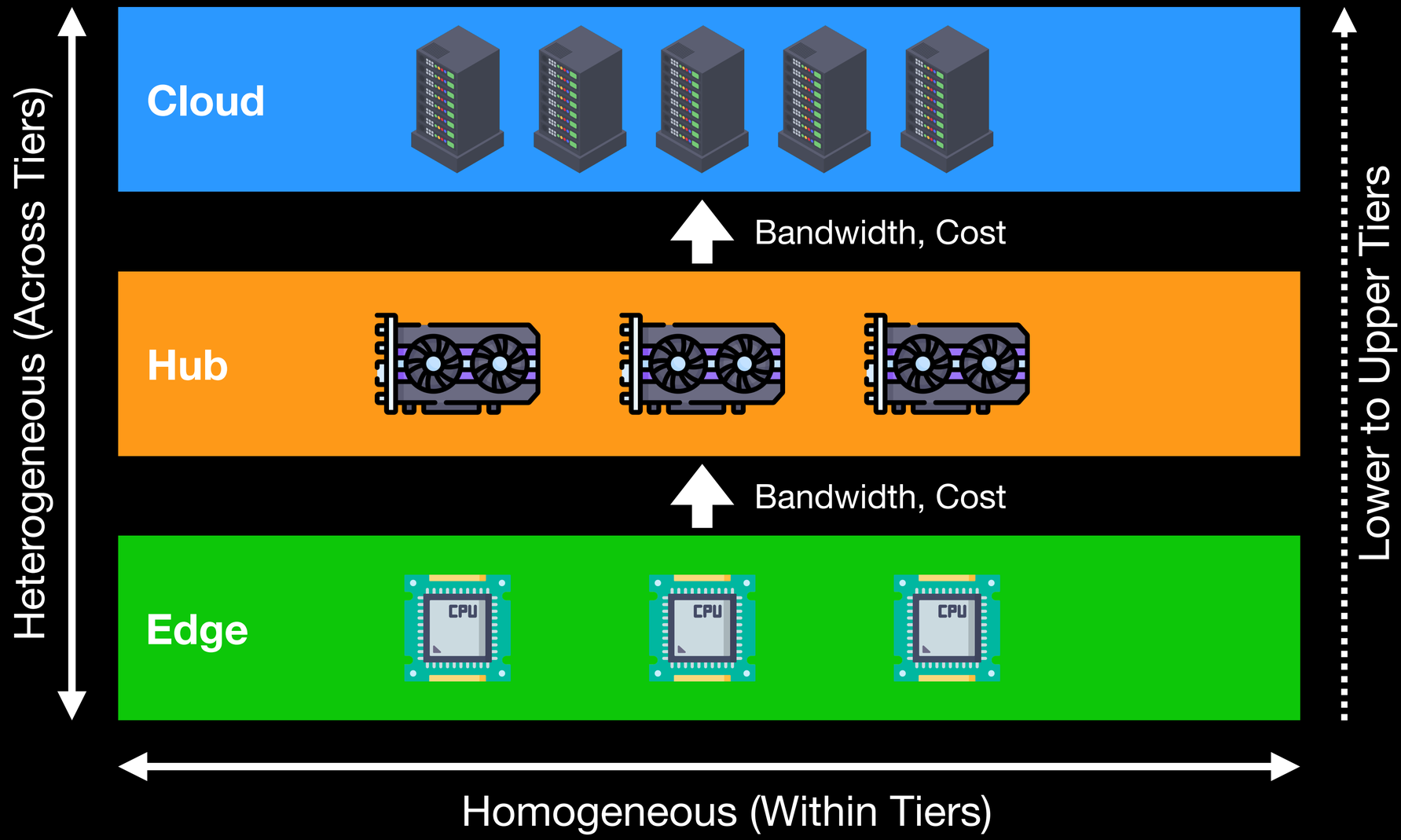
Targets for minimum accuracy and supported throughput

User Inputs: Model Choices



Many model variants with different accuracy-efficiency tradeoffs

User Inputs: Infrastructure Tiers

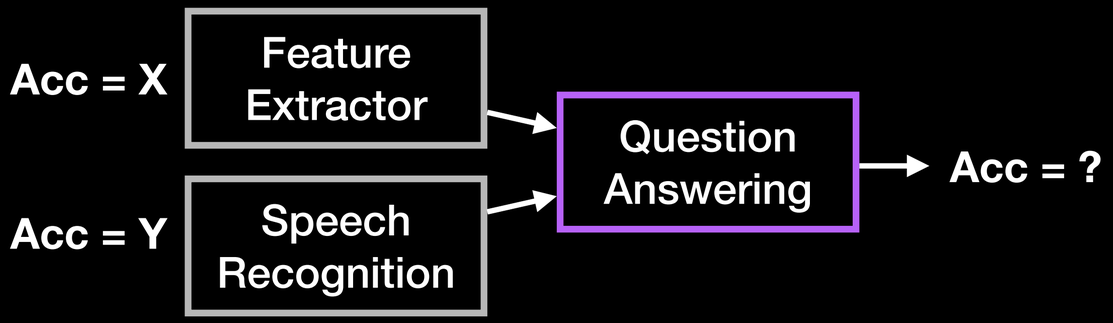


Offline Profiling: Impact of Model Selection?

1. Compute mean throughput for model on each infrastructure worker

2. Compute accuracy response for model based on direct upstream models

We assume output accuracy is monotonically increasing w.r.t. input accuracy



Query Optimizer: Objective Function

a: worker assignment
s: model selection

Compute Cost

Based on unit compute cost **C_C** for worker **w** to execute selected model **s(v)**

Communication Cost

Based on bandwidth cost between workers **C_B** and the consumed bandwidth **R** for output of one model as input to the other

$$\arg\min_{a,\,s} \sum_{v \in V} \sum_{w \in a(v)} C_c(s(v), w) + \sum_{(u,v) \in E} \sum_{(w_u, w_v) \in a(u) \times a(v)} C_B(w_u, w_v) R(u, v)$$

$$s.t.\ acc \geq A,\ t_{v_{out}}^{a(v_{out})} \geq T$$

Assumptions

“Communication among workers within a tier is free” → **C_B** set to 0

“Communication is one way from lower to upper tiers” → **C_B** set to ∞

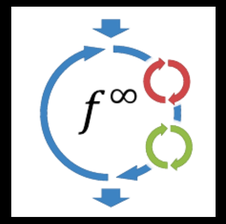
Query Optimizer: Approaches to MS and WA

Model Selection → **Worker Assignment**

- Go in reverse topological order
  - Final accuracy must satisfy the user's minimum constraint
  - Keep top selections in the beam based on simplified cost model (assume most powerful worker)

- Go in topological order
  - Reasonable approximation given realistic workflows (and one-way assumption)
- Greedy choice of lowest-cost workers (preferring lower tiers)
  - Keep top assignments in the beam based on accurate cost model (knowledge of worker)

Query Processor



Timely Dataflow provides a low-overhead dataflow abstraction (**Naiad**)

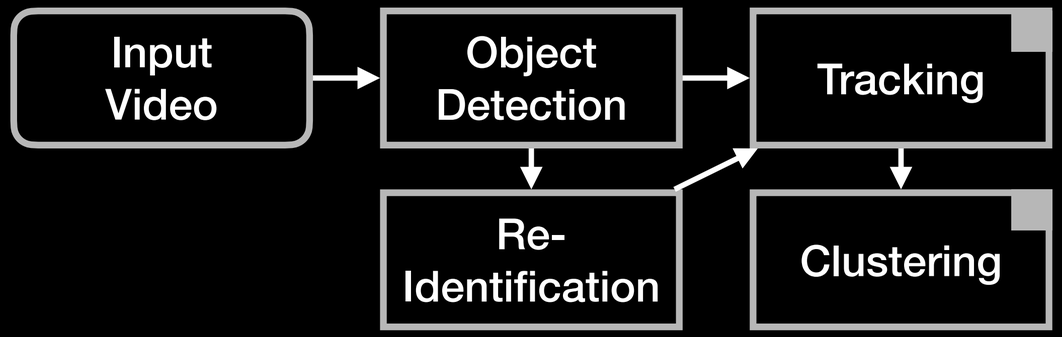
Our Extensions for JellyBean:

1. Support for operator-level parallelism instead of data-level parallelism
2. Support heterogeneous runtimes
3. Support relational operators (e.g., filter, join)

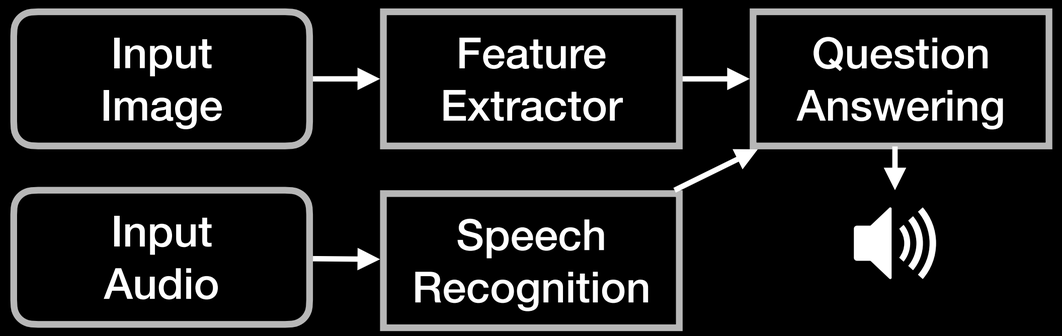
Evaluation: Inputs

Workflows

NVIDIA AI City Challenge (AICity)

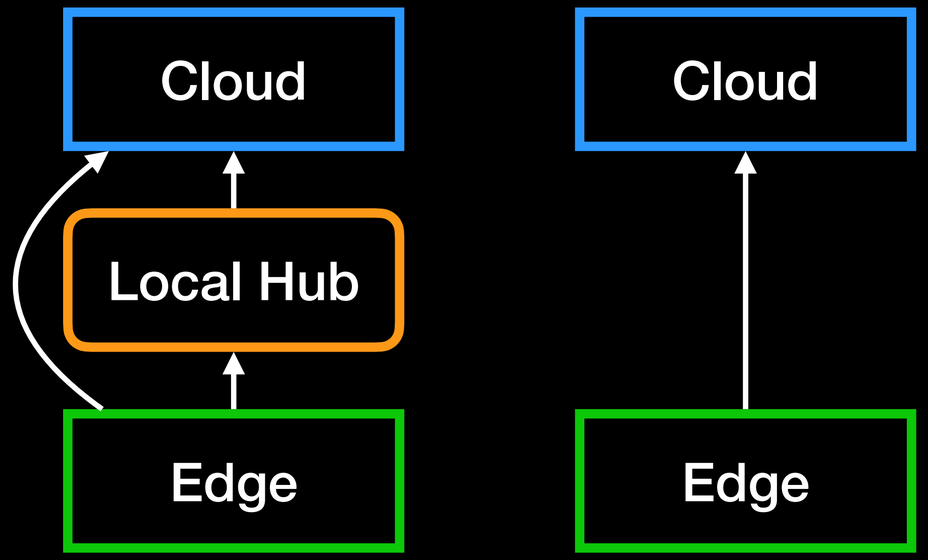


Visual Query Answering (VQA)



Infrastructures

5/9/15/30 workers (CPU & GPU)
divided into multiple tiers

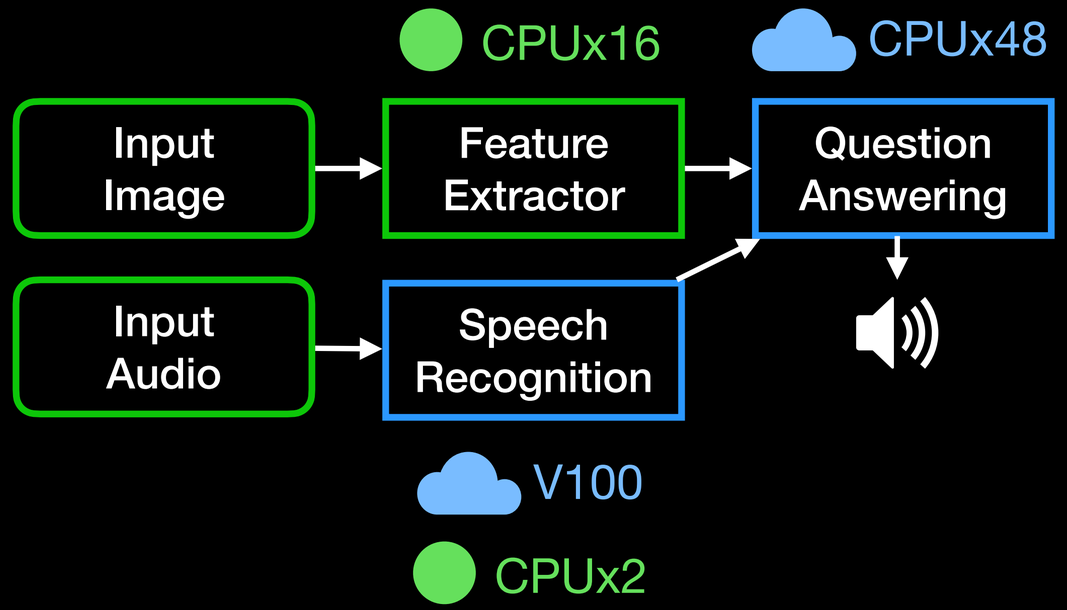


AI City **VQA**

Example: JellyBean Execution Plan for VQA

“Medium” Infrastructure:

- CPUx2
- CPUx4
- CPUx8
- CPUx8
- CPUx16
- CPUx48
- V100
- V100
- V100

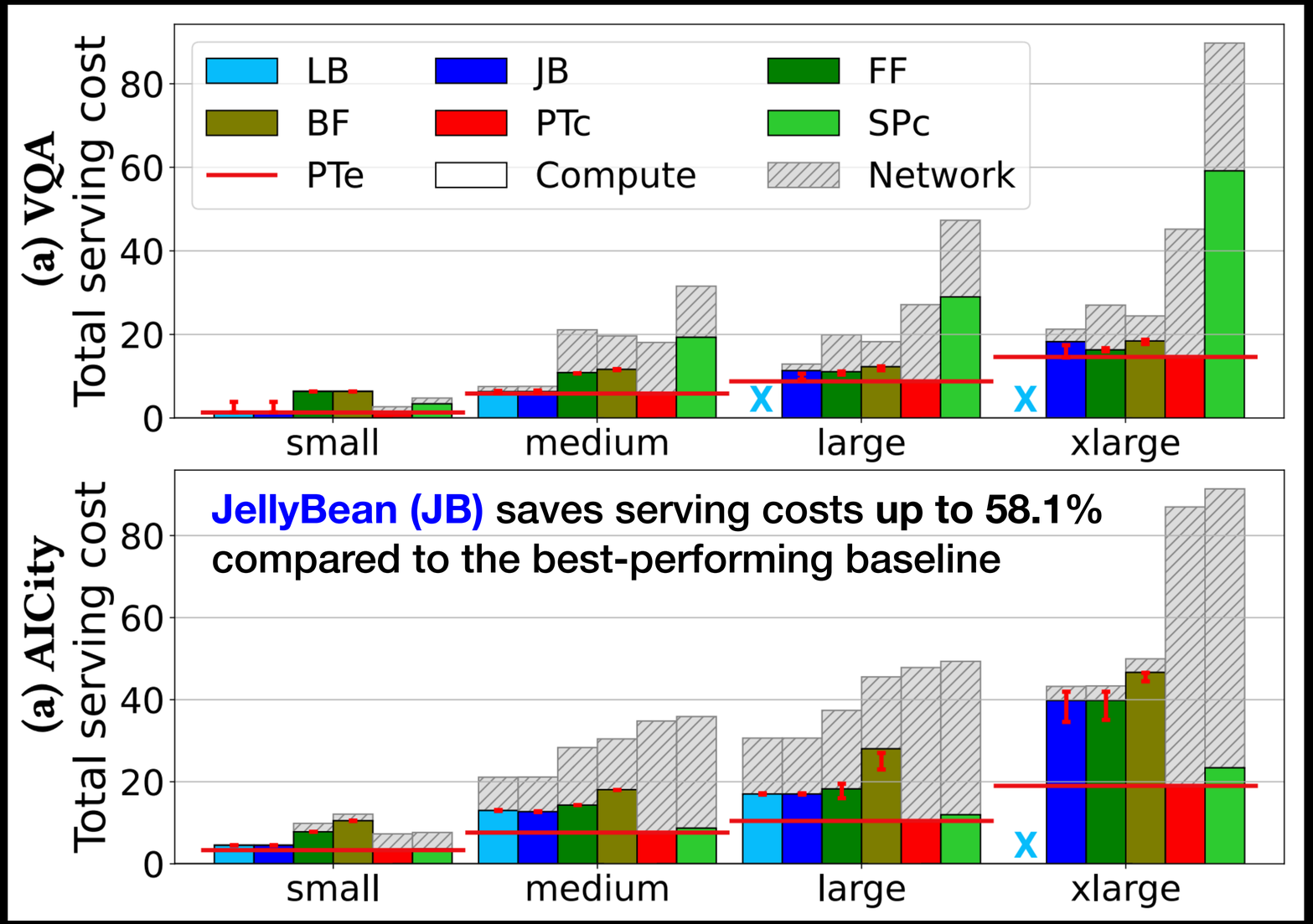


Constraints:

Throughput ≥ 20 rps

Accuracy ≥ 0.56

Evaluation: Total Serving Cost



Baselines:

Note: All use most accurate models

FF = First Fit
(Cheapest worker on any tier)

BF = Best Fit
(Cheapest worker on current tier)

PTc = PyTorch on Cloud
(One cloud GPU worker)

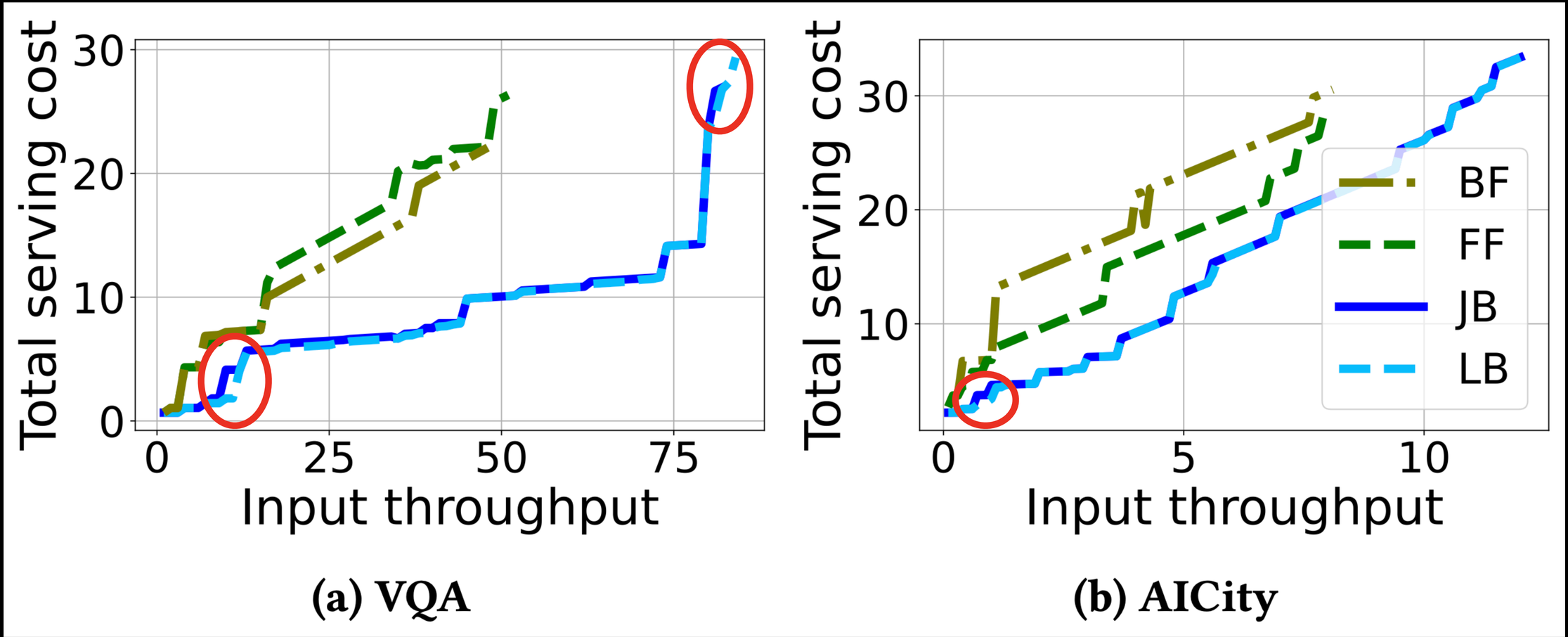
SPc = Spark with PyTorch
(All cloud GPU workers)

Ideal:

LB = Lower Bound
(Brute force over all plans)

PTe = PyTorch on Edge
(One edge GPU worker)

Evaluation: Serving Cost vs Throughput



JellyBean (JB) can keep up with increasing throughputs (and is near optimal)

Summary

JellyBean: Automatically optimizes deployments for ML workflow serving

- Jointly leverages worker assignment and model selection
- Optimized plans can significantly reduce the total serving cost

https://github.com/libertyeagle/JellyBean