Focus: Querying Large Video Datasets with Low Latency and Low Cost

Authors: Kevin Hsieh, Ganesh Ananthanarayanan, Peter Bodik, Shivaram Venkataraman, Paramvir Bahl, Matthai Philipose, Phillip B. Gibbons, Onur Mutlu

Presenters: Wei-Chung Liao, Jonah Rosenblum, Yiran Si

Motivation

- **Current systems for processing "after the fact" queries incur either high cost at ingest time or high latency at query time**
- **Processing fully at ingest time is expensive**
  - Typically only a small fraction of recorded video will be queried → Most of the works at ingest time will be wasted
  - Common methods for reducing costs (i.e. compression) also lower recall and precision
- **Processing fully at query time incurs high latency**

Goals

- Enable low-latency and low-cost querying over large historical video datasets
- Achieve low latency for queries while retaining high recall and precision from GT-CNN (ground-truth CNN)
- Enable trade-offs between cost at ingest time and latency at query time

Observations

- **Query: "find all frames in the video that contain objects of class X"**
- **Key characteristics of real-world videos towards supporting these queries**
  - Large portion of videos can be excluded
  - Only a limited set of object classes occur in each video
  - Objects of the same class have similar feature vector

Solutions

- Large portion of videos can be excluded
- Only a limited set of object classes occur in each video
- Objects of the same class have similar feature vector

Solutions

- Large portion of videos can be excluded
- Only a limited set of object classes occur in each video
- Objects of the same class have similar feature vector

Solutions

- **Large portion of videos can be excluded**
  - Use cheap ingest CNN to exclude them at ingest time (approximate indexing)
- Only a limited set of object classes occur in each video
- Objects of the same class have similar feature vector

Solutions

- Large portion of videos can be excluded
  - Use cheap ingest CNN to exclude them at ingest time (approximate indexing)
- Only a limited set of object classes occur in each video
- Objects of the same class have similar feature vector

Solutions

- Large portion of videos can be excluded
  - Use cheap ingest CNN to exclude them at ingest time (approximate indexing)
- **Only a limited set of object classes occur in each video**
  - Use fewer classes and thus can reduce the size of cheap CNN
- Objects of the same class have similar feature vector

Solutions

- Large portion of videos can be excluded
  - Use cheap ingest CNN to exclude them at ingest time (approximate indexing)
- Only a limited set of object classes occur in each video
  - Use fewer classes and thus can reduce the size of cheap CNN
- Objects of the same class have similar feature vector

Solutions

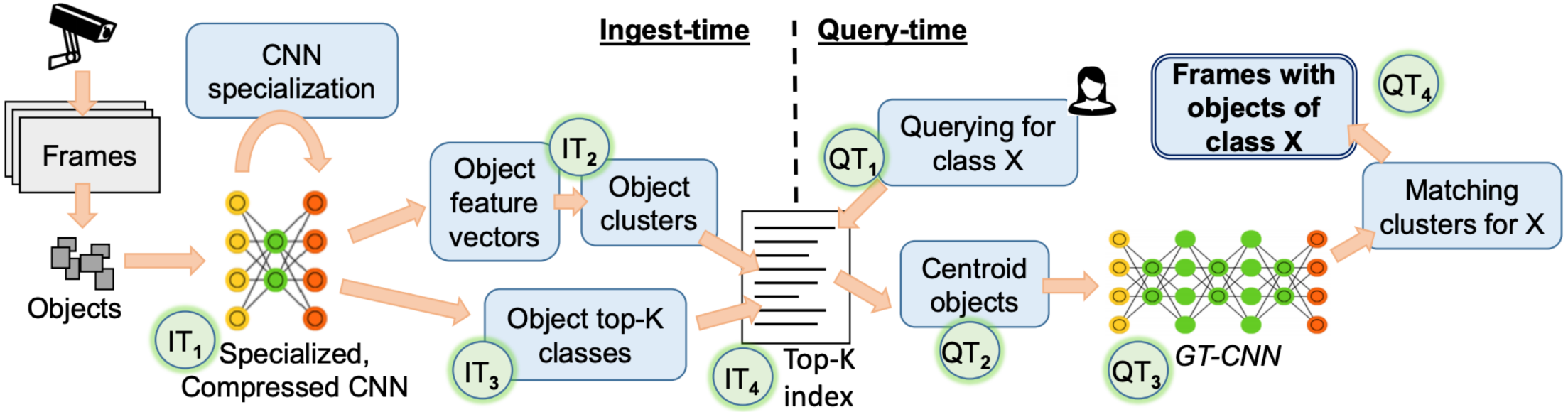
- Large portion of videos can be excluded
  - Use cheap ingest CNN to exclude them at ingest time (approximate indexing)
- Only a limited set of object classes occur in each video
  - Use fewer classes and thus can reduce the size of cheap CNN
- Objects of the same class have similar feature vector
  - Use clustering to avoid repeatedly processing on similar objects (redundancy elimination)

Solutions

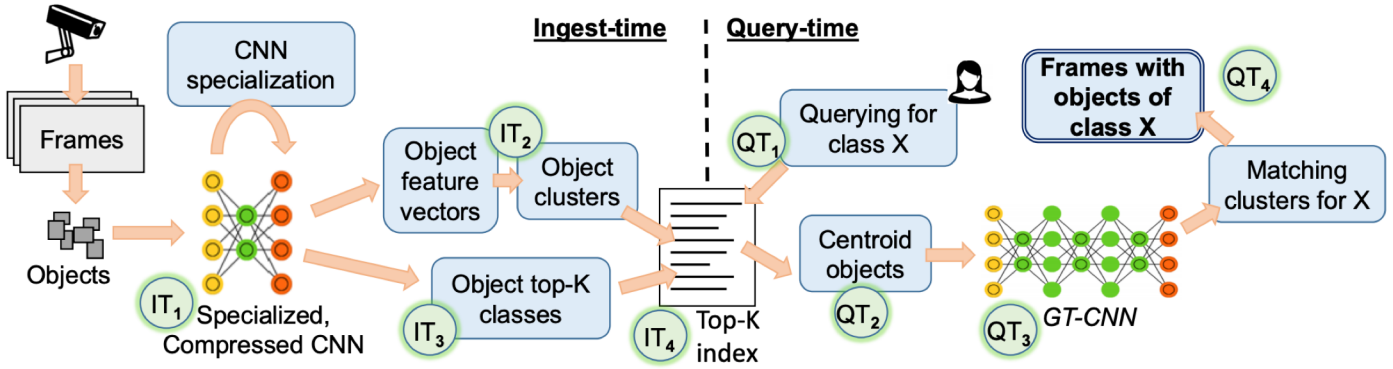
- Large portion of videos can be excluded
  - Use cheap ingest CNN to exclude them at ingest time (approximate indexing)
- Only a limited set of object classes occur in each video
  - Use fewer classes and thus can reduce the size of cheap CNN
- Objects of the same class have similar feature vector
  - Use clustering to avoid repeatedly processing on similar objects (redundancy elimination)

→ Do some preprocessing at ingest time to decrease the number of frames that need to be analyzed at query time

Overview of Focus

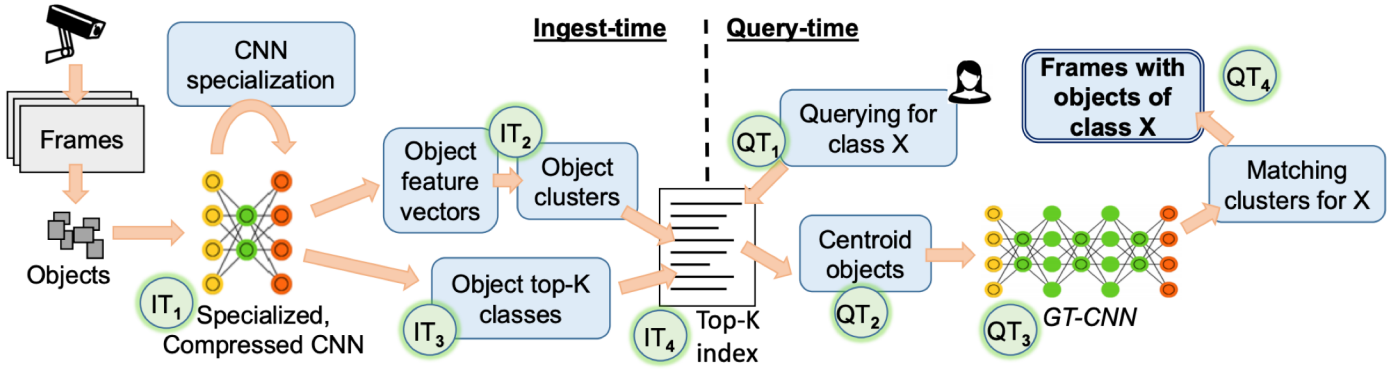


Overview of Focus



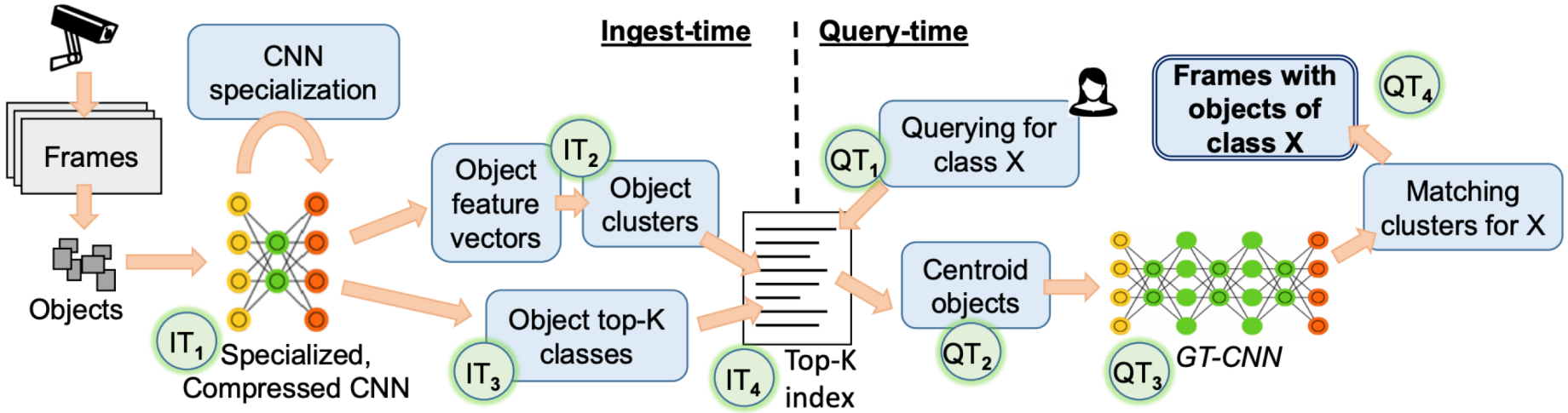
- At ingest time, do cheap indexing and object clustering.
  - Achieve high recall but low precision
- When queried, only analyze indexed cluster centroids, and use more sophisticated CNN (GT-CNN) to improve precision.
  - Achieve high recall AND high precision

Overview of Focus



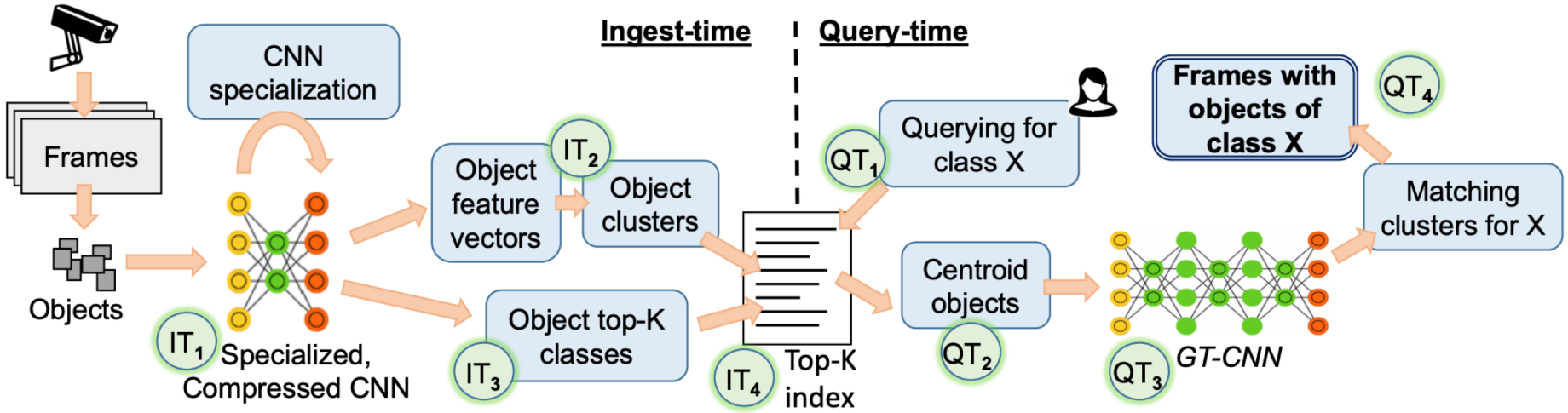
- At ingest time, do cheap indexing and object clustering
  - Achieve high recall but low precision
- When queried, only analyze indexed cluster centroids, and use more sophisticated CNN (GT-CNN) to improve precision
  - Achieve high recall AND high precision
- Ingest is cheap because it runs on cheap CNN instead of GT-CNN
- Query is fast because it only runs on selected frames (pre-index cluster centroids)
- More techniques and optimizations for both ingest and query later.

Pipeline



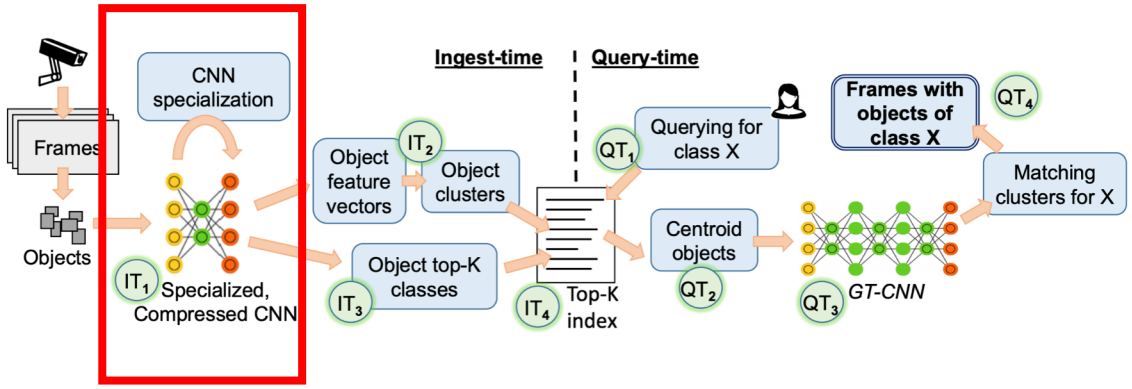
- Use highly compressed and specialized alternatives of the GT-CNN model (IT1)
- Cluster objects based on their feature vectors (IT2)
- Assign to each cluster the top K most likely classes (IT3)
- Create top-K index, which maps each class to the set of object clusters (IT4)

Pipeline



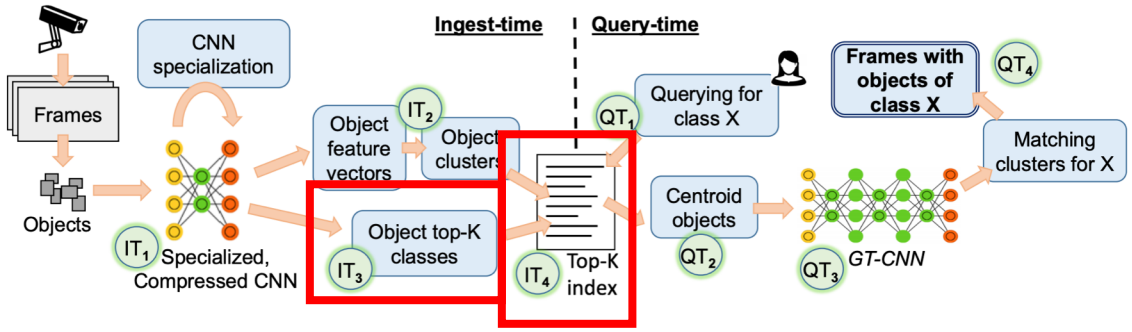
- When the user queries for a certain class X (QT1)
- Focus retrieves the matching clusters from the top-K index (QT2)
- Run the centroids of the clusters through GT-CNN (QT3)
- Return all frames from the clusters whose centroids were classified by GT-CNN as class X (QT4)

Cheap Ingest-time CNN



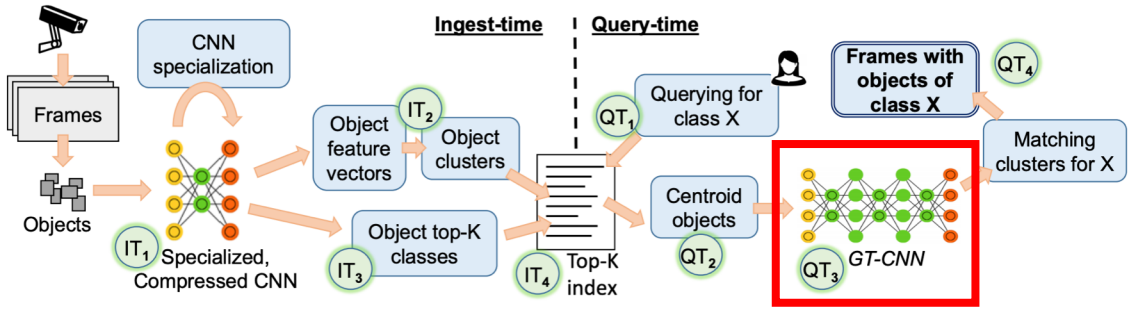
- **Ground-truth CNNs (GT-CNNs) are expensive.**
- **Top-K indexing and object elimination (clustering) do not need to be very accurate**
  - False positives are OK
  - Can use GT-CNN later to improve precision
  - Reminder: only a small fraction of recorded video will be queried.
  - So, we want the CNN to be cheap here
- **Use specialized, compressed CNN to save cost**

Top-K Index



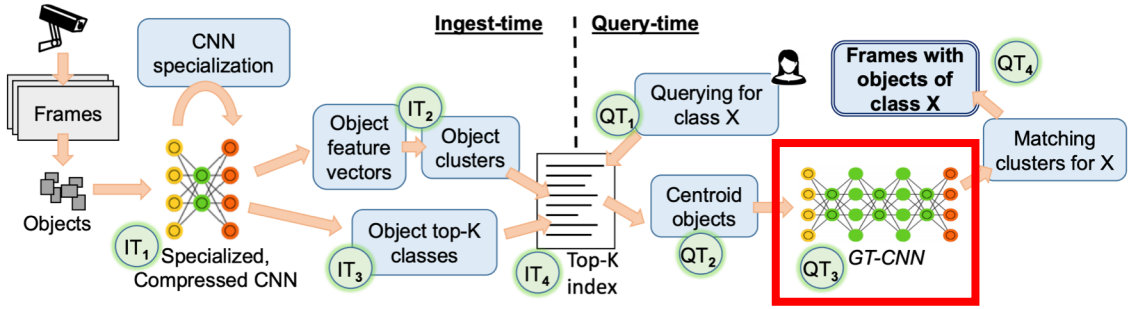
- **Index each frame with the top-K results**
- **Map each class to the set of possible object clusters**
- **Why K?**
  - Cheap CNN isn't very accurate
  - However, "right" answer is often in the top-K answers
  - Therefore, we use top-K to improve recall
- **K is small compared to the total number of classes**
  - Reach 99% recall when K = 4 for 80 classes (video stream from a static camera)
- **Higher K means higher recall but larger indexes and higher latency**

Achieve Precision



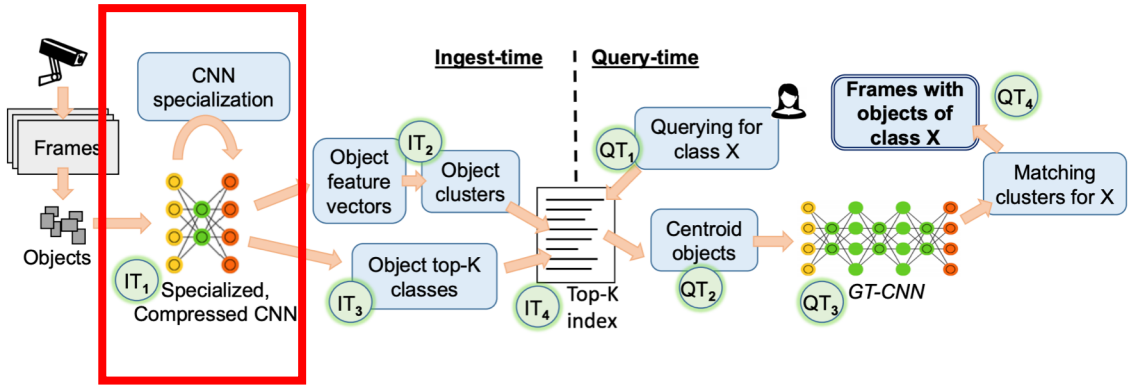
- Filtering for objects of the queried class X using top-K index has high recall but very low precision
- To improve precision, use expensive GT-CNN to determine the actual class of objects from the top-K index

Skip GT-CNN if High Confident



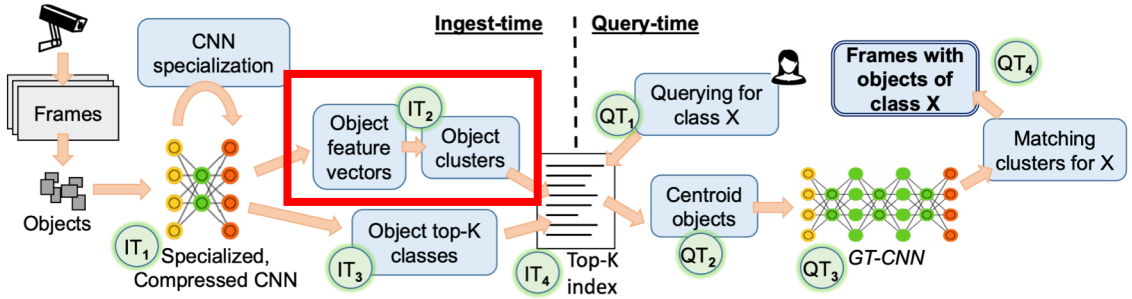
- GT-CNN is expensive. We want to skip GT-CNN if we are confident about the result.
- Record prediction confidence with the top-K results
- Skip GT-CNN evaluation for high-confidence indexes
- Apply a query-specialized binary classifier on the frames need to be checked before invoking GT-CNN

Ingest CNN Specialization



- **Only a limited set of object classes occur in each video**
  - We can use fewer classes for ingest CNN
- **Just a handful of classes often account for a dominant majority of the objects**
  - Combine the unfrequently occurring object classes into one class "OTHER"
  - Fewer classes for ingest CNN
  - If the queried class X is in OTHER, always run GT-CNN.

Cluster Similar Objects



- Avoid repeatedly processing on similar objects
- Assume that objects in the same cluster are in the same class
- At ingest time, only cluster centroids need to do top-K indexing
- At query time, only cluster centroids run through GT-CNN
  - Apply the classified result from the GT-CNN to all objects in the cluster

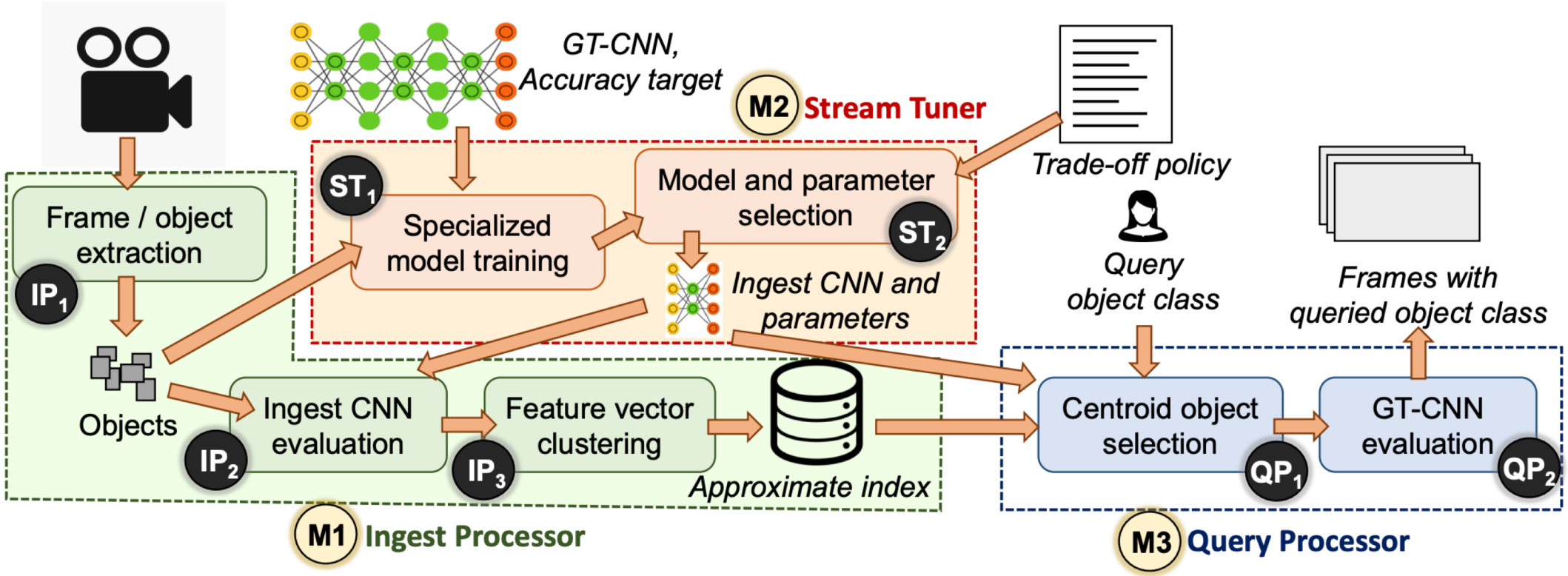
Ingest Cost and Query Latency Trade-off

- **Some parameters affect ingest cost**
  - K
  - The number of popular classes for specialized model (others are combined into OTHER)
  - Cheap CNN
  - Distance threshold for clustering algorithm
- **Some parameters affect query latency**
  - Confidence threshold to skip invoking GT-CNN
  - Distance threshold for clustering algorithm

Ingest Cost and Query Latency Trade-off

- We can pick different parameters to improve one of the two costs for a small worsening of the other cost
- Only select combinations that meet the precision and recall targets

Architecture



Evaluation

- **Baseline**
  - Ingest-heavy: use heavy GT-CNN for ingest
  - NoScope: a state-of-the-art video query system

Evaluation

- **Performance**
  - 48× cheaper on average than the Ingest-heavy in processing videos and 125× faster on average than NoScope in query latency simultaneously
  - All the while achieving at least 99% precision and recall
- **Trade-off**
  - If optimized for ingest cost, 65× cheaper on average than Ingest-heavy while reducing query latency 100× on average than NoScopre
  - If optimized for query latency, 202× faster queries on average than NoScope with 53× cheaper ingest on average than Ingest-heavy

Related Work: NoScope

- NoScope filters frames by using lightweight binary classifiers for the queried class before running heavy CNNs (at query time)
- Similarity: both try to avoid invoking heavy CNN
- Difference: Focus improves on this by further filtering frames at ingest time doing indexing and clustering

Questions?

Nexus: A GPU Cluster Engine for Accelerating DNN-Based Video Analysis

Authors: Haichen Shen, Lequn Chen, Yuchen Jin, Liangyu Zhao, Bingyu Kong, Matthai Philipose, Arvind Krishnamurthy, Ravi Sundaram

Presenters: Yiran Si, Jonah Rosenblum, Wei-Chung Liao

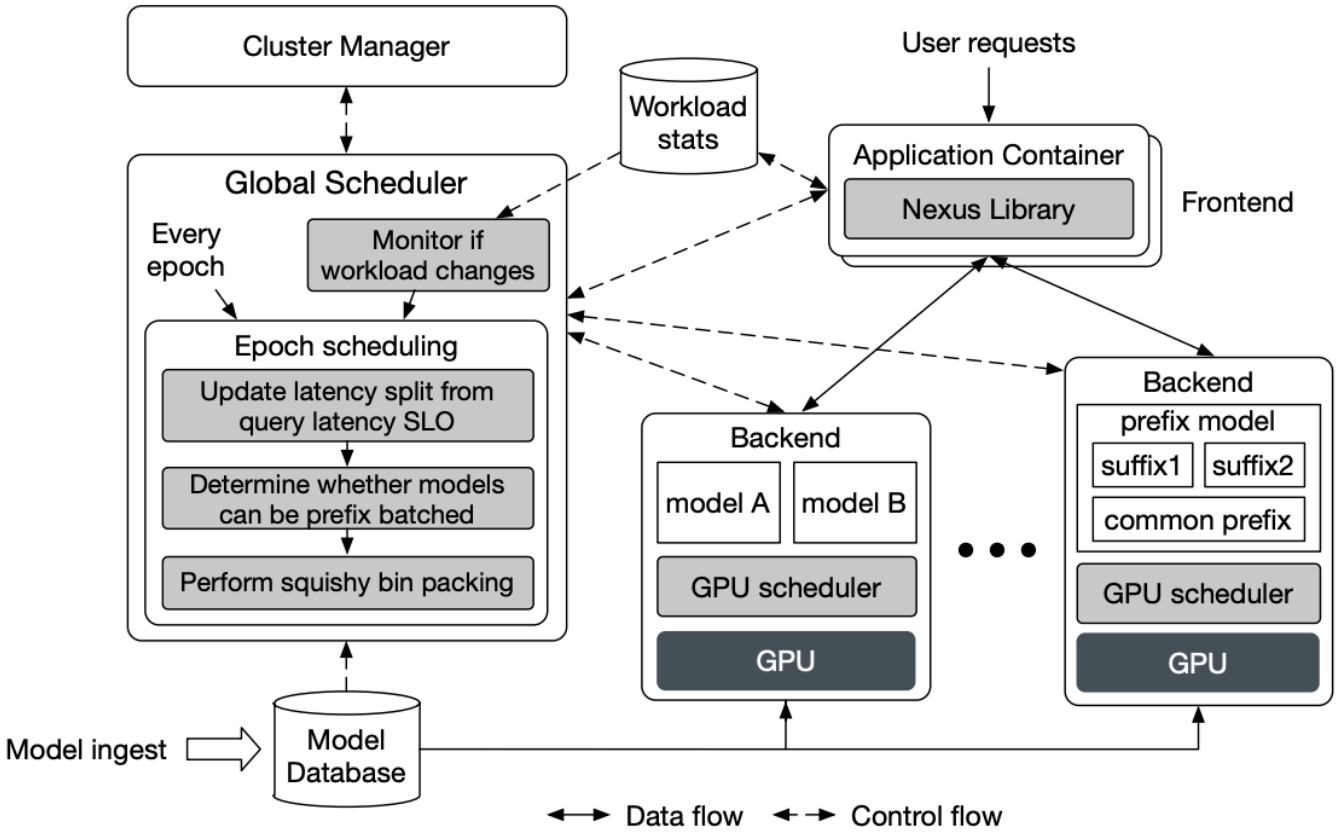
Motivation

1. Accelerators like GPUs and TPUs are expensive with huge capacities
2. Existing serving systems do not optimize resources for DNN applications
3. Fundamental challenge: distribute large workload onto a cluster of accelerators with high utilization and latency requirements

Goal

- High GPU utilization
- High throughput while meeting SLO requirements
- Flexibility (multi-application, auto-scale, more models)

Systems Overview



Frontend

Global Scheduler

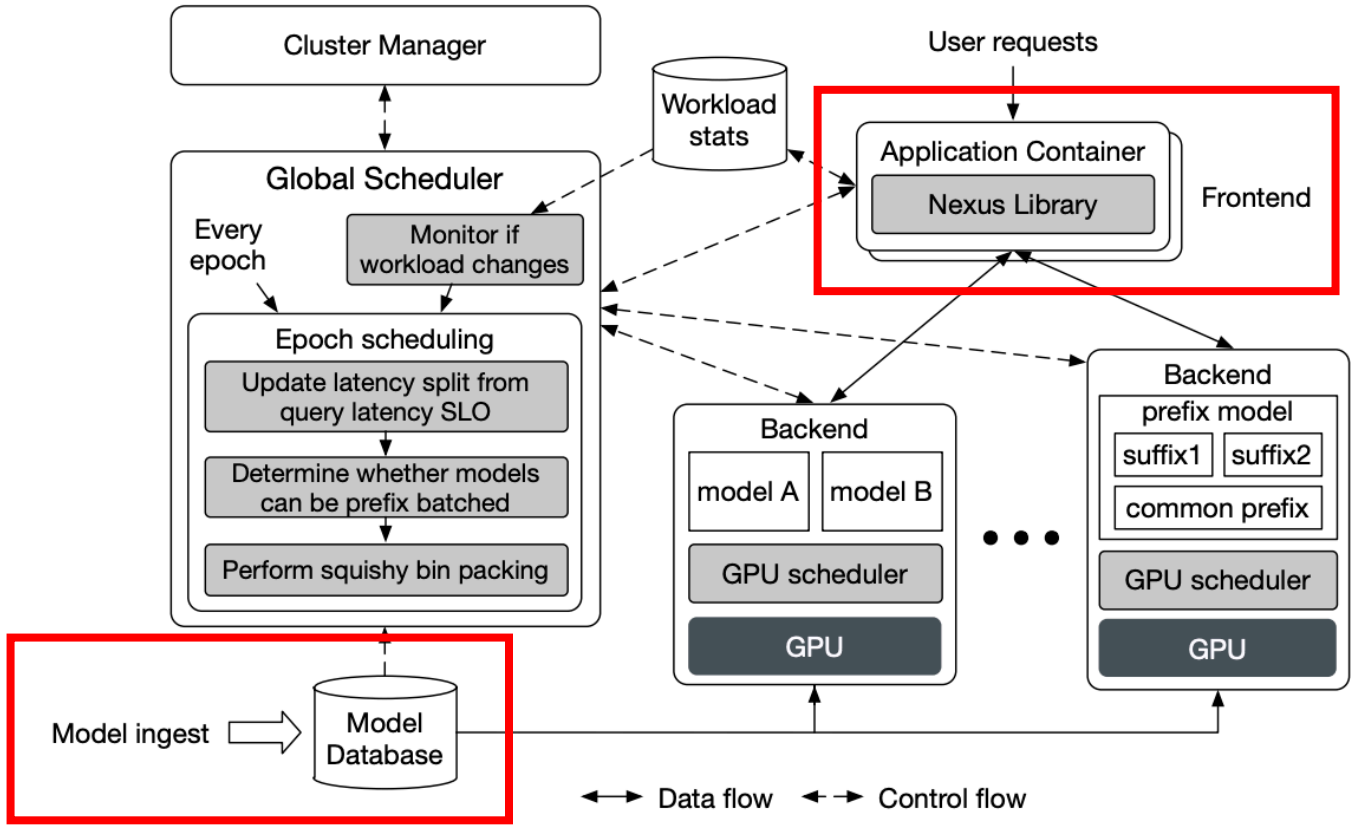
Backend

Model Database

Systems Overview

- Flexible management plane to upload models and deploy applications
- Fast data plane for forwarding messages from frontend to backend.
- Slow control plan for heavyweight scheduling tasks: resource allocation, packing and load-balancing.
- Provides query processing, task allocation, and subtask scheduling.

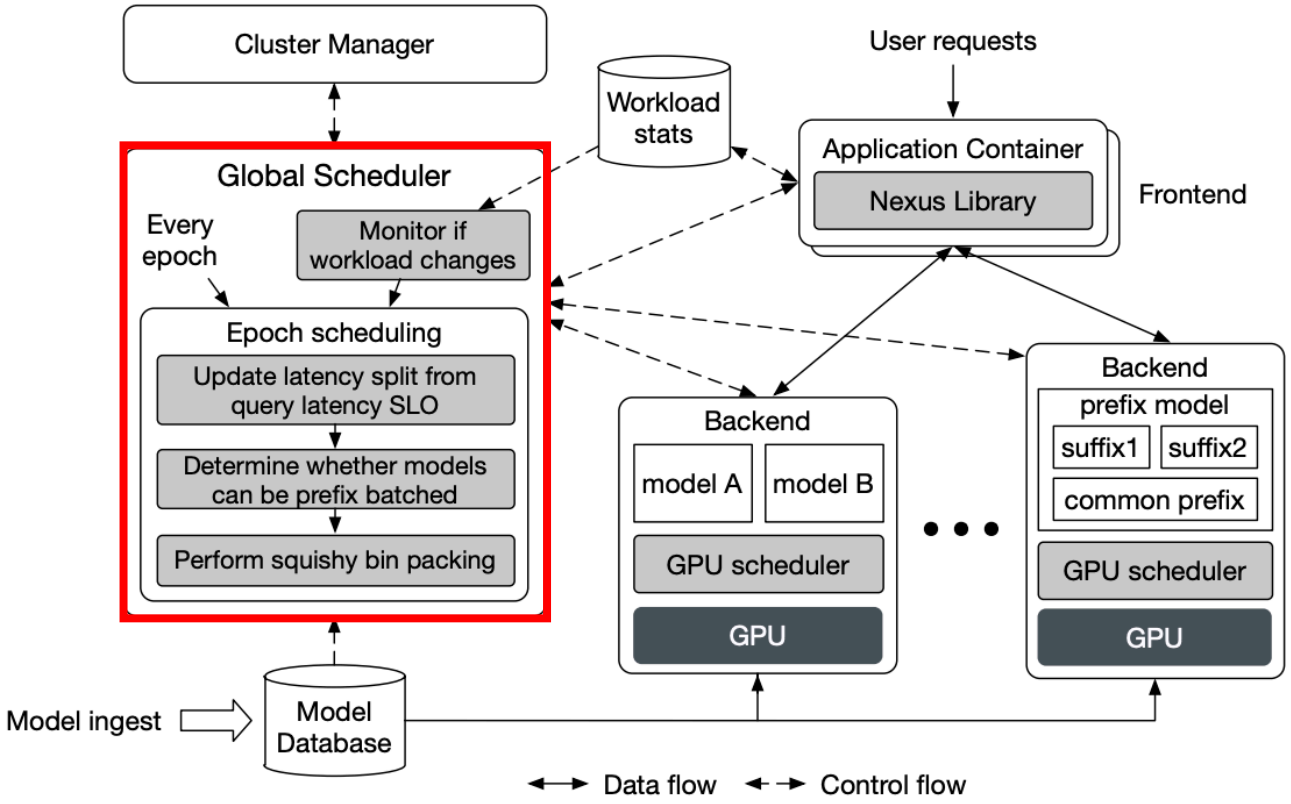
Management Plane



Management Plane

- Allows models to be ingested along with batch profile/sample data
- Users can upload applications and then instruct nexus when to deploy

Control Plane



Control Plane

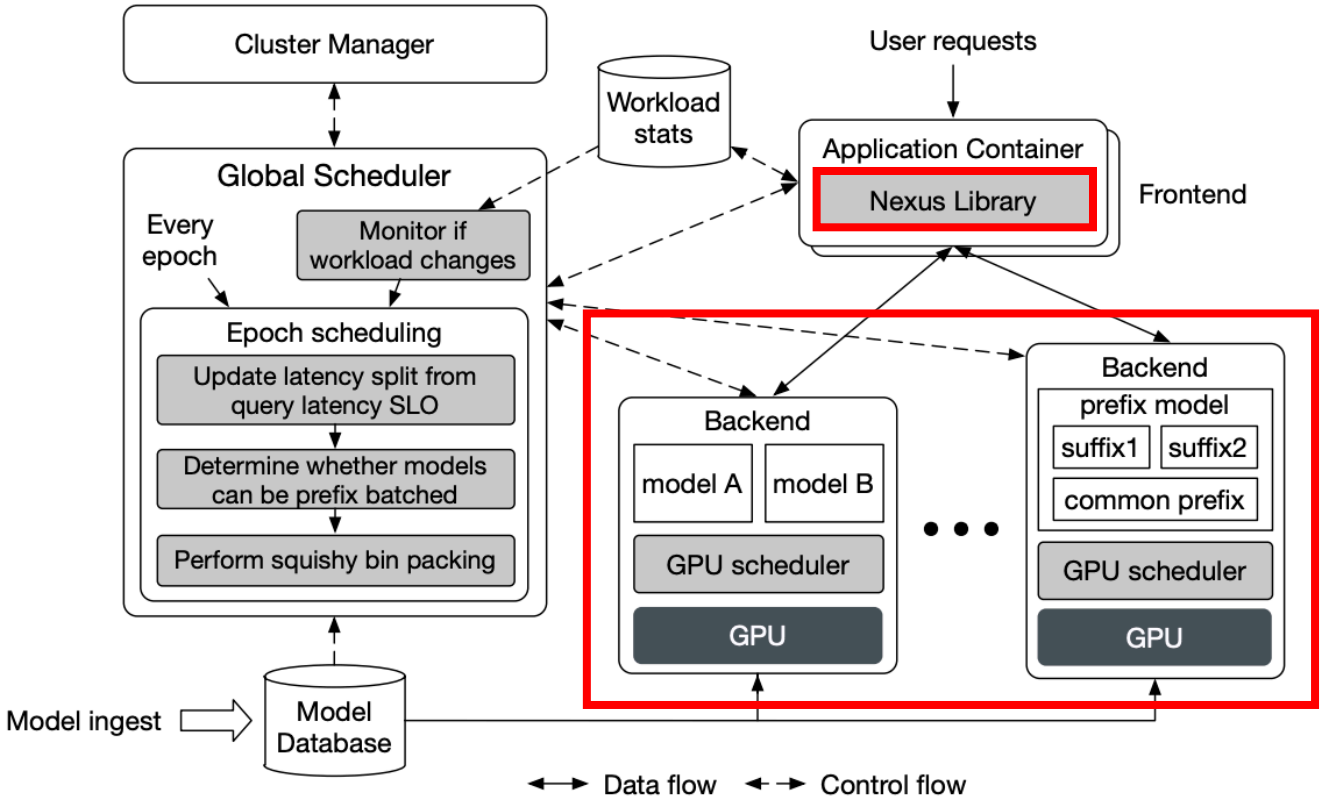
Global Scheduler:

- Cluster-wide resource manager
- Uses load statistics from runtime

Epoch Scheduling:

- Which model to execute
- Which backend to place the model

Data Plane


Cluster Manager
Workload stats
User requests
Application Container
Nexus Library
Frontend
Global Scheduler
Every epoch
Monitor if workload changes
Epoch scheduling
Update latency split from query latency SLO
Determine whether models can be prefix batched
Perform squishy bin packing
Backend
model A
model B
GPU scheduler
GPU
Backend
prefix model
suffix1
suffix2
common prefix
GPU scheduler
GPU
Model ingest
Model Database
Data flow
Control flow

Data Plane

- API calls link application to backend
- Dispatches requests quickly by consulting routing table
- Backend of dataplane runs multiple threads to handle incoming requests

Main Contributions of Nexus

1. Prefixed Batching
2. Query Latency Analysis
3. Batch-aware Resource Allocator

Goal

Achieve high throughput under latency SLO (Server Level Objectives)

Main Contributions of Nexus

1. Prefixed Batching
2. Query Latency Analysis
3. Batch-aware Resource Allocator

Goal

Achieve high throughput under latency SLO (Server Level Objectives)

Prefixed Batching

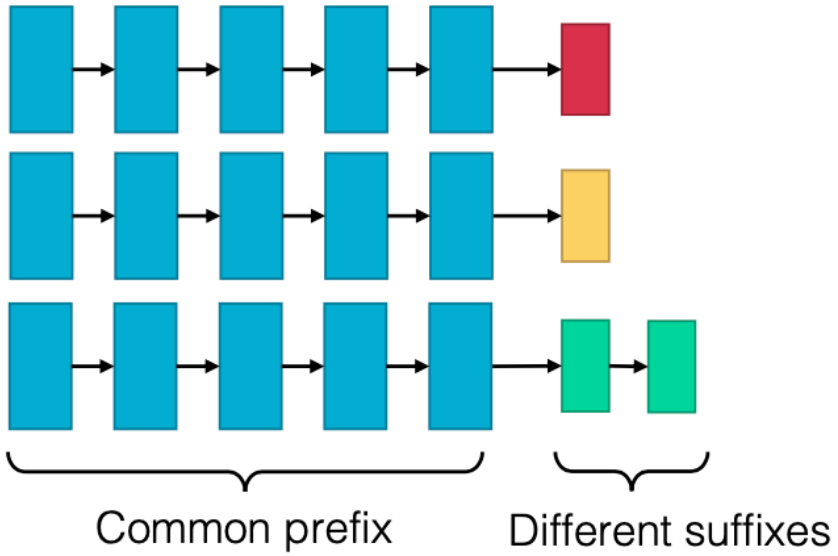
Observation:

in the common setting of model specialization, several models might differ only by their output layer.

→

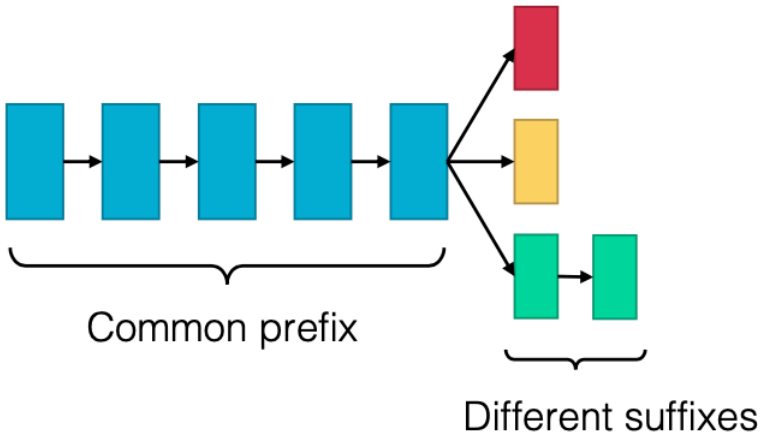
Optimization:

Batching the execution of all but the output layer

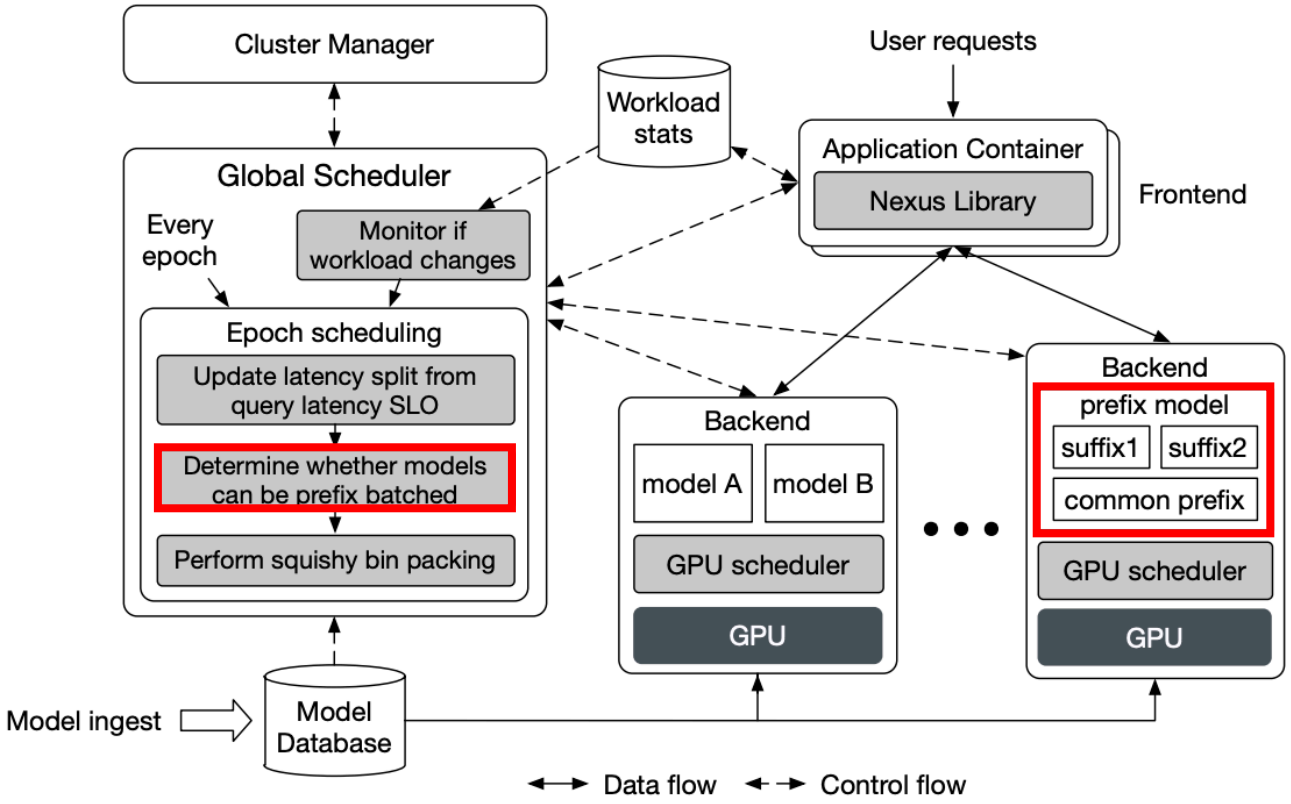


Prefixed Batching

1. Hash every subtree in the model to find common prefix
2. Load common prefix once + many different suffixes
3. Batch common prefix, execute suffixes sequentially



Prefixed Batching



Main Contributions of Nexus

1. Prefixed Batching
2. Query Latency Analysis
3. Batch-aware Resource Allocator

Goal

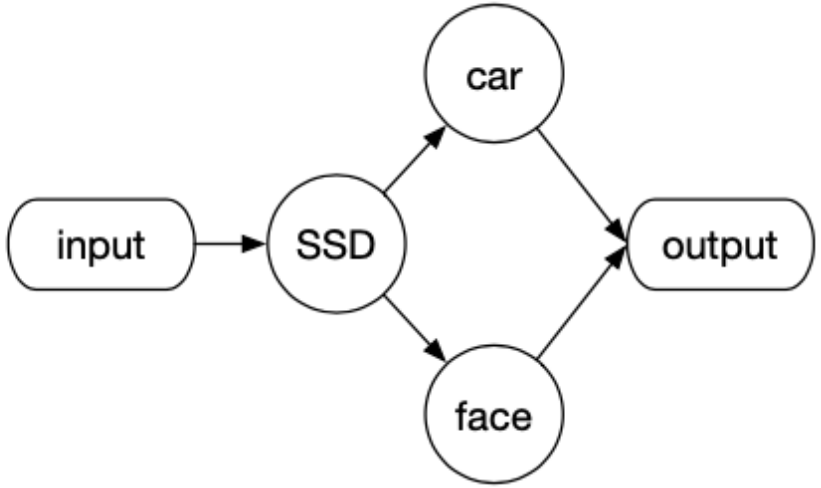
Achieve high throughput under latency SLO (Server Level Objectives)

Complex Queries Analysis

Objective:

find the best latency split for each model to minimize the # of GPUs required

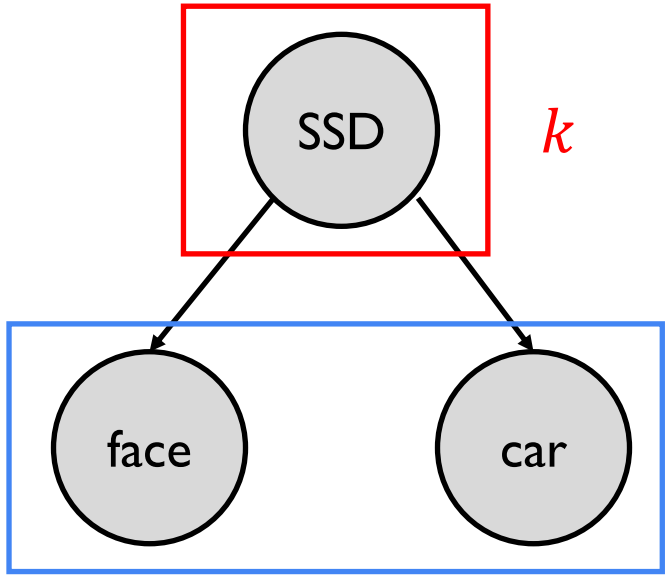
while meeting the latency requirements



Complex Queries Analysis

Solution: dynamic programming

minimize # of GPUs required along each path till the leaf nodes



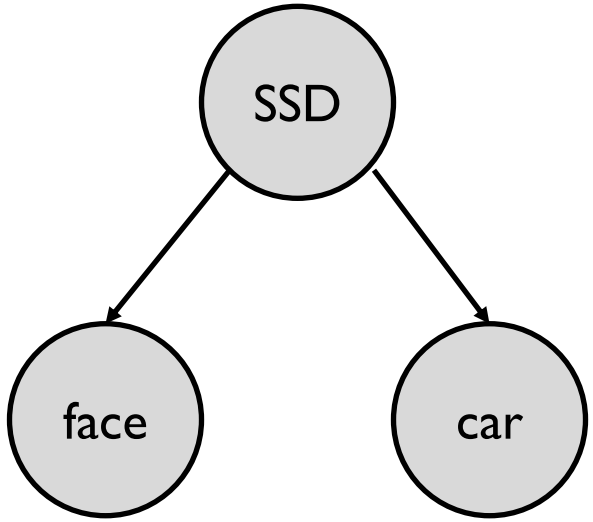
f(u,t) = #GPUs required to run model u and subtree of u given time t

$$f(u,t) = \min_{k:k\le t}\left\{\min_{b:l_u(b)\le k} R_u\frac{l_u(b)}{b} + \min_{t':t'\le t-k}\sum_{v:M_u\to M_v} f(v,t')\right\}$$

#GPUs for SSD

#GPUs for subtrees

Complex Queries Analysis

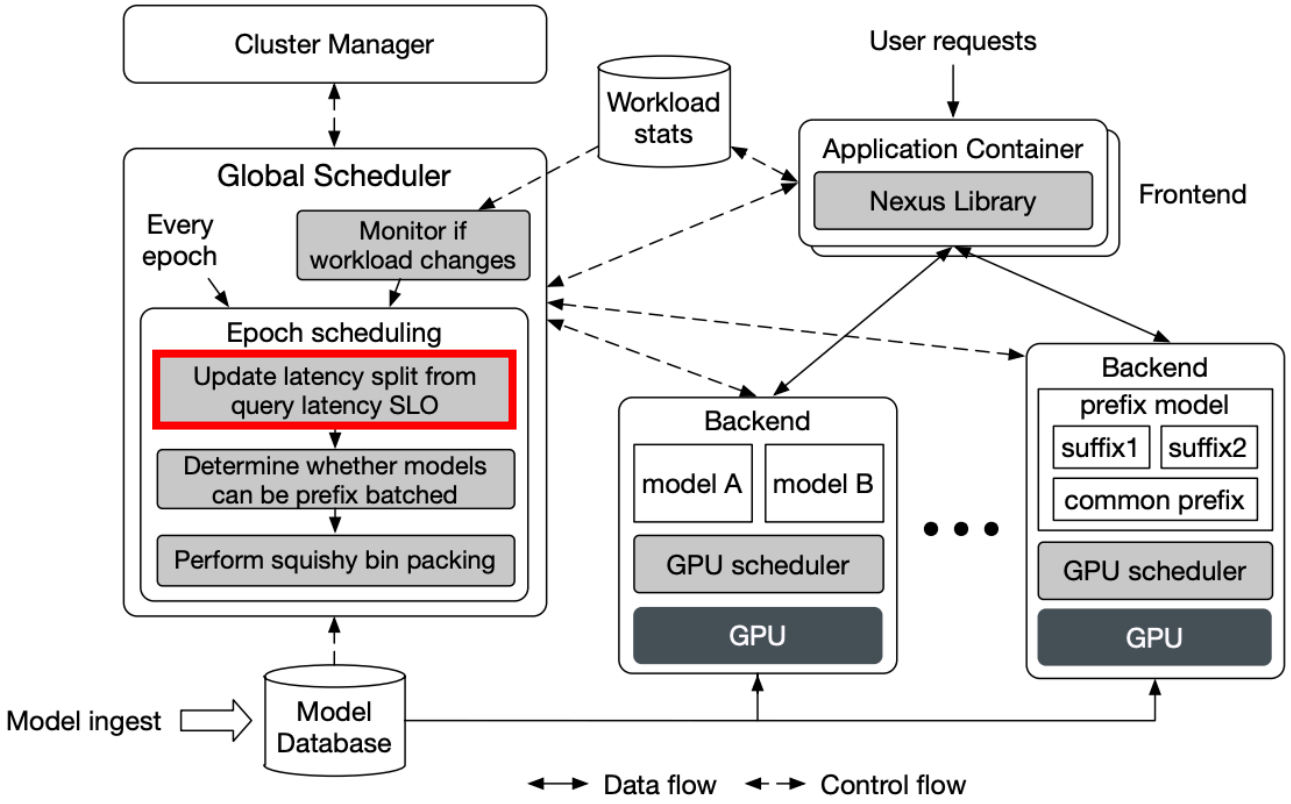


$$f(u,t) = \min_{k:k\le t}\left\{\min_{b:l_u(b)\le k} R_u\frac{l_u(b)}{b} + \min_{t':t'\le t-k}\sum_{v:M_u\to M_v} f(v,t')\right\}$$

#GPUs for SSD

#GPUs for subtrees

Complex Queries Analysis



Main Contributions of Nexus

1. Prefixed Batching
2. Query Latency Analysis
3. Batch-aware Resource Allocator

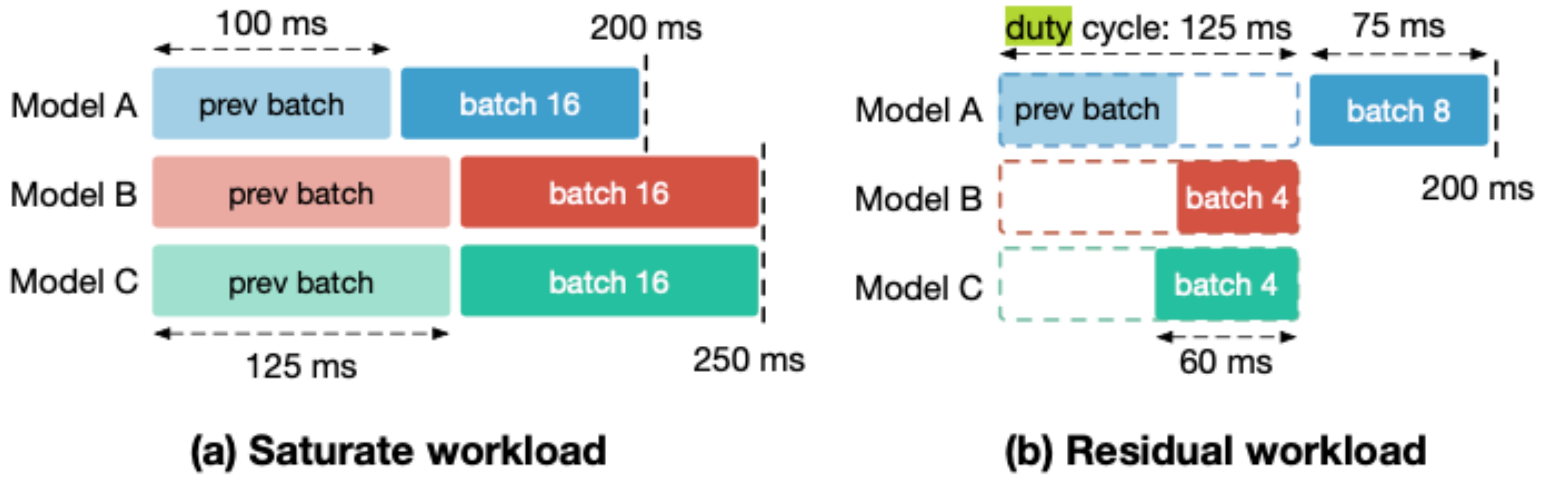
Goal

Achieve high throughput under latency SLO (Server Level Objectives)

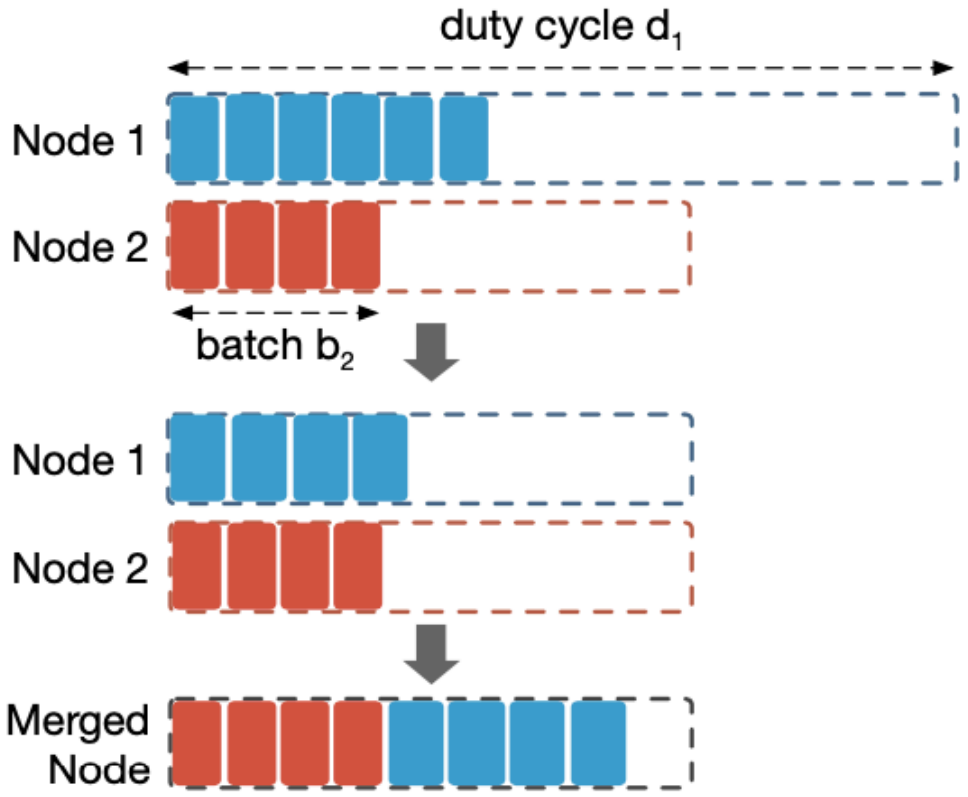
Batch-aware Resource Allocator and Scheduling

With multiple models are sent to one GPU, the actual latency largely depends on the duty cycle

*Duty cycle: a time period a GPU cycles through tasks(tasks are executed in a Round-Robin fashion)



Squishy Bin Packing



Merging 2 nodes:

1. take min duty cycle
2. adjust batch size

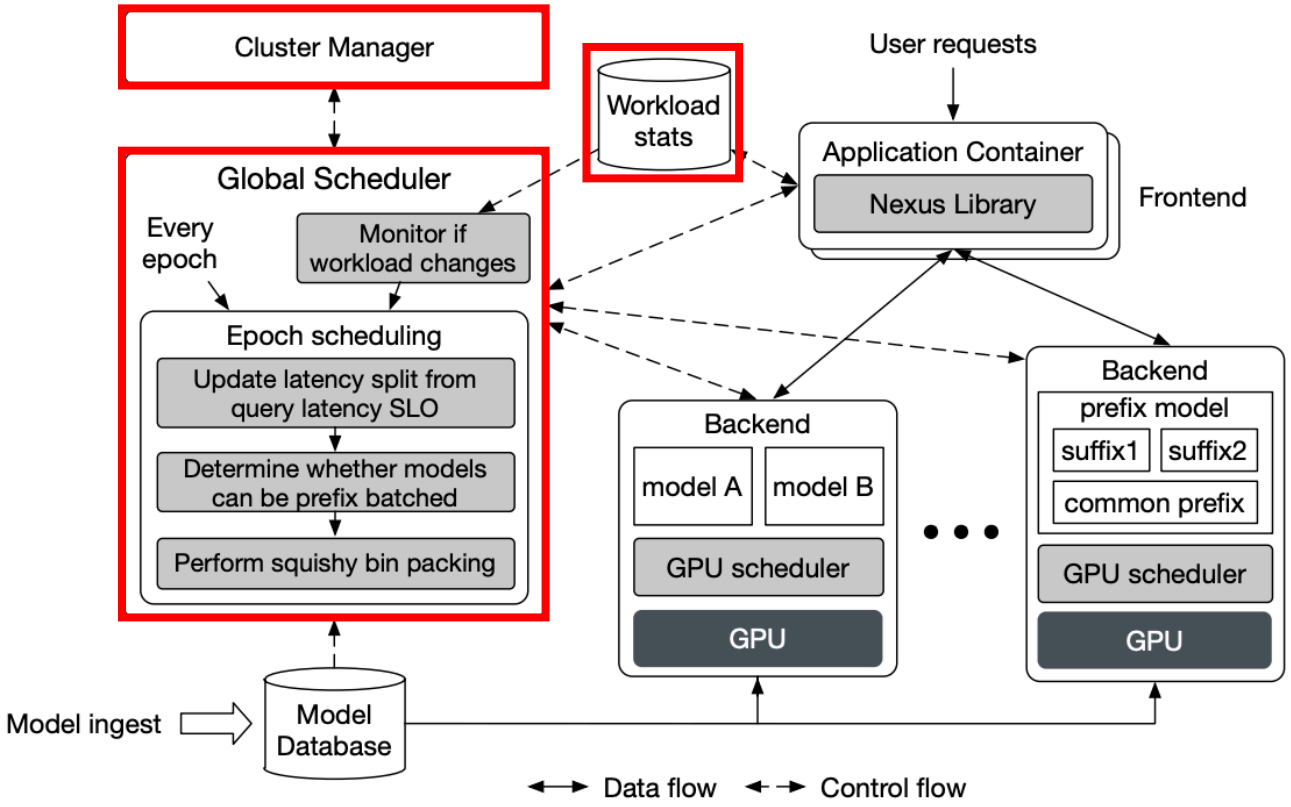
such that the occupancy of the merged node ≤ 1

Squishy Bin Packing

Extend the algorithm to be across epochs:

- Relocate sessions from infrequently used backends
- Release backends if no longer execute sessions
- Drop the cheapest sessions if overworked
  - Relocate dropped sessions

Batch-aware Resource Allocator and Scheduling



Performance

Key:

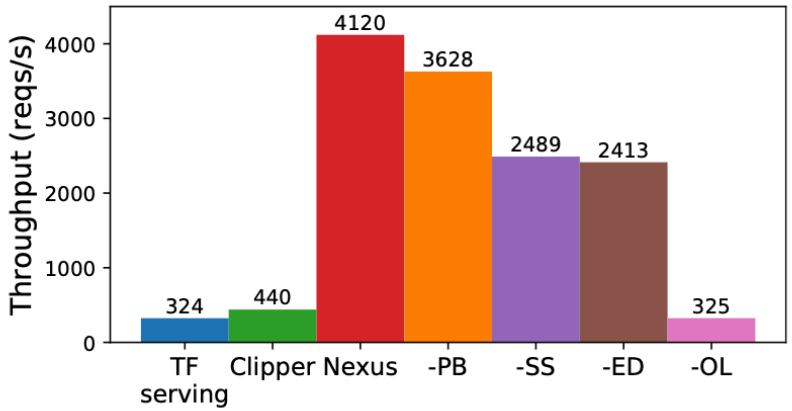
prefix batching = PB,

squishy scheduling = SS,

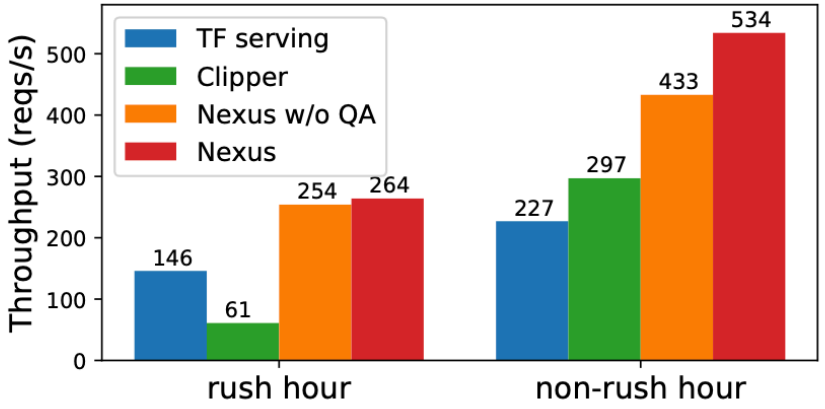
early drop = ED,

overlapped processing = OL,

query analysis = QA



Game analysis ablation study



Diurnal throughput variation for traffic analysis

Performance

Long running multiple application deployment

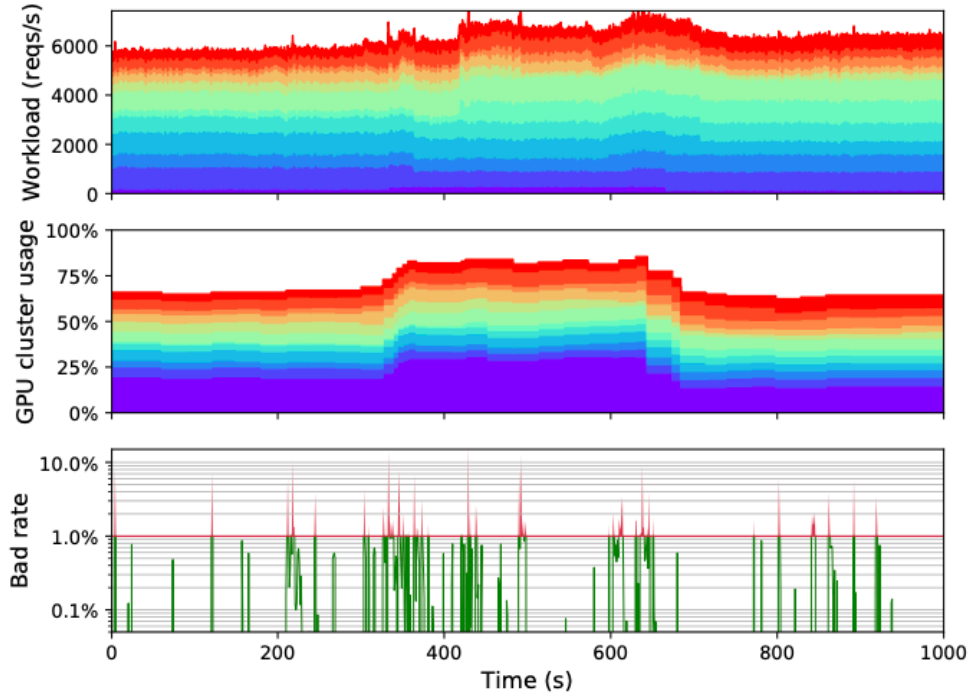


Figure 13: A 1000 sec window from our large-scale deployment.

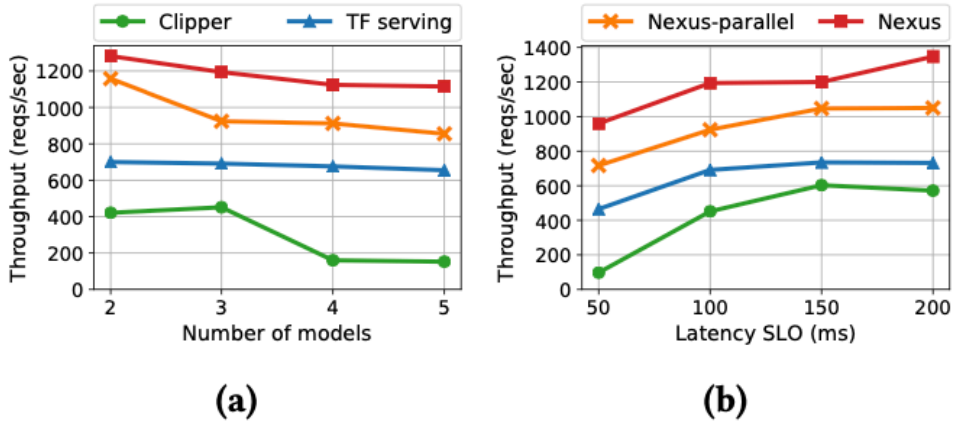


Figure 14: Impact on throughput of varying numbers of models served (a) and latency SLOs (b) under GPU multiplexing.

Summary

Optimizations Techniques:

- Prefixed Batching
- Query Latency Analysis
- Batch-aware Resource Allocator and Scheduling

Framework Improvement:

- Scale
- Granularity
- Expressivity

Questions?