MACHINE LEARNING FOR SYSTEM

Stage: Query Execution Time Prediction in Amazon Redshift

Best Practice, Lessons Learned and Open Questions from ML for Redshift

Zhengchun Liu

Senior Applied Scientist, AWS

Ziniu Wu[1], Ryan Marcus[2], Zhengchun Liu[3], Parimarjan Negi[1], Vikram Nathan[3], Pascal Pfeil[3], Gaurav Saxena[3], Mohammad Rahman[3], Balakrishnan Narayanaswamy[3], Tim Kraska[1,3]

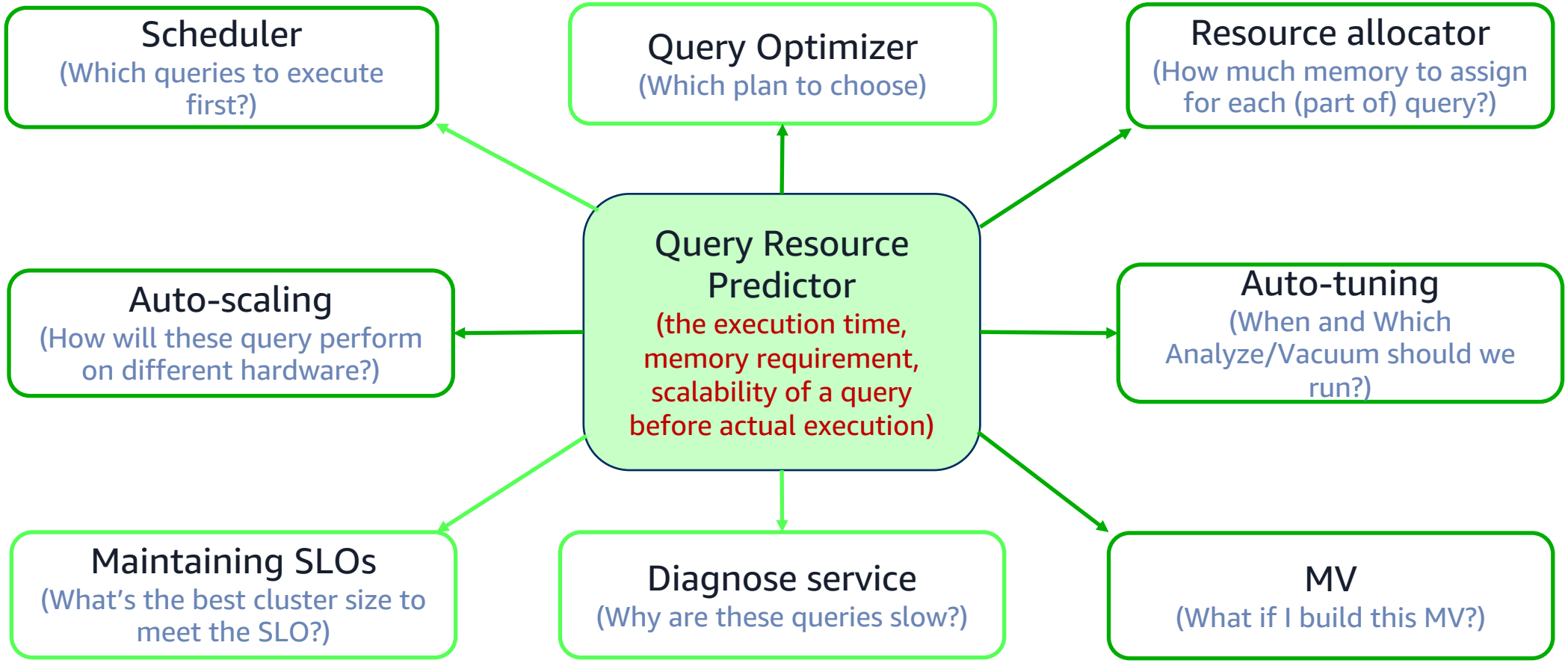
1 MIT, 2 Upenn, 3 AWS

Agenda

1. Motivation
2. Challenge
3. Our Solution
4. Impact in production environment
5. Lessons Learnt

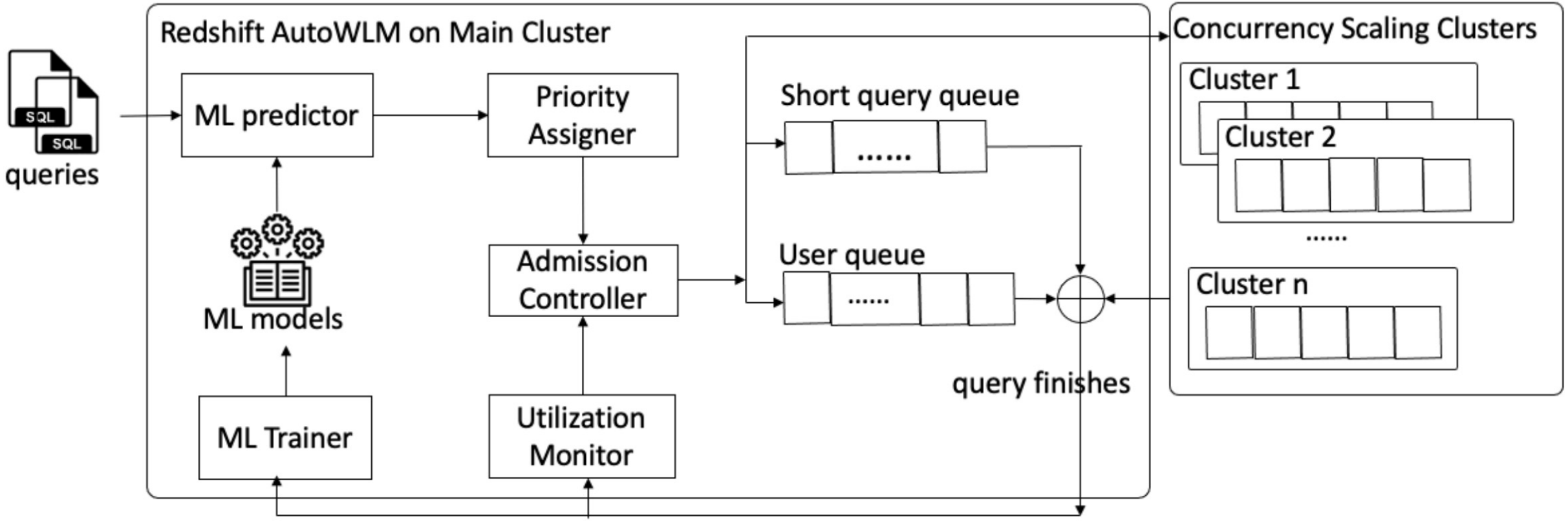
Motivation

ML is Widely Used in Redshift to Optimize Redshift



Application 1: Automatic Workload Management

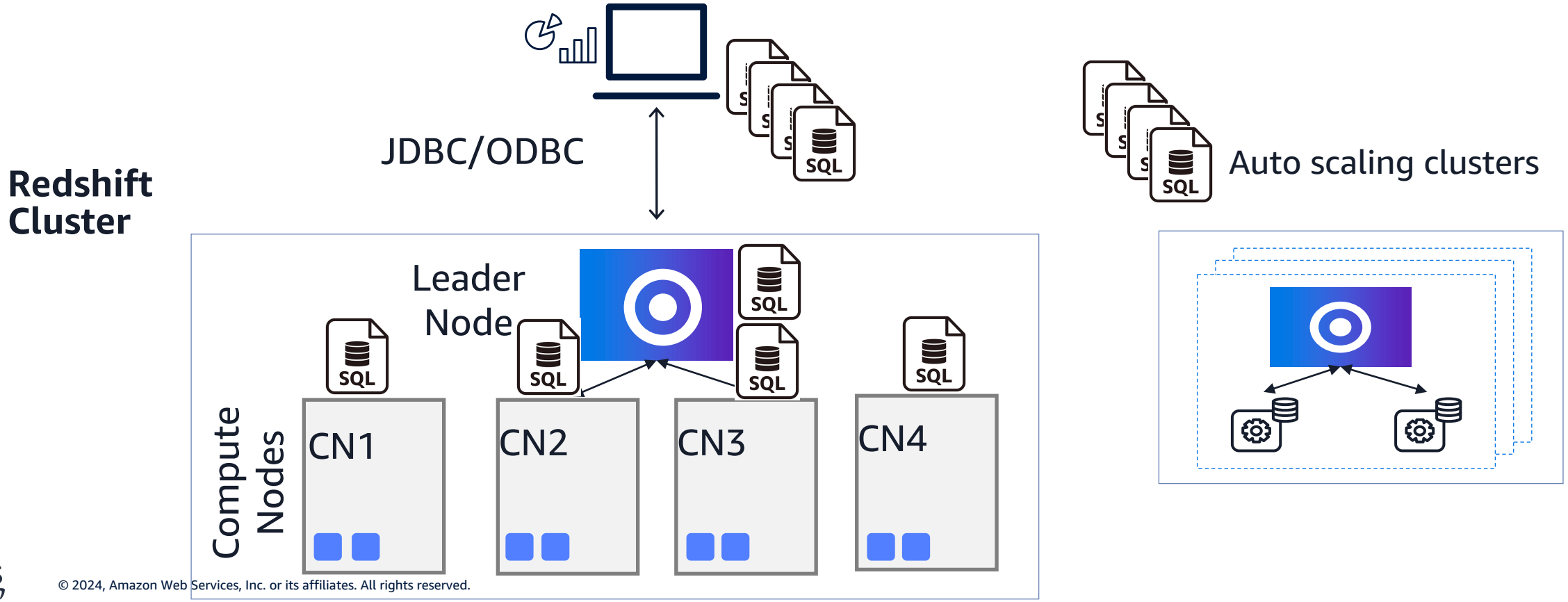
Query Admission via Predicted Query Memory Requirement



Application 2: Query Scalability Prediction

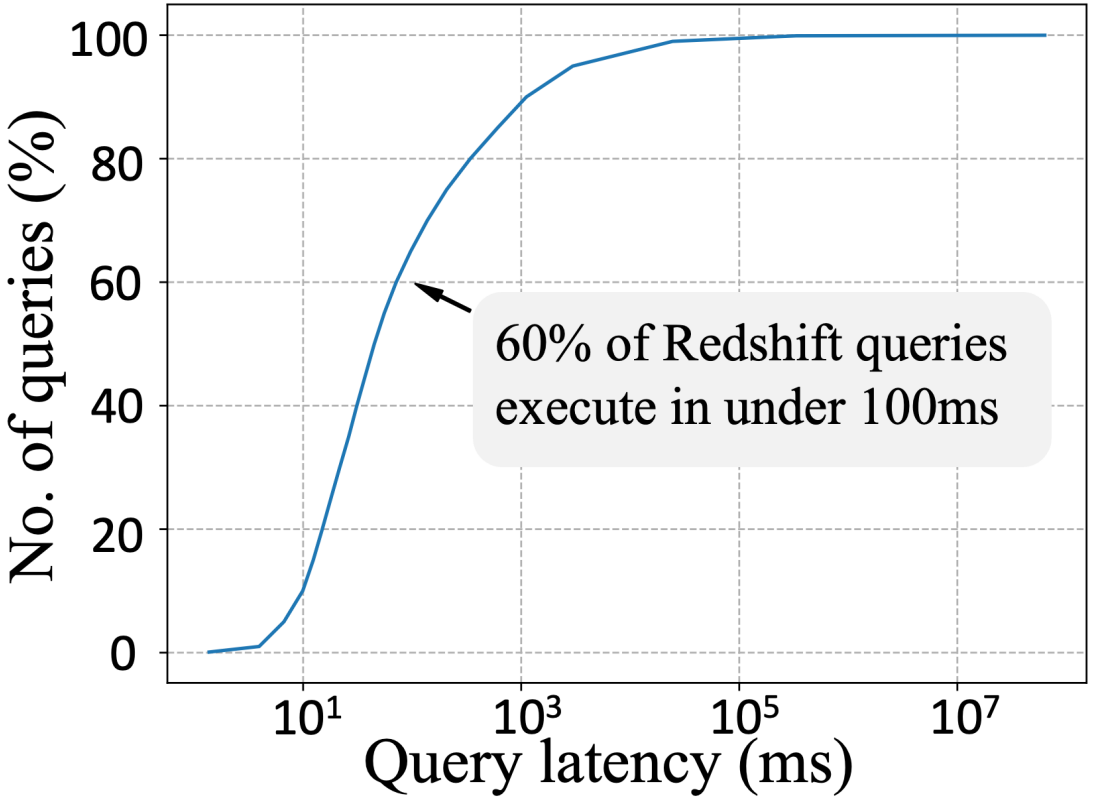
AI-powered Scaling and Optimization for Amazon Redshift Serverless
Instead of deciding whether route query to burst cluster based on length of queue, we burst based on complexity and scalability of query for better price-performance.

So, upon query *q* arrival, we need answer of
"**How long will it take *if* we run *q* on a *n*-node burst cluster?"** So that we know the best price-performance to run *q*.



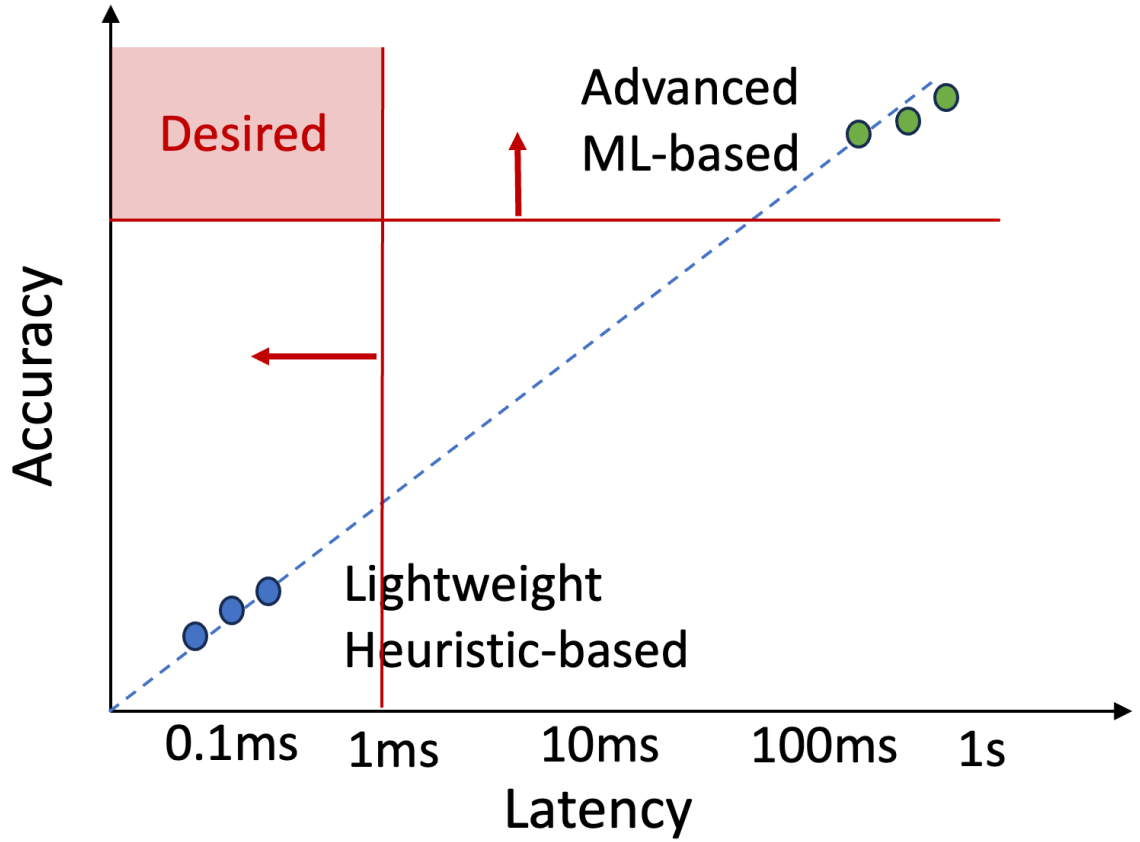
Challenge: Prediction for Decision *in Critical Path*

Training:
- Limited data available on the cluster, i.e., cold-start.
- Limited computing resource available for ML
- Limited memory/storage to save data collected from user queries.



Inference Latency:
- it is on the critical path of query execution
- 60% queries are under 100ms
- General goal is under 1ms per query

There must be a balance between High prediction accuracy and Low inference latency.



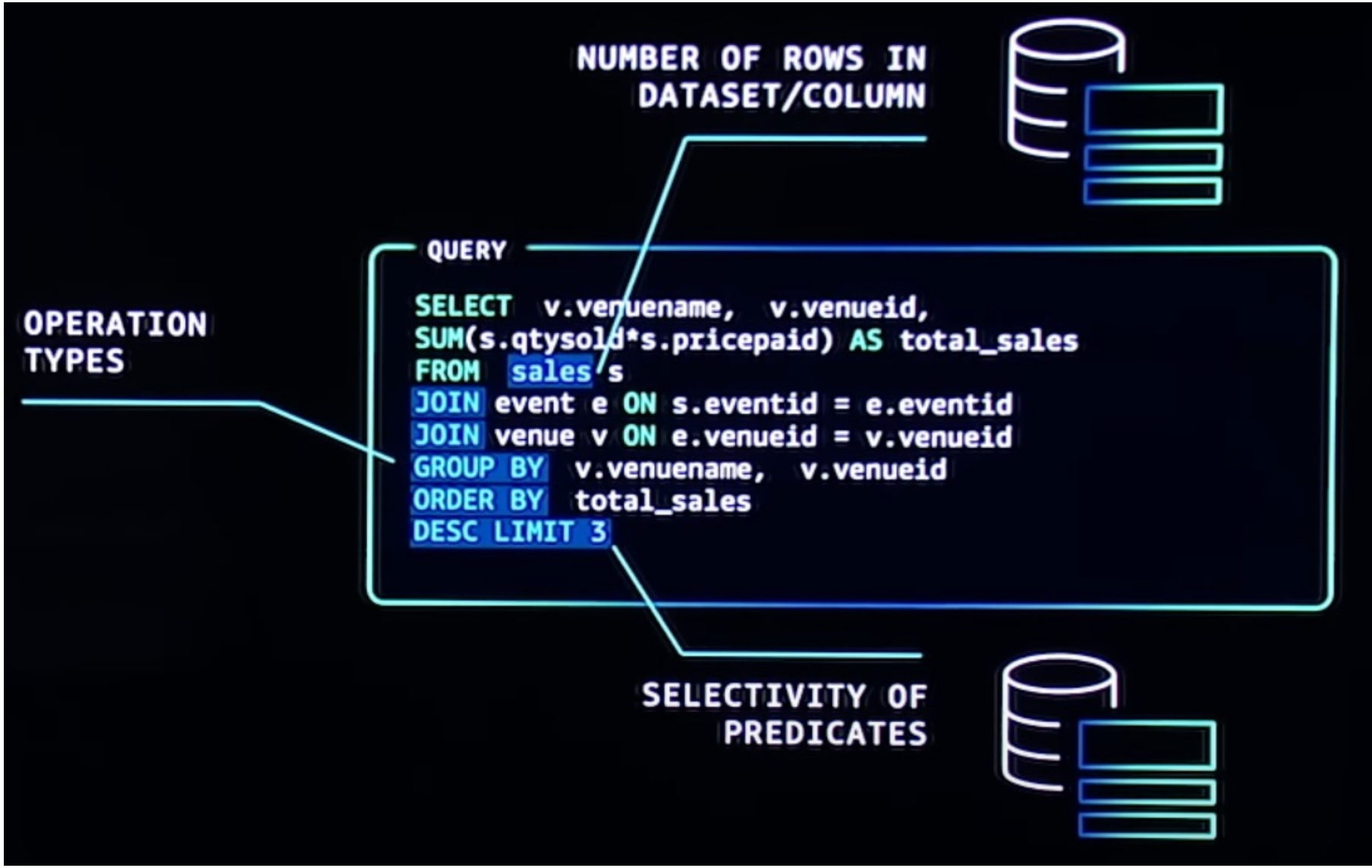
Solution: *Staged* for Query Resource Prediction

- Uncertainty and Goes Up with Stage
- Inference Latency Increase with Stage
- Accuracy Goes up with Stage

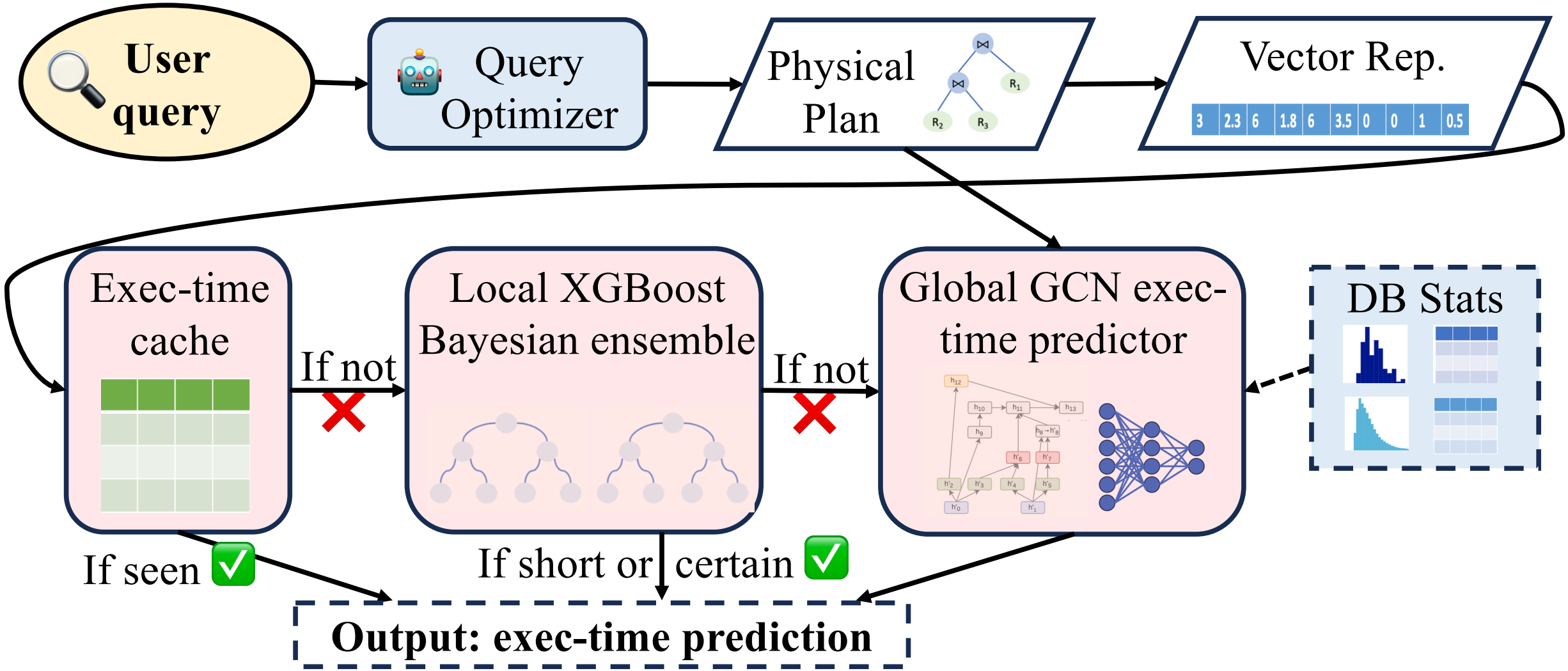
Solution: Creating query feature embeddings

Creating query feature **embeddings**

A representation of the SQL itself *and* table stats.



Stage: Tech Overview



Stage 1: Exactly the same query on the same tables

A simple but effective method leveraging the fact of repeated queries in Redshift.

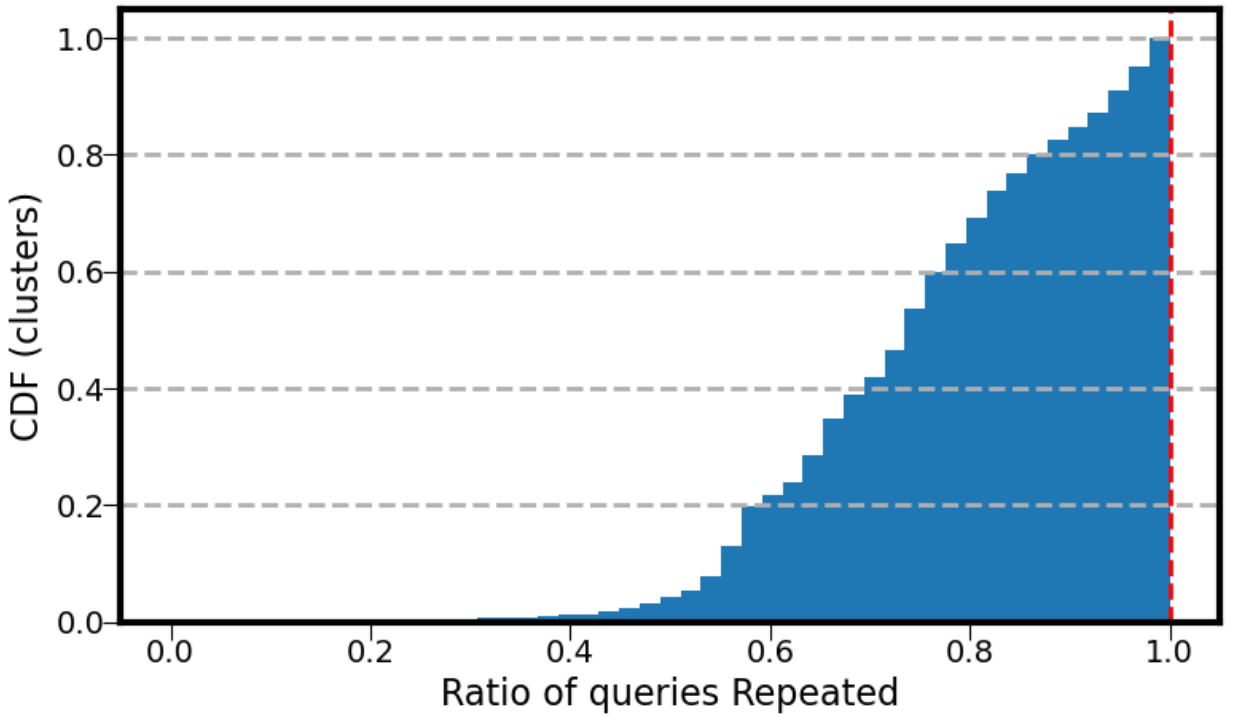
No Transformer 😮‍💨

Let's define **repeated queries** as the 2nd and plus occurrence based on the plan tree.

Ratio of queries Repeated is defined as fraction of queries that are repeated (at least once).

Repeating queries are very common, for example,

- At least 60% queries are repeated queries for 80% of clusters,
- Overall, 80% of queries have been seen before.



It is an **opportunity** to improving existing ML predictors which are not aware of such pattern by design!

Stage 2: A similar query on this cluster

How

1. Removed duplicated queries from training dataset for better coverage without increasing training data size.

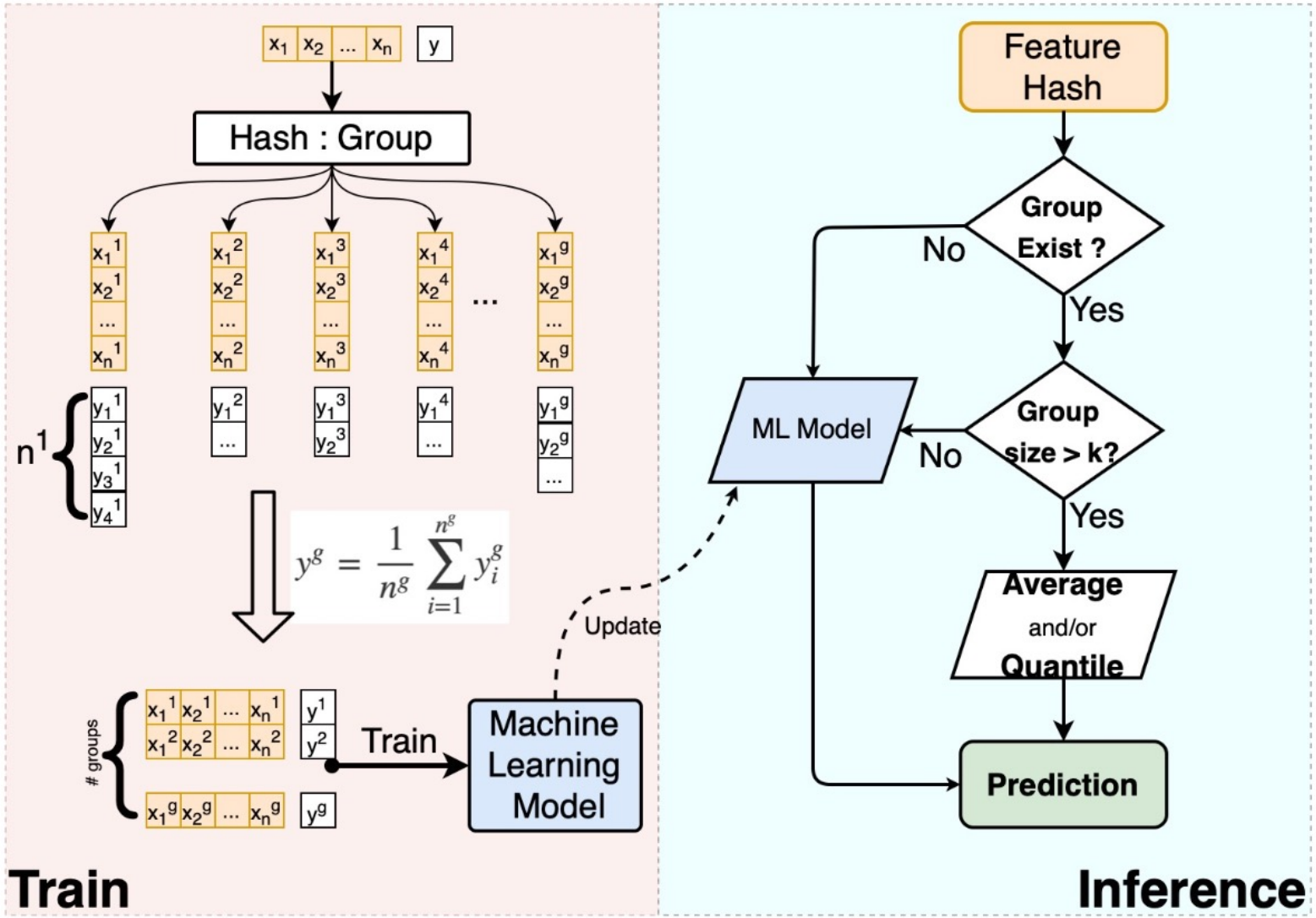
2. Make use of high repeating queries by caching truth of previous runs for the prediction of future repetitions.

Mixture of Experts (MoE)

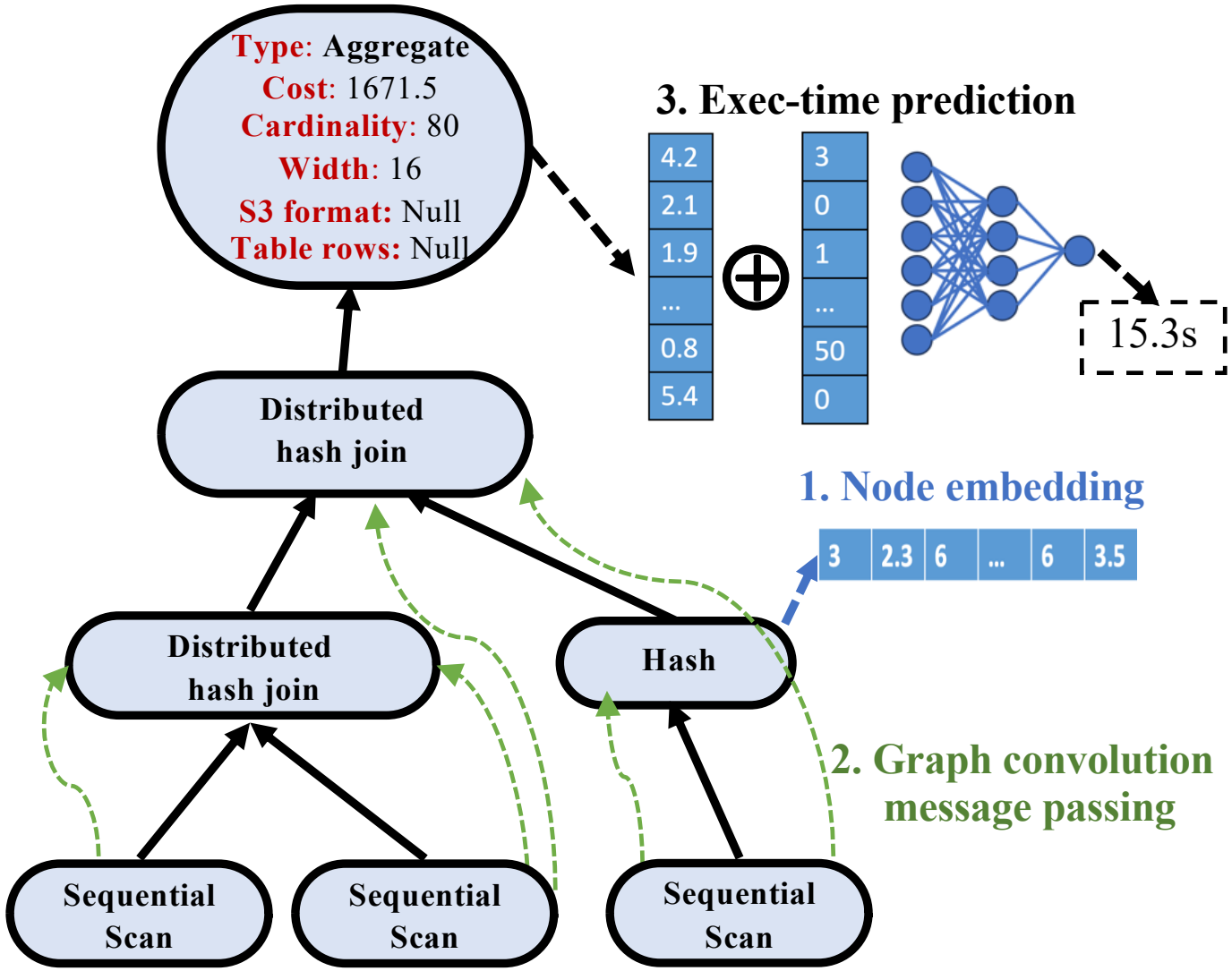
- One expert (kNN) for repeated queries
- One expert (ML regressor) for the rest

Bonus:

- ***Explicit*** memorialization instead of ***best-effort*** learning.
- Guaranteed to not making-mistake-twice.



Stage 3: anything like this over the fleet?



The Data-Driven way:

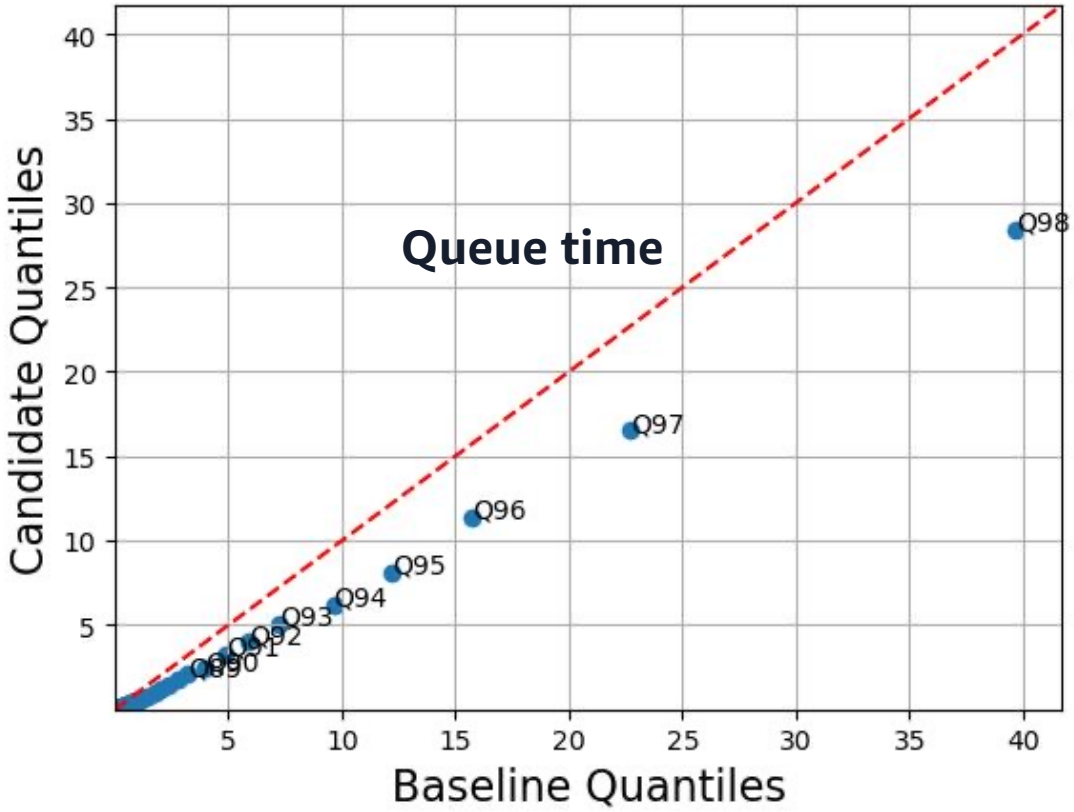
1. MLP to encode plan node feature
2. GCN message passing to understand correlation between plan nodes
3. Concatenate final query feature with features on hardware type and concurrency level to make final prediction.

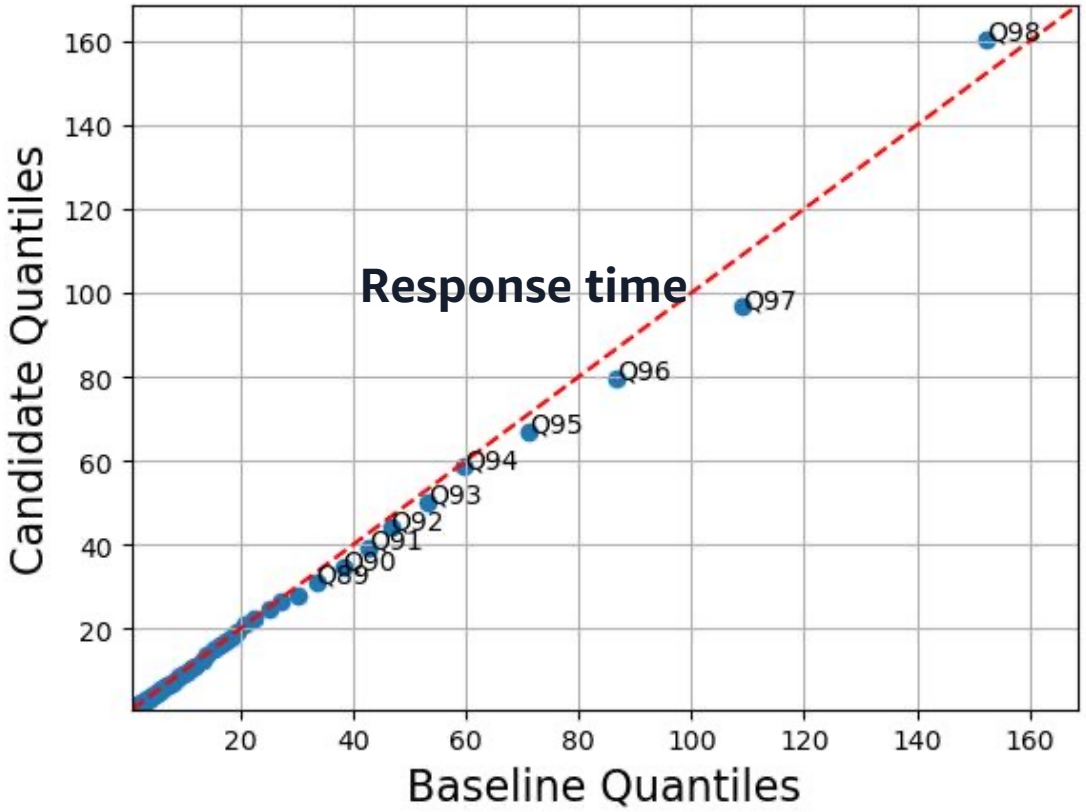
Impact from Production Environment

From a random sample of heavily queued clusters:

- P50 queue time reduced by 3x from 20ms to 5.8ms.
- P90 queue time reduced by 1.6x from 4s to 2.4s
- P99 queue time reduced by 1.6x from 103s to 66s.

- Time spent on Spilled queries were reduced to half.
- Huge same for Serverless customer as they paid for each second of use.





Conclusions and Lessons/Open-Q Learnt

1) *Data-Driven* is better than *Heuristic* driven as customers workload are so diverse.
2) Fleet Data from *Production* are more useful than it from *TPC-DS* experiments.
3) Data *Quality* is as important as *Quantity*
4) High Quality data with complicated models like NN are more accurate but also more expensive.
5) Staged MoE points out a way to practically integrate expensive machine learning models on the critical path of customer-facing production systems.

6) Similar as Optimizer, Stale or missing table stats makes prediction a challenge task.
7) Queries barely run in isolation, contention (not only with other user query, but also system jobs) can void pre-run prediction.

Thank you!

Interested in working/interning with us on ML for (database)system?
drop us an email: zcl@amazon.com