

# Low Rank Approximation for Learned Query Optimization

Zixuan Yi<sup>1</sup>, Yao Tian<sup>2</sup>, Zack Ives<sup>1</sup>, Ryan Marcus<sup>1</sup>

<sup>1</sup> University of Pennsylvania <sup>2</sup> The Hong Kong University of Science and Technology

Simple, low-overhead  
**Linear Methods** can perform  
nearly as effective as complex  
**deep learning approach** for  
**Offline Learned QO**.

## Offline Learned QO

**Why?** Current Learned QOs cause **unpredictable regressions**. (“my query was fast yesterday, why is it slow today?”)

**How?** verify that potential new query plans are actually better than the default plan.

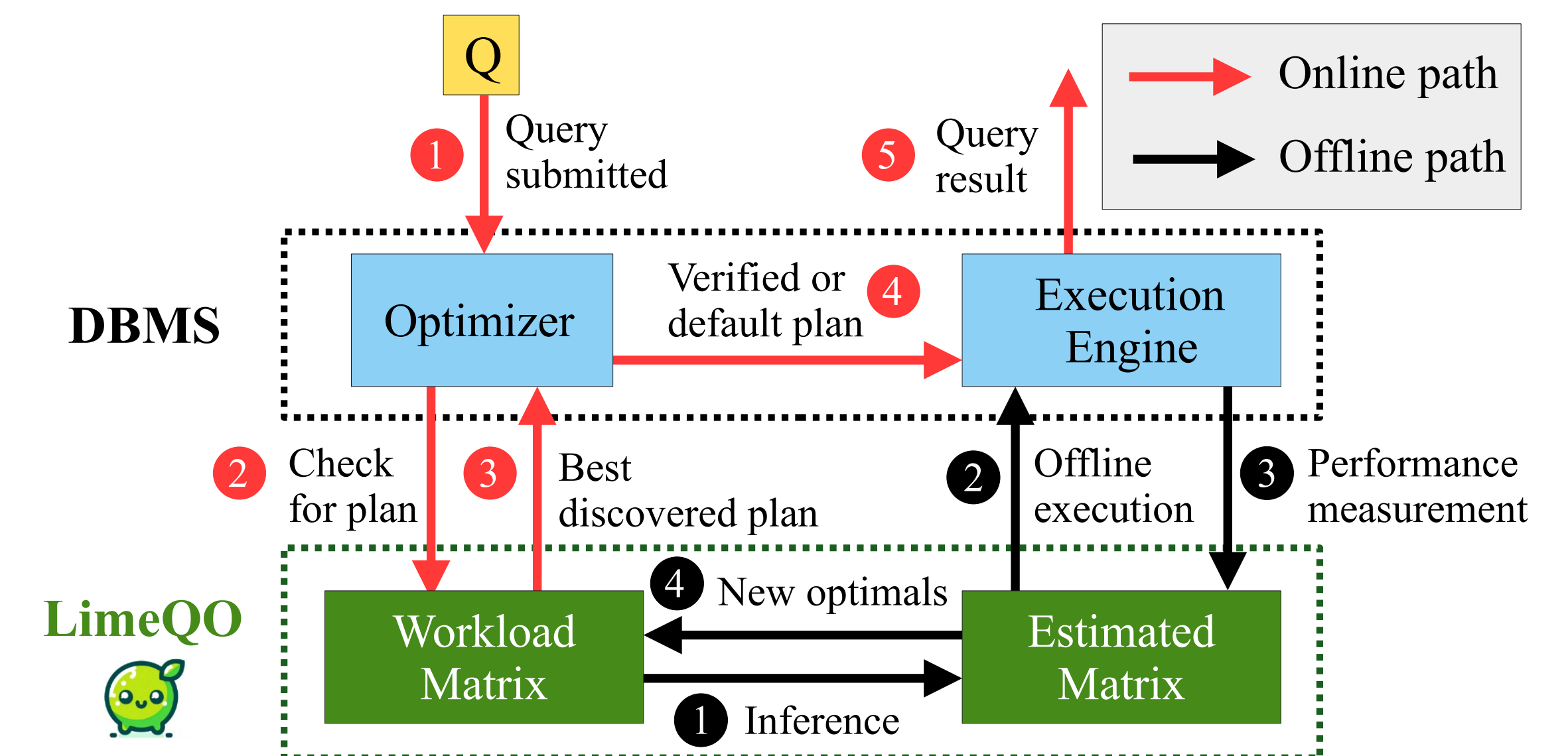
**Setting:** Repetitive workload!

**Goal:** simultaneously minimize the workload latency and the total offline exploration time, while maintaining the “no-regressions” guarantee.



Checkout the paper for more detailed info:

[zixy17.github.io/pdf/aidm.pdf](https://zixy17.github.io/pdf/aidm.pdf)



## Low Rank Workload Matrix

### Workload Matrix $M$ :

Each row represents a SQL query.  
Each column represents a hint (parameterization of the QO).

One possible hint:

**Disable** Nested Loop Join  
Enable Hash Join  
Enable Merge Join  
Enable Index Scan  
Enable Seq Scan  
Enable Index-only Scan

Each entry represents the latency time for DB to execute the query under the hint.

### $M$ is **LOW RANK**

Intuition: two queries that behave similarly on some hints are likely to behave similarly on other hints as well.

### Option1: LimeQO (Linear Method Only)

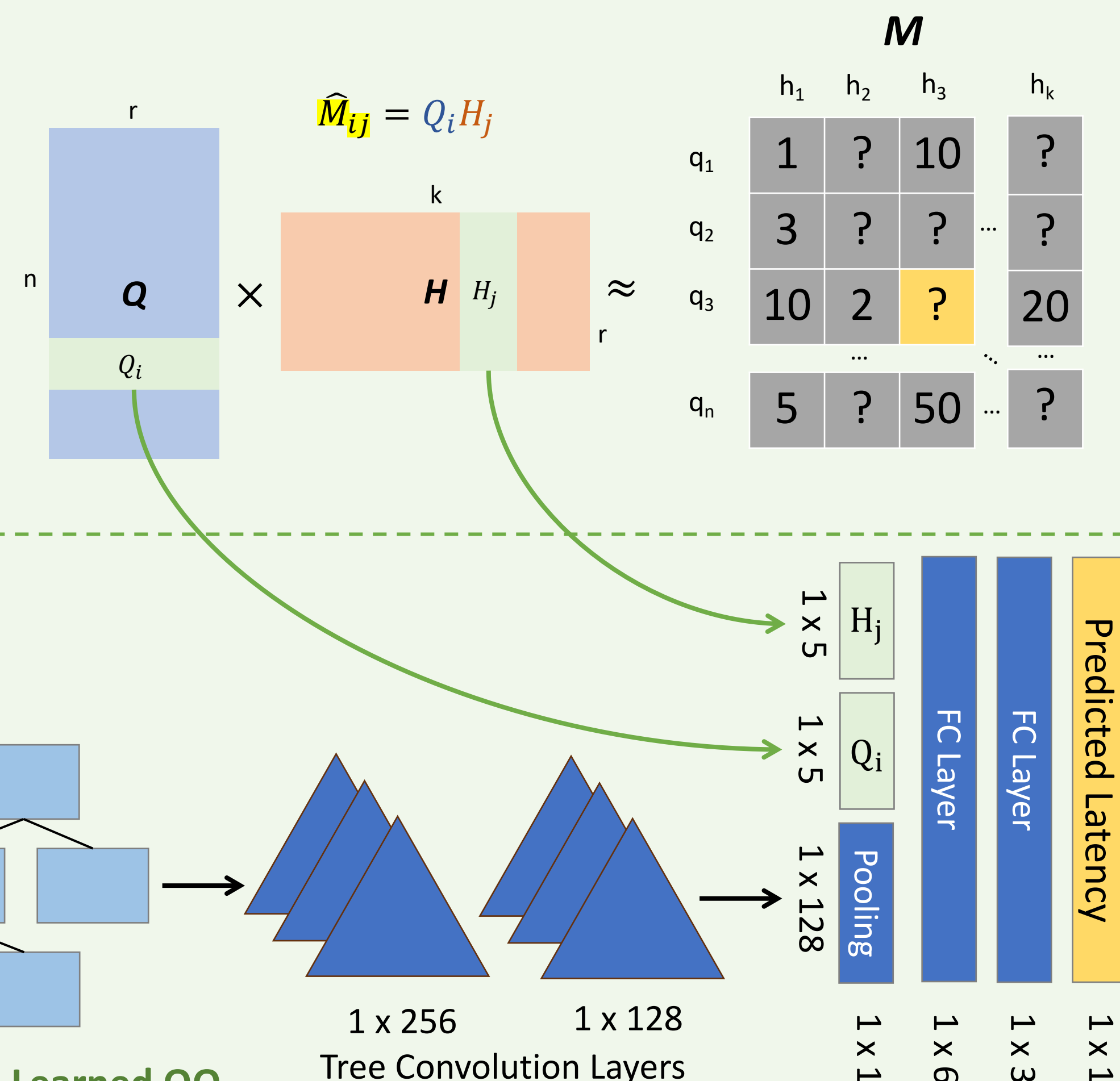
Use **Alternating Least Squares** Algorithm to recover the unobserved entries from the observed ones.

### Option2: LimeQO+ (Adding Query Features in)

Use query plan features in tree structure (including cardinality estimation result and cost) and QH Matrix **embeddings** as input.

### LimeQO strategy for Offline Learned QO

Generate the full matrix, then explore the queries with the **biggest potential gain ratio** (current min observed value – predicted row min) / predicted row min

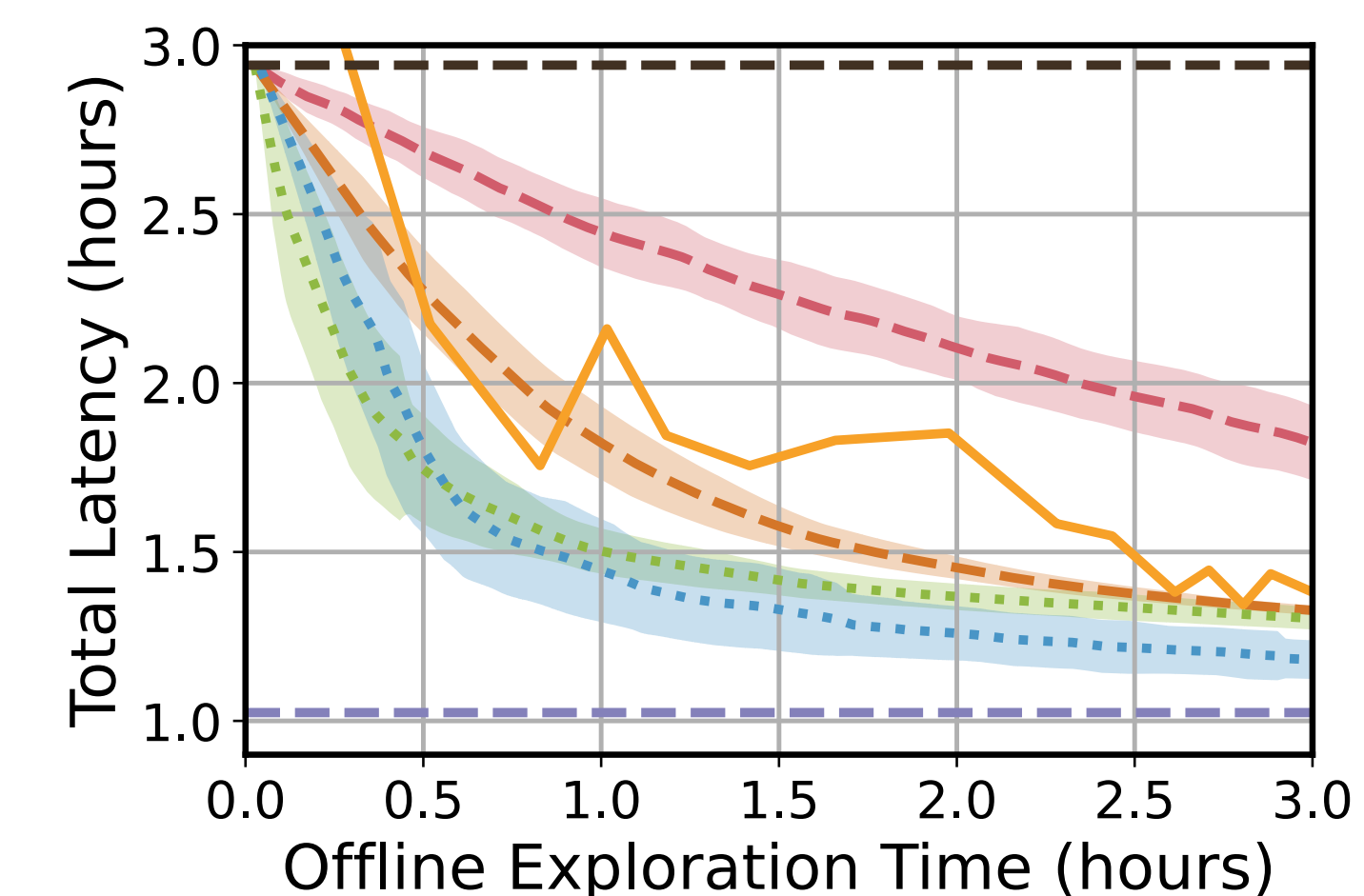


## Experiments

Dataset: CEB core workload

- 3133 queries in total
- takes ~3 hours for PostgreSQL default to finish
- ~1 hour if every query is chosen the optimal hint

Random Greedy LimeQO LimeQO+ Offline-Bao



**Caption:** Both LimeQO and LimeQO+ outperform Bao. Even without any features, pure linear method (LimeQO) can perform nearly as effective as the one using complex Neural Network (LimeQO+).

**Random** randomly explore unobserved entries.

**Greedy** explore the tail latency queries first.

**LimeQO** uses only Linear Method to predict.

**LimeQO+** uses query features and matrix embeddings to train and predict.

**Offline-Bao** uses TCNN to select unobserved entries to explore. It does not verify plans before selecting them so regressions happens.

**Total Latency Time** is simply adding up the observed row minimum in the workload matrix. **Offline Exploration Time** is the total time to execute the query plan + overhead time of the technique. We also applied timeout and censored techniques to reduce offline time.