Self-Driving Components

Weichen Li Saatvik Shah

Progress

✓ 75% Goal: Complete

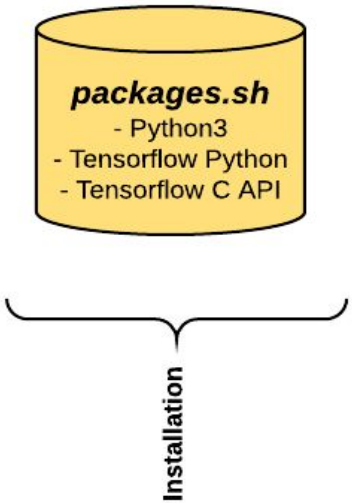
✓ 100% Goal: Complete

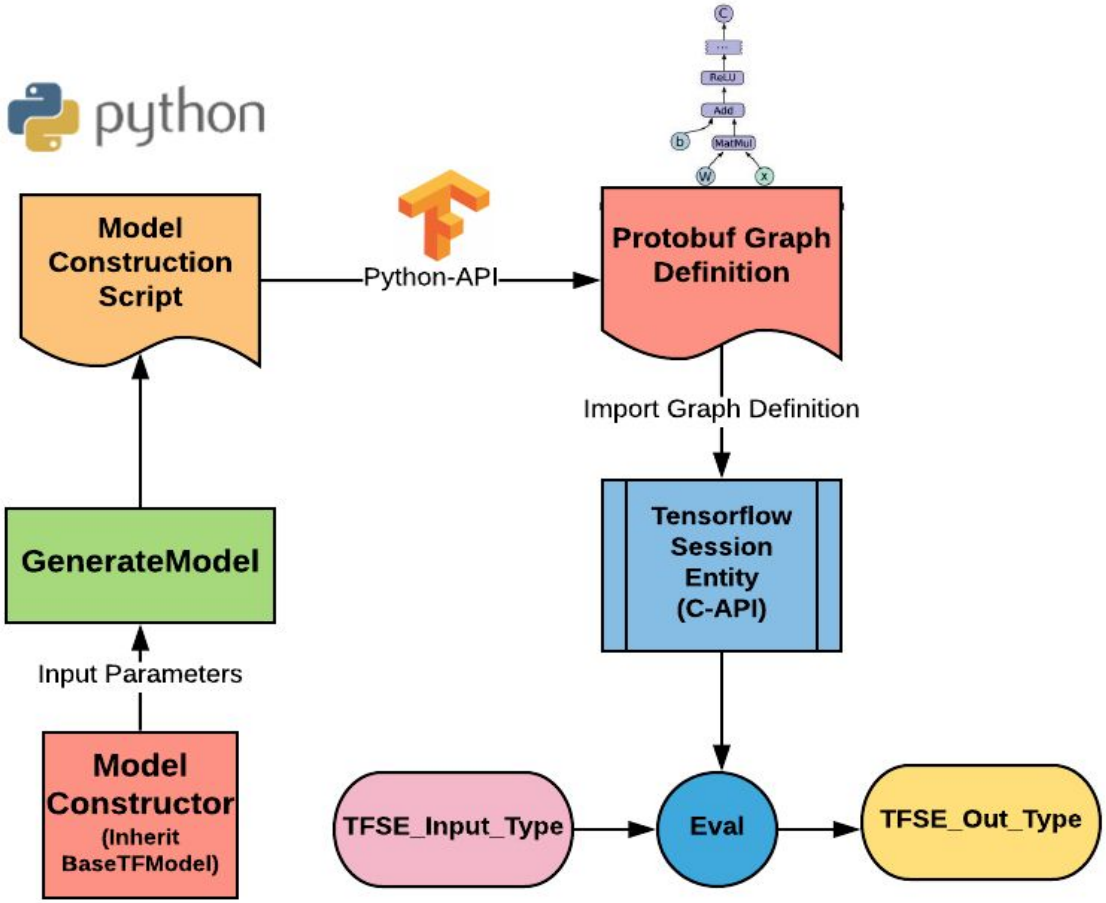
☹ 125% Goal: Ongoing

75% Goal

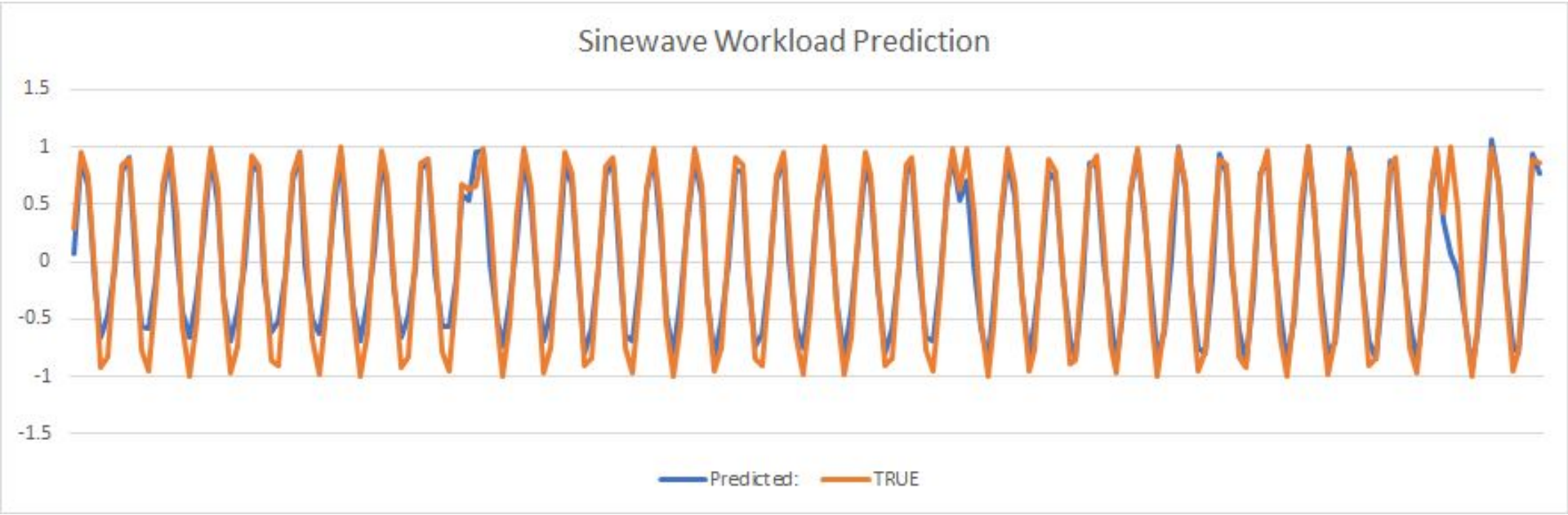
Tensorflow Integration

- ✓ Model Development Pipeline
- ✓ Working Example:
  - ❏ LSTM for Workload Prediction
- ✓ Tests:
  - ❏ Basic Tests
  - ❏ Multiple-Sine Wave Workload Forecasting



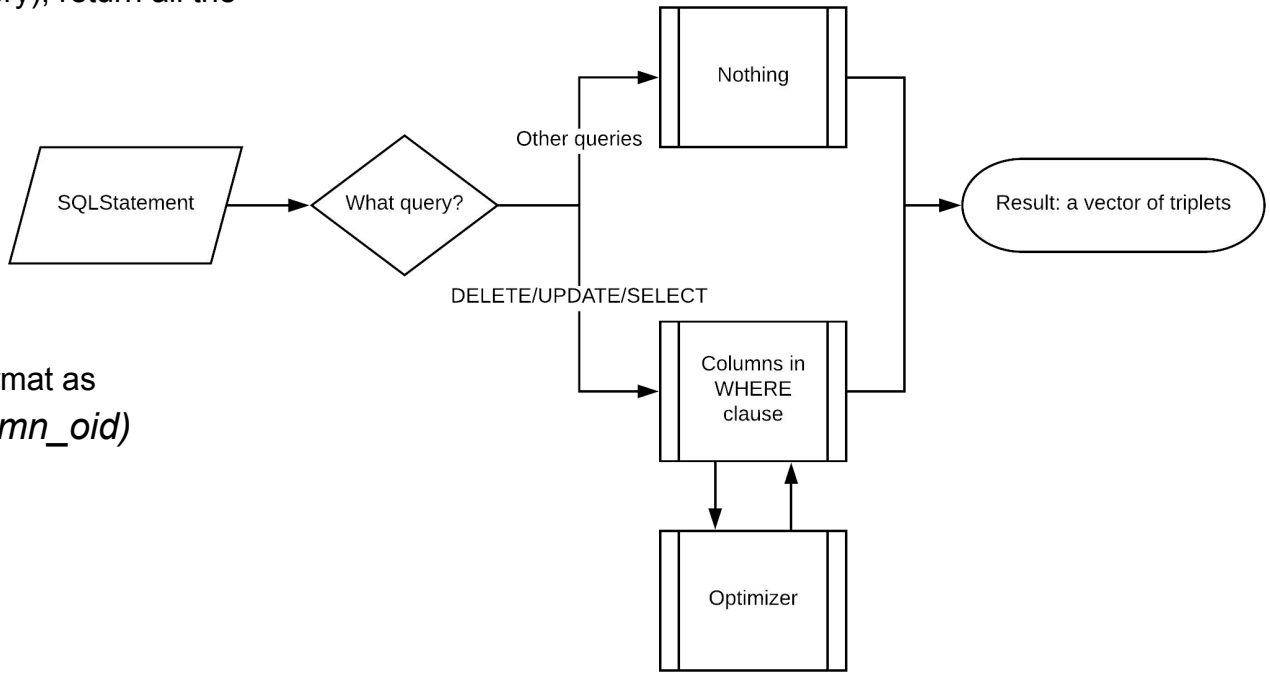


Workload Prediction



Recommended Indexes

Task: Given a *SQLStatement* (a query), return all the *indexable columns* in a vector.

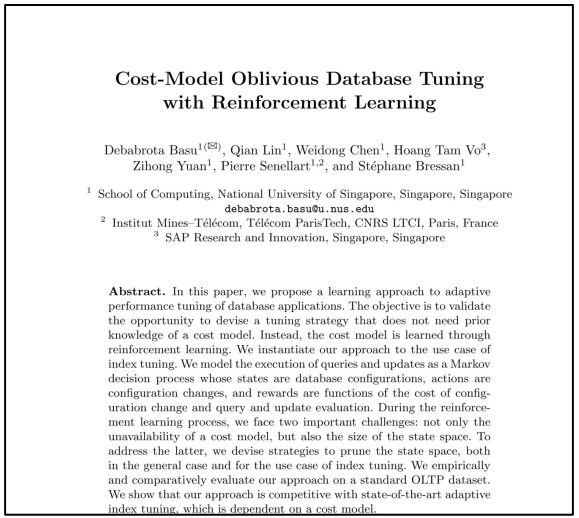


Returned vector: a vector of triplets
Each element in the vector has a format as
(dababase_oid, table_oid, column_oid)

100% Goal

Setting up a RL framework for Dynamic Knob Changes

Recreation of NUS Dynamic Index Tuning "COREIL"



Cost-Model Oblivious Database Tuning with Reinforcement Learning

Debabrota Basu[1(✉)], Qian Lin[1], Weidong Chen[1], Hoang Tam Vo[3], Zihong Yuan[1], Pierre Senellart[1,2], and Stéphane Bressan[1]

[1] School of Computing, National University of Singapore, Singapore, Singapore
debabrota.basu@u.nus.edu
[2] Institut Mines–Télécom, Télécom ParisTech, CNRS LTCI, Paris, France
[3] SAP Research and Innovation, Singapore, Singapore

Abstract. In this paper, we propose a learning approach to adaptive performance tuning of database applications. The objective is to validate the opportunity to devise a tuning strategy that does not need prior knowledge of a cost model. Instead, the cost model is learned through reinforcement learning. We instantiate our approach to the use case of index tuning. We model the execution of queries and updates as a Markov decision process whose states are database configurations, actions are configuration changes, and rewards are functions of the cost of configuration change and query and update evaluation. During the reinforcement learning process, we face two important challenges: not only the unavailability of a cost model, but also the size of the state space. To address the latter, we devise strategies to prune the state space, both in the general case and for the use case of index tuning. We empirically and comparatively evaluate our approach on a standard OLTP dataset. We show that our approach is competitive with state-of-the-art adaptive index tuning, which is dependent on a cost model.

Birds Eye View

Get Batch
(10 queries)

Workload

Execute

Metrics
Collection

Ask for Metrics

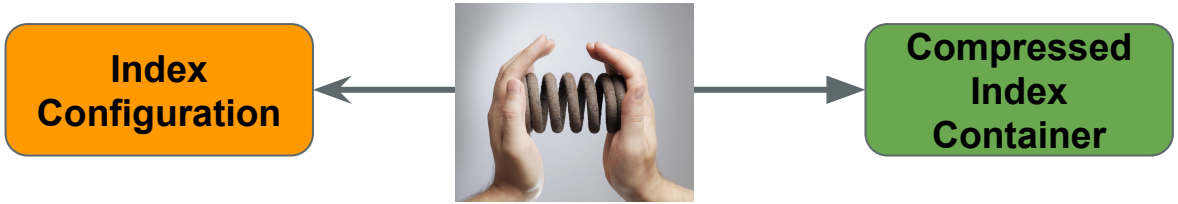
- (Query, Metric) Pair

Update

Tunable Knobs

Testing Correctness - I

- ➔ **Compressed Index Configuration Modules**
  - Equality of Compression vs. Decompression on varying sample Index Configurations
  - Tested Add and Drop candidates against what might be expected



- ➔ **ML Models**
  - Synthetic Workloads
    - LSTM: Sine Wave workload
    - RLSE: Linear Workload

Testing Correctness - II

- ➔ **“RL” Implementation**
  - We setup synthetic Cyclic Workloads*
    - Simple: Cyclic SELECTs with varying WHERE clauses on a Single Table
    - Complex: Cyclic SELECTs/INSERTs/UPDATEs clauses on Multiple Tables
  - Checks for drop in Cost over time
  - Checks for drop in Cost over an untuned DBMS

*Modifications on top of the Autoadmin teams Testing Utility

Quality of Code Assessment

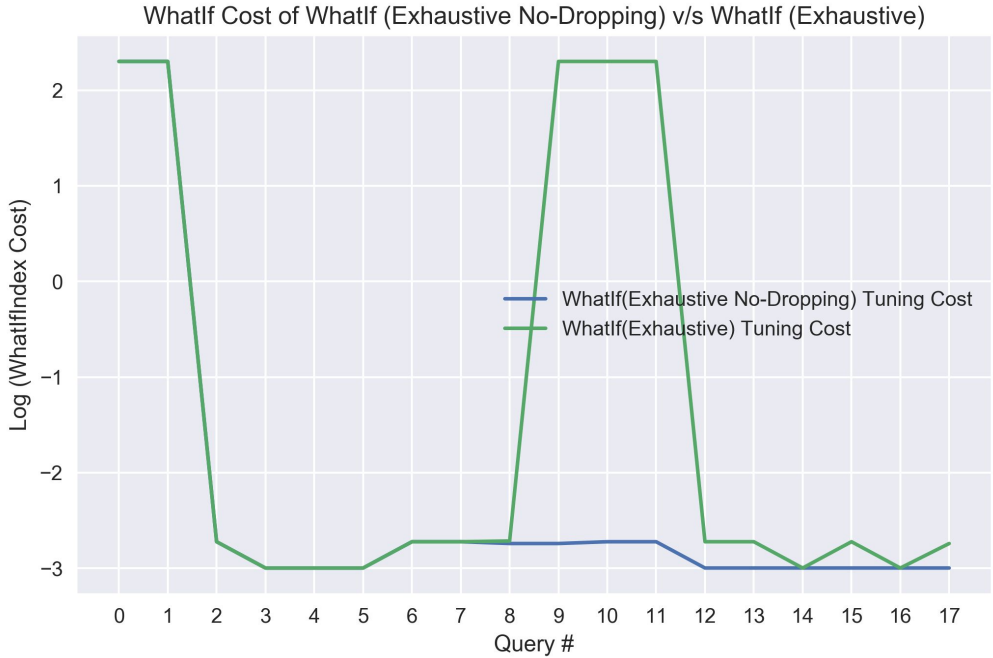
Code Quality (1: need more work to improve 2: good 3: production-quality)

- (3) ***compressed_index_config.h/.cpp***: internal representation of index configurations
- (3) ***compressed_index_config_util.h/.cpp***: utility functions to manipulate index configs
- (2) ***compressed_idx_config_test.cpp***: test file for index configs
- (3) ***lspi_tuner.h/.cpp***: Least-Squares Policy Iteration class
- (3) ***lstdq.h/.cpp***: LSTDQ Estimator class
- (2) ***rlse.h/.cpp***: Recursive Least Squares Estimator
- (2) ***lspi_test.cpp***: test file for LSPI tunner
- 75% Work: All in Production

Benchmarks and Results

Tuning Modes Analyzed

1. **No Tuning**
2. **What-If Exhaustive/Brute Force**
   a. *With Index Dropping*
   b. *Without Index Dropping*
3. **RL(LSPI) Tuning**
   a. *Non-Exhaustive: Only Single Column Indexes*
   b. *Exhaustive: With Multicolumn Indexes*

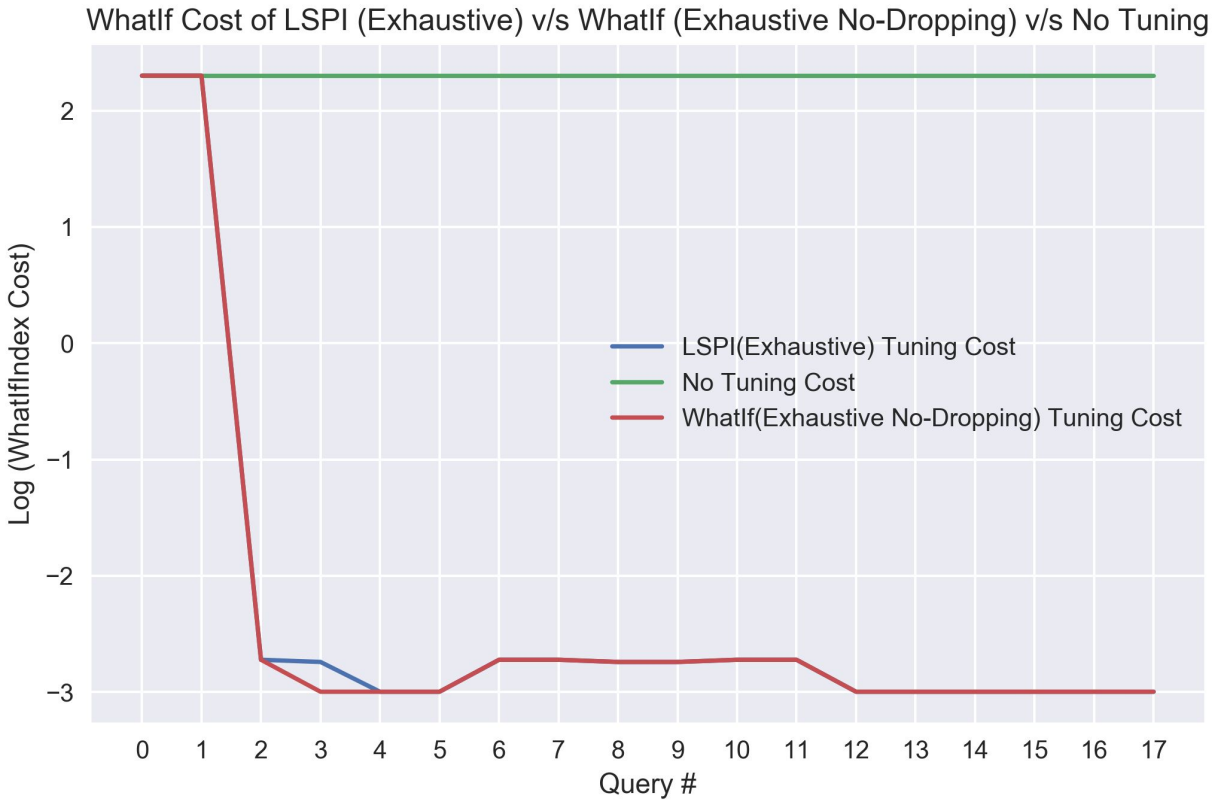


Why Index Dropping is Tricky?

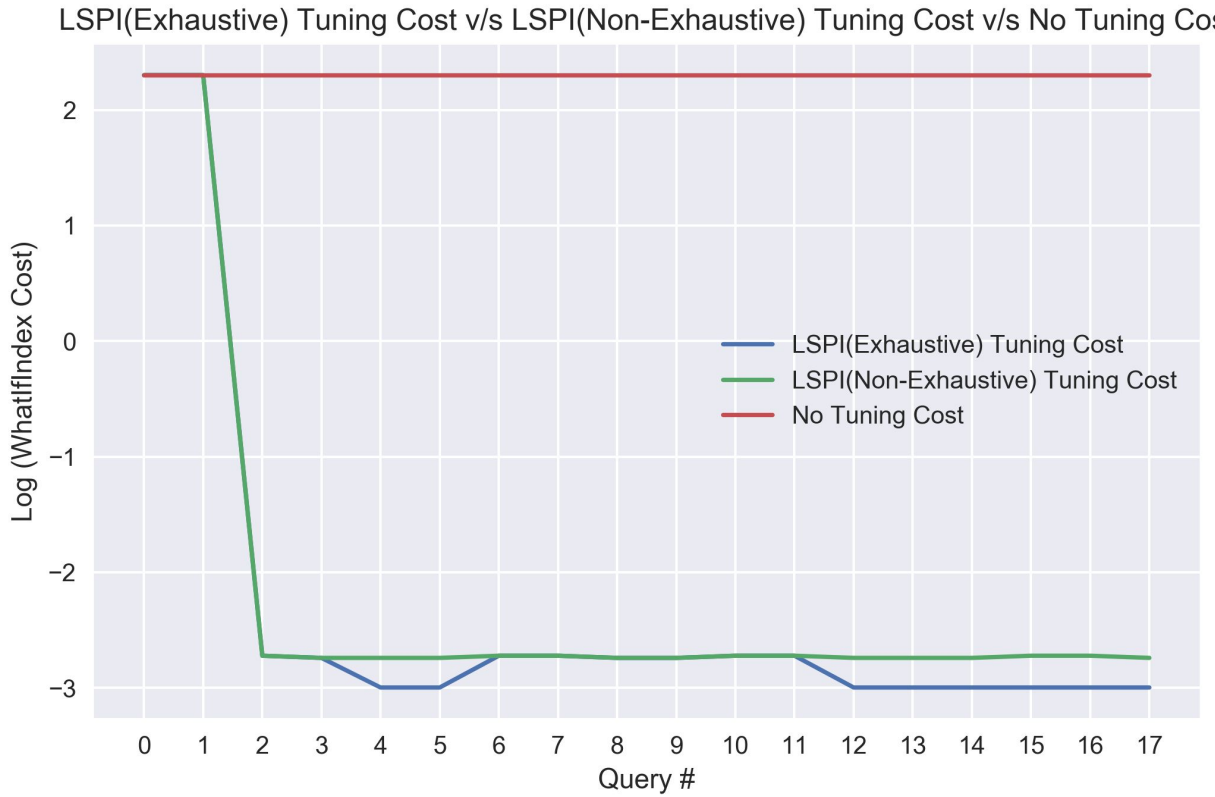
Simple Cyclic Workloads

- ➔ Simple Mix of SELECTs with a variable number of WHERE clauses on Single Table: ~10 queries
- ➔ Number of Cycles Set to 2
- ➔ Since we have SELECTs only, the RL algorithm will not perform Drops

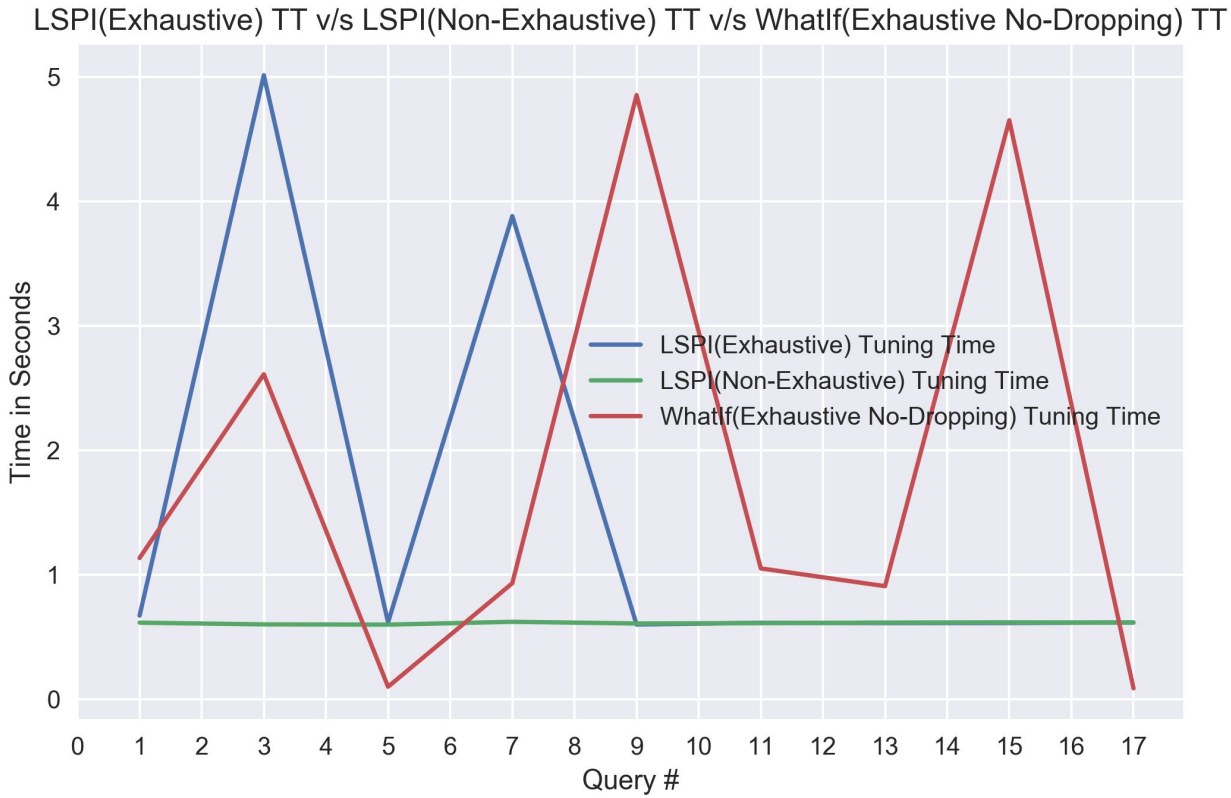
Simple Cyclic Workloads: Cost Analysis of RL vs. Best Case



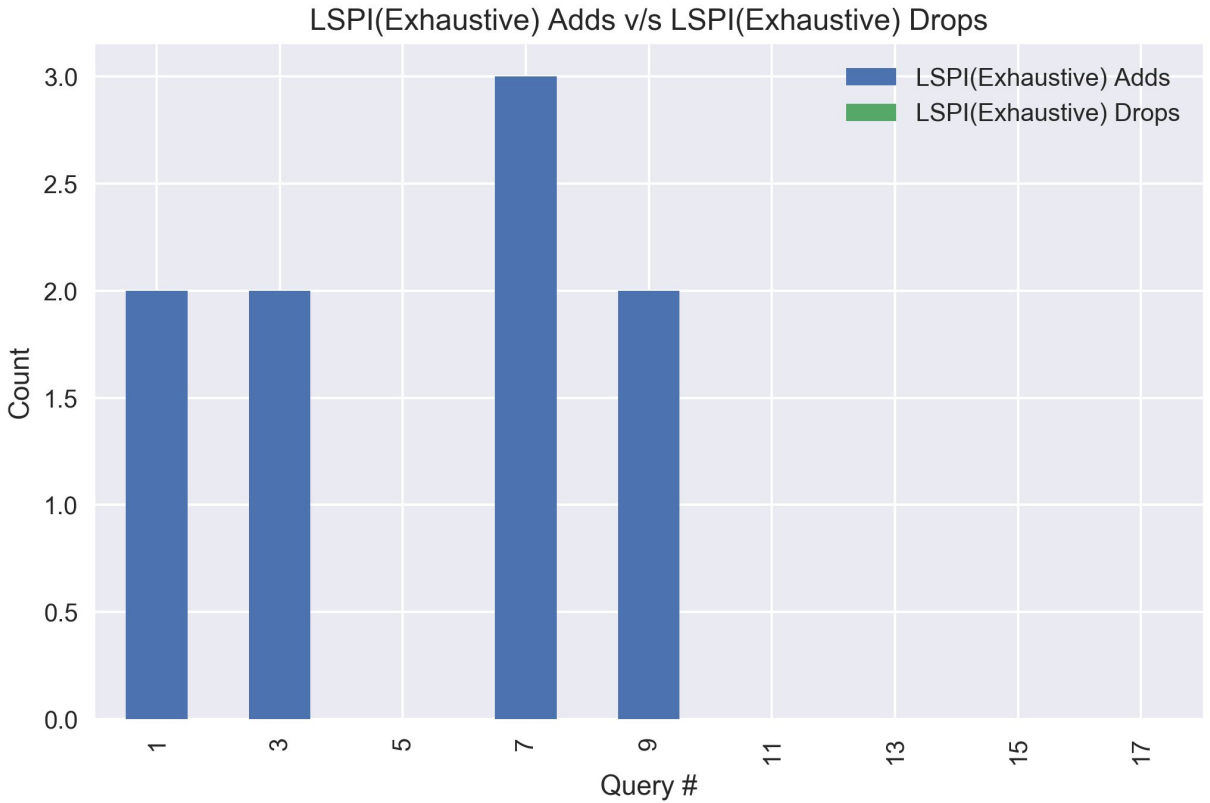
Simple Cyclic Workloads: Cost Analysis of MultiCol vs. SingleCol Search



Simple Cyclic Workloads: Time Analysis



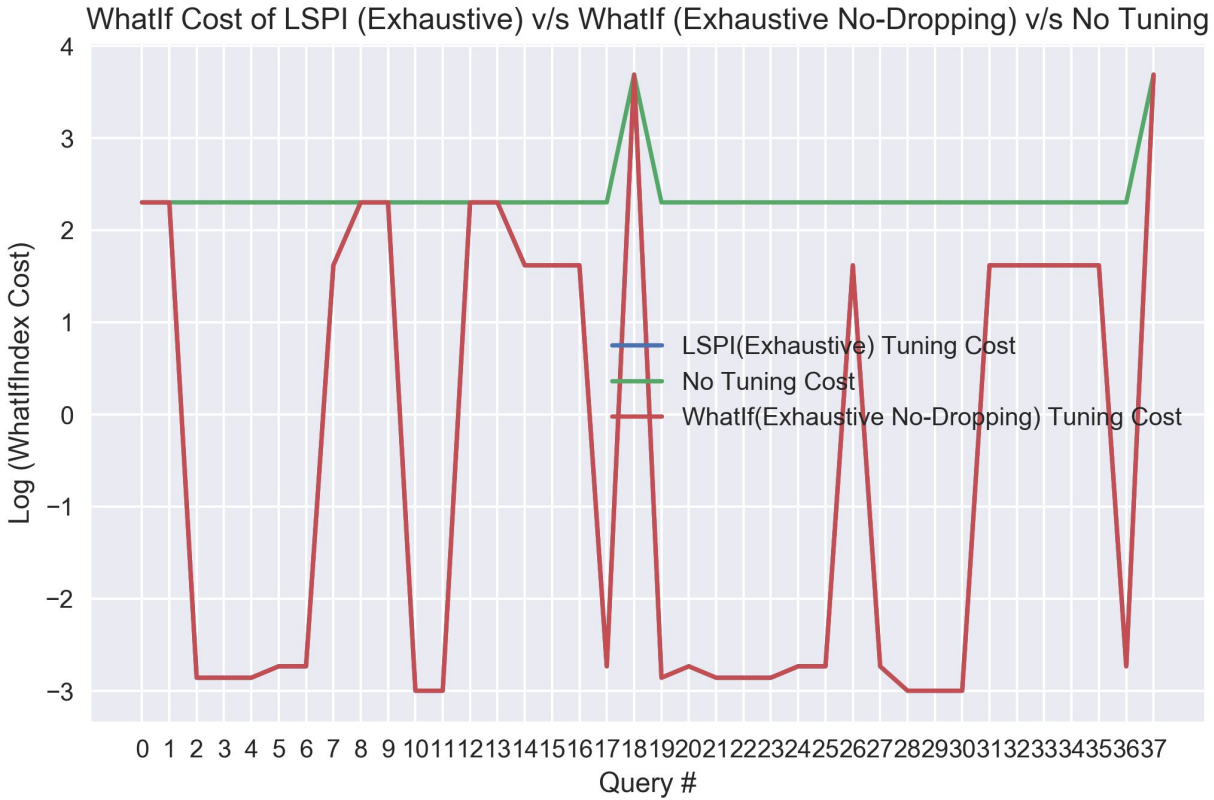
Simple Cyclic Workloads: LSPI Trend of Adding Indexes



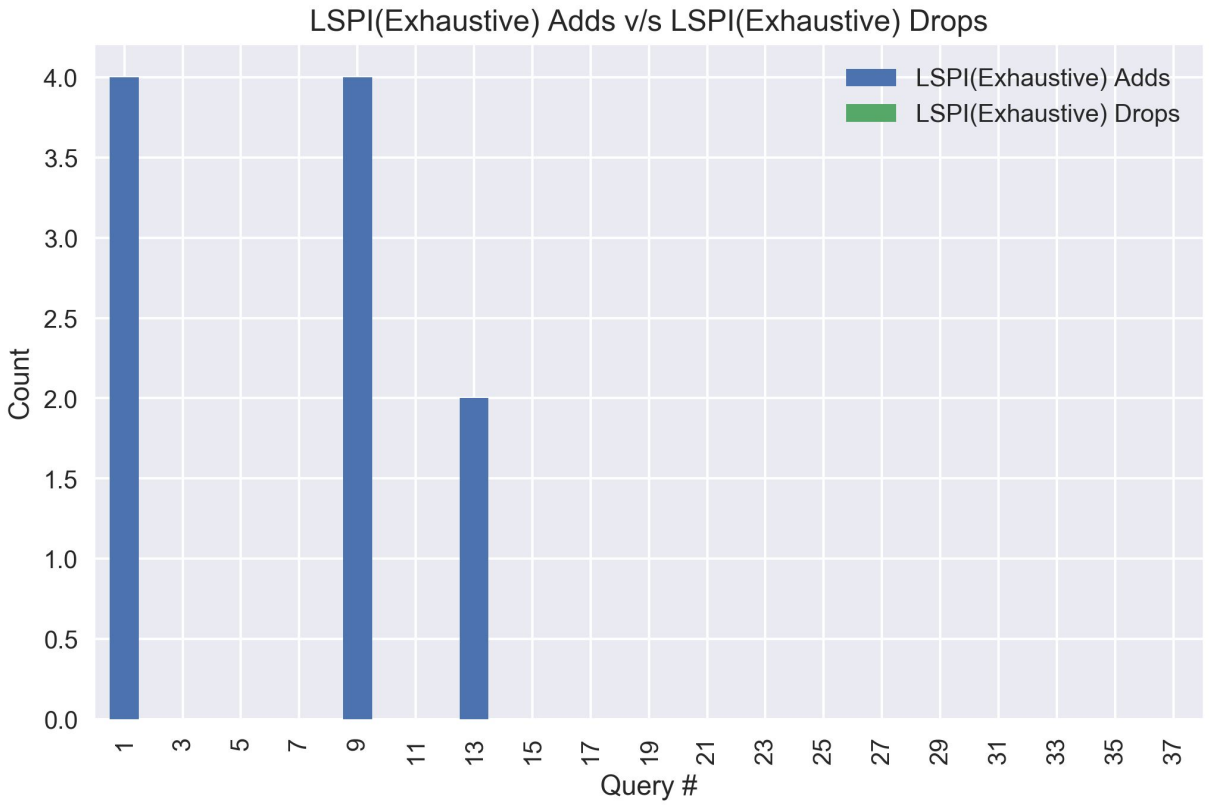
Complex Cyclic Workloads

- Mix of SELECTs/UPDATE/DELETE/JOIN over Multiple Tables: ~20 queries
- Number of Cycles Set to 2
- Two Cases
  - Dropping Disabled
  - Dropping Enabled

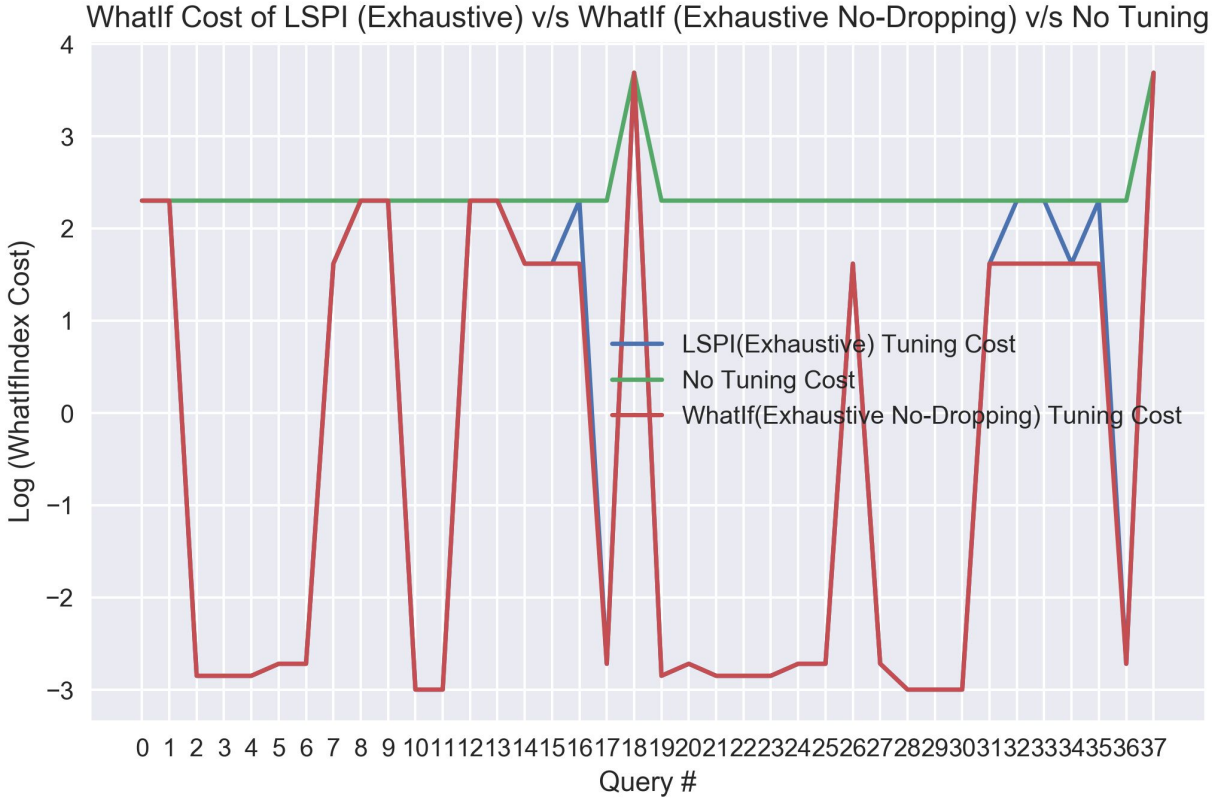
Complex Cyclic Workloads(Dropping Disabled): Cost Analysis of RL vs. Best Case



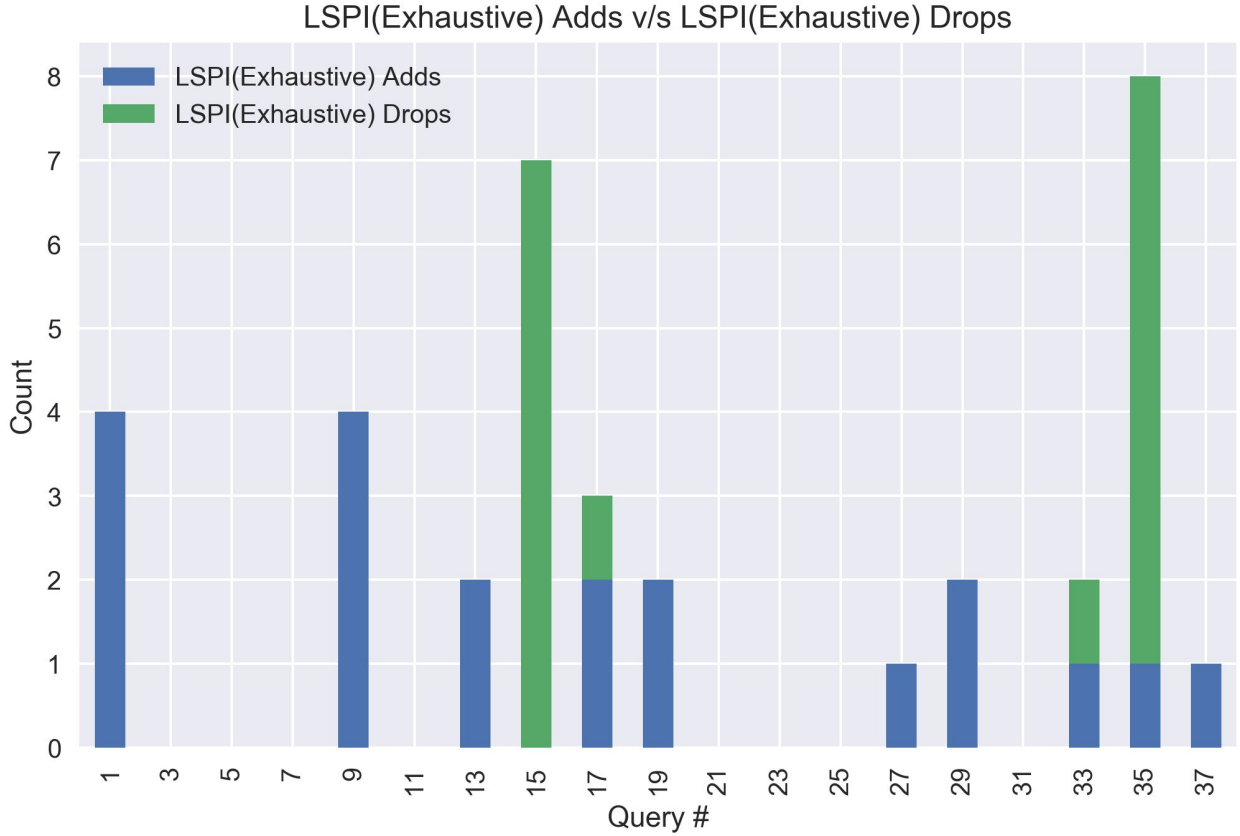
Complex Cyclic Workloads(Dropping Disabled): LSPI Trend of Adding Indexes



Complex Cyclic Workloads(Dropping Enabled): Cost Analysis of RL vs. Best Case



Complex Cyclic Workloads(Dropping Enabled): LSPI Trend of Adding/Dropping Indexes



Cumulative Costs

Workload	No Tuning	LSPI (Exhaustive)	WhatIf (Exhaustive)	LSPI (Non-Exhaustive)	WhatIf (Exhaustive No-Dropping)
Simple Cyclic	180.000000	20.921036	50.772381	21.044231	20.905663
Complex Cyclic (No Dropping)	440.000000	191.685303	261.120544	191.824112	191.680405
Complex Cyclic (With Dropping)	440.000000	211.481110	261.116150	-	191.680405

Future Work

- ➔ TPCC!!
- ➔ Incorporate AutoAdmin Indexes as Add Candidates with Prefix Closures
- ➔ Incorporate Better Metrics:
  - ◆ Number of Tuples Read
  - ◆ Latency(??)
- ➔ More Error Analysis
  - ◆ Effects across different sizes of Multicolumn Indexes
  - ◆ Methods to Improve Quality of Drops
  - ◆ Longer Training Sessions
  - ◆ More Real World Datasets
- ➔ Application of Better RL algorithms
  - ◆ We have Tensorflow + Eigen within Peloton at our disposal
- ➔ More Knobs!!