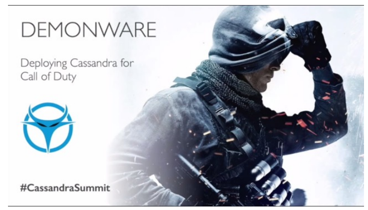
Tuning Log-Structured Merge Trees

Andy Huynh

February 25th 2025

Serving Diverse Data Domains

Low Latency



Distributed IOT

Bigmate

HIPAA Compliant

Amazon Keyspaces for Apache Cassandra

NoSQL Distributed Database

cassandra

LLM Vector Databases

DATASTAX ASTRA DB

Time Series

NETFLIX

Messaging*

Discord

Flexibility Due To Configuration

NoSQL Distributed Database

cassandra

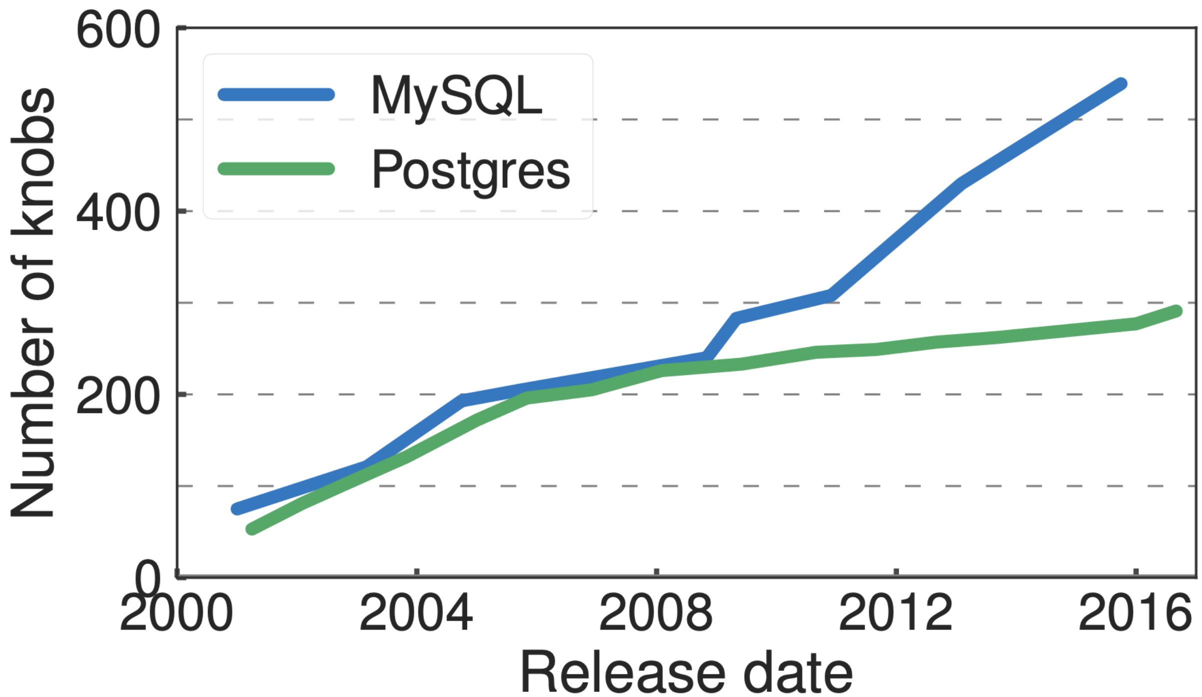
cassandra.yaml

- num_tokens
- max_hints_delivery_threads
- batchlog_endpoint_strategy
- credentials_validity
- row_cache_size
- column_index_size
- column_index_cache_size

...

- memtable_flush_writers

Database Complexity



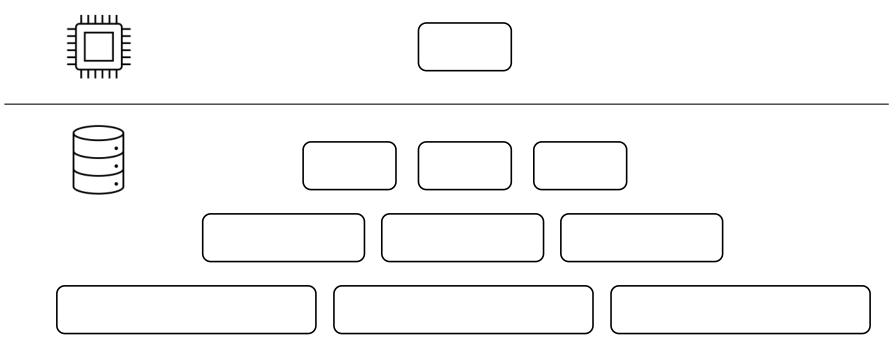
Van Aken D. et. al., "Automatic Database Management System Tuning Through Large-scale Machine Learning". SIGMOD 2017

LSM Trees Are Everywhere!

NoSQL Distributed Database

cassandra

Log-Structured Merge-Trees



RocksDB
levelDB
SQLite
WT
accumulo
DynamoDB
APACHE HBASE

Outline

A Primer on LSM Trees

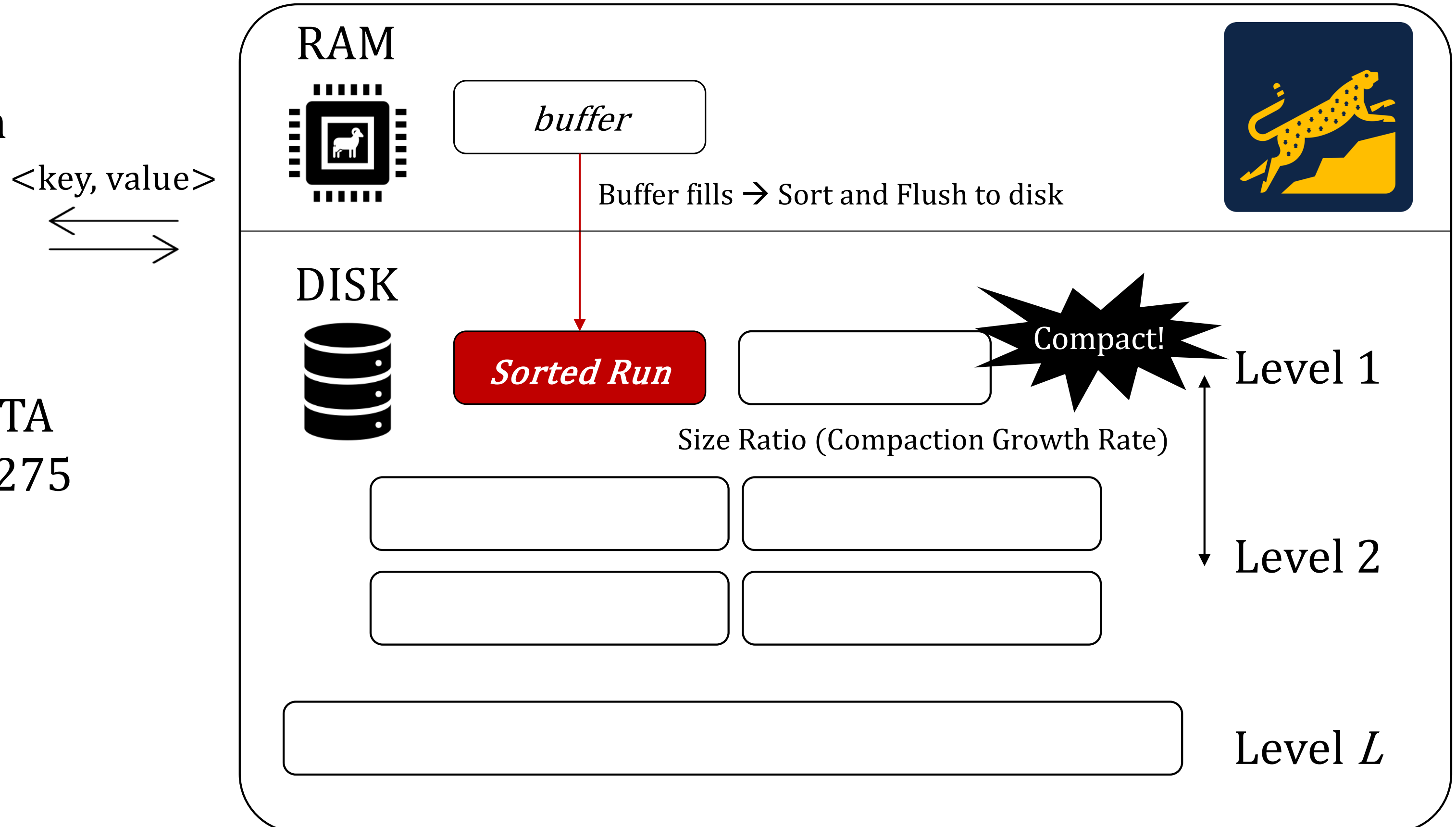
- Flexibility in Compaction Design **[VLDB-J 24]**
- Modeling LSM Trees

Tuning LSM Trees

- ENDURE: Finding Robust Tunings for LSM Trees **[VLDB 22]**
- AXE: Learning to Tune LSM Trees By Task Decomposition **[UR*]**

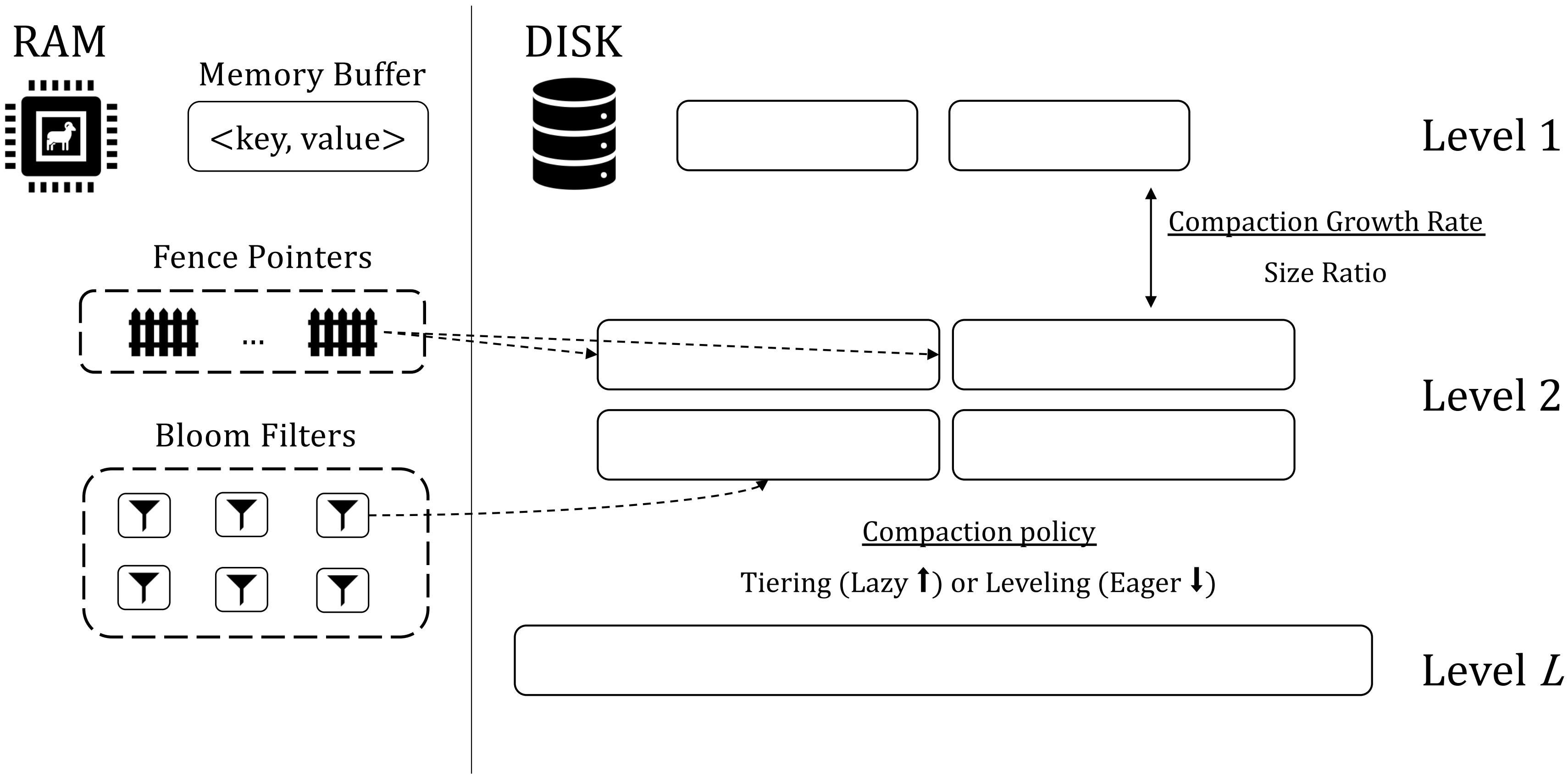
Primer: Log-Structured Merge-Trees

Application

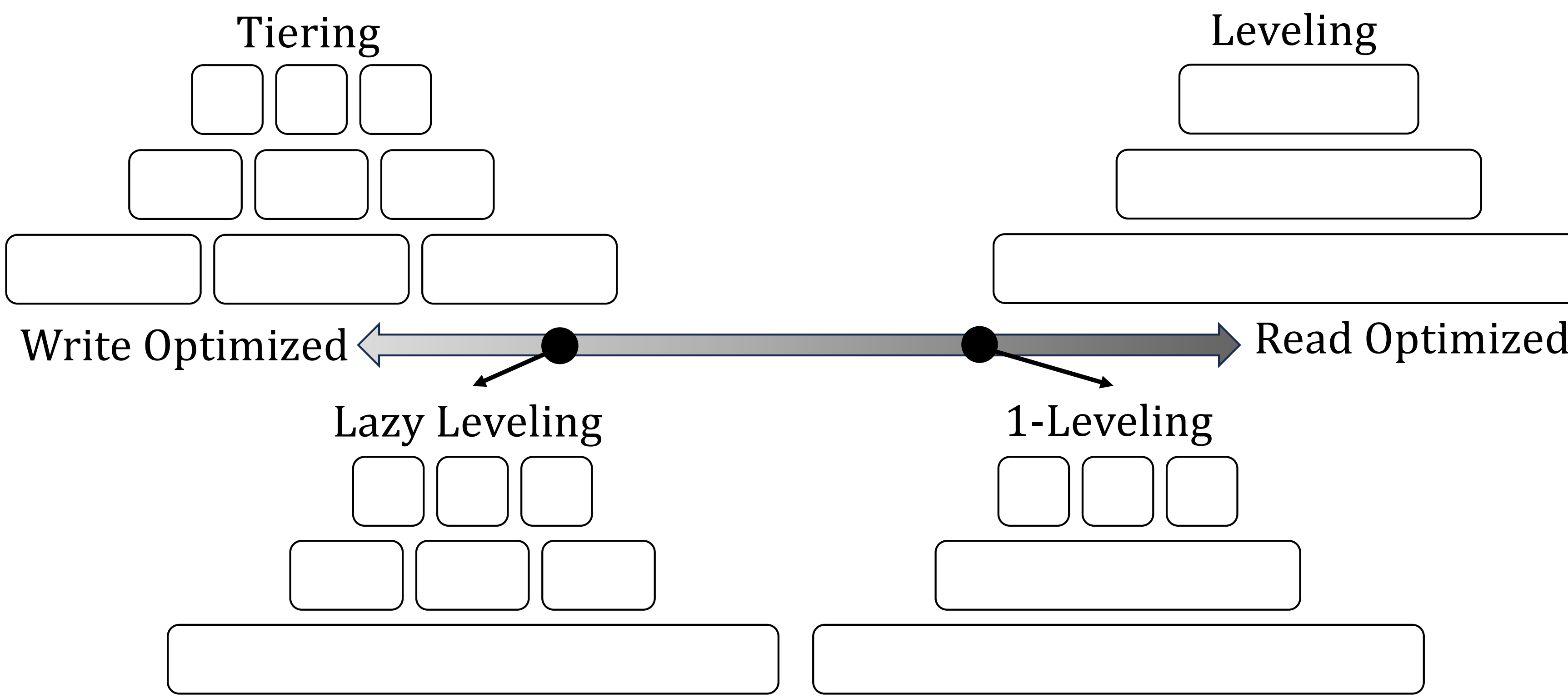


GET 25
PUT 87, DATA
SCAN 10 – 275
PUT 86,
GET 99
GET 47
...
PUT 25

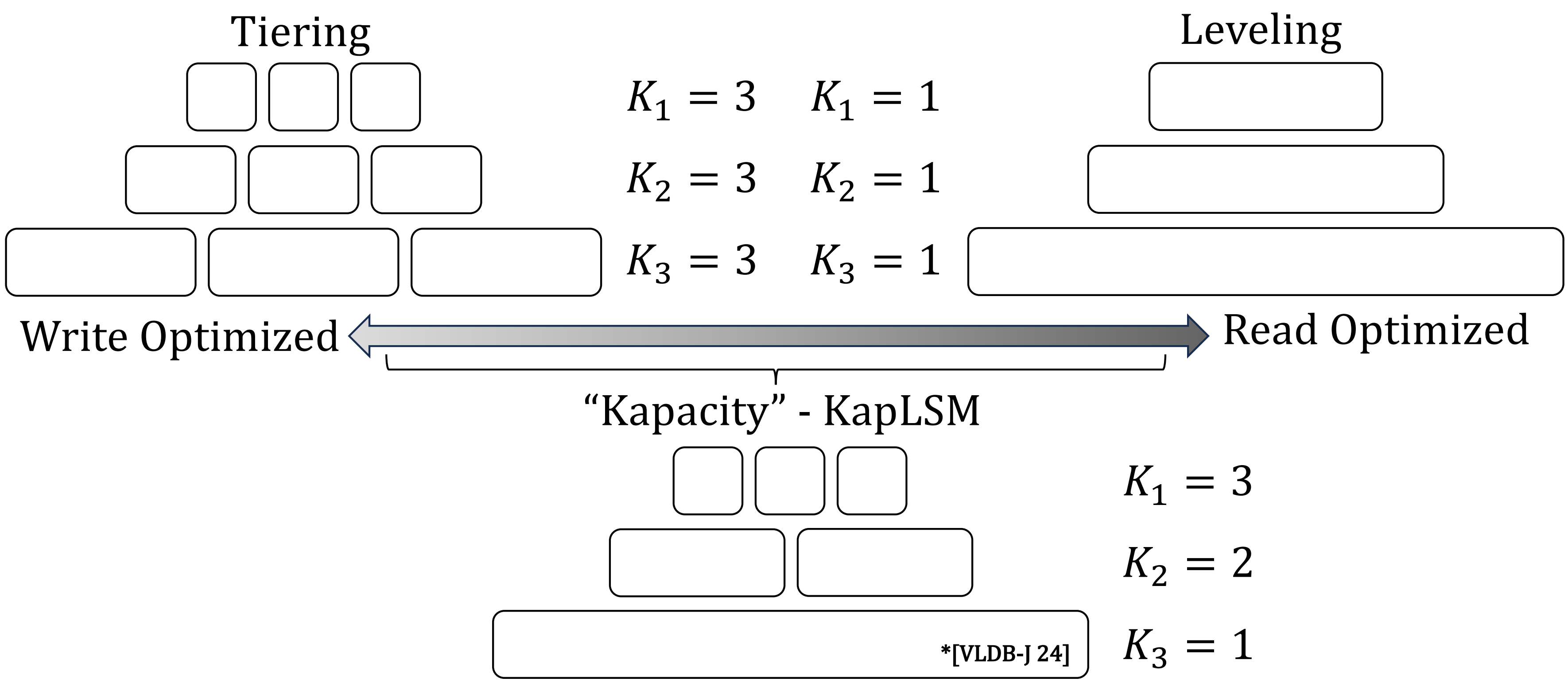
Primer: Log-Structured Merge-Trees



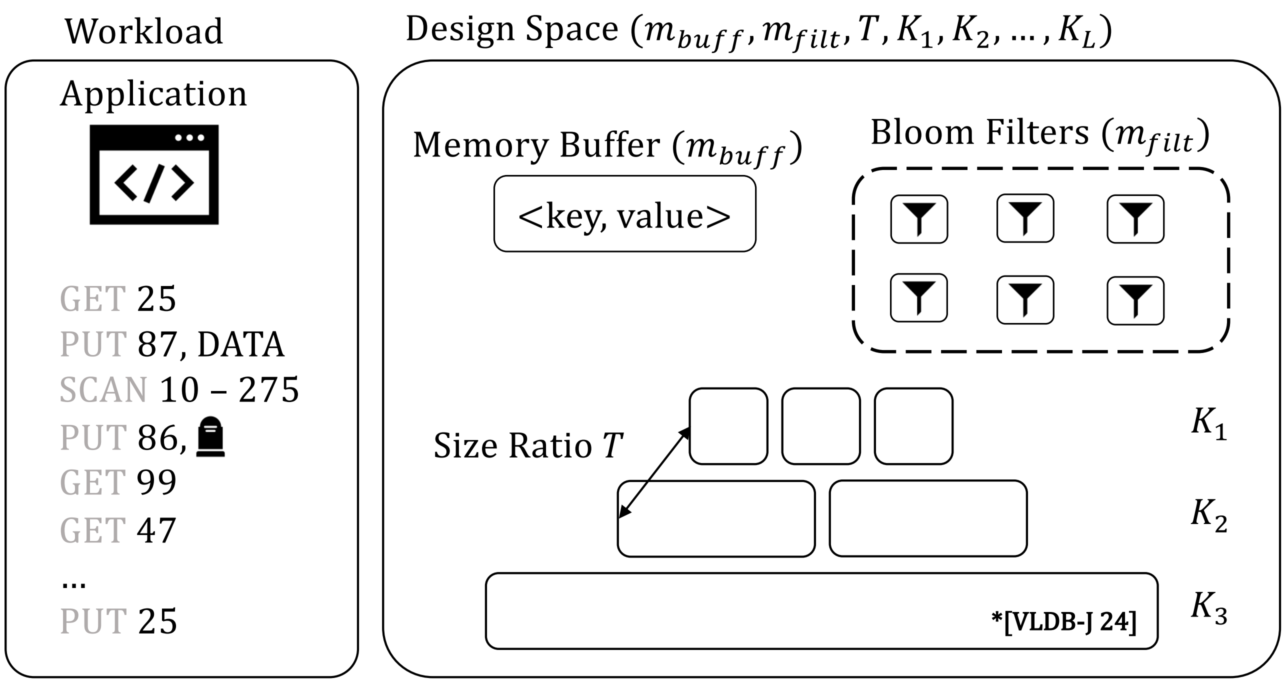
Compaction Design



Unifying Compaction Design Language



Navigating Flexibility



How do we decide on the best configuration for a specific application?

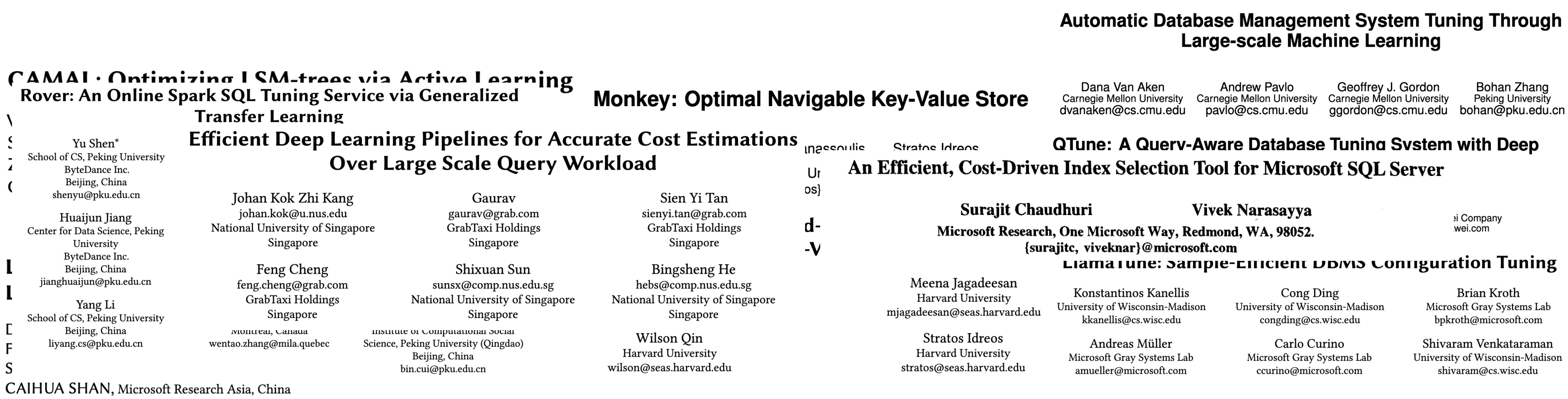
The Tuning Problem

w : Workload

φ : Design $\quad (m_{buff}, m_{filter}, T, K_1, \ldots, K_L)$

C : Cost

$$\varphi^* = argmin_{\varphi}\, C(w, \varphi)$$



The LSM Tuning Problem

w : Workload

φ : Design $\quad (m_{buff}, m_{filter}, T, K_1, \ldots, K_L)$

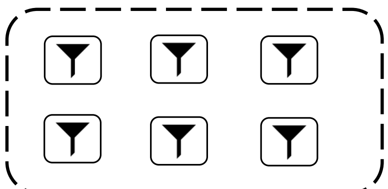
C : Cost

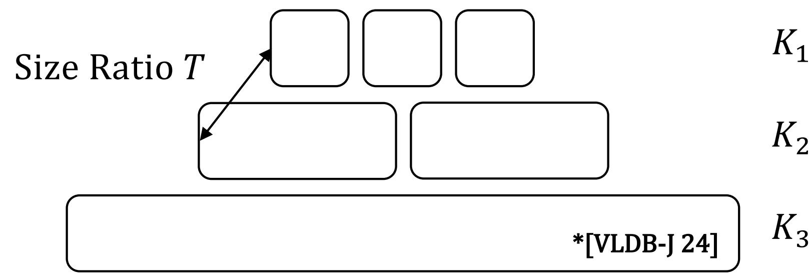
$$\varphi^* = argmin_{\varphi}\, C(w, \varphi)$$

Memory Buffer (m_{buff})

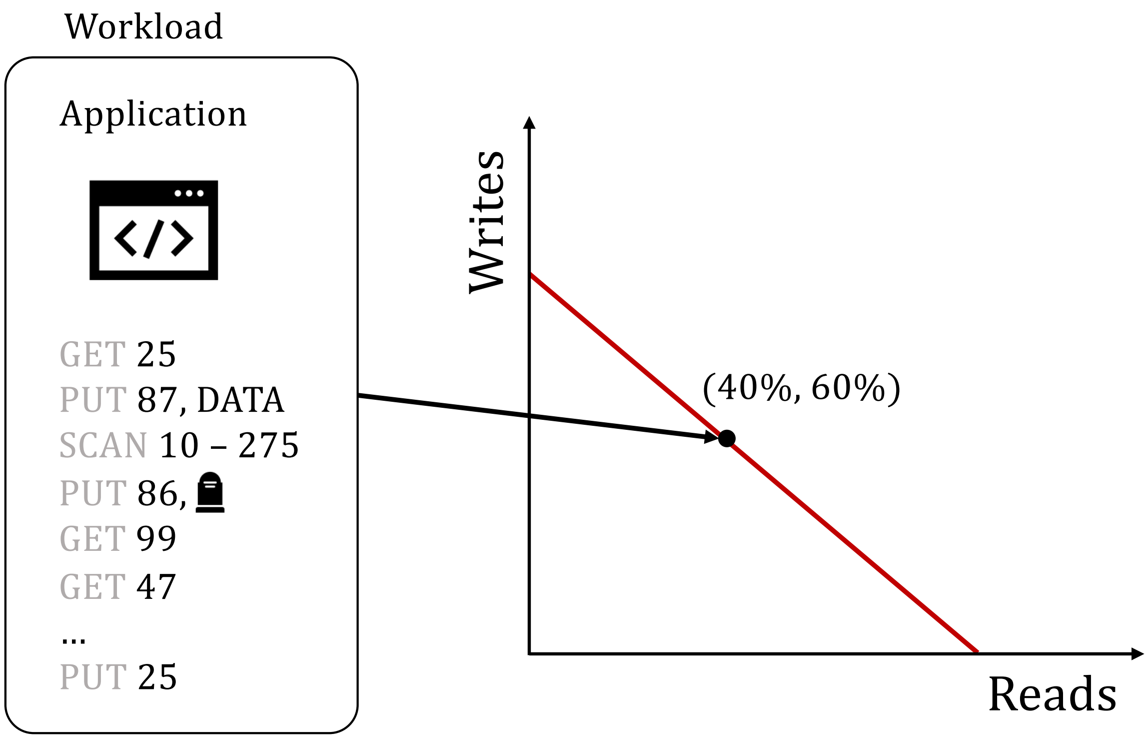
<key, value>

Bloom Filters (m_{filt})

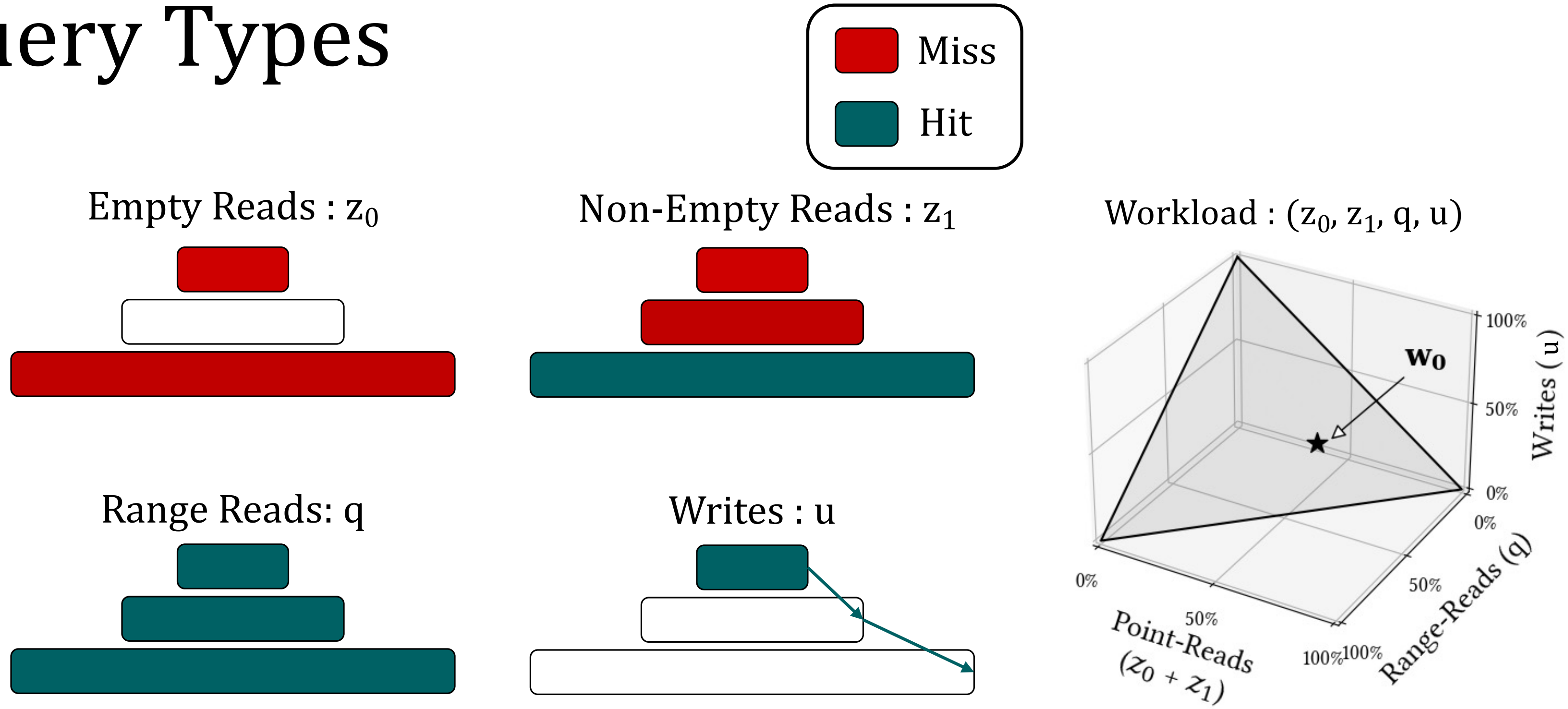




Workloads


Workload
Application
GET 25
PUT 87, DATA
SCAN 10 – 275
PUT 86,
GET 99
GET 47
...
PUT 25
Writes
(40%, 60%)
Reads

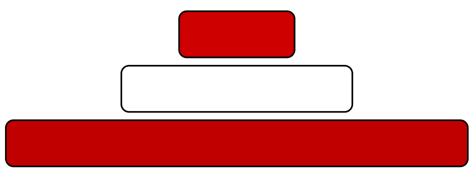
Query Types


Miss
Hit
Empty Reads : z_0
Non-Empty Reads : z_1
Workload : (z_0, z_1, q, u)
Range Reads: q
Writes : u
w_0
100%
50%
0%
Writes (u)
0%
50%
100%
Range-Reads (q)
0%
50%
100%
Point-Reads ($z_0 + z_1$)

Point Reads

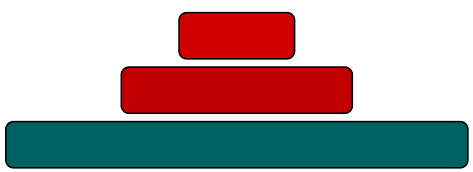
Additional Notation	
E	Num entries in 1 page
$L(T)$	Total levels
$f_i(T)$	BF false positive rate at level i
$N_f(T)$	Total keys if tree was full

Empty Reads : z_0



$$Z_0(\varphi) = \overbrace{\sum_{i=1}^{L(T)} K_i \cdot f_i(T)}^{\text{Sum of false positives}}$$

Non-Empty Reads : z_1



$$Z_1(\varphi) = \overbrace{\sum_{i=1}^{L(T)} \frac{(T-1)\cdot T^{i-1}}{N_f(T)} \cdot \frac{m_{buf}}{E}}^{\text{Probability query is satisfied at Level } i} \overbrace{\left(1 + \sum_{j=1}^{i-1} K_j \cdot f_j(T) + \frac{K_i - 1}{2} \cdot f_i(T)\right)}^{\text{False positives from levels above}}$$

Andy Huynh, Harshal A. Chaudhari, Evimaria Terzi, and Manos Athanassoulis. 2024.
Towards flexibility and robustness of LSM trees. The VLDB Journal (2024), 1–24.

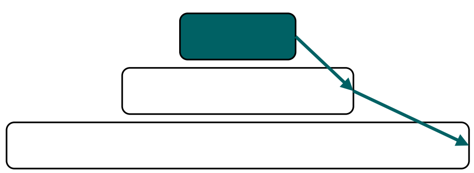
Range-Reads and Writes

Additional Notation	
B	Num. entries in buffer
$L(T)$	Total levels
S_{RQ}	Range query selectivity
f_{seq}	Sequential access cost

Range Reads: q

$$Q(\varphi) = f_{seq} \cdot \overbrace{S_{RQ}}^{\text{Sequential read based on selectivity}} \cdot \frac{N}{B} + \left.\sum_{i=1}^{L(T)} K_i \right\} \text{1 I/O per Seek per level}$$

Writes : u



$$U(\varphi) = f_{seq} \cdot \underbrace{\frac{1+f_a}{B}}_{\text{Writes only flush once buffer is full}} \cdot \overbrace{\sum_{i=1}^{L(T)} \frac{T-1+K_i}{2K_i}}^{\text{Average number of merges a write will participate in}}$$

Andy Huynh, Harshal A. Chaudhari, Evimaria Terzi, and Manos Athanassoulis. 2024. Towards flexibility and robustness of LSM trees. The VLDB Journal (2024), 1–24.

The LSM Tuning Problem

w : Workload (z_0, z_1, q, w)

φ : Design $(m_{buff}, m_{filter}, T, K_1, K_2, \ldots, K_L)$

C : Cost (I/O)

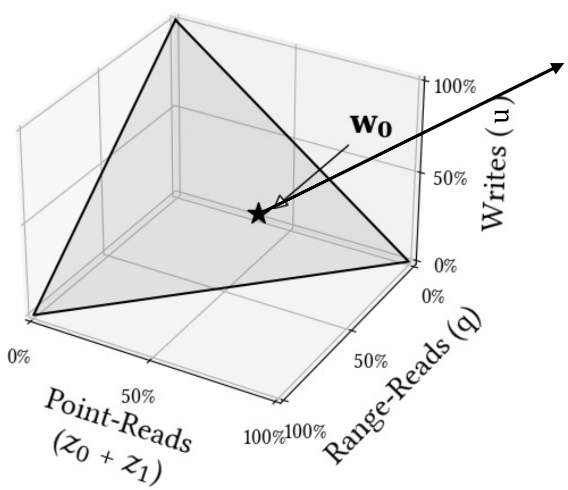
$$\varphi^* = argmin_{\varphi}\, C(w, \varphi)$$

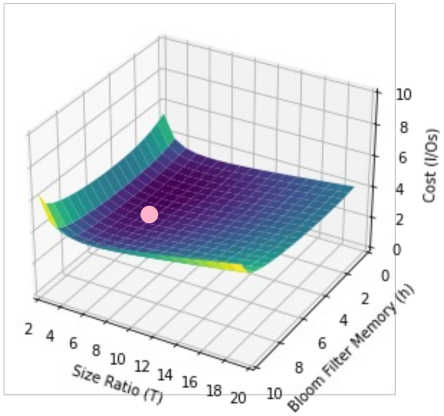
Cost function is a weighted sum over each operation type

$$C_S(w, \varphi) = z_0 \cdot Z_0(\varphi) + z_1 \cdot Z_1(\varphi) + q \cdot Q(\varphi) + u \cdot U(\varphi)$$

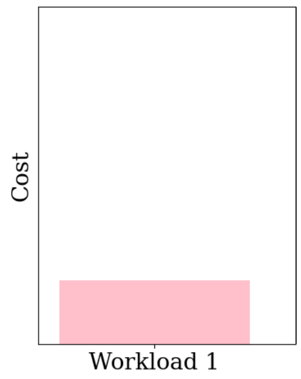
Tuning Problems

$\mathbf{w_0}$: Workload (z_0, z_1, q, u)





Best Configuration



Outline

A Primer on LSM Trees

Flexibility in Compaction Design **[VLDB-J 24]**

Modeling LSM Trees

Tuning LSM Trees

ENDURE: Finding Robust Tunings for LSM Trees **[VLDB 22]**

AXE: Learning to Tune LSM Trees By Task Decomposition

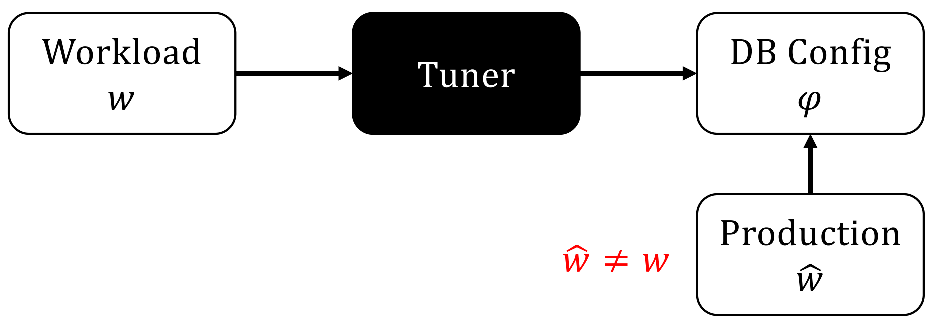
We Are Not ORACLEs

w : Workload (z_0, z_1, q, u)

φ : LSM Tree Design $(m_{buff}, m_{filter}, T, K_1, K_2, \ldots, K_L)$

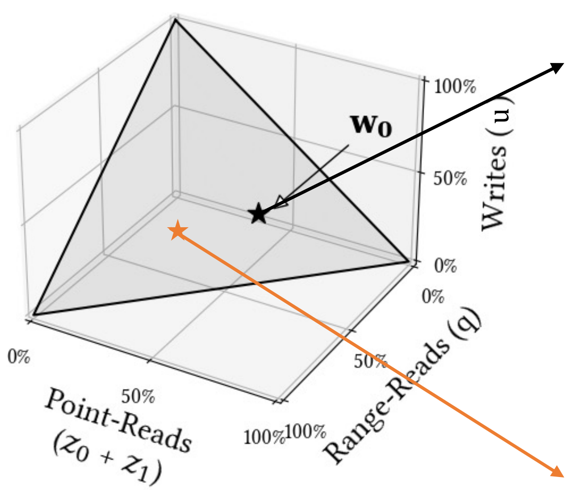
C : Cost (I/O)

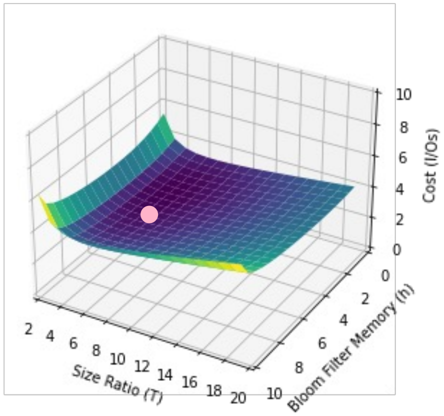
$$\varphi^* = argmin_{\varphi}\, C(w, \varphi)$$

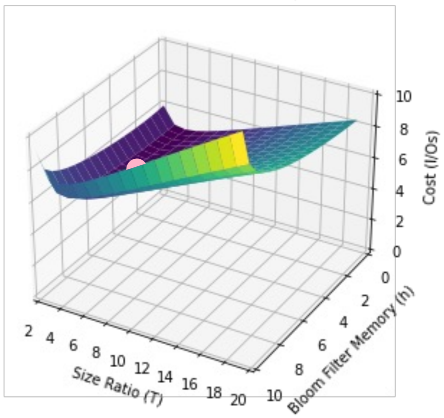


Tuning Problems

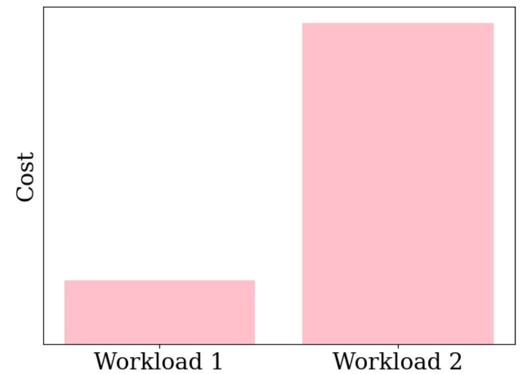
$\mathbf{w_0}$: Workload (z_0, z_1, q, u)







'Best' Configuration



Optimal tuning depends on workload

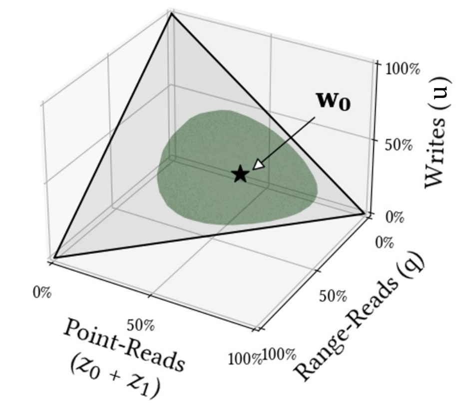
Workload uncertainty leads to **<u>sub-optimal</u>** tuning

The LSM-Tuning Problem

w : Workload (z_0, z_1, q, u)

φ : Design $(m_{buff}, m_{filter}, T, K_1, K_2, \dots, K_L)$

C : Cost (I/O)



$$\varphi^* = argmin_{\varphi}\, C(w, \varphi)$$

Nominal

U_w^{ρ}: Uncertainty Neighborhood of Workloads

ρ : Size of this neighborhood

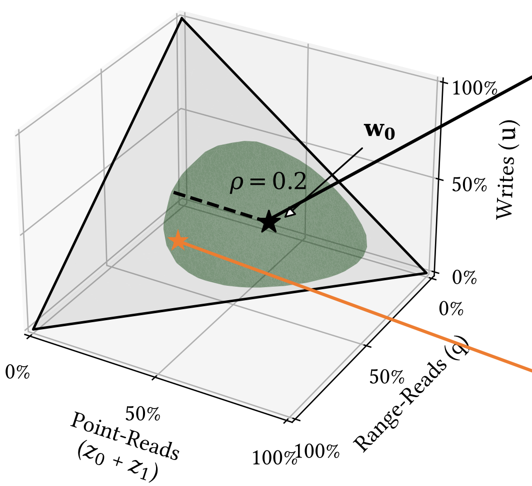
Robust

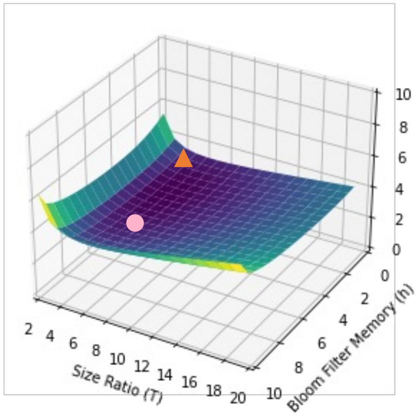
$$\varphi^* = \text{argmin}_{\varphi}\, C(\widehat{w}, \varphi)$$

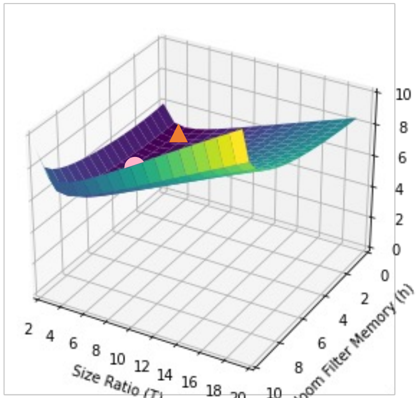
$$s.t., \qquad \widehat{w} \in U_w^{\rho}$$

Robust Tuning

w : Workload (z_0, z_1, q, u)



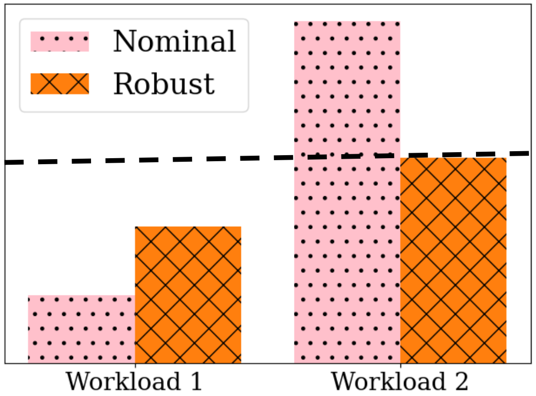




$$\varphi^* = \mathrm{argmin}_{\varphi}\ C(\hat{w}, \varphi)$$

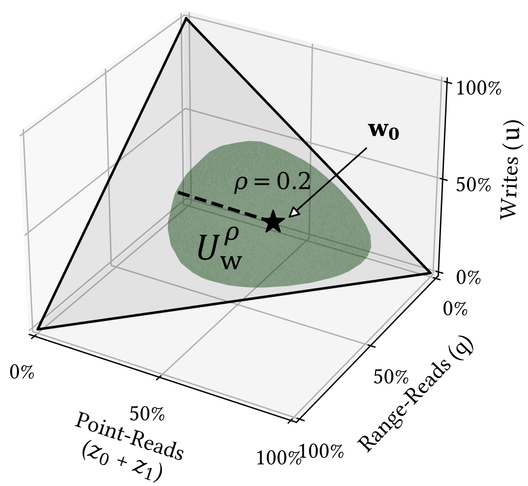
$$s.t., \quad \hat{w} \in U_w^{\rho}$$

- Optimal configuration for expected workload
- Robust configuration for the workload neighborhood



Uncertainty Neighborhood

Workload Characteristic



Neighborhood of workloads (ρ) via the KL-divergence

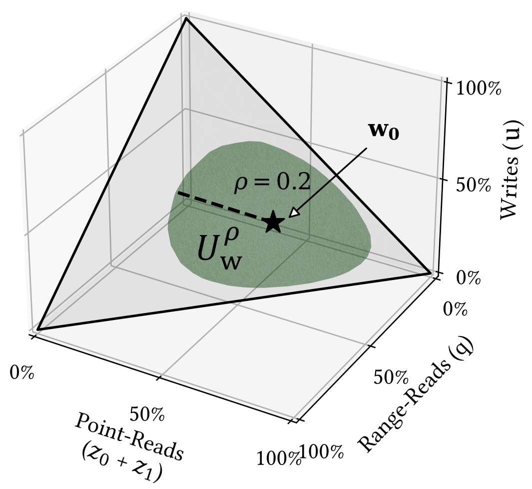
$$I_{KL}(\hat{w}, w) = \sum_{i=1}^{m} \hat{w}_i \cdot \log\left(\frac{\hat{w}_i}{w_i}\right)$$

U_{w}^{ρ}: Uncertainty Neighborhood of Workloads

ρ : Size of this neighborhood

Calculating Neighborhood Size

Workload Characteristic



Historical workloads
maximum/average uncertainty among workload pairings

User provided workload uncertainty

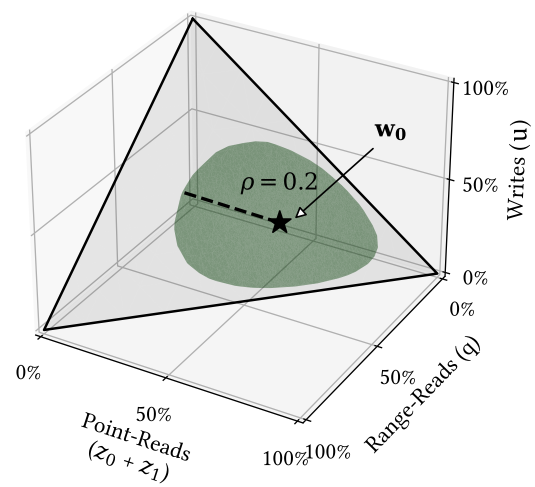
U_{w}^{ρ}: Uncertainty Neighborhood of Workloads
ρ : Size of this neighborhood

Solving Robust Problem

Iterating over every possible workload is expensive

$$\varphi_R = \arg\min_{\varphi} C_S(\hat{w}, \varphi)$$

$$s.t., \ \ \hat{w} \in \mathcal{U}_w$$



Solving Robust Problem

Iterating over every possible workload is expensive

$$\varphi_R = \arg\min_{\varphi} C_S(\hat{w}, \varphi)$$
$$s.t., \quad \hat{w} \in \mathcal{U}_w$$

$$\Downarrow$$

$$\min_{\varphi} \max_{\hat{w} \in \mathcal{U}_w} C_S(\hat{w}, \varphi)$$

$$\Downarrow$$

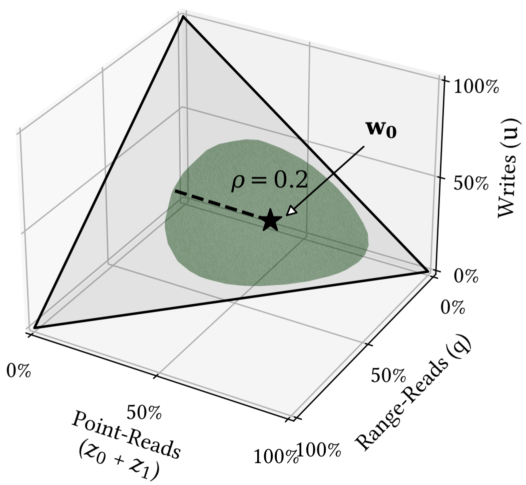
Formulation of the dual problem to reduce to a feasible problem‡

$$\min_{\varphi, \lambda \geq 0, \eta} \left\{ \eta + \rho\lambda + \lambda \sum_{i=1}^{m} w_i \phi_{KL}^{*} \left(\frac{c_i(\varphi) - \eta}{\lambda} \right) \right\}$$

‡Aharon Ben-Tal, Dick den Hertog, Anja De Waegenaere, Bertrand Melenberg, and Gijs Rennen. 2013. Robust Solutions of Optimization Problems Affected by Uncertain Probabilities.

ENDURE Pipeline

Workload Characteristic



System Information

Page Size

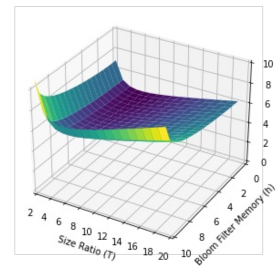
Memory Budget

ENDURE

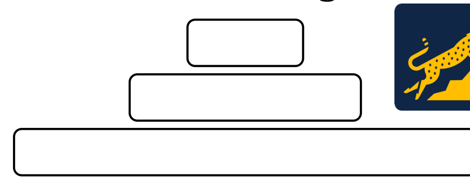
Solves the

Robust Problem

Expected performance



RocksDB Configuration

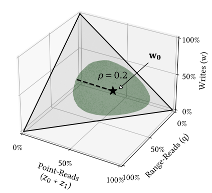


Testing Suite

ENDURE in Python, implemented in tandem with RocksDB

Uncertainty benchmark

- 15 expected workloads
- 10K randomly sampled workloads as a test-set



Normalized delta throughput

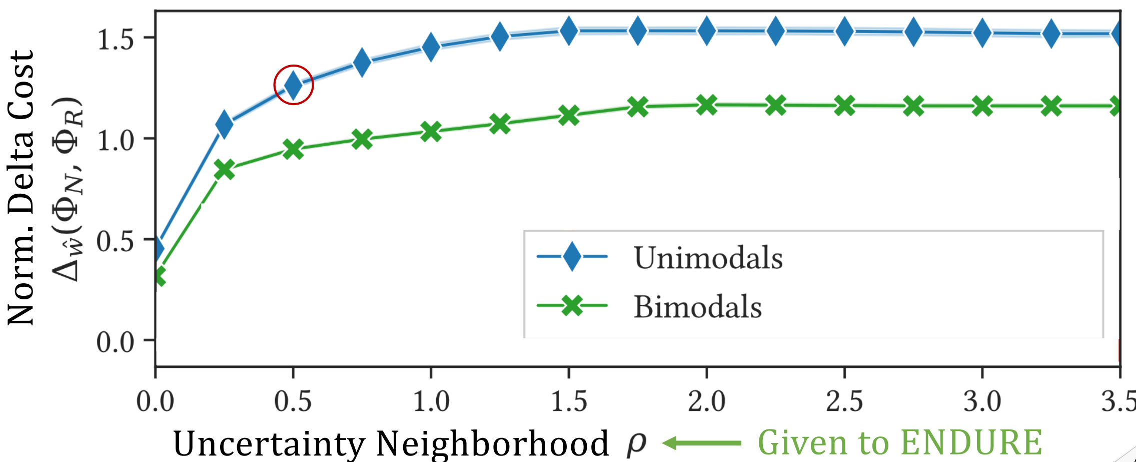
$$\Delta_{\mathbf{w}}(\Phi_1, \Phi_2) = \frac{1/C(\mathbf{w},\Phi_2) - 1/C(\mathbf{w},\Phi_1)}{1/C(\mathbf{w},\Phi_1)}$$

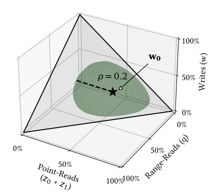
Nominal vs Robust: > 0 is better

1 means 2x speedup

Index	(z_0, z_1, q, u)				Type
0	25%	25%	25%	25%	Uniform
1	97%	1%	1%	1%	Unimodal
2	1%	97%	1%	1%	
3	1%	1%	97%	1%	
4	1%	1%	1%	97%	
5	49%	49%	1%	1%	Bimodal
6	49%	1%	49%	1%	
7	49%	1%	1%	49%	
8	1%	49%	49%	1%	
9	1%	49%	1%	49%	
10	1%	1%	49%	49%	
11	33%	33%	33%	1%	Trimodal
12	33%	33%	1%	33%	
13	33%	1%	33%	33%	
14	1%	33%	33%	33%	

Impact of Workload Type

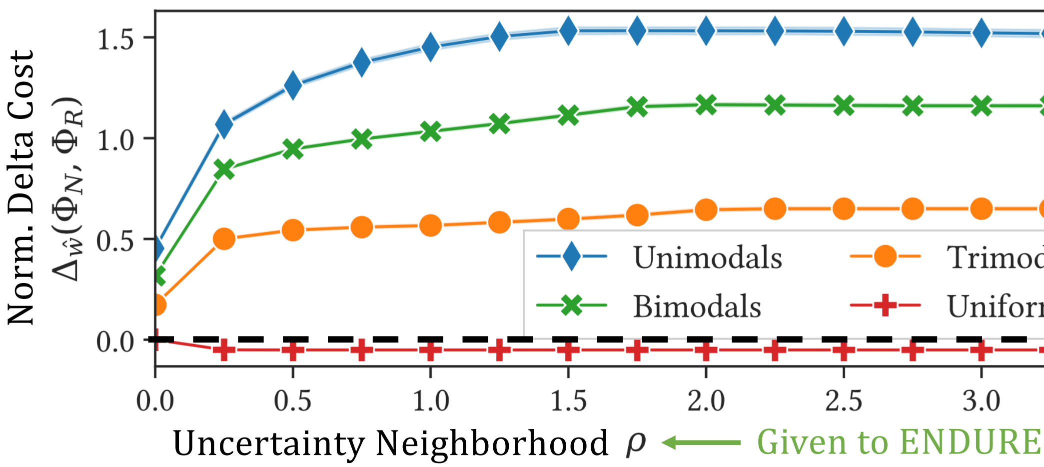




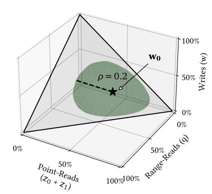
Index	(z_0, z_1, q, u)				Type
0	25%	25%	25%	25%	**Uniform**
1	97%	1%	1%	1%	**Unimodal**
2	1%	97%	1%	1%	
3	1%	1%	97%	1%	
4	1%	1%	1%	97%	
5	49%	49%	1%	1%	**Bimodal**
6	49%	1%	49%	1%	
7	49%	1%	1%	49%	
8	1%	49%	49%	1%	
9	1%	49%	1%	49%	
10	1%	1%	49%	49%	
11	33%	33%	33%	1%	**Trimodal**
12	33%	33%	1%	33%	
13	33%	1%	33%	33%	
14	1%	33%	33%	33%	

Unbalanced workloads result in overfitted nominal tunings

Impact of Workload Type



Index	(z_0, z_1, q, u)				Type
0	25%	25%	25%	25%	**Uniform**
1	97%	1%	1%	1%	**Unimodal**
2	1%	97%	1%	1%	
3	1%	1%	97%	1%	
4	1%	1%	1%	97%	
5	49%	49%	1%	1%	**Bimodal**
6	49%	1%	49%	1%	
7	49%	1%	1%	49%	
8	1%	49%	49%	1%	
9	1%	49%	1%	49%	
10	1%	1%	49%	49%	
11	33%	33%	33%	1%	**Trimodal**
12	33%	33%	1%	33%	
13	33%	1%	33%	33%	
14	1%	33%	33%	33%	

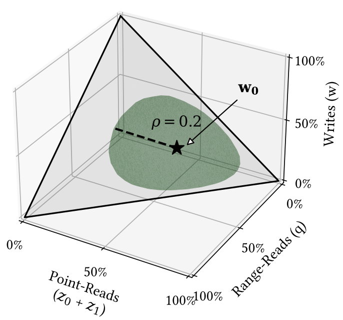


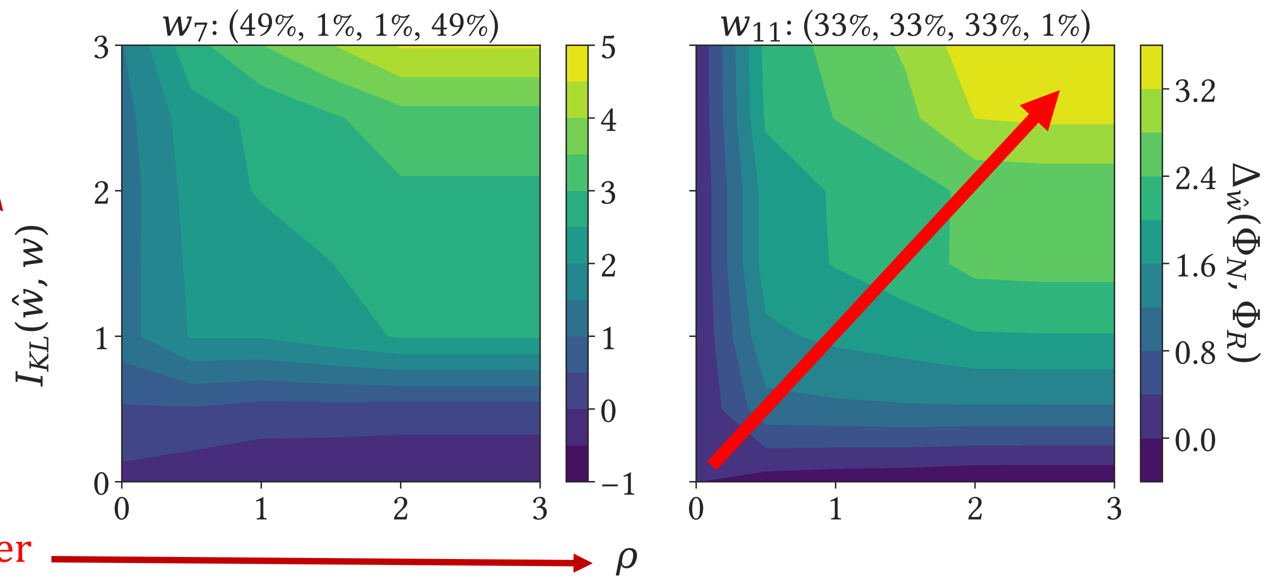
Unbalanced workloads result in overfitted nominal tunings

Tuning with uncertainty ($\rho > 0.5$) provides benefits

Relationship of Expected and Observed ρ

Observed ρ: distance from executed workload to expected workload



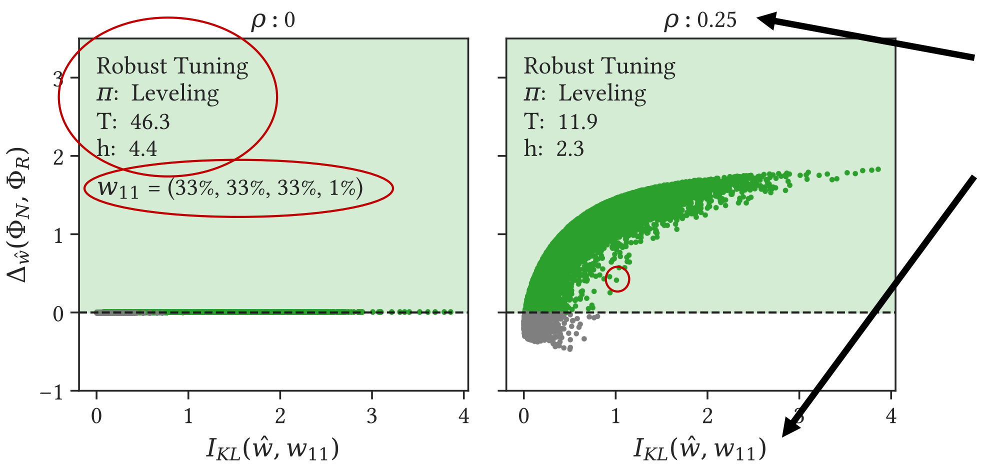


Expected ρ: workload given to tuner

Highest throughput when observed and expected ρ **match**

Lowest throughput when ρ is **mismatched**

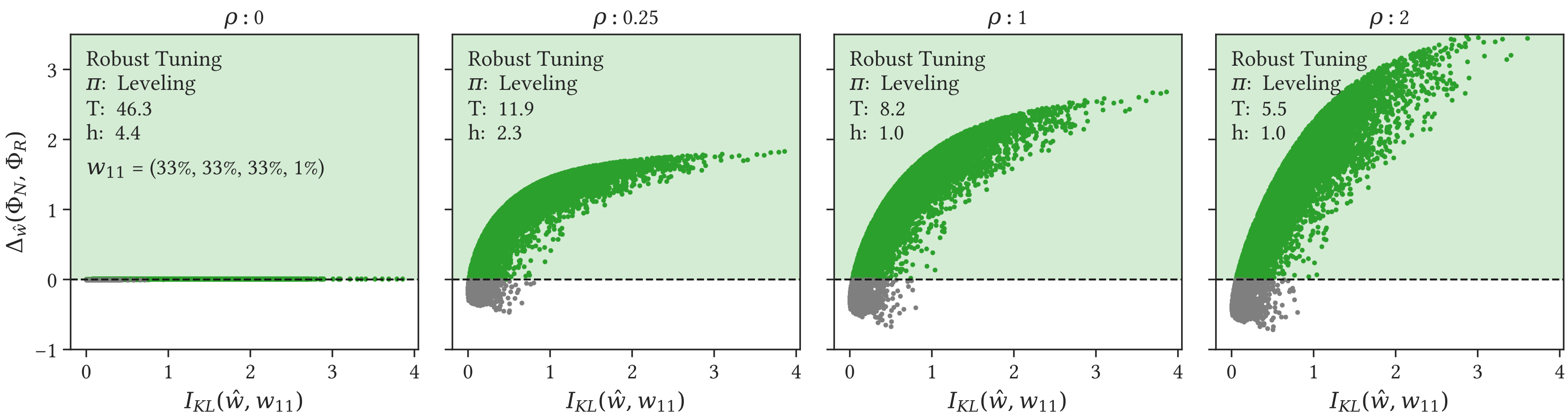
Impact of Observed vs Expected ρ



Expected ρ: given to tuner

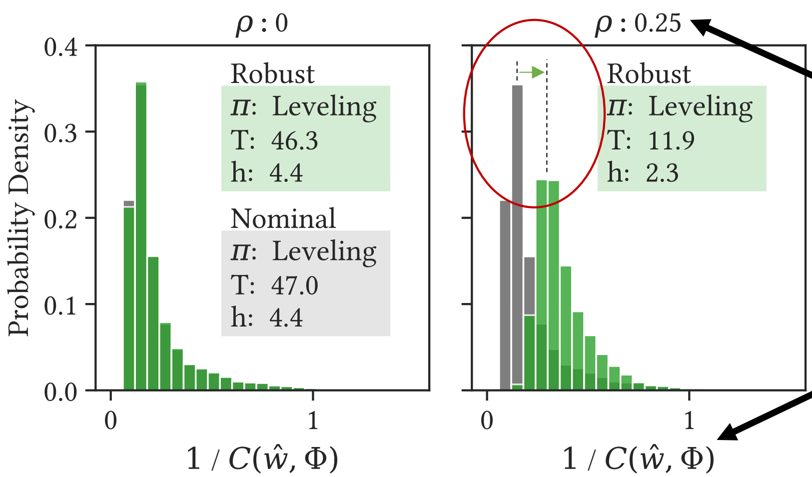
Observed ρ: distance from executed workload to expected workload

Impact of Observed vs Expected ρ



- Higher expected ρ accounts for more uncertainty,
- Potential speed up of 4x
- Higher expected ρ → anticipates writes → shallow tree

ρ and Performance Gain Distribution

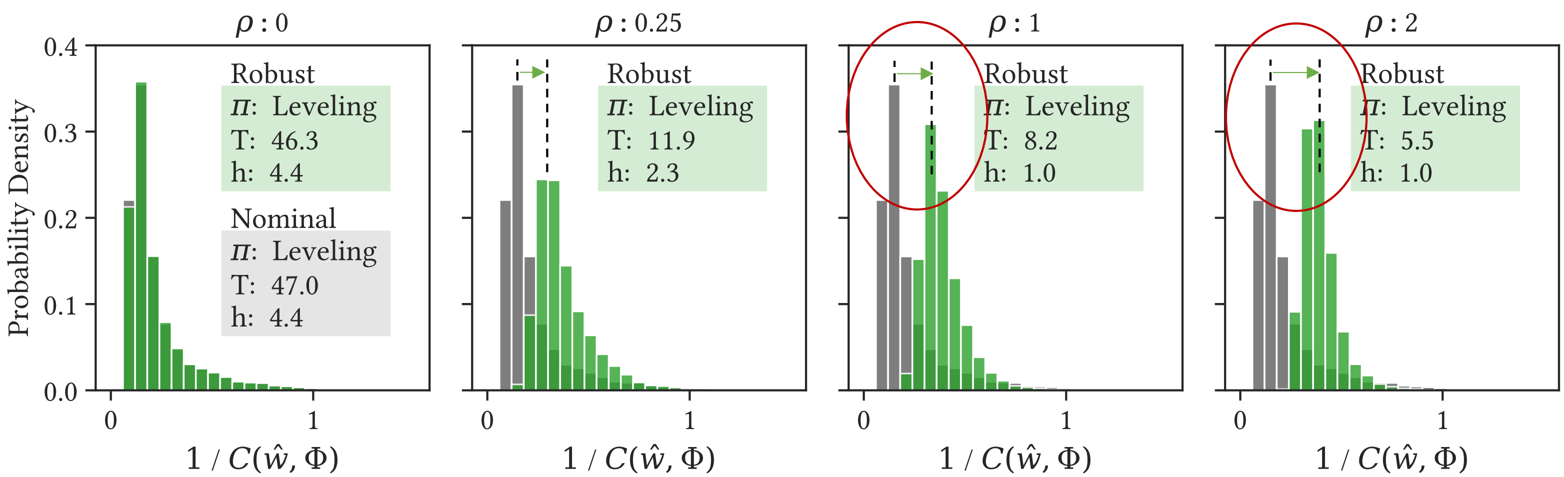


Expected ρ: given to tuner

Throughput

$$w_{11} = (33\% , 33\%, 33\%, 1\%)$$

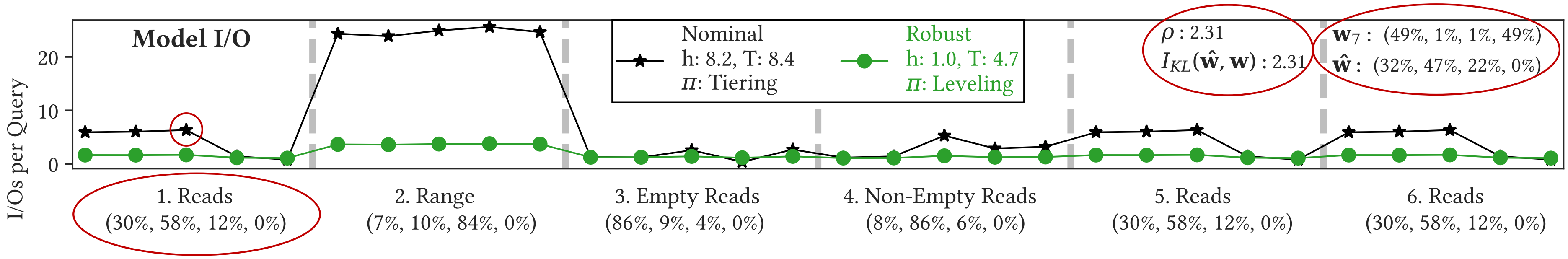
ρ and Performance Gain Distribution



$w_{11} = (33\%, 33\%, 33\%, 1\%)$

Peak of the distribution moves towards higher throughput as we consider higher uncertainty

Workload Sequence on RocksDB

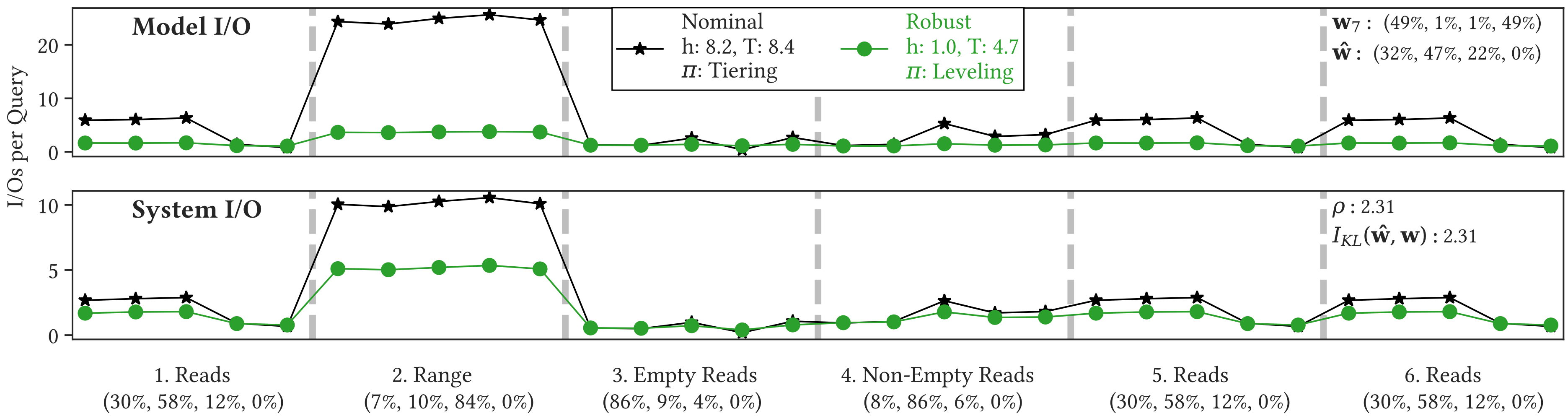


RocksDB instance setup with 10 million unique key-value pairs of size 1KB

Each observation period is 200K queries, with 5 observations per session 6 million queries to the DB

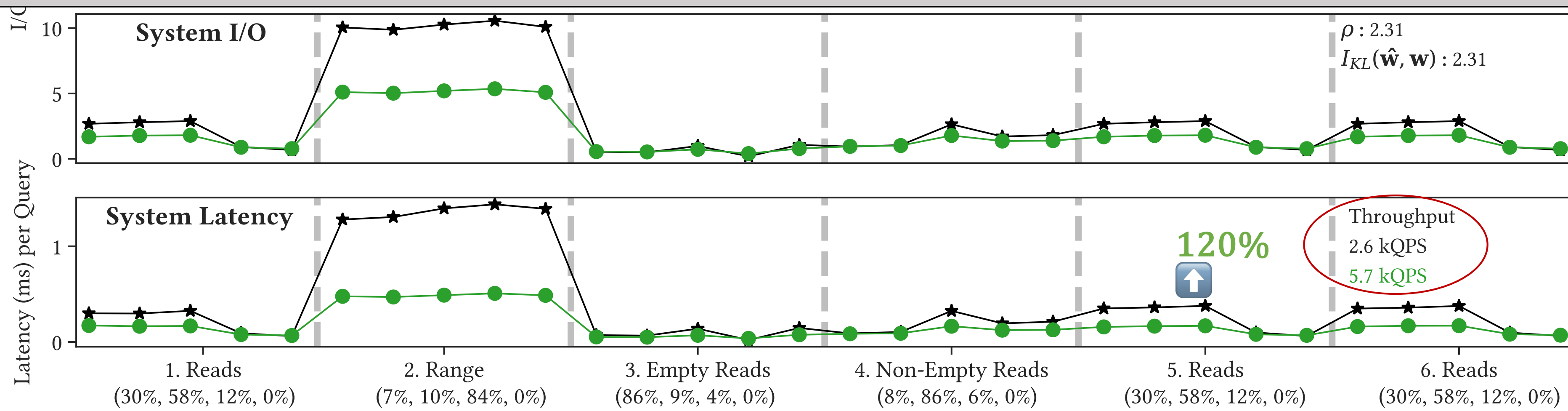
Writes are unique, range queries average 1-2 pages per level

Workload Sequence



Workload Sequence

Small subset of results! Take a look at the paper for a more detailed analysis



Outline

A Primer on LSM Trees

Flexibility in Compaction Design **[VLDB-J 24]**

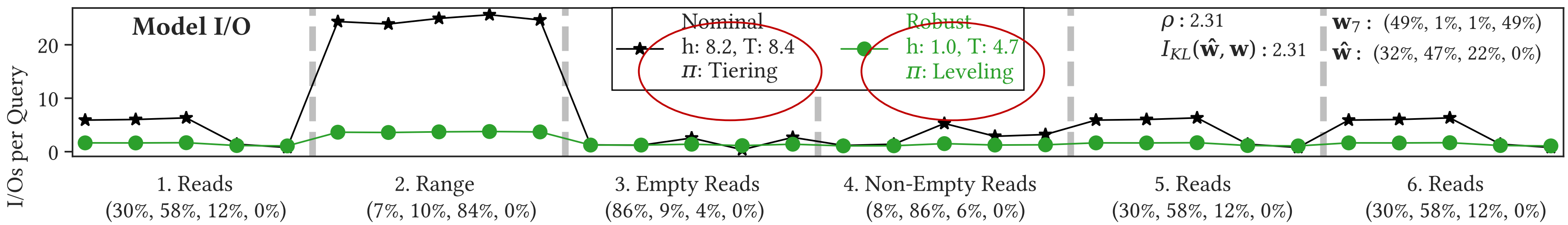
Modeling LSM Trees

Tuning LSM Trees

ENDURE: Finding Robust Tunings for LSM Trees **[VLDB 22]**

AXE: Learning to Tune LSM Trees By Task Decomposition

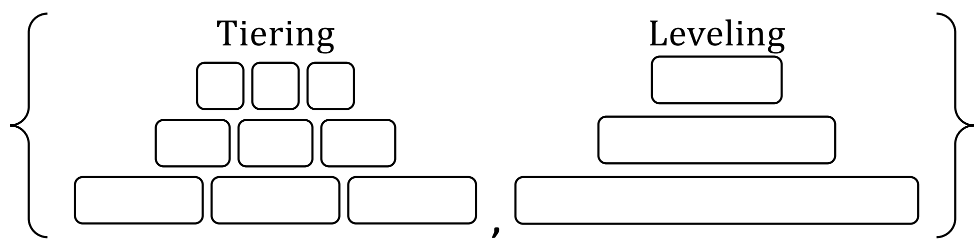
Limitations in ENDURE



Optimizers tends to scale poorly as the decision space grows, hence, we limited the decision to Tiering or Leveling

Cost model took **a lot** of man hours

$\pi \in$



Model

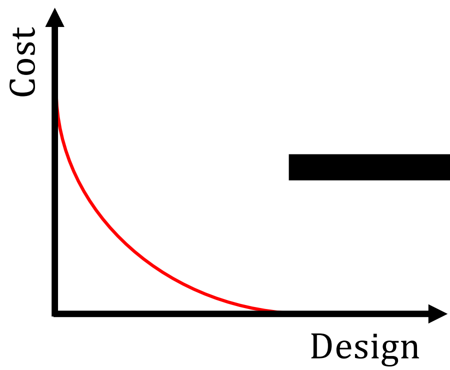
$$Z_0(\varphi) = \sum_{i=1}^{L(T)} K_i \cdot f_i(T)$$

$$U(\varphi) = f_{seq} \cdot \frac{1+f_a}{B} \cdot \sum_{i=1}^{L(T)} \frac{T-1+K_i}{2K_i}$$

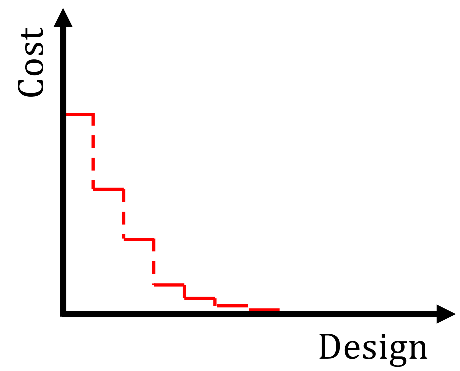
Optimizer

Complex Decision Space

Knob	Value
K_1	1.0
K_2	4.0
K_3	2.0
K_4	3.0
T	6.0



Knob	1	2	3	4	5	6
K_1	1	0	0	0	0	0
K_2	0	0	0	1	0	0
K_3	0	1	0	0	0	0
K_4	0	0	1	0	0	0
T	0	0	0	0	0	1



We like to think of knobs as continuous values

Reality is they take on a set number of values

Solution: Pretend we're in a continuous space then round off

Linear Program Solver

Knob	Value
K_1	1.27
K_2	4.46

Round Off

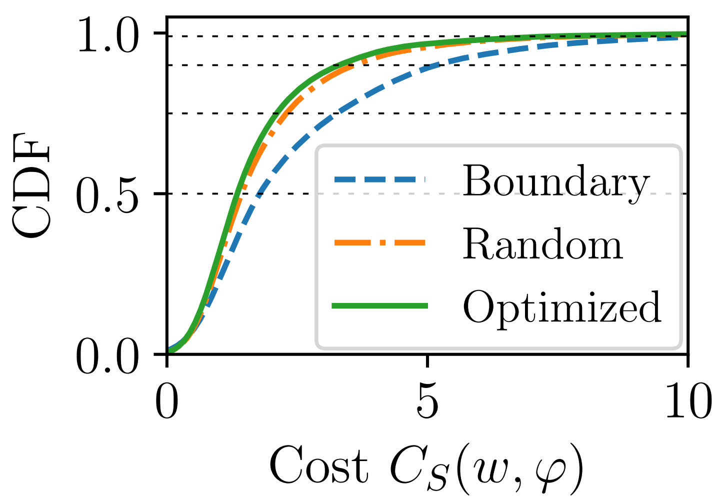
Knob	Value
K_1	1
K_2	4

Problem: Errors propagate fast

Sensitive Optimizers

1. Build optimizer with different initial starting conditions
2. For each optimizer, test recommended designs for various workloads

<u>No</u> workload drift. ($\boldsymbol{w_0} = \hat{\boldsymbol{w}}$)



Median Slowdown from Optimal

Percentile	Boundary	Random
P50%	1.0x	1.0x
P75%	1.7x	1.1x
P90%	2.4x	1.2x
P99%	4.1x	1.7x
P99.9%	6.2x	3.3x

Tackling Discrete Optimization

Machine learning strategies proven to work well in discrete optimization tasks

Adapt techniques for systems tuning

NEURAL INFORMATION PROCESSING SYSTEMS

Machine Learning for Combinatorial Optimization COMPETITION 2021

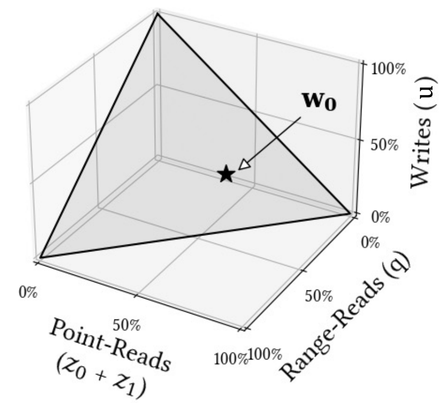
ECOLE Extensible Combinatorial Optimization Learning Environments

The LSM Tuning Problem

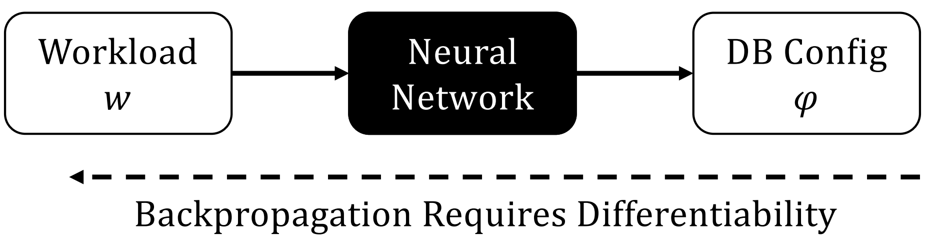
w : Workload (z_0, z_1, q, w)

φ : Design $(m_{buff}, m_{filter}, T, K_1, K_2, \ldots, K_L)$

C : Cost (I/O)



$$\varphi^* = argmin_{\varphi}\, C(w, \varphi)$$

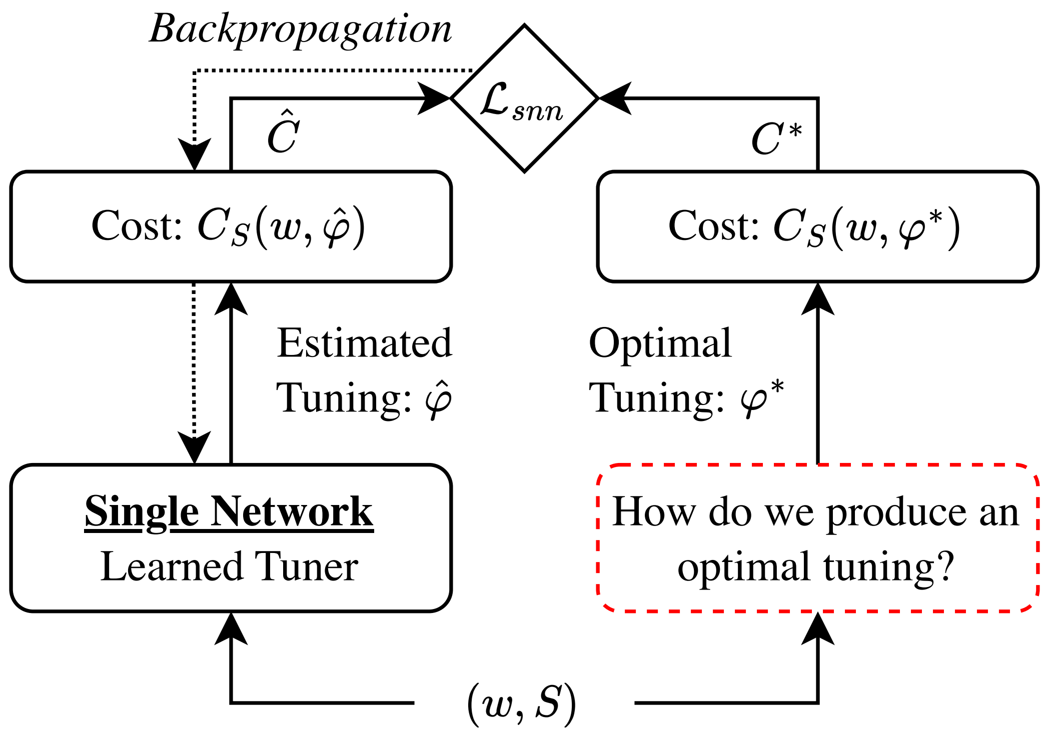


Training a Tuner

$$\mathcal{L}_{snn} = \mathop{\mathbb{E}}_{(w,S)\sim D}\left[C_S(w,\hat{\varphi}) - C^*\right]$$

How would we train a tuner?

Single neural network for our task is infeasible

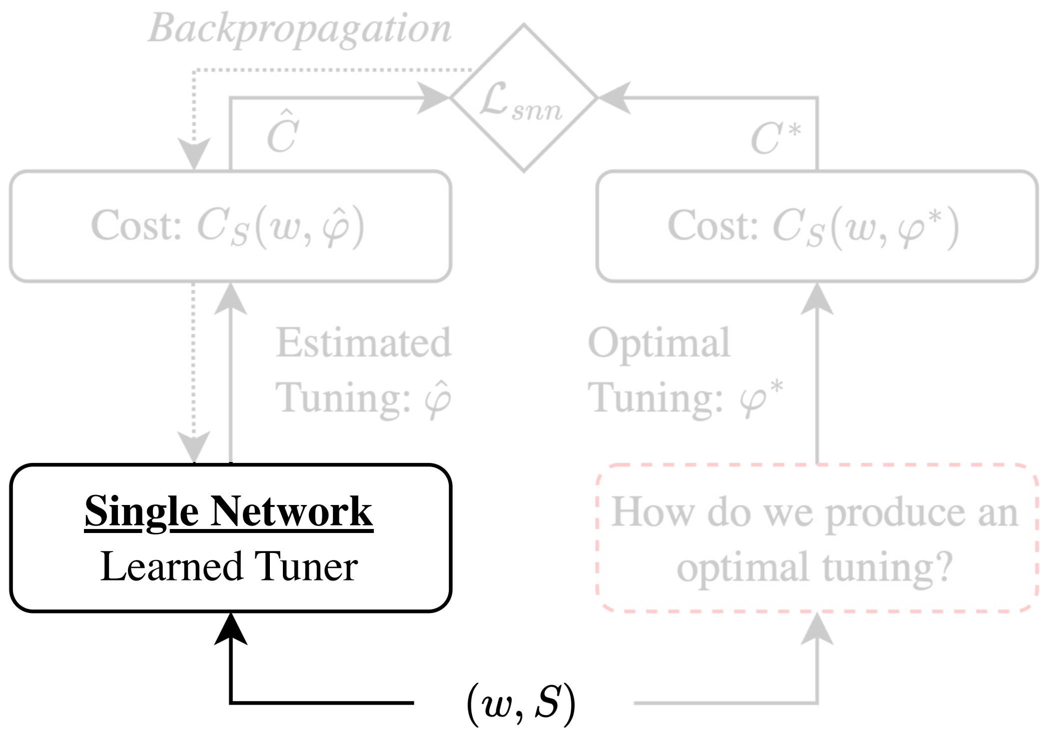


Training a Tuner

$$\mathcal{L}_{snn} = \mathop{\mathbb{E}}_{(w,S)\sim D}\left[C_S(w,\hat{\varphi}) - C^*\right]$$

How would we train a tuner?

Single neural network for our task is infeasible

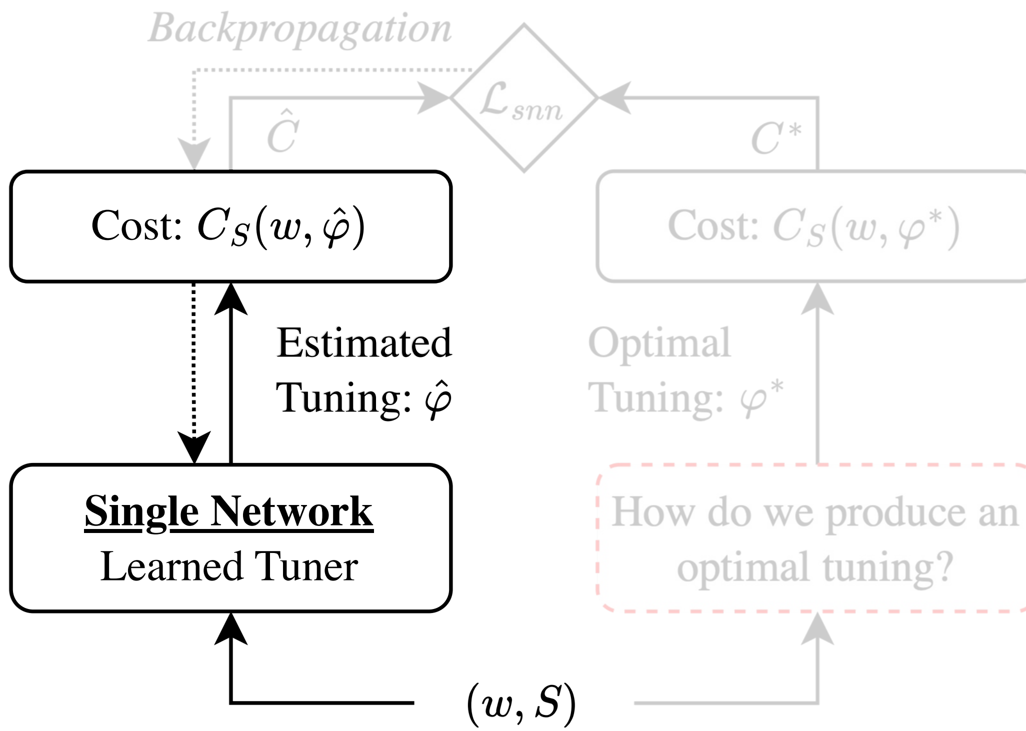


Training a Tuner

How would we train a tuner?

Single neural network for our task is infeasible

$$\mathcal{L}_{snn} = \mathop{\mathbb{E}}_{(w,S)\sim D}\left[C_S(w,\hat{\varphi}) - C^*\right]$$

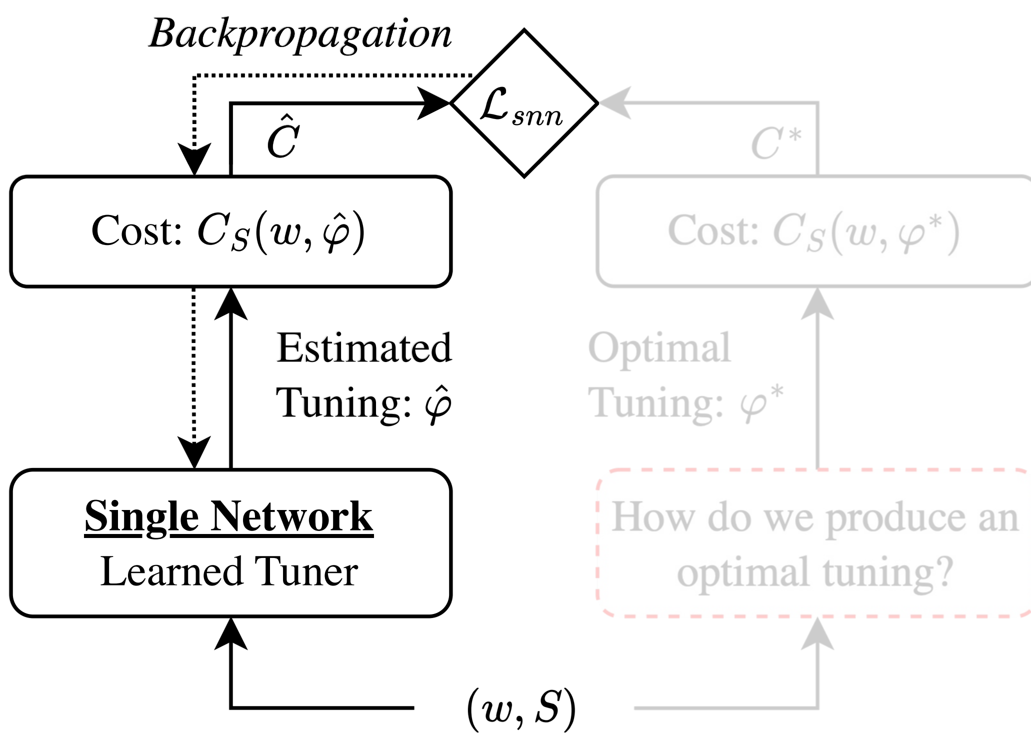


Training a Tuner

$$\mathcal{L}_{snn} = \mathop{\mathbb{E}}_{(w,S)\sim D}\left[C_S(w,\hat{\varphi}) - C^*\right]$$

How would we train a tuner?

Single neural network for our task is infeasible

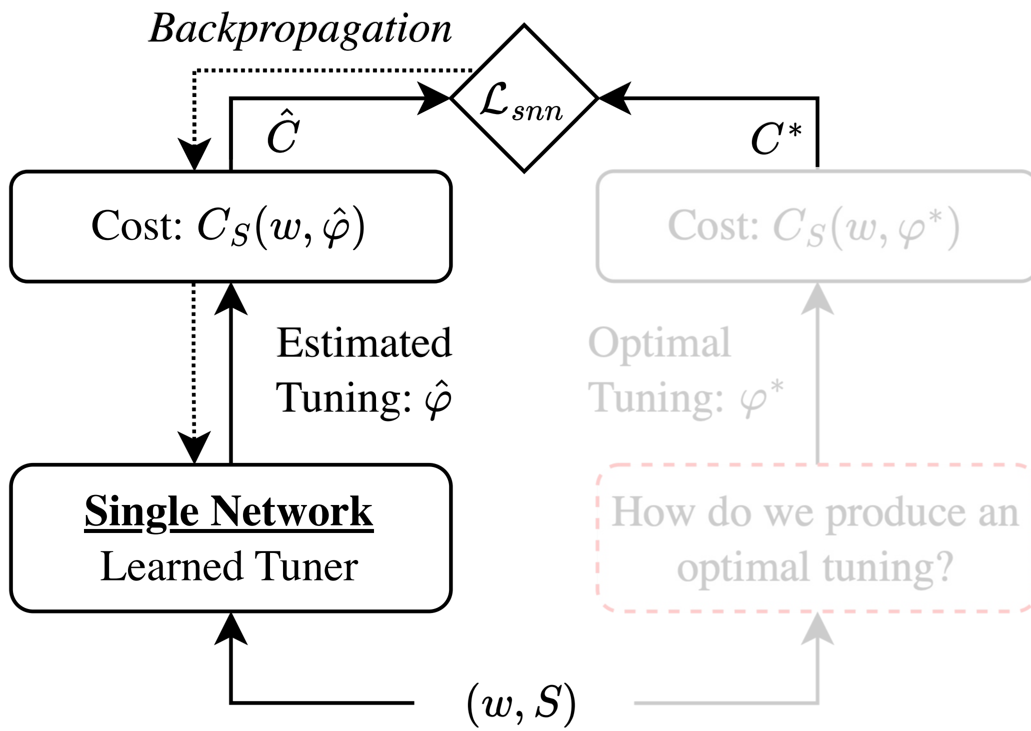


Training a Tuner

$$\mathcal{L}_{snn} = \mathop{\mathbb{E}}_{(w,S)\sim D}\left[C_S(w,\hat{\varphi}) - C^*\right]$$

How would we train a tuner?

Single neural network for our task is infeasible

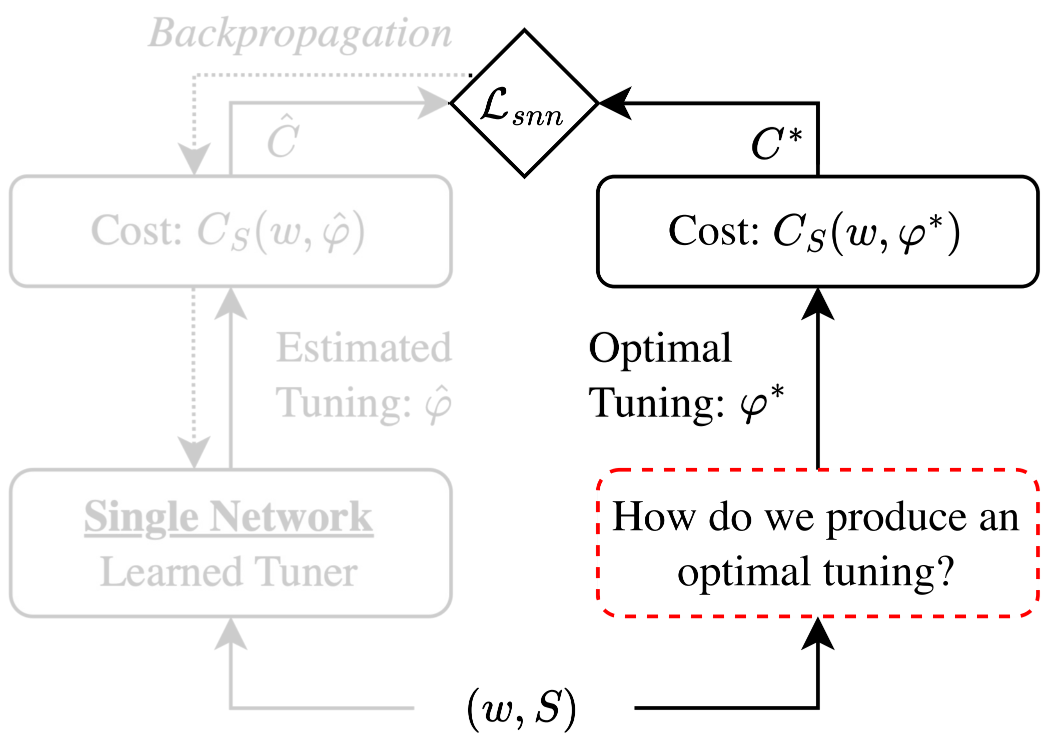


Training a Tuner

$$\mathcal{L}_{snn} = \mathop{\mathbb{E}}_{(w,S)\sim D} \left[C_S(w,\hat{\varphi}) - C^*\right]$$

How would we train a tuner?

Single neural network for our task is infeasible



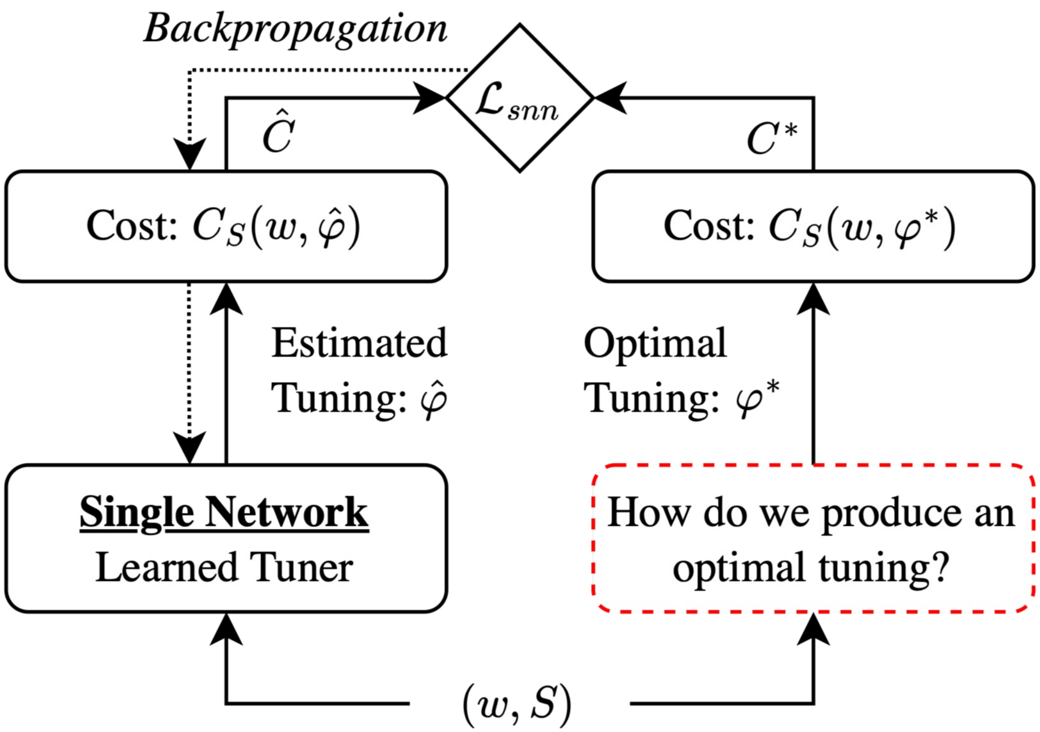
Training a Tuner

How would we train a tuner?

Single neural network for our task is infeasible

We would need to have already solved the problem to obtain the correct data

$$\mathcal{L}_{snn} = \mathop{\mathbb{E}}_{(w,S)\sim D}\left[C_S(w,\hat{\varphi}) - C^*\right]$$

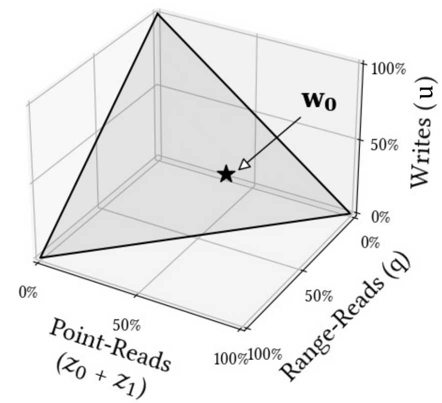


The LSM Tuning Problem

w : Workload (z_0, z_1, q, w)

φ : Design $(m_{buff}, m_{filter}, T, K_1, K_2, \ldots, K_L)$

C : Cost (I/O)



$$\varphi^* = \boxed{argmin_{\varphi}} \boxed{C(w, \varphi)}$$

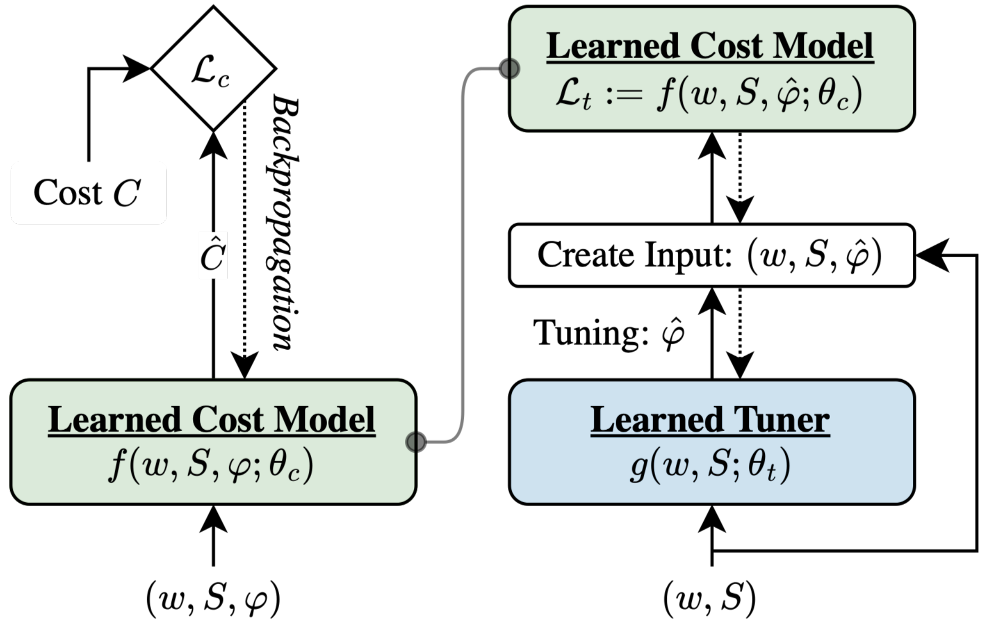
What is the **optimal tuning** for a specific **workload?**

What is the **cost** of a **workload** executed on a **tuning**?

Task Decomposition of the Tuning Problem

We approach each subtask with a neural network

$$\mathcal{L}_c = \mathop{\mathbb{E}}_{(w,S,\varphi,C)\sim D}\left[||\hat{C} - C||\right]$$



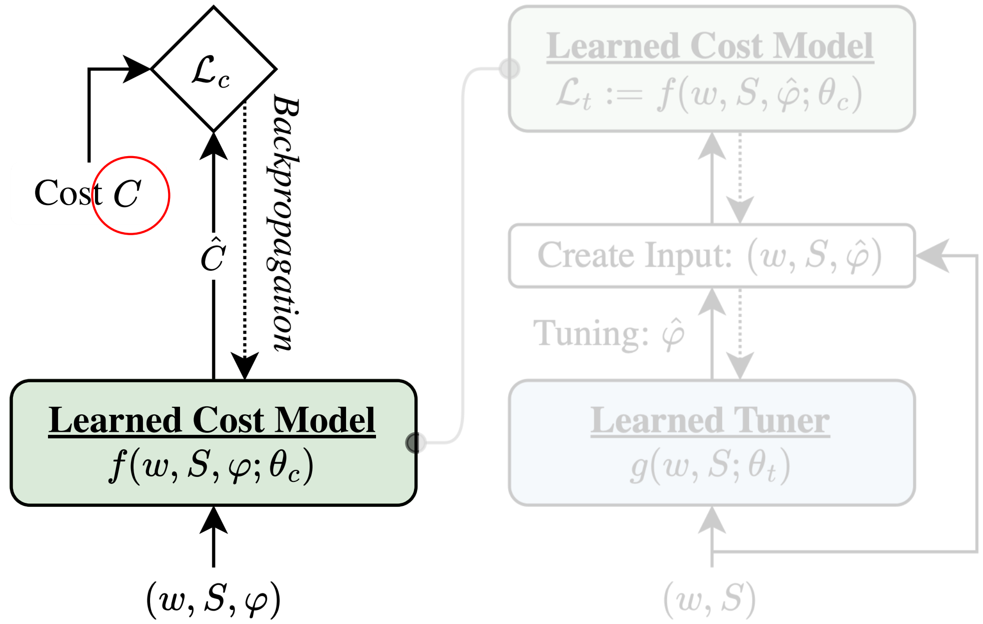
(a) Learned Cost Model **(b)** Learned Tuner

Task Decomposition of the Tuning Problem

We approach each subtask with a neural network

$$\mathcal{L}_c = \mathop{\mathbb{E}}_{(w,S,\varphi,C)\sim D}\left[||\hat{C} - C||\right]$$

What is the **cost** of a **workload** executed on a **tuning**?



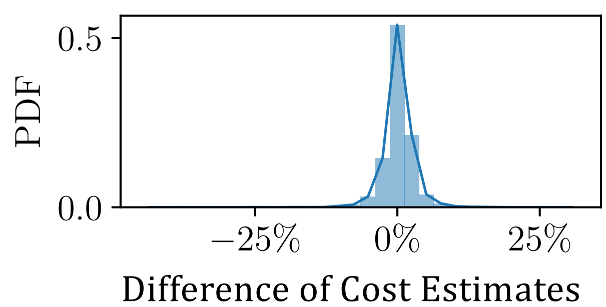
(a) Learned Cost Model

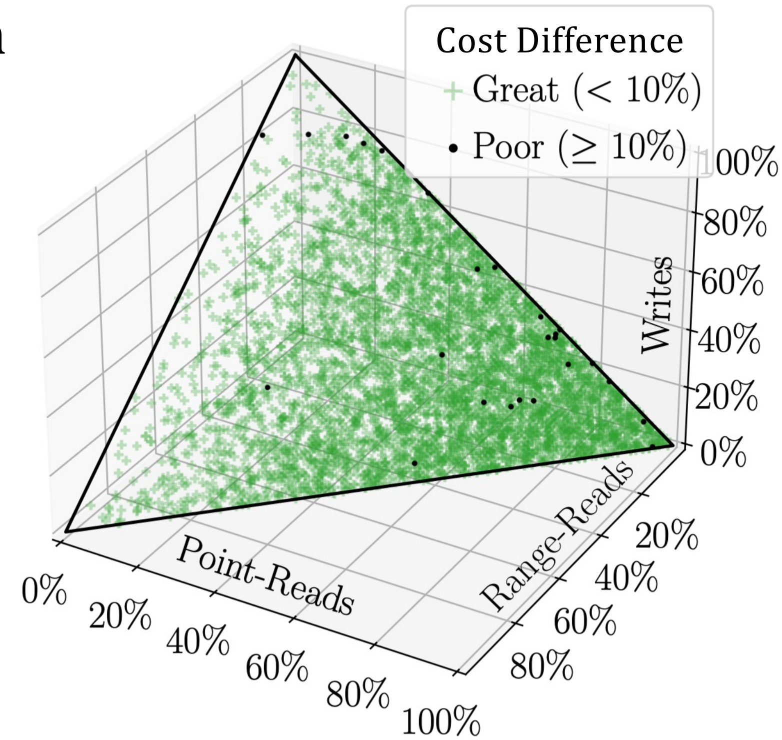
(b) Learned Tuner

Learned Cost Model Performance

Randomly sampled workloads with a fixed environment and tuning

99% of points evaluated are within 10% of the true cost





Task Decomposition of the Tuning Problem

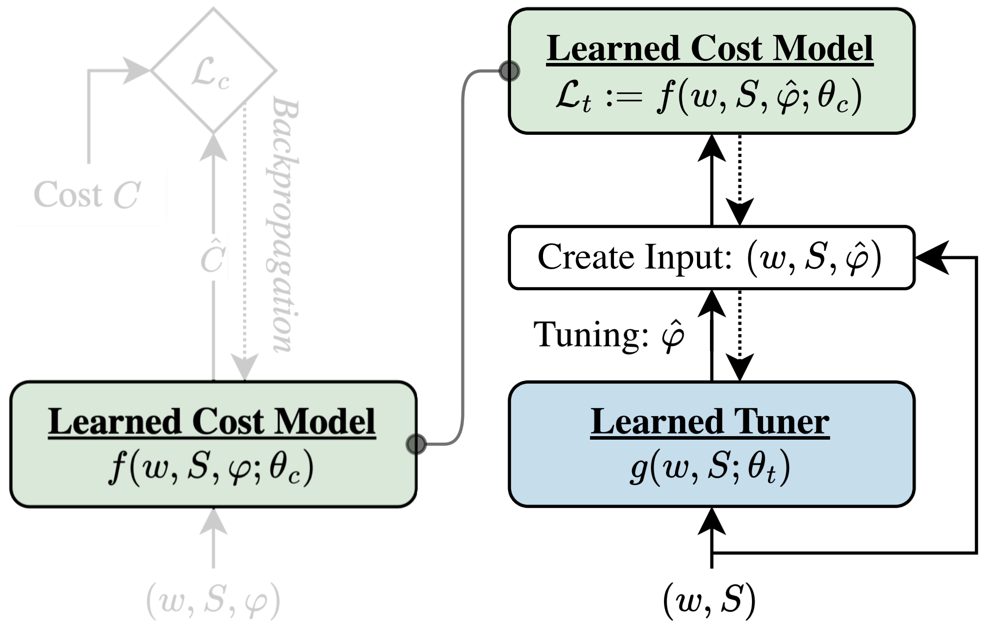
What is the **optimal tuning** for a specific **workload?**

$$\varphi^* = \arg\min_{\varphi} C_S(w, \varphi)$$

$$\mathcal{L}_t = \mathop{\mathbb{E}}_{(w,S)\sim\mathcal{D}} [C_S(w, \hat{\varphi})]$$

$$\mathcal{L}_t = \mathop{\mathbb{E}}_{(w,S)\sim\mathcal{D}} [f(w, S, \hat{\varphi})]$$

Learned cost model is used as the loss for the tuner



(a) Learned Cost Model **(b)** Learned Tuner

BOSTON UNIVERSITY

Task Decomposition of the Tuning Problem

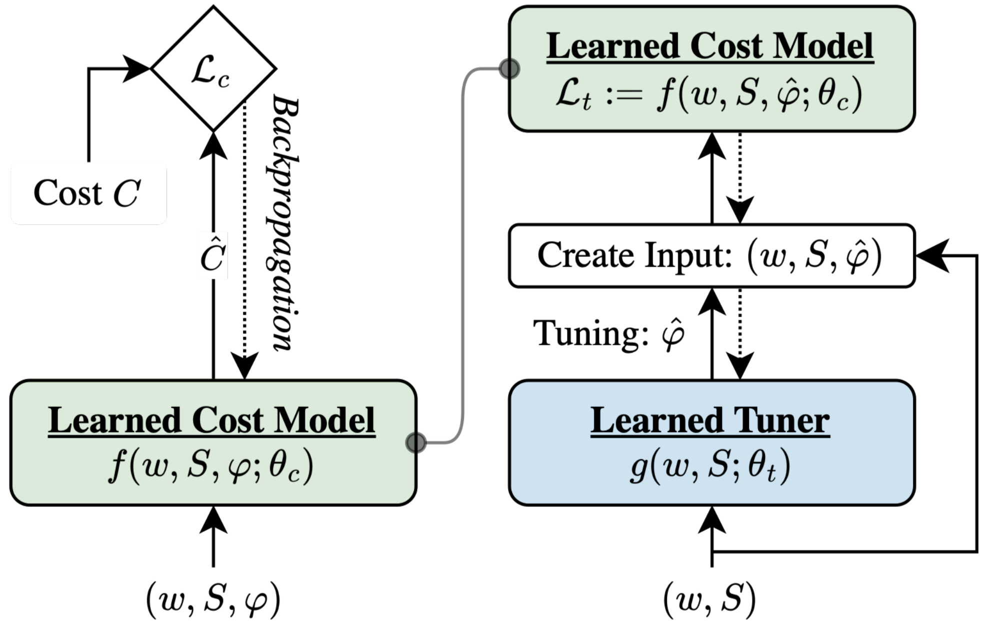
What is the **optimal tuning** for a specific **workload?**

$$\varphi^* = \arg\min_{\varphi} C_S(w, \varphi)$$

$$\mathcal{L}_t = \mathop{\mathbb{E}}_{(w,S)\sim\mathcal{D}} [C_S(w, \hat{\varphi})]$$

$$\mathcal{L}_t = \mathop{\mathbb{E}}_{(w,S)\sim\mathcal{D}} [f(w, S, \hat{\varphi})]$$

Learned cost model is used as the loss for the tuner

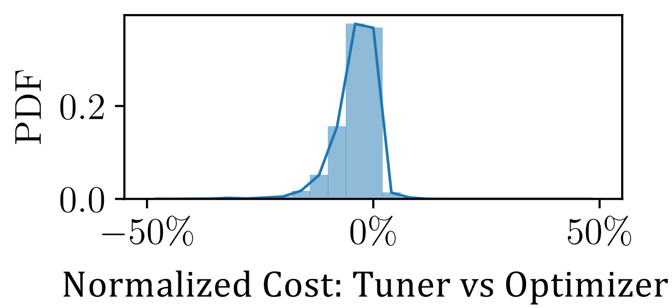


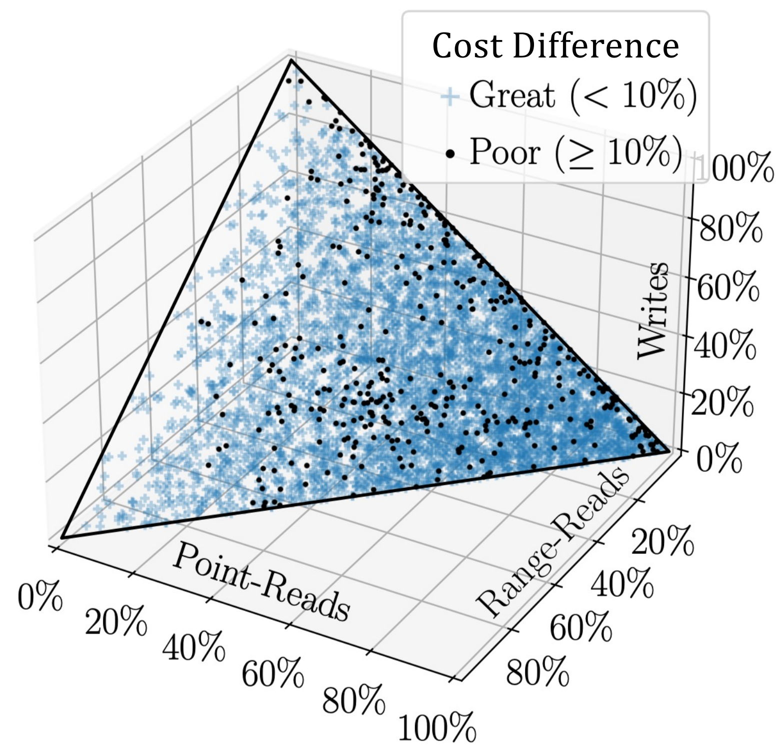
(a) Learned Cost Model **(b)** Learned Tuner

Learned Tuner Performance

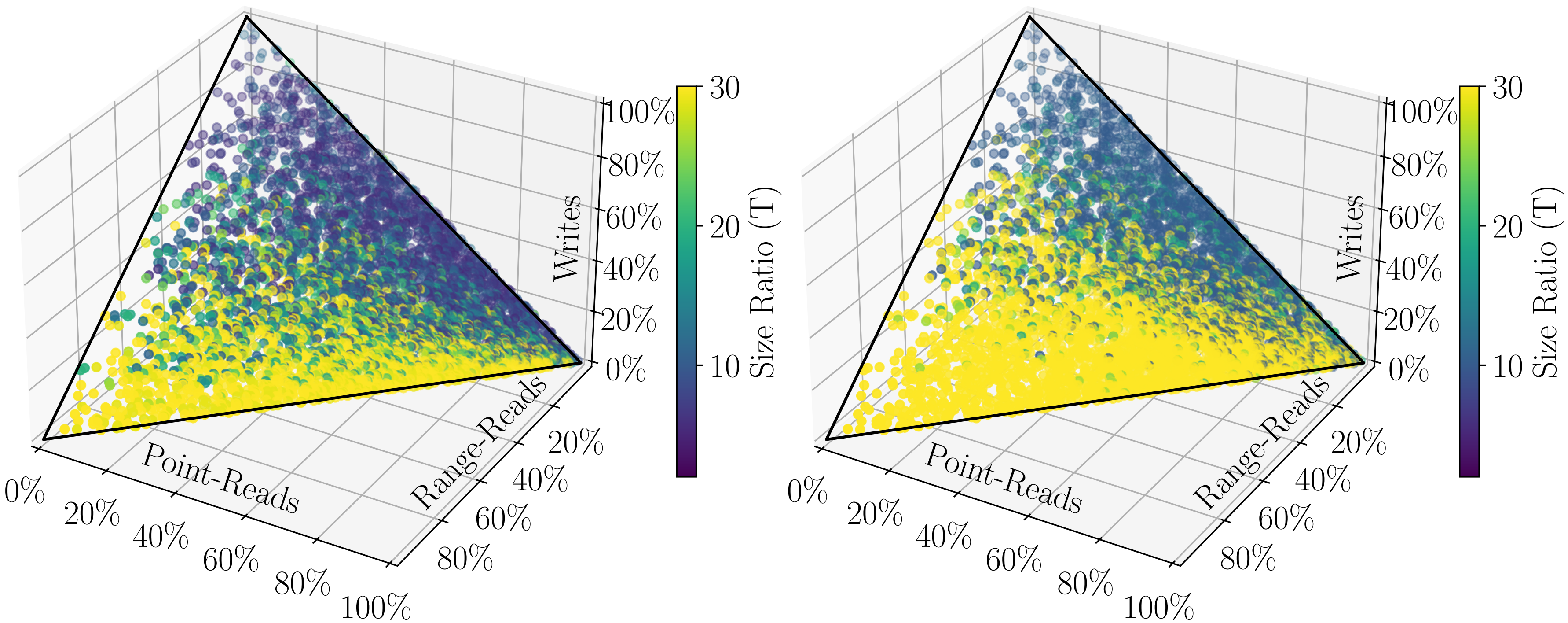
Comparison to an expert configured analytical optimizer

88% of tunings are within 10% from the optimal



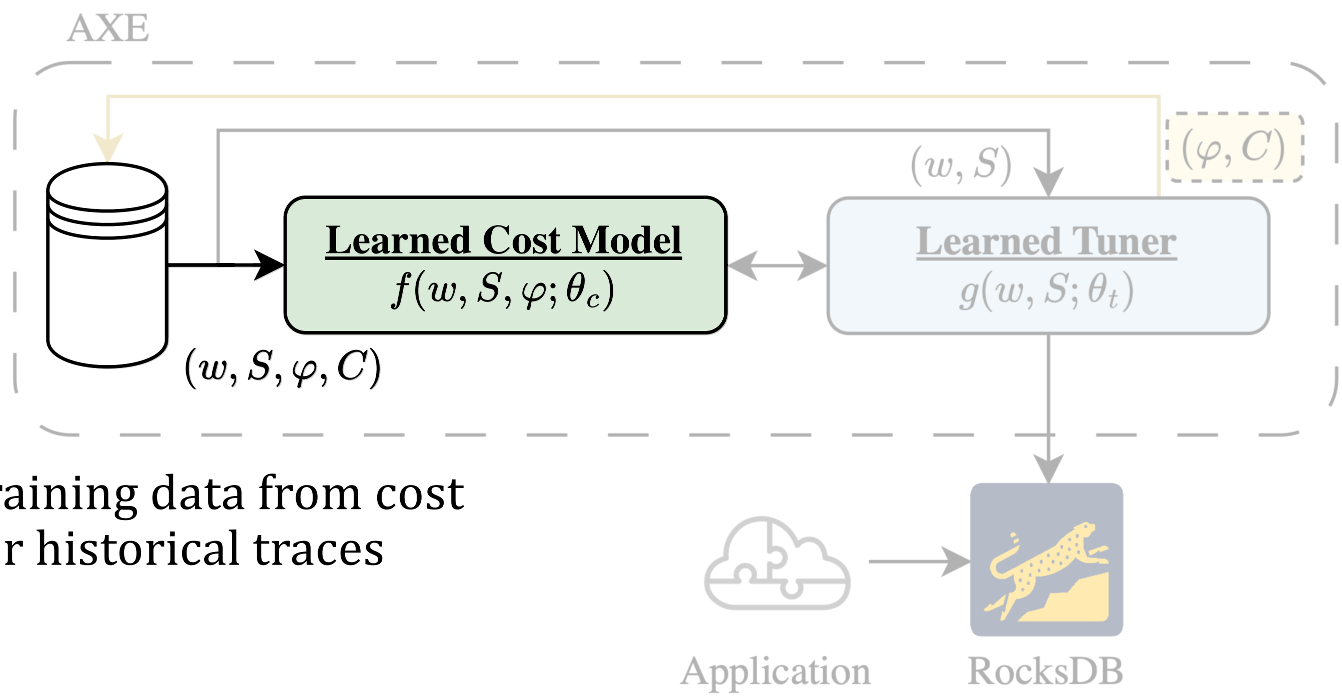


Size Ratio Recommendations



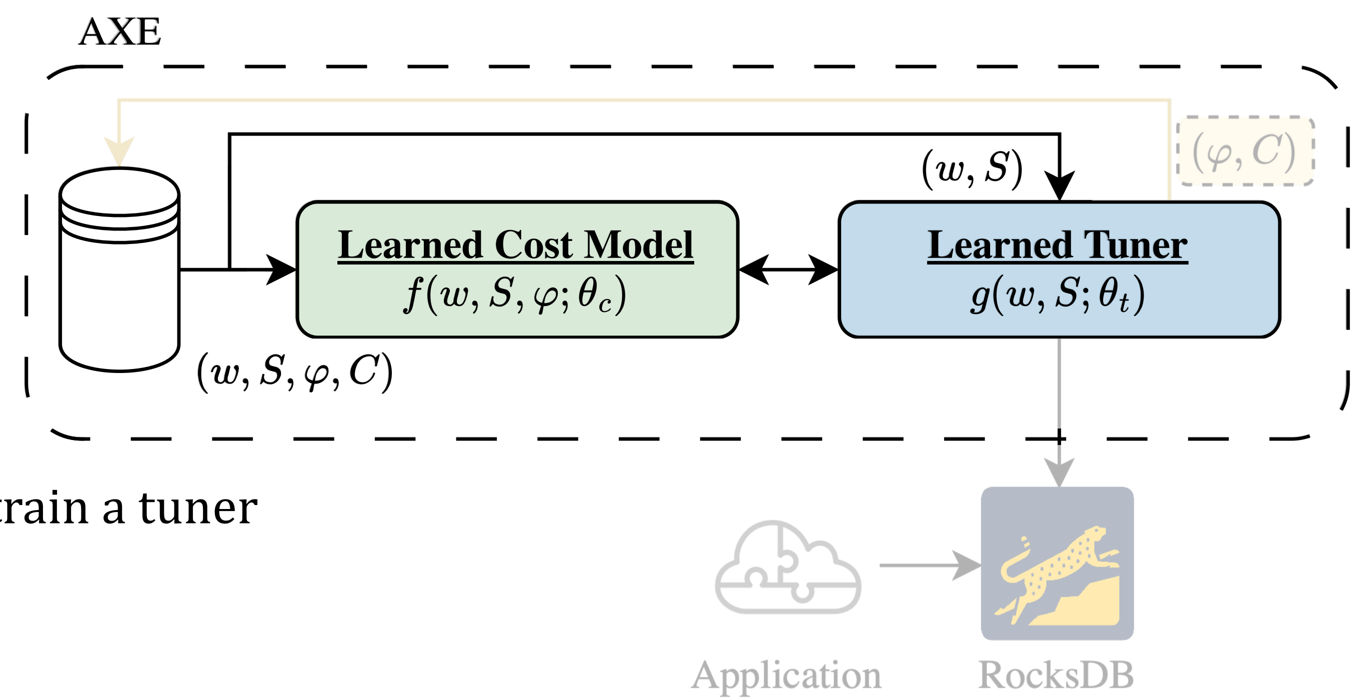
Analytical (L) and Learned Tuner (R) effectively navigates optimal size ratios

Building AXE



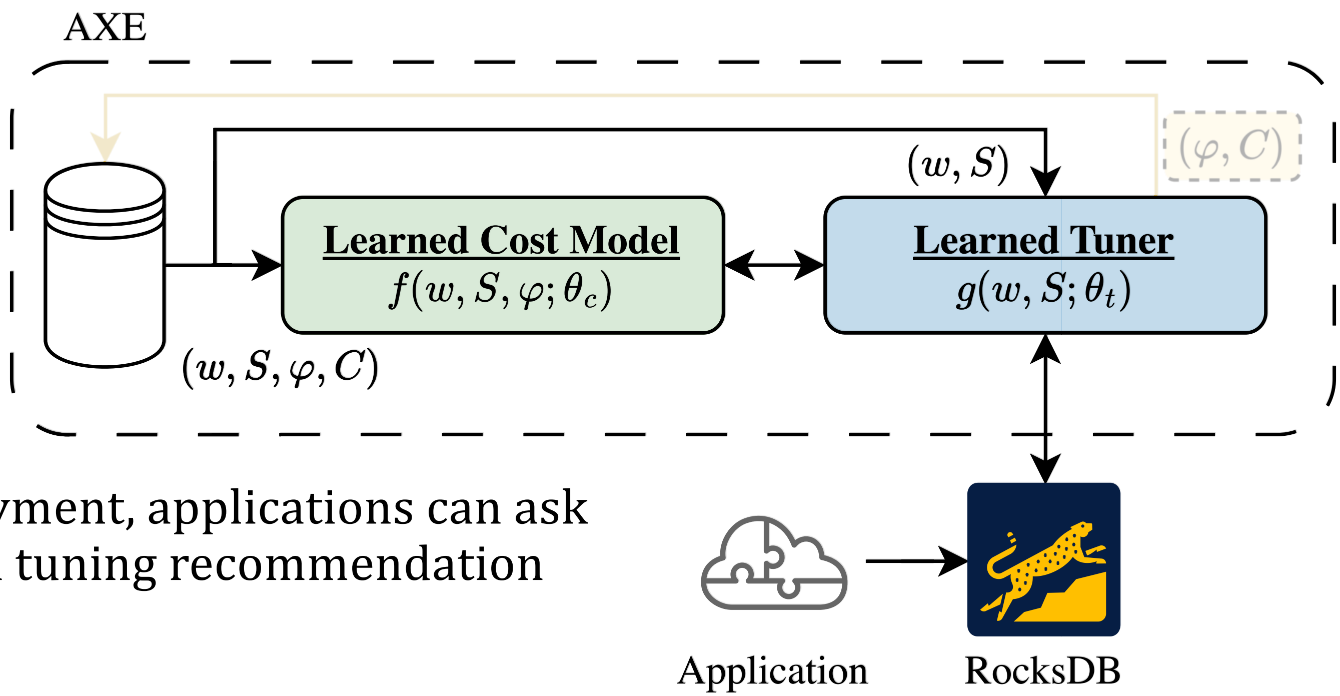
Collect training data from cost models or historical traces

Building AXE



Offline, train a tuner

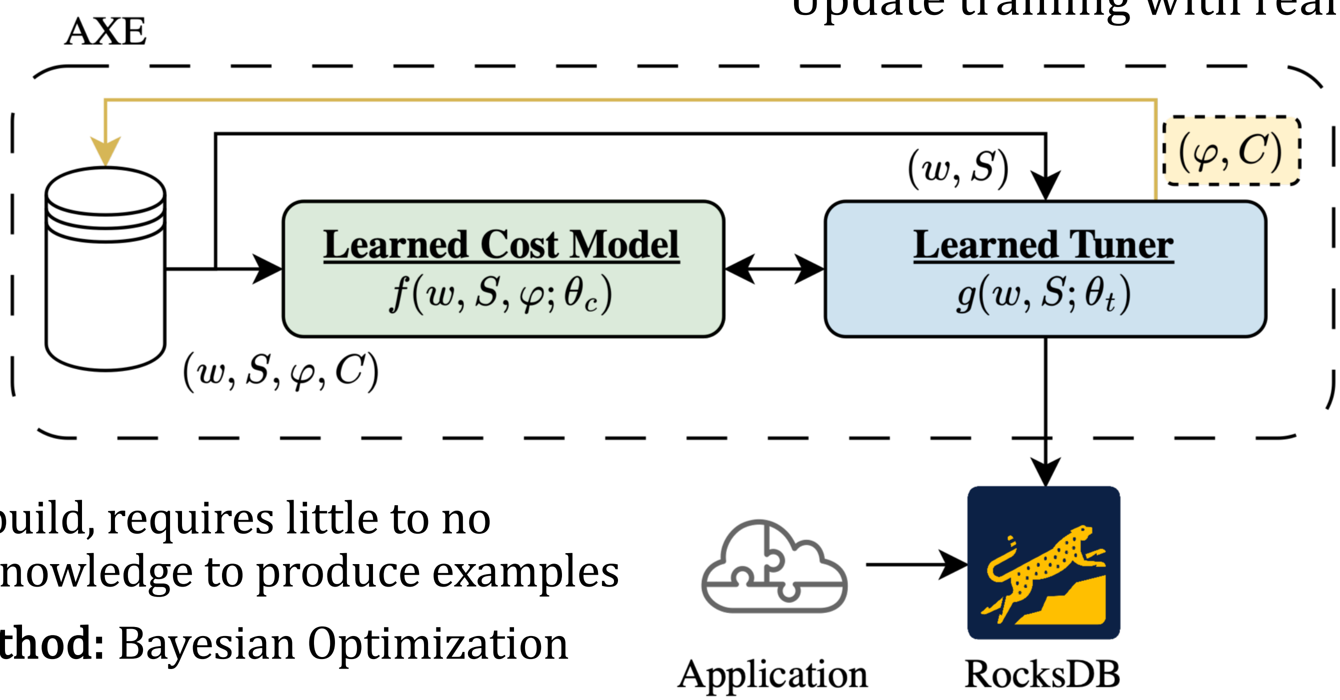
Building AXE



At deployment, applications can ask AXE for a tuning recommendation

Building AXE

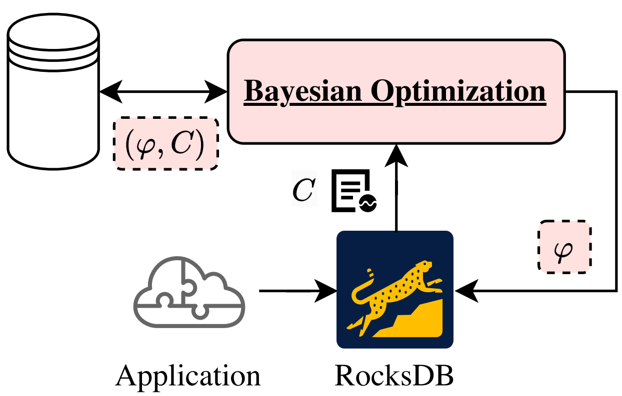
Update training with real traces

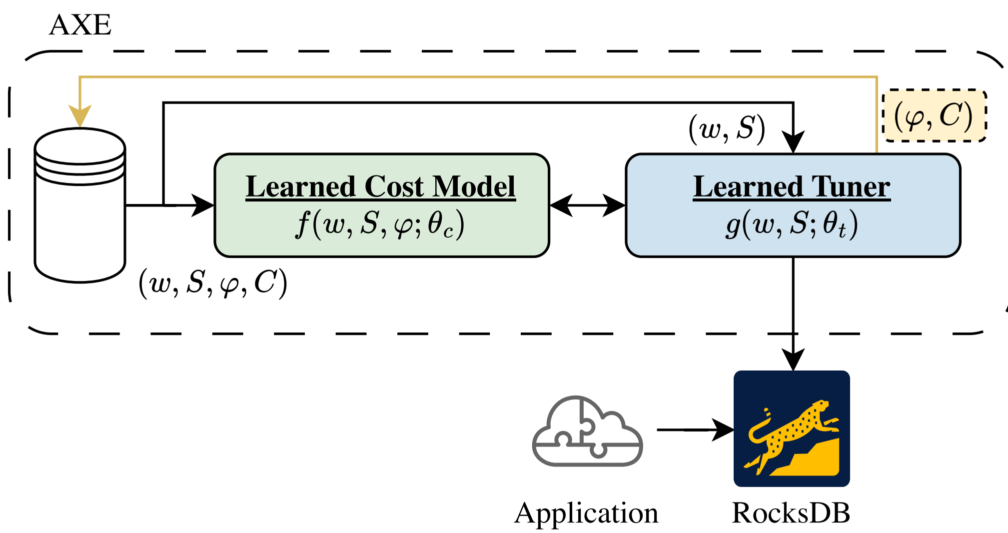


Easily to build, requires little to no systems knowledge to produce examples

Other method: Bayesian Optimization

AXE Compared To BO



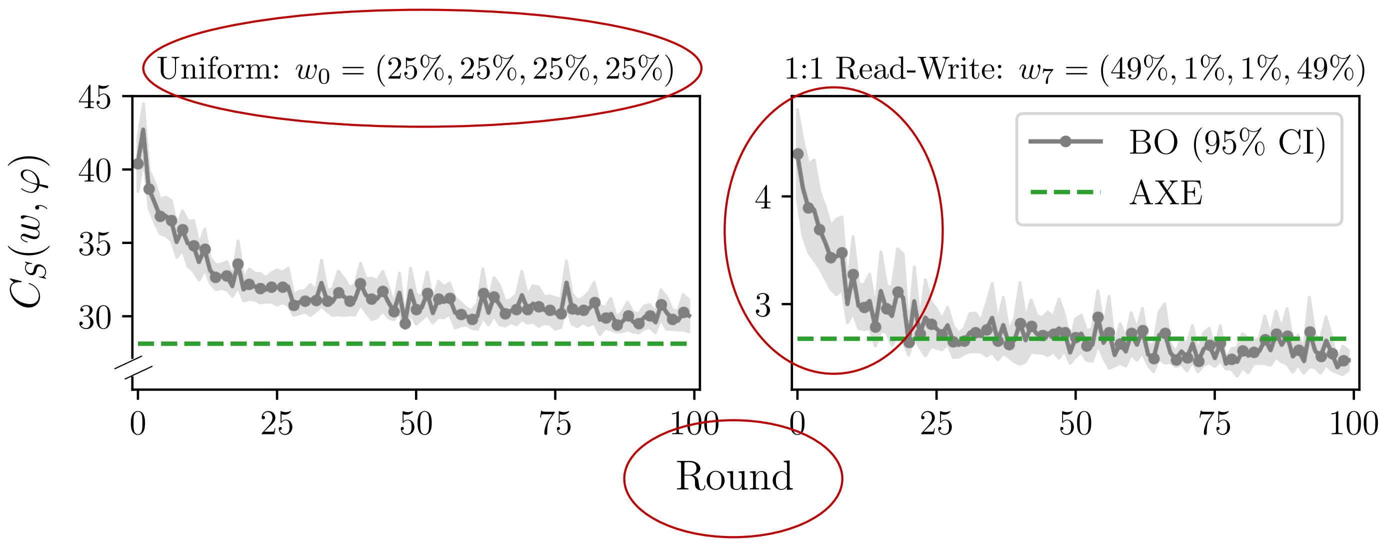


BO adapts to new workloads

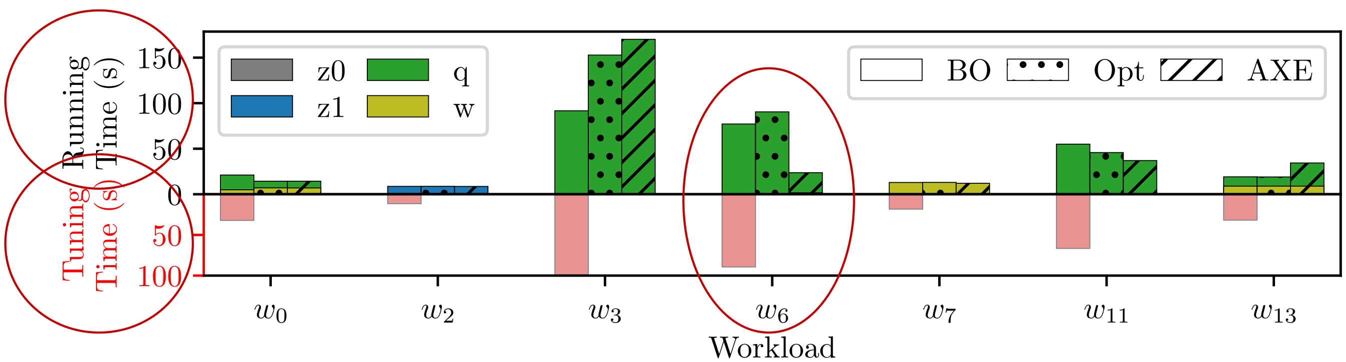
Mechanism for feedback is expensive

The Cost of AXE is Marginal

Pays a **hefty** upfront cost compared to AXE.



Tunings Deployed on RocksDB



Index	(z_0, z_1, q, u)				Type
0	25%	25%	25%	25%	**Uniform**
1	97%	1%	1%	1%	**Unimodal**
2	1%	97%	1%	1%	
3	1%	1%	97%	1%	
4	1%	1%	1%	97%	
5	49%	49%	1%	1%	**Bimodal**
6	49%	1%	49%	1%	
7	49%	1%	1%	49%	
8	1%	49%	49%	1%	
9	1%	49%	1%	49%	
10	1%	1%	49%	49%	
11	33%	33%	33%	1%	**Trimodal**
12	33%	33%	1%	33%	
13	33%	1%	33%	33%	
14	1%	33%	33%	33%	

AXE on RocksDB creates competitive tunings

Less domain expertise needed compared to an optimizer

Less upfront cost compared to iterative tuning

Thanks!

Learn more at

disc.bu.edu/
www.ndhuynh.com/