LlamaTune: Sample-Efficient DBMS Configuration Tuning

연세대학교 컴퓨터과학과 권세인

2022년 9월

과제명: IoT 환경을 위한 고성능 플래시 메모리 스토리지 기반 인메모리 분산 DBMS 연구개발

과제번호: 2017-0-00477

INTRODUCTION

First, tuning a smaller configuration space can lead to significant improvements in the number of samples required.

Yet, which knobs are important varies by workload, and existing methods for identifying important knobs are expensive and unreliable.

Second, we observe that not all DBMS knob values are the same.

Additionally, some parameters may have very large valid ranges.

CONTRIBUTION

First, we design a randomized low-dimensional projection approach where we perform a projection of the configuration space from all dimensions (D) to a lower-dimensional subspace (d) and then use a BO-method to tune this lower-dimensional subspace.

Second, to handle configuration **knobs** that **have a different behavior for special values**, we design a new biased-sampling based approach that can be used during the random initialization of BO methods.

We combine the above techniques to design **LlamaTune**, an end-to-end tuning framework that can **tune a DBMS for new workloads without using any prior knowledge.**

MOTIVATION

1. Only Tuning Important Knobs

The main reason optimizers need a large number of iterations is that the configuration search space exposed to them is very high-dimensional.

All existing methods for automatically identifying important knobs are based on a **ranking** oriented process.

Limitation

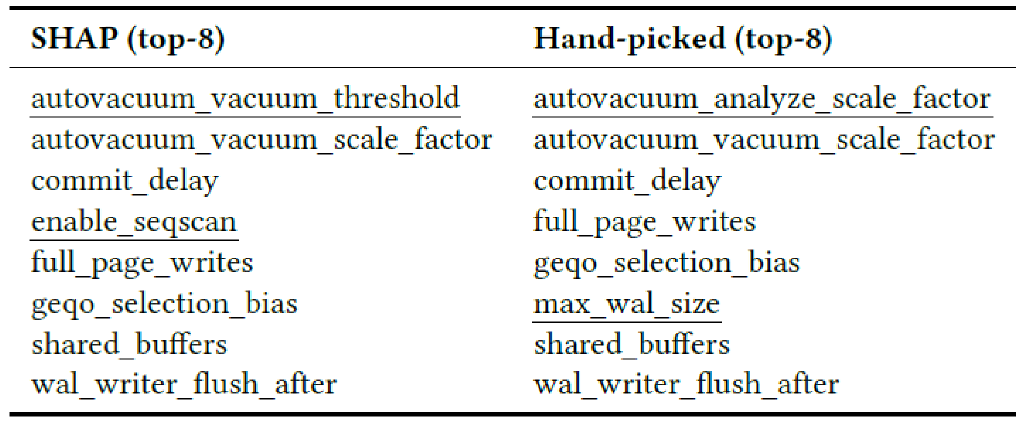
(1) not always reliable, which can lead to finding worse configurations

(2) the important knobs found for one workload may not work for others.

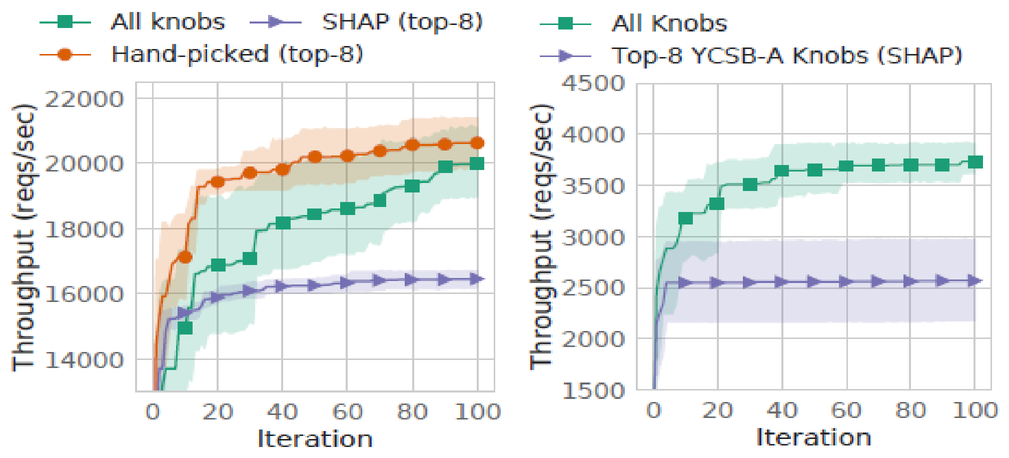
(3) challenging to identify and select the correct set of important knobs for different workloads or systems.

MOTIVATION

Table 1: SHAP's top-8 knobs vs hand-picked ones for YCSB-A. Underlined knobs indicate the differences between the two.



SHAP (top-8)	Hand-picked (top-8)
autovacuum_vacuum_threshold	autovacuum_analyze_scale_factor
autovacuum_vacuum_scale_factor	autovacuum_vacuum_scale_factor
commit_delay	commit_delay
enable_seqscan	full_page_writes
full_page_writes	geqo_selection_bias
geqo_selection_bias	max_wal_size
shared_buffers	shared_buffers
wal_writer_flush_after	wal_writer_flush_after



(a) SHAP vs Manual (YCSB-A) (b) Top-8 YCSB-A to TPC-C

Figure 2: Best performance on YCSB-A, when tuning SHAP's top-8 knobs; a hand-picked of top-8 knobs, and all knobs are baselines (left plot). Best performance on TPC-C, when tuning YCSB-A's top-8 knobs ranked by SHAP (right plot).[3]

2. Identifying Important Knobs

tuning a low-dimensional space, generated by a set of important knobs can lead to finding better configurations with fewer iterations compared to the state-of-the-art SMAC algorithm.

However, existing statistical-based methods for selecting important knobs **cannot be utilized as they are expensive, not always reliable and can lead to worse tuning outcomes.**

Randomized Low Dimensional Tuning

Present a new approach which bypasses the need to **identify the exact set of important knobs, yet realizes the benefits of low dimensional tuning.**

Due to the high dimensionality D of the input search space X_D, modeling the DBMS performance across the whole space accurately enough requires the evaluation of many points.

The BO method received as an input, a smaller d-dimensional space X_d.

The fewer points would be needed to effectively learn the (smaller) space X_d .

Therefore, choosing a smaller X_d instead of X_D has the potential to improve the BO method performance.

Synthetic Search Spaces

We can use "artificial” dimensions (or synthetic knobs) to generate our low dimensional space, X_d .

⇒Mapping from the synthetic knob values (i.e. approximated space X_d)
to the physical DBMS configuration knobs (i.e. original input space X_D).

This enables us to **realize the benefits of optimizing a low-dimensional space**, while **avoiding the need to identify important knobs.**

We believe that **low-dimensional BO-methods** are a promising way to **improve the optimizer performance for database tuning**.

Random Low-dimensional Projections

REMBO

① Defines d-dimensional search space.

$X_d = [-\sqrt{d}, \sqrt{d}]^d$

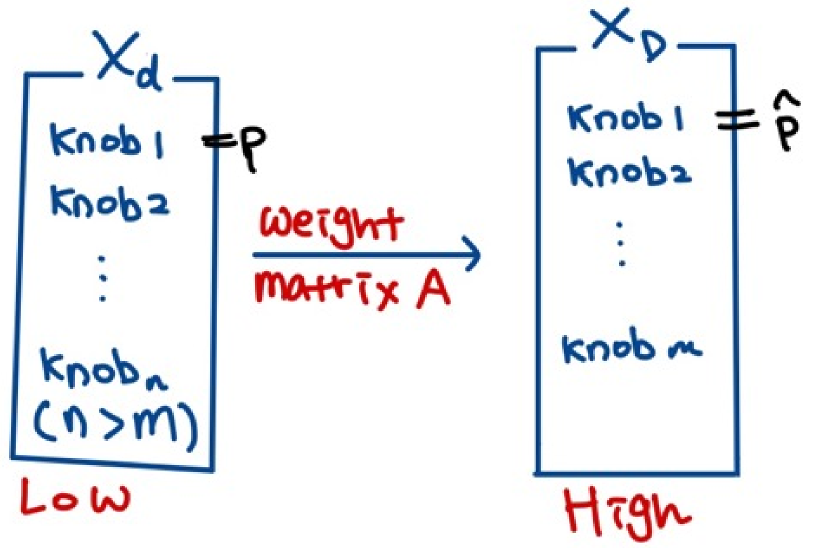
② given as input to the BO method.

Suggest $p \in X_d$

③ REMBO generates a random projection matrix.

$A \in R^{D \times d}$ by using $N(0,1)$

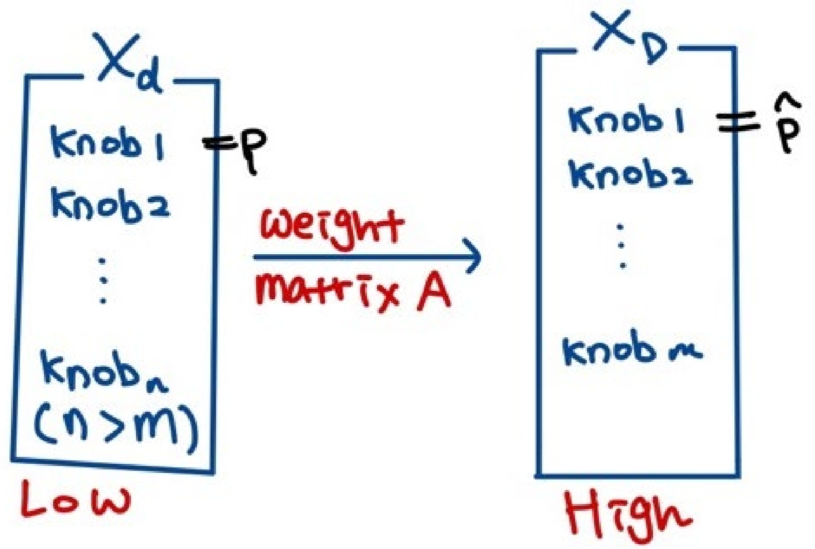
④ A is used to project a point suggested by the BO, p to a corresponding point $\hat{p}$.



$\therefore \hat{p} = Ap$

Random Low-dimensional Projections

REMBO



If the original space X_D is unbounded, then the projected point will be a valid point in X_D .

Yet, in many problems (including DBMS tuning), there exist constraints associated with X_D .

REMBO handles this by **"clipping"** the projected $\hat{p}$ to the **nearest point that belongs in X_D.**

This heuristic forces the optimization to be done on the facets of X_D , thus neglecting almost all interior points

Random Low-dimensional Projections

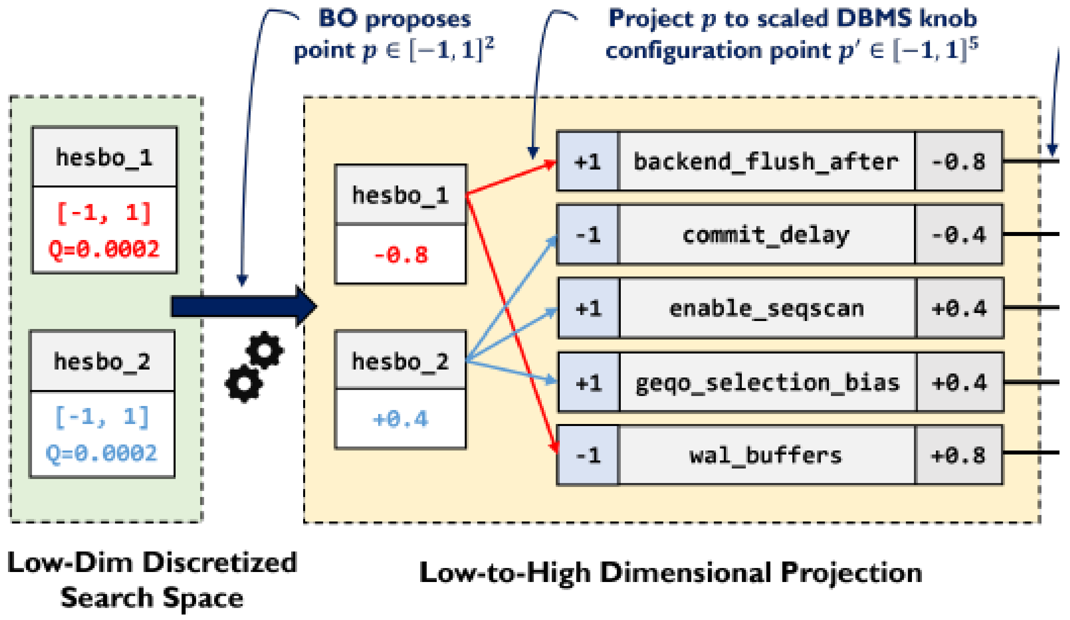
HeSBO

① Defines d-dimensional search space

$X_d = [-1, 1]^d$

② HeSBO generates a random projection matrix.

$A \in R^{D \times d}$, each row contains exactly one non-zero element in a random column that is set to ± 1.


BO proposes point $p \in [-1,1]^2$
Project p to scaled DBMS knob configuration point $p' \in [-1,1]^5$
hesbo_1
[-1, 1]
Q=0.0002
hesbo_2
[-1, 1]
Q=0.0002
hesbo_1
-0.8
hesbo_2
+0.4
+1 backend_flush_after -0.8
-1 commit_delay -0.4
+1 enable_seqscan +0.4
+1 geqo_selection_bias +0.4
-1 wal_buffers +0.8
Low-Dim Discretized Search Space
Low-to-High Dimensional Projection

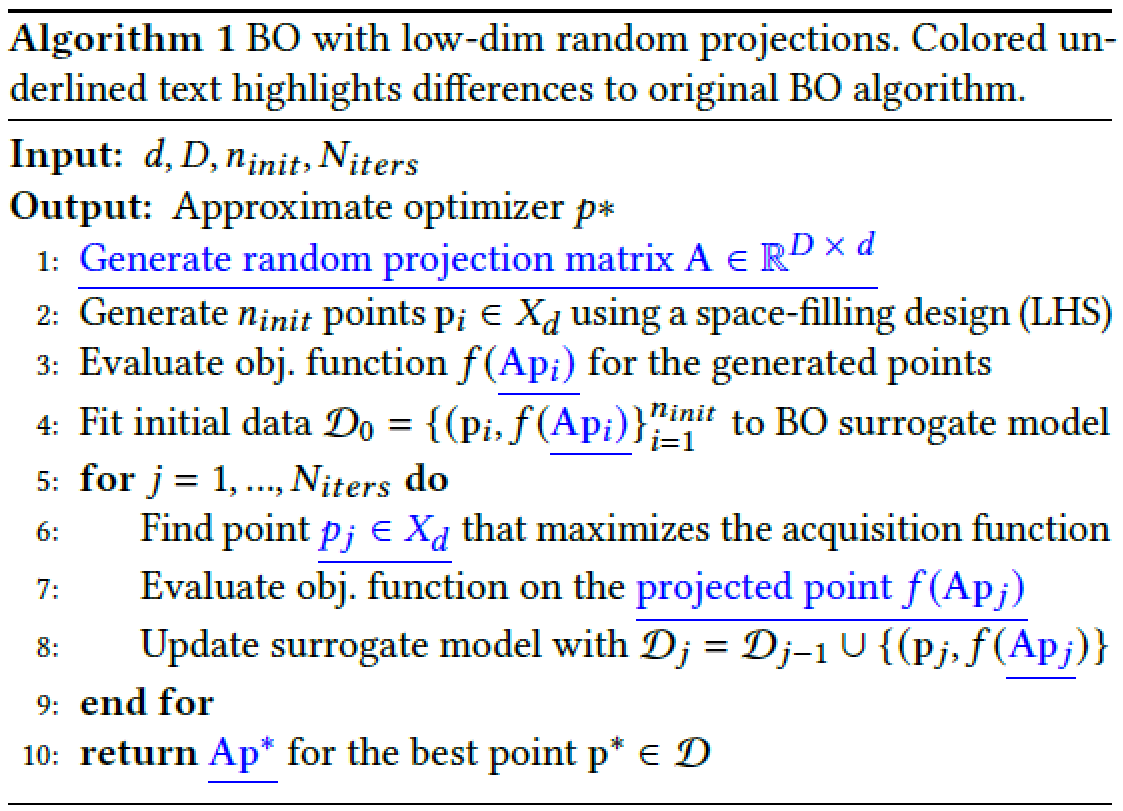
The index of the column and the sign of the value are sampled independently and uniformly at random.

Essentially, ***A*** provides a one-to-many mapping.

Every original knob ***i*** in X_D is controlled by exactly one synthetic knob j in X_d, while each synthetic knob can control multiple original ones.

Thus, it is impossible for any projection to fall outside of X_D .

BO with Random Projections



Algorithm 1 BO with low-dim random projections. Colored underlined text highlights differences to original BO algorithm.

```
Input: $d, D, n_{init}, N_{iters}$
Output: Approximate optimizer $p*$
 1: Generate random projection matrix $A \in \mathbb{R}^{D \times d}$
 2: Generate $n_{init}$ points $p_i \in X_d$ using a space-filling design (LHS)
 3: Evaluate obj. function $f(Ap_i)$ for the generated points
 4: Fit initial data $\mathcal{D}_0 = \{(p_i, f(Ap_i))\}_{i=1}^{n_{init}}$ to BO surrogate model
 5: for $j = 1, ..., N_{iters}$ do
 6:     Find point $p_j \in X_d$ that maximizes the acquisition function
 7:     Evaluate obj. function on the projected point $f(Ap_j)$
 8:     Update surrogate model with $\mathcal{D}_j = \mathcal{D}_{j-1} \cup \{(p_j, f(Ap_j))\}$
 9: end for
10: return $Ap^*$ for the best point $p^* \in \mathcal{D}$
```

d : the number of dimensions of the low dimensional space.
n_{init} : the number of initial random samples to be generated.
N_{iters} : the number of iterations to perform.

Converting Points to DBMS Knob Values

Both REMBO and HeSBO project the point suggested by the BO method to the uniform high-dimensional space $X_D = [-1, 1]^D$.
However**, in reality a DBMS configuration knob space is much more heterogeneous compared to $\boldsymbol{X_D}$.**

1. Numerical
The [min, max] **range of values can significantly vary across different knobs**
$\Rightarrow$ employ min-max uniform scaling.

translate a value $x \in [x_{m_i n}, x_{m_{ax}}]$ to a y ∈ $[y_{min}, y_{max}]$

$$y = y_{min} + \frac{x - x_{min}}{x_{max} - x_{min}}(y_{max} - y_{min})$$

Converting Points to DBMS Knob Values

Both REMBO and HeSBO project the point suggested by the BO method to the uniform high-dimensional space $X_D = [-1, 1]^D$.
However**, in reality a DBMS configuration knob space is much more heterogeneous compared to $\boldsymbol{X_D}$.**

2. Categorical

we perform a **two-step** conversion process.
We first **rescale** the projected value x ∈ [−1, 1] to a value y ∈ [0, 1], and then we split this range equally to as many bins, as is the number of different choices.

The final categorical knob value is defined by which bin the value y falls into.

Dimensionality of the Projected Space

Prior works have shown good performance improvements when tuning around the 10−20% most important, of all knobs considered for tuning.

we show next, setting *d* to an even higher value (i.e. > 16) is not detrimental to the optimizer performance, which also demonstrates the robustness of our proposed method.

First, we want to evaluate **how effective REMBO and HeSBO methods are at finding a good projection** of the original DBMS configuration space, which can result in better performance.

Second, we want to explore how much our selection of *d* influences the optimizer performance.

Dimensionality of the Projected Space

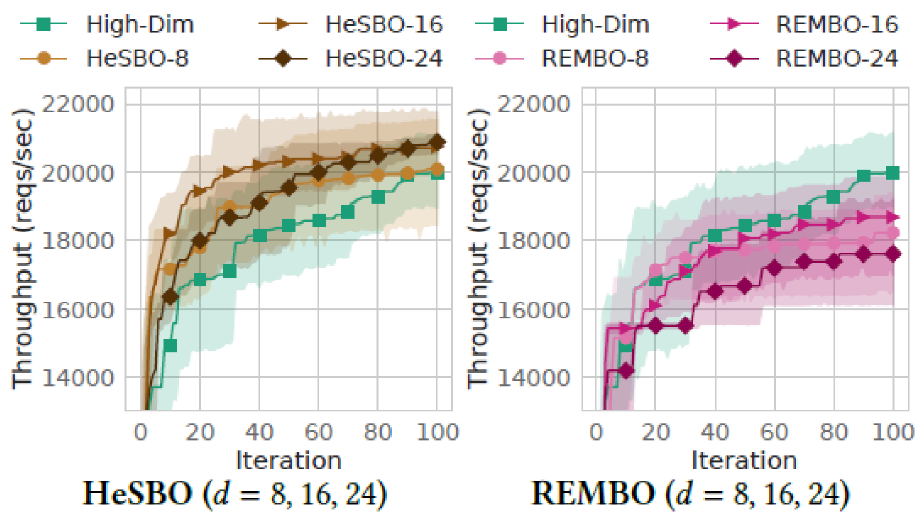


Figure 3: Best throughput on YCSB-A, when using SMAC to optimize a low-dimensional space from REMBO or HeSBO projections. Baseline (green line) tunes the original space. Shaded areas correspond to 95% confidence interval ranges.[3]

Case study
using YCSB-A.
Tuning 90 knobs of PostgreSQL v9.6 for 100 iterations. (first 10 are LHS-generated samples.)
experiment with both REMBO and HeSBO, where we set d to 8, 16, or 24 dimensions.

Special Knob Values in Databases

The configuration search space
- Categorical knobs

- Numerical knobs

- hybrid knobs
  : numerical knobs do have special values (e.g. −1, 0) that inevitably break natural order.

If such a knob is set to its special value, it does something very different compared to what it normally does. (e.g., disables some feature).

We identified 20 hybrid knobs for PostgreSQL v9.6 from the official documentation; special values are explicitly mentioned in the knob description.

Special Knob Values in Databases

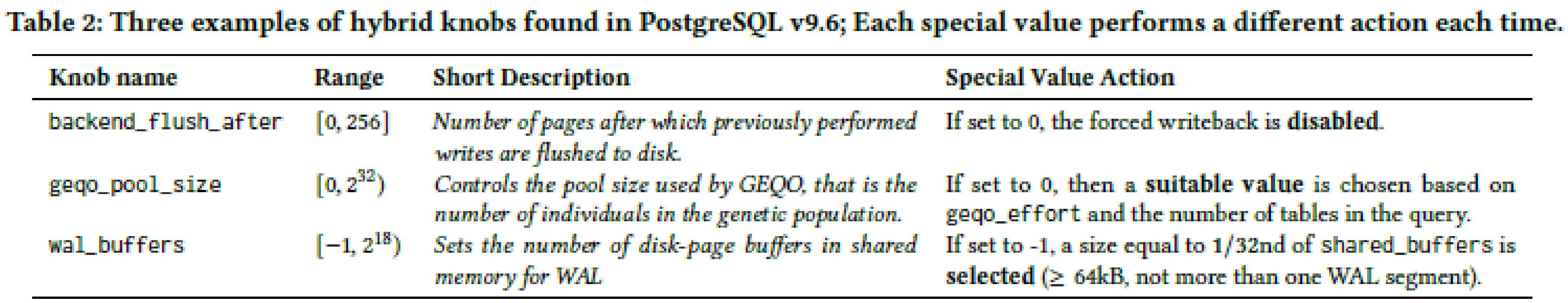


Table 2: Three examples of hybrid knobs found in PostgreSQL v9.6; Each special value performs a different action each time.

Knob name	Range	Short Description	Special Value Action
`backend_flush_after`	[0, 256]	*Number of pages after which previously performed writes are flushed to disk.*	If set to 0, the forced writeback is **disabled**.
`geqo_pool_size`	$[0, 2^{32})$	*Controls the pool size used by GEQO, that is the number of individuals in the genetic population.*	If set to 0, then a **suitable value** is chosen based on `geqo_effort` and the number of tables in the query.
`wal_buffers`	$[-1, 2^{18})$	*Sets the number of disk-page buffers in shared memory for WAL.*	If set to -1, a size equal to 1/32nd of `shared_buffers` is **selected** ($\geq$ 64kB, not more than one WAL segment).

1. Disabling some feature
2. Inferring this knob's value based on some other's knob
3. Setting this knob's value using an internal heuristic.

Special values make the modeling of the DBMS performance more difficult because they represent a discontinuity i
objective
Function output relative to its input.

Special Knob Values in Databases

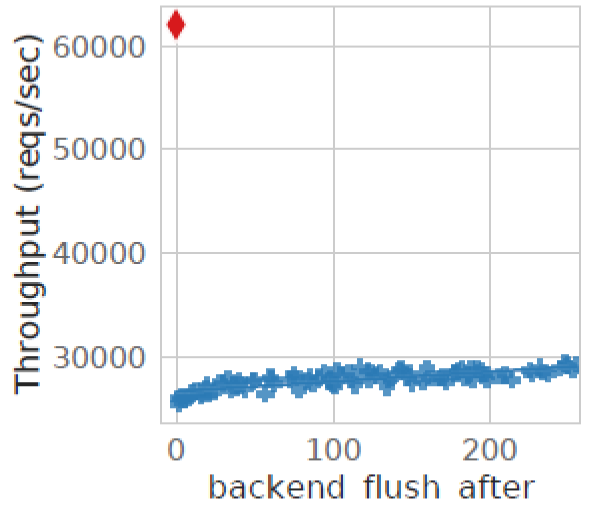


Figure 4: Effect on perf. of special value "0".

A value of "0" (red diamond in the figure 4), resulting in far higher throughput for this workload.

However, if the optimizer tuning this knob lacked the knowledge about the special meaning of value 0, it would be **unlikely that it would ever choose this value during tuning.**

The optimizer would focus on exploring the **"neighborhood" of larger values for this knob**, which seem more promising.

Special Knob Values in Databases

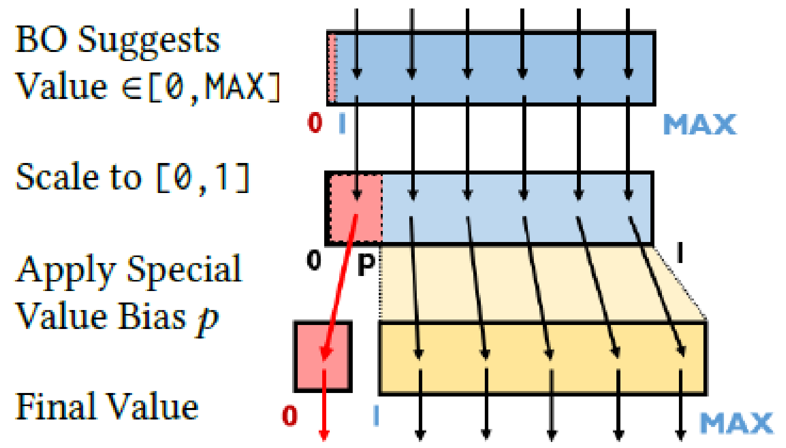


Figure 5: Applying bias to special value "0" of a hybrid knob.

- *Biasing Special Value Sampling Probability*

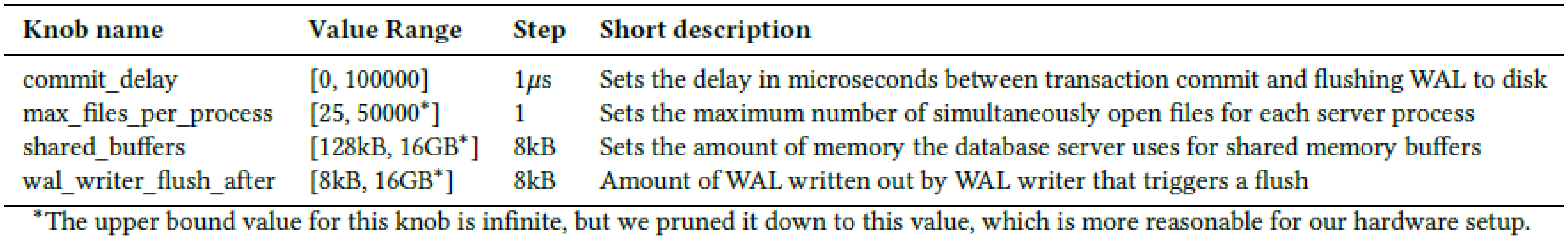
We propose a methodology that enables the optimizer to observe the effect of a special value early on in the tuning process.

Change the hybrid knobs values proposed, such that we bias the probability of the special value being evaluated. we keep the same probability *p* for all hybrid knobs we tune.

Finally, the resulting value is used at the DBMS configuration to be evaluated.

Configuration Space Bucketization

Table 3: Examples of discrete knobs with large value range for PostgreSQL v9.6



Knob name	Value Range	Step	Short description
commit_delay	[0, 100000]	1μs	Sets the delay in microseconds between transaction commit and flushing WAL to disk
max_files_per_process	[25, 50000*]	1	Sets the maximum number of simultaneously open files for each server process
shared_buffers	[128kB, 16GB*]	8kB	Sets the amount of memory the database server uses for shared memory buffers
wal_writer_flush_after	[8kB, 16GB*]	8kB	Amount of WAL written out by WAL writer that triggers a flush

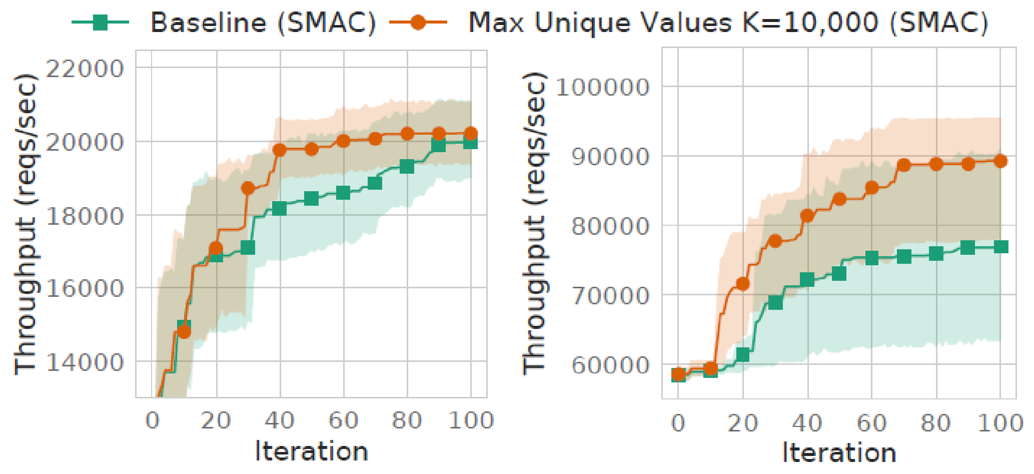
*The upper bound value for this knob is infinite, but we pruned it down to this value, which is more reasonable for our hardware setup.

How **discrete configuration knobs** in DBMS' can affect the exploitation (or local search) phase.

A broader range of values gives more fine-grained control of the underlying mechanism.
Our observation is that for many configuration knobs with large values ranges, **small changes are unlikely to affect the DBMS performance significantly.**

we explore bucketizing the knob value space.
We can do this by limiting the number of unique values that any configuration knob can take to K.

Configuration Space Bucketization



YCSB-A YCSB-B

Figure 6: Best performance achieved when tuning the bucketized space vs. the original space (top-left is better).[3]

Case Study.
We choose P% = 50% (i.e. half of all knobs), which leads to K = 10, 000 for our set of 90 knobs in PostgreSQL v9.6.

For **YCSB-A**, while the performance in the **first 20 iterations are similar**, in later iterations the **bucketized (smaller) space reaches a better performing** configuration faster.

For **YCSB-B** we see **larger benefits starting at iteration 10.**

Thus, we see that the benefits from bucketization varies across workloads.

LlamaTune Design

DBMS specific observations that can improve the configuration optimizer.

1. random low-dimensional projections

2. biasing the special value of hybrid knobs

3. bucketization for knobs with many unique values.

A unified design should satisfy three main requirements.

First, the **optimizer should always operate on the low-dimensional space**, and not in the original DBMS knob configuration one.

Second, the **special value biasing should be performed only over the set of hybrid knobs.**

Finally, the **bucketized value space for the set of knobs should be exposed to the optimizer,** so it is aware of the larger sampling intervals; otherwise it will still continue to sample at finer granularities.

LlamaTune Design

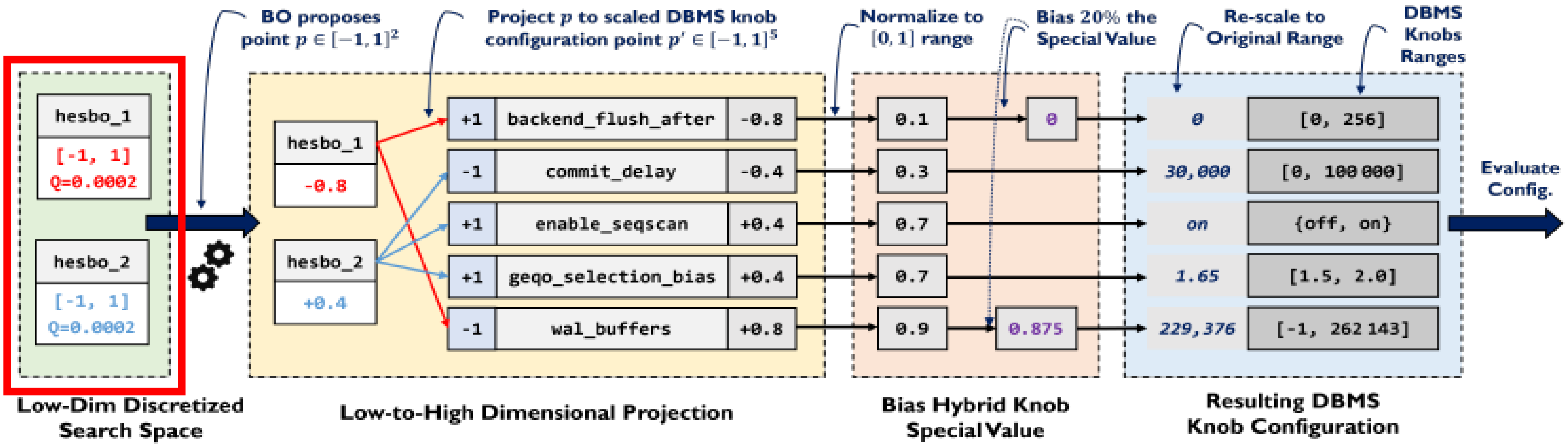
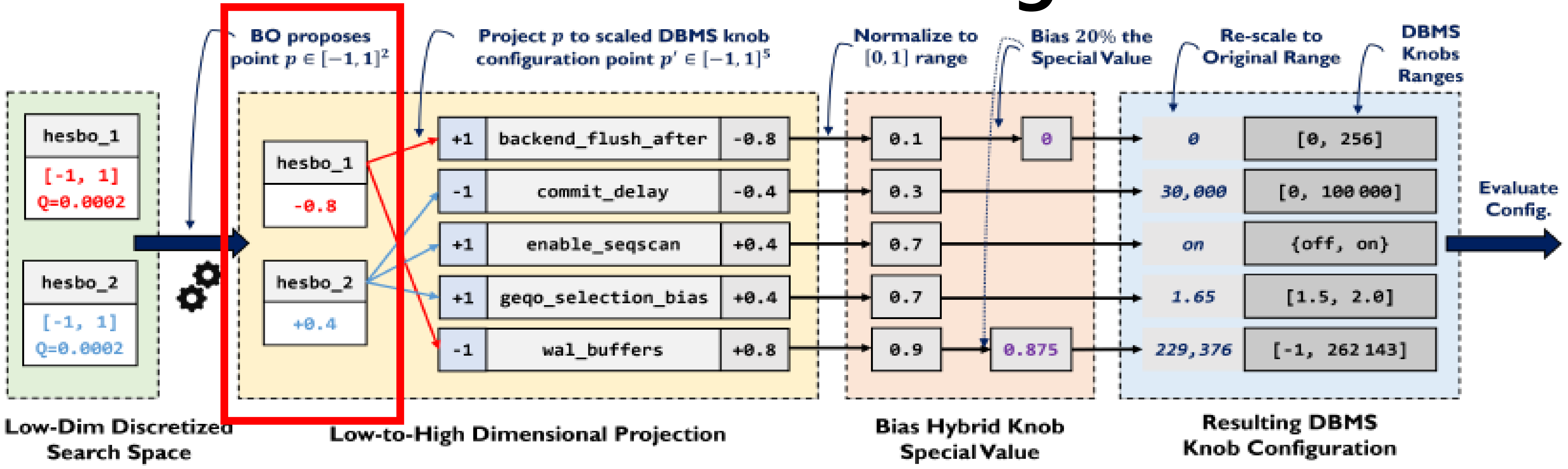

BO proposes point $p \in [-1, 1]^2$
Project p to scaled DBMS knob configuration point $p' \in [-1, 1]^5$
Normalize to [0, 1] range
Bias 20% the Special Value
Re-scale to Original Range
DBMS Knobs Ranges
hesbo_1
[-1, 1]
Q=0.0002
hesbo_2
[-1, 1]
Q=0.0002
hesbo_1
-0.8
hesbo_2
+0.4
+1 backend_flush_after -0.8
-1 commit_delay -0.4
+1 enable_seqscan +0.4
+1 geqo_selection_bias +0.4
-1 wal_buffers +0.8
0.1
0.3
0.7
0.7
0.9
0
0.875
0 [0, 256]
30,000 [0, 100 000]
on {off, on}
1.65 [1.5, 2.0]
229,376 [-1, 262 143]
Evaluate Config.
Low-Dim Discretized Search Space
Low-to-High Dimensional Projection
Bias Hybrid Knob Special Value
Resulting DBMS Knob Configuration

Figure 7: LlamaTune: Tuning example that highlights the unified end-to-end pipeline.

1. We tune 5 DBMS knobs using a 2-dimensional random projection derived by HeSBO.
   Q denotes the size of the fixed interval (2/K, K=10,000).

LlamaTune Design



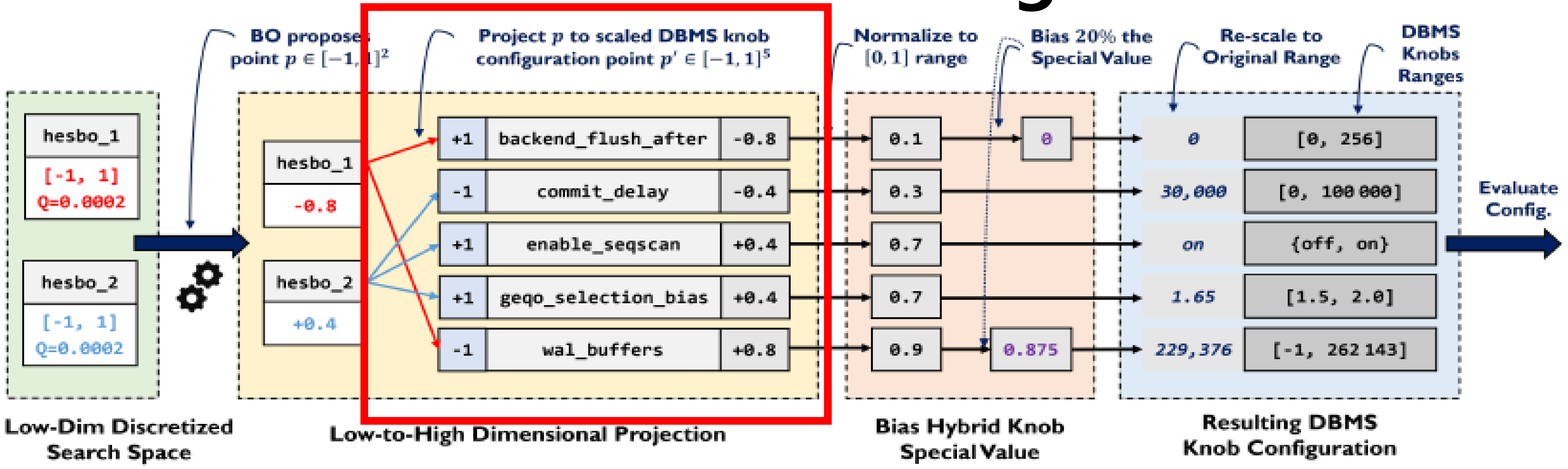
BO proposes point $p \in [-1,1]^2$
Project p to scaled DBMS knob configuration point $p' \in [-1,1]^5$
Normalize to [0, 1] range
Bias 20% the Special Value
Re-scale to Original Range
DBMS Knobs Ranges
hesbo_1
[-1, 1]
Q=0.0002
hesbo_2
[-1, 1]
Q=0.0002
hesbo_1
-0.8
hesbo_2
+0.4
+1 backend_flush_after -0.8
-1 commit_delay -0.4
+1 enable_seqscan +0.4
+1 geqo_selection_bias +0.4
-1 wal_buffers +0.8
0.1
0.3
0.7
0.7
0.9
0
0.875
0 [0, 256]
30,000 [0, 100 000]
on {off, on}
1.65 [1.5, 2.0]
229,376 [-1, 262 143]
Evaluate Config.
Low-Dim Discretized Search Space
Low-to-High Dimensional Projection
Bias Hybrid Knob Special Value
Resulting DBMS Knob Configuration

Figure 7: LlamaTune: Tuning example that highlights the unified end-to-end pipeline.

1. We tune 5 DBMS knobs using a 2-dimensional random projection derived by HeSBO.
   Q denotes the size of the fixed interval (2/K, K=10,000).

2. The optimizer suggests a point [-0.8, 0.4] that belongs to the low-dim $[-1, 1]^d$ bucketized space.

LlamaTune Design



BO proposes point $p \in [-1,1]^2$
Project p to scaled DBMS knob configuration point $p' \in [-1,1]^5$
Normalize to [0, 1] range
Bias 20% the Special Value
Re-scale to Original Range
DBMS Knobs Ranges
hesbo_1
[-1, 1]
Q=0.0002
hesbo_2
[-1, 1]
Q=0.0002
hesbo_1
-0.8
hesbo_2
+0.4
+1 backend_flush_after -0.8
-1 commit_delay -0.4
+1 enable_seqscan +0.4
+1 geqo_selection_bias +0.4
-1 wal_buffers +0.8
0.1
0.3
0.7
0.7
0.9
0
0.875
0 [0, 256]
30,000 [0, 100 000]
on {off, on}
1.65 [1.5, 2.0]
229,376 [-1, 262 143]
Evaluate Config.
Low-Dim Discretized Search Space
Low-to-High Dimensional Projection
Bias Hybrid Knob Special Value
Resulting DBMS Knob Configuration

Figure 7: LlamaTune: Tuning example that highlights the unified end-to-end pipeline.

3. This point is then projected to a high-dimensional point in $[-1, 1]^D$ (i.e., D = 5).
    The first synthetic knob is mapped to both hybrid knobs (but with opposite ± signs), while the second one maps to the remaining three.

LlamaTune Design

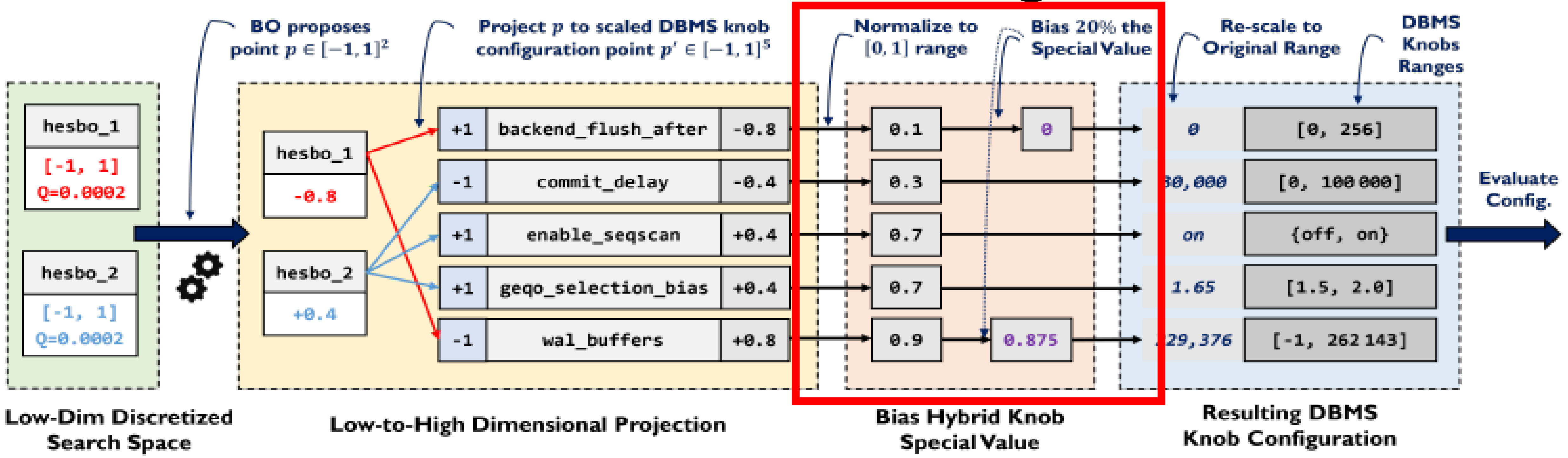

BO proposes point $p \in [-1,1]^2$
Project p to scaled DBMS knob configuration point $p' \in [-1,1]^5$
Normalize to [0, 1] range
Bias 20% the Special Value
Re-scale to Original Range
DBMS Knobs Ranges
hesbo_1
[-1, 1]
Q=0.0002
hesbo_2
[-1, 1]
Q=0.0002
hesbo_1
-0.8
hesbo_2
+0.4
+1 backend_flush_after -0.8
-1 commit_delay -0.4
+1 enable_seqscan +0.4
+1 geqo_selection_bias +0.4
-1 wal_buffers +0.8
0.1
0.3
0.7
0.7
0.9
0
0.875
0
30,000
on
1.65
229,376
[0, 256]
[0, 100 000]
{off, on}
[1.5, 2.0]
[-1, 262 143]
Evaluate Config.
Low-Dim Discretized Search Space
Low-to-High Dimensional Projection
Bias Hybrid Knob Special Value
Resulting DBMS Knob Configuration

Figure 7: LlamaTune: Tuning example that highlights the unified end-to-end pipeline.

3. This point is then projected to a high-dimensional point in $[-1, 1]^D$ (i.e., D = 5).
The first synthetic knob is mapped to both hybrid knobs (but with opposite ± signs), while the second one maps to the remaining three.

4. Next, we normalize all knob values to [0, 1], and apply special value biasing only for the two hybrid knob values.

LlamaTune Design

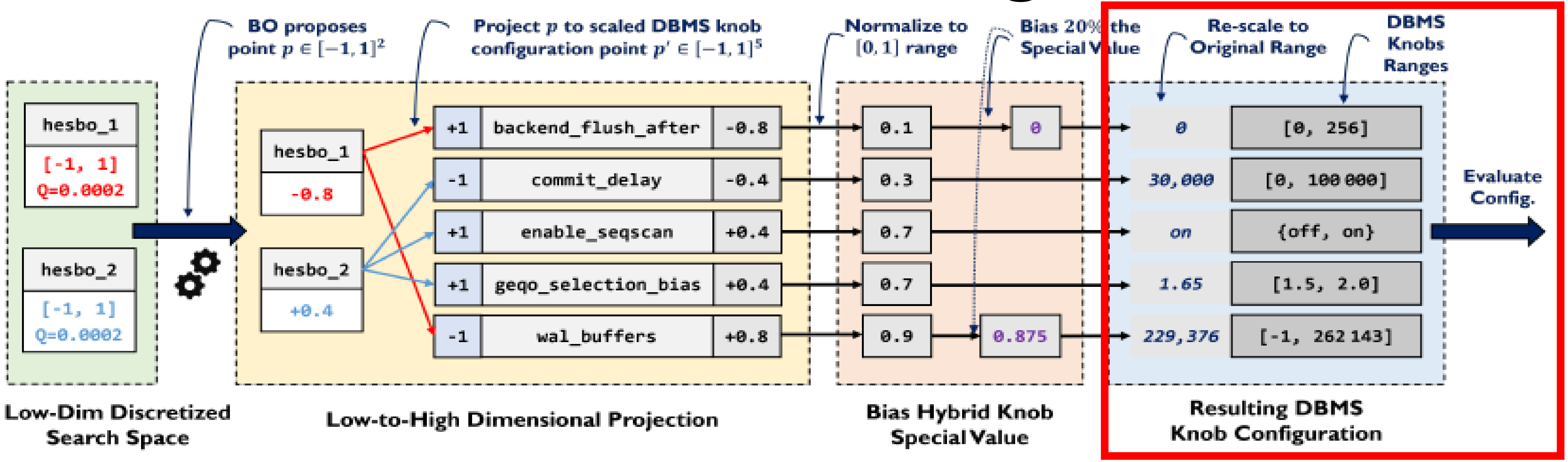

BO proposes point $p \in [-1,1]^2$
Project p to scaled DBMS knob configuration point $p' \in [-1,1]^5$
Normalize to [0, 1] range
Bias 20% the Special Value
Re-scale to Original Range
DBMS Knobs Ranges
hesbo_1
[-1, 1]
Q=0.0002
hesbo_2
[-1, 1]
Q=0.0002
hesbo_1
-0.8
hesbo_2
+0.4
+1 backend_flush_after -0.8
-1 commit_delay -0.4
+1 enable_seqscan +0.4
+1 geqo_selection_bias +0.4
-1 wal_buffers +0.8
0.1
0.3
0.7
0.7
0.9
0
0.875
0 [0, 256]
30,000 [0, 100 000]
on {off, on}
1.65 [1.5, 2.0]
229,376 [-1, 262 143]
Evaluate Config.
Low-Dim Discretized Search Space
Low-to-High Dimensional Projection
Bias Hybrid Knob Special Value
Resulting DBMS Knob Configuration

Figure 7: LlamaTune: Tuning example that highlights the unified end-to-end pipeline.

3. This point is then projected to a high-dimensional point in $[-1, 1]^D$ (i.e., D = 5).
   The first synthetic knob is mapped to both hybrid knobs (but with opposite ± signs), while the second one maps to the
   remaining three.

4. Next, we normalize all knob values to [0, 1], and apply special value biasing only for the two hybrid knob values.

5. Finally, all normalized values are converted to the corresponding physical knob values.

Experiment Setup

Hardware : CloudLab

Tuning Settings : PostgreSQL v9.6 as our target DBMS
select a set of tunable 90 knobs that could affect DBMS performance.
repeat each experiment five times using different random seeds as input to our optimizer.
first 10 iterations are generated randomly using LHS.

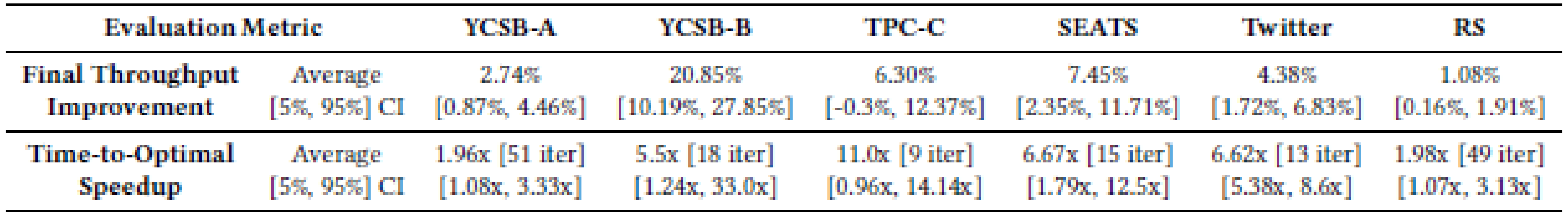
Workload : six workloads

Optimizers : we use two state-of-the-art BO-based optimizers: SMAC, and GP-BO.
Evaluation Metrics : final DBMS throughput (latency) improvement and time-to-optimal throughput (latency).

Efficiency Evaluation

Optimizing for Throughput

Table 4: Evaluation of LlamaTune compared to SMAC. We report both average and [5%, 95%] confidence interval numbers.



Evaluation Metric		YCSB-A	YCSB-B	TPC-C	SEATS	Twitter	RS
Final Throughput Improvement	Average	2.74%	20.85%	6.30%	7.45%	4.38%	1.08%
	[5%, 95%] CI	[0.87%, 4.46%]	[10.19%, 27.85%]	[-0.3%, 12.37%]	[2.35%, 11.71%]	[1.72%, 6.83%]	[0.16%, 1.91%]
Time-to-Optimal Speedup	Average	1.96x [51 iter]	5.5x [18 iter]	11.0x [9 iter]	6.67x [15 iter]	6.62x [13 iter]	1.98x [49 iter]
	[5%, 95%] CI	[1.08x, 3.33x]	[1.24x, 33.0x]	[0.96x, 14.14x]	[1.79x, 12.5x]	[5.38x, 8.6x]	[1.07x, 3.13x]

First, LlamaTune reaches the performance of the baseline SMAC optimum configuration **by ~ 5.62X faster on average**.

For four of the total six workloads, it **reaches that performance before iteration 20**, which highlights its increased sample efficiency.

Second, LlamaTune can improve the average final throughput (after 100 iterations) on all workloads, **by ~7.13% on average**, compared to vanilla SMAC

Third, even when we account for the variance from different runs (i.e. [5%, 95%] confidence interval), **LlamaTune outperforms the baseline average performance on all experiments.**

Efficiency Evaluation

Optimizing for Tail Latency

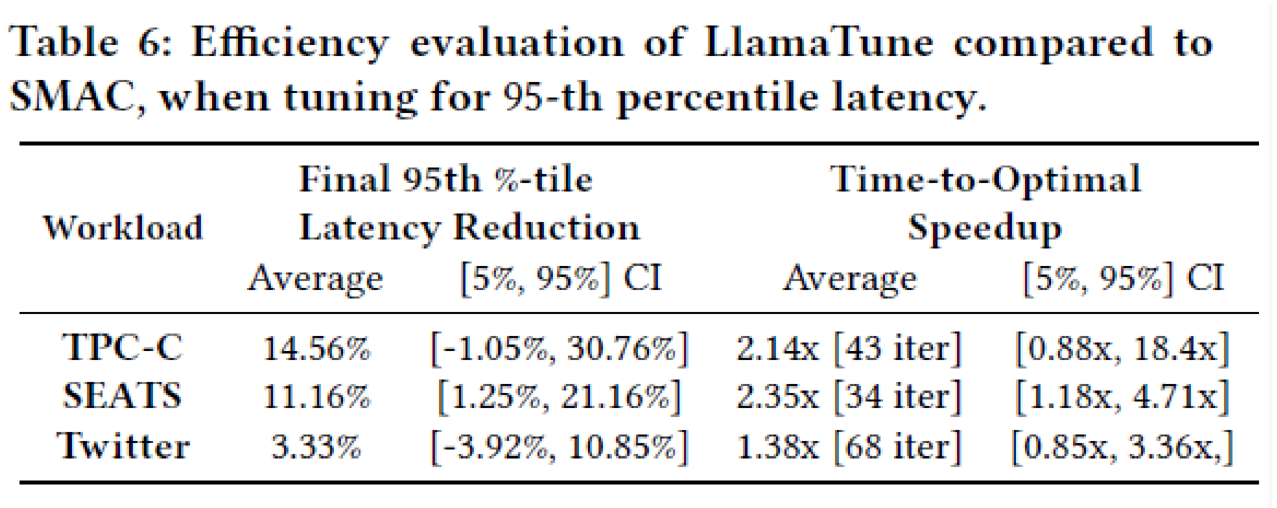


Table 6: Efficiency evaluation of LlamaTune compared to SMAC, when tuning for 95-th percentile latency.

Workload	Final 95th %-tile Latency Reduction		Time-to-Optimal Speedup	
	Average	[5%, 95%] CI	Average	[5%, 95%] CI
TPC-C	14.56%	[-1.05%, 30.76%]	2.14x [43 iter]	[0.88x, 18.4x]
SEATS	11.16%	[1.25%, 21.16%]	2.35x [34 iter]	[1.18x, 4.71x]
Twitter	3.33%	[-3.92%, 10.85%]	1.38x [68 iter]	[0.85x, 3.36x,]

2, 000 requests per second for TPC-C, 8, 000 for SEATS, and 60, 000 for Twitter.

We observe that **LlamaTune outperforms** the baseline SMAC on all three workloads.

On average LlamaTune yields a **~1.96X time-to-optimal speedup**, and **~9.68% better final tail latency.**

Ablation Study

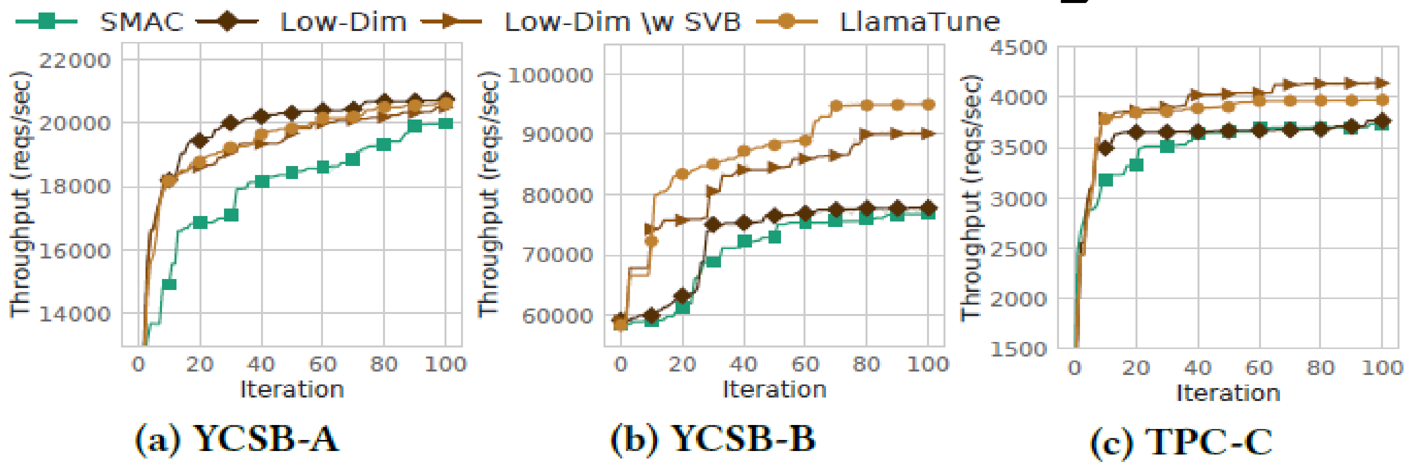


Figure 9: Ablation study for the three components of LlamaTune.[3]

For YCSB-B we see that only using HeSBO-16 achieves ~2X time-to-optimal speedup, while using the other techniques this speedup is improved to ~5.5X.
With all our methods applied on top of each other, both the final throughput and the convergence curve clearly improve.

For YCSB-A, the plain HeSBO-16 projection performs the best, while it seems that the special value biasing hinders LlamaTune's convergence.
This is because special knobs values do not improve the YCSB-A performance.

For TPC-C, we observe that HeSBO-16 and special value biasing do have a positive effect on optimization performance.
Yet, we note a small performance degradation when applying the search space bucketization.

Ablation Study

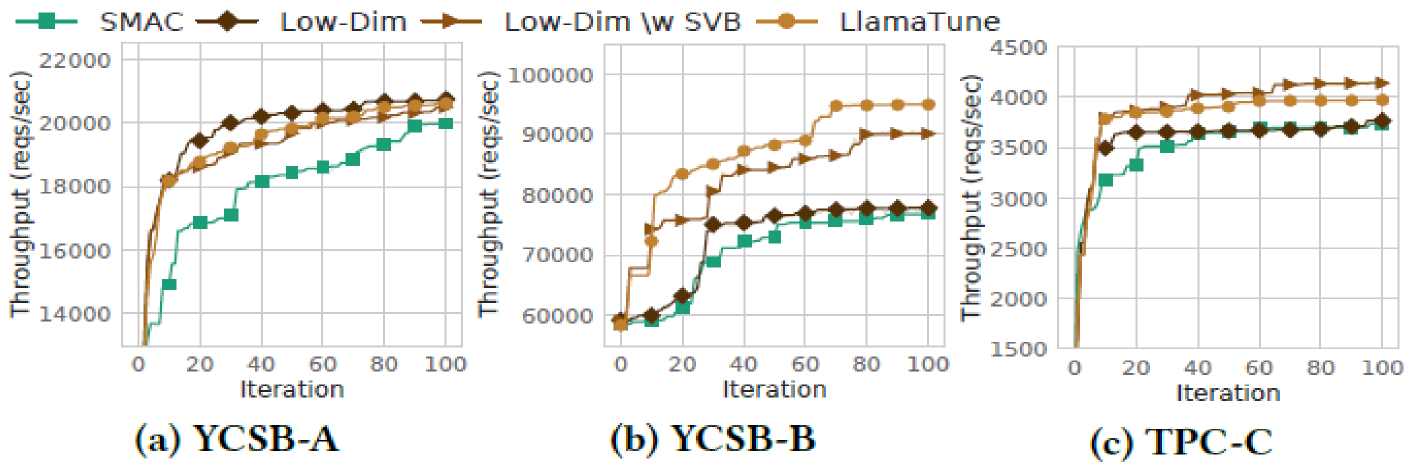
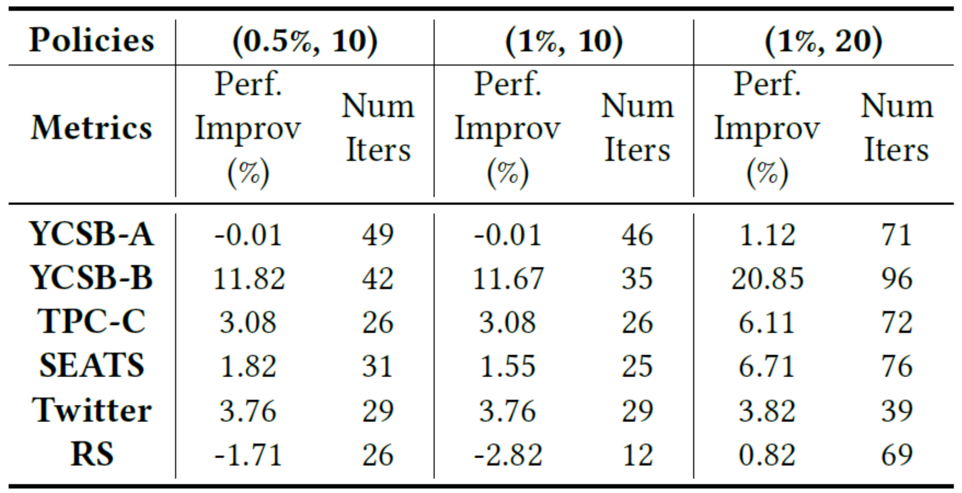


Figure 9: Ablation study for the three components of LlamaTune.[3]

The result of LlamaTune's tuning pipeline primary limitation, which imposes a bucketized space for all knobs, not just for the ones with a large number of unique values.

Finally, we note that even though the integration of special value biasing and space bucketization does not always help.

Deployment Scenario



Policies	(0.5%, 10)		(1%, 10)		(1%, 20)	
Metrics	Perf. Improv (%)	Num Iters	Perf. Improv (%)	Num Iters	Perf. Improv (%)	Num Iters
YCSB-A	-0.01	49	-0.01	46	1.12	71
YCSB-B	11.82	42	11.67	35	20.85	96
TPC-C	3.08	26	3.08	26	6.11	72
SEATS	1.82	31	1.55	25	6.71	76
Twitter	3.76	29	3.76	29	3.82	39
RS	-1.71	26	-2.82	12	0.82	69

Table 8: Evaluation for three early-stopping policies. We report mean perf. improvement, and the iteration when the session was stopped.

Our goal here is to study how the gains in terms of convergence speed (i.e. time-to-optimal) can be realized in this setting.

(minimum best performance improvement , the maximum number of K iterations to wait for this improvement to be made).

For ResourceStresser, the policy terminates very early (12th iteration), which results in slight decrease in final throughput.

We can alleviate this behavior by choosing a more conservative policy.

Optimizer Overhead

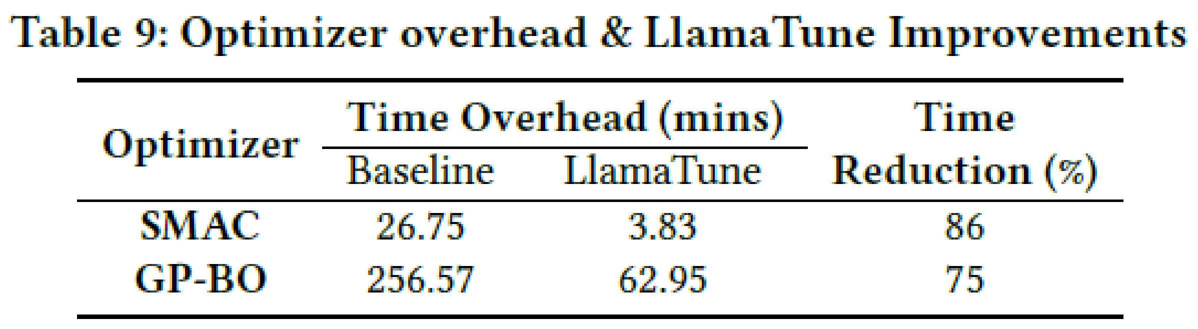


Table 9: Optimizer overhead & LlamaTune Improvements

Optimizer	Time Overhead (mins)		Time Reduction (%)
	Baseline	LlamaTune	
SMAC	26.75	3.83	86
GP-BO	256.57	62.95	75

We measure the **time taken by the optimizer to suggest the next configuration** across the entire tuning session (i.e., 100 iterations).
+)This time includes updating the underlying surrogate models

The **reduced number of dimensions**, resulting from the low-dimensional projection **is the primary reason**, as the optimizers have to model a much smaller space.

This is especially important for the GP-BO optimizer, due to the non-linear sampling behavior of GPs.