SIGMOD PODS BERLIN GERMANY 2025

Autotuning Systems

Techniques, Challenges, and Opportunities

https://aka.ms/sigmod-2025-autotuning-tutorial

BRIAN KROTH, SERGIY MATUSEVYCH, YIWEN ZHU

MICROSOFT GRAY SYSTEMS LAB

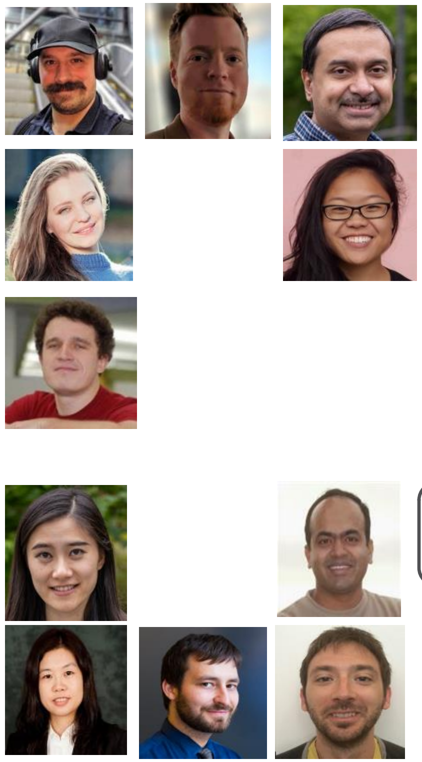
Outline

- Overview (15 mins)
- Offline Tuning (45 mins)
  - Basic Architectural Overview
  - Running Example
  - Optimization
    - Classic Search
    - Bayesian Optimization
  - Systems Challenges
- Online Tuning (20 mins)
  - Basic Architectural Overview
  - Optimization
    - Reinforcement Learning (RL)
    - Genetic Algorithms (GA)
  - Systems Challenges
- Future Directions (10 mins)
  - Workload Identification

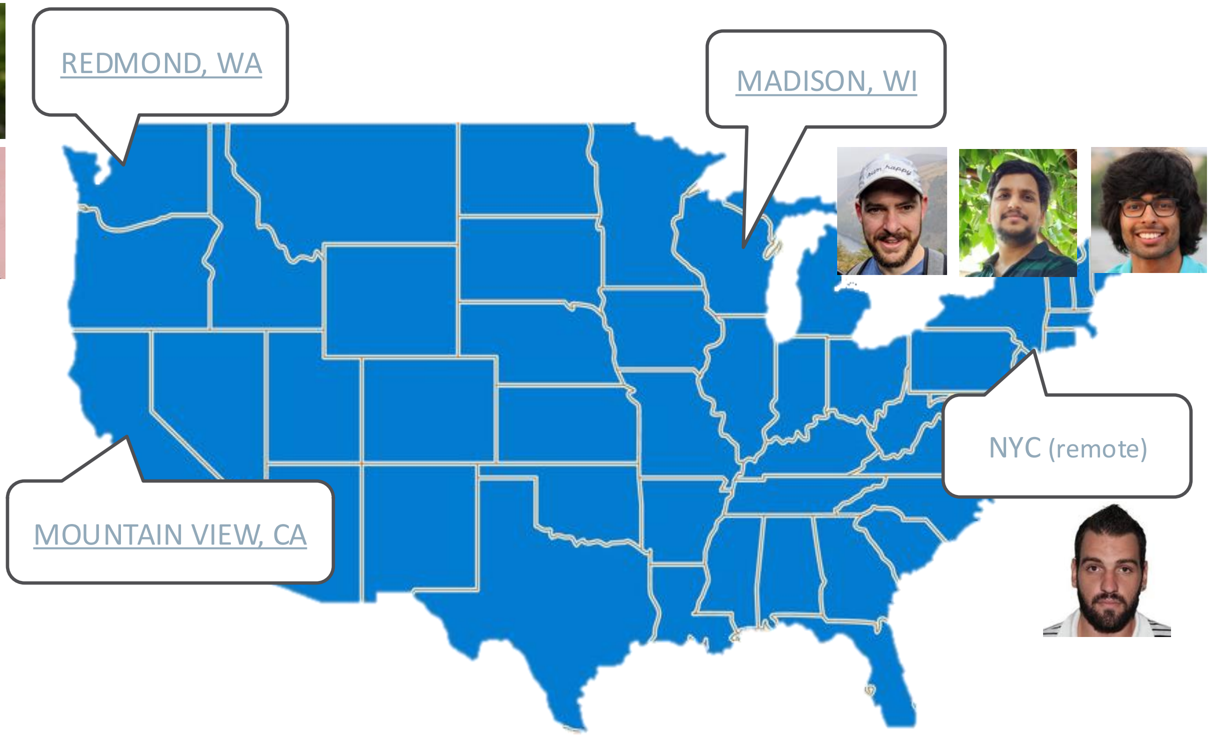
We are interested in interns and collaborations!
Come find us at the Microsoft booth ☺

Gray Systems Lab

https://aka.ms/gsl



(remote)



GSL is an applied and embedded research group, comprised of Data-Scientists, Engineers, and Researchers.

Team

[Brian Kroth]

General systems nerd from UW-Madison with years of experience in both industry and research and generally excited to mentor and learn from others and make things go faster and more efficiently.

Along with Sergiy, Brian is one of the main developers and leads of the MLOS framework for generalized systems autotuning at MS.

[Sergiy Matusevych]

Sergiy is a data scientist, engineer, researcher, and passionate hacker, among other things.

At GSL, he applies machine learning for systems optimization and builds ML models for workload identification and analysis.

You can identify Sergiy by a large camera that he always has in his hand.

[Yiwen Zhu]

Yiwen is a Principal Scientist at Microsoft's Gray Systems Lab (GSL). Her research interests center on the vision of autonomous cloud systems, utilizing machine learning, statistical inference, and operation research techniques.

What is "Autotuning"?

Efficiently auto selecting a system configuration

for a workload and its execution environment

to improve (optimize) one or more metrics

- Sampling of prior works:
  - **DB**: OtterTune, DBTune, CDBTune, DB-Bert, GPTuner, …
  - **HPC**: ATLAS, CLTune
  - **Compilers**: MILEPOST GCC,
  - **Cloud**, **Resource Management**: Autoscalers (e.g., CaaSPER)
  - **ML Hyperparams** (AutoML): Keras Tuner, skopt, optuna, BoTorch, HyperMapper, …
  - **Generic**: OpenTuner, MLOS

Where is “Autotuning”?

```
max_connections = 40
shared_buffers = 8GB
effective_cache_size = 24GB
maintenance_work_mem = 2GB
checkpoint_completion_target = 0.9
wal_buffers = 16MB
default_statistics_target = 100
random_page_cost = 1.1
effective_io_concurrency = 200
work_mem = 174762kB
huge_pages = try
min_wal_size = 2GB
max_wal_size = 8GB
max_worker_processes = 8
max_parallel_workers_per_gather = 4
max_parallel_workers = 8
max_parallel_maintenance_workers = 4
```

Efficiently auto selecting a <u>system configuration</u>

for a workload and its execution environment

to improve (optimize) one or more metrics

- VM Size
- DB Indexes
- CPU Speed, Cache Size/Associativity
- …
- Talk Focus: *<u>Software Configurations</u>*
  - Sometimes called “knob tuning”
  - What level?
    - *Build Time* (e.g., compile options)
    - *Startup Time* (e.g., BP size)
    - *Runtime* (e.g., QO rules, buffer sizes, etc.)
  - “Where” affects deployment, tuning options

How to “Autotuning”?

***Efficiently auto selecting* a system configuration**

for a workload and its execution environment

to improve (optimize) one or more metrics

- Heuristics
  - Encoded “best practices”
    - E.g., mysqltuner, pgtune, …
- Search Based
  - Grid search
  - Simulated Annealing
- Model Guided
  - Bayesian Optimization
  - Reinforcement Learning
- Key: ***sample efficiency***
- *More in the rest of the talk*

Where and What are we "Autotuning"?

Efficiently auto selecting a system configuration

for a <u>workload</u> and its execution <u>environment</u>

to improve (optimize) one or more metrics

- "Execution Environment"
  - HW Config (CPU, RAM, Disk, Network, GPU, …)
  - VM Size
  - OS
  - System: Redis, MySQL, Postgres, Nginx, …
  - …
- "Workload"
  - YCSB
  - TPC-C
  - TPC-H
  - Other?
    - User or customer workloads
  - …

"*Context*"

What are we "Autotuning" for?

Efficiently auto selecting a system configuration

for a workload and its execution environment

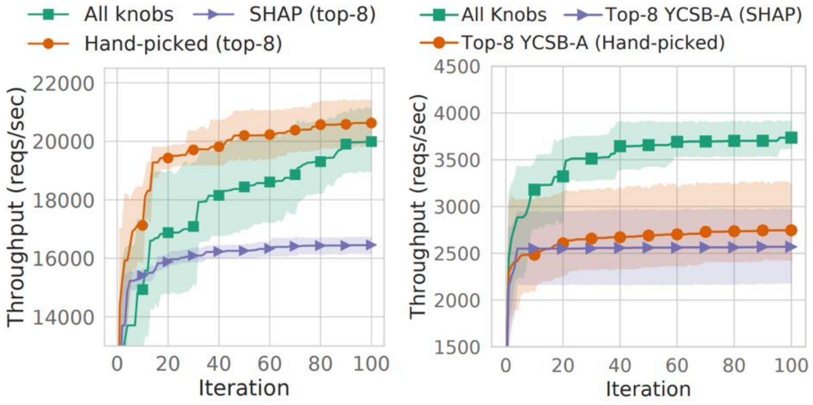
to <u>improve</u> (optimize) one or more <u>metrics</u>

- Minimize Latency
  - Avg, Med, P95, …
- Maximize Throughput
- Minimize Cost
- Minimize Resource Usage
  - Pack more into less with good perf
  - Reduce power!
- Maximize "Robustness"
  - Availability
  - Sensitivity to changes in environment

All of these? *At once*?

Why Tune? – Performance!

- “Properly tuned database systems can achieve *4-10x* higher throughput” ([Van Aken, VLDB 2021](#))
- 68% reduction in P95 latency for Redis
  - Tuning Kernel Scheduler Parameters

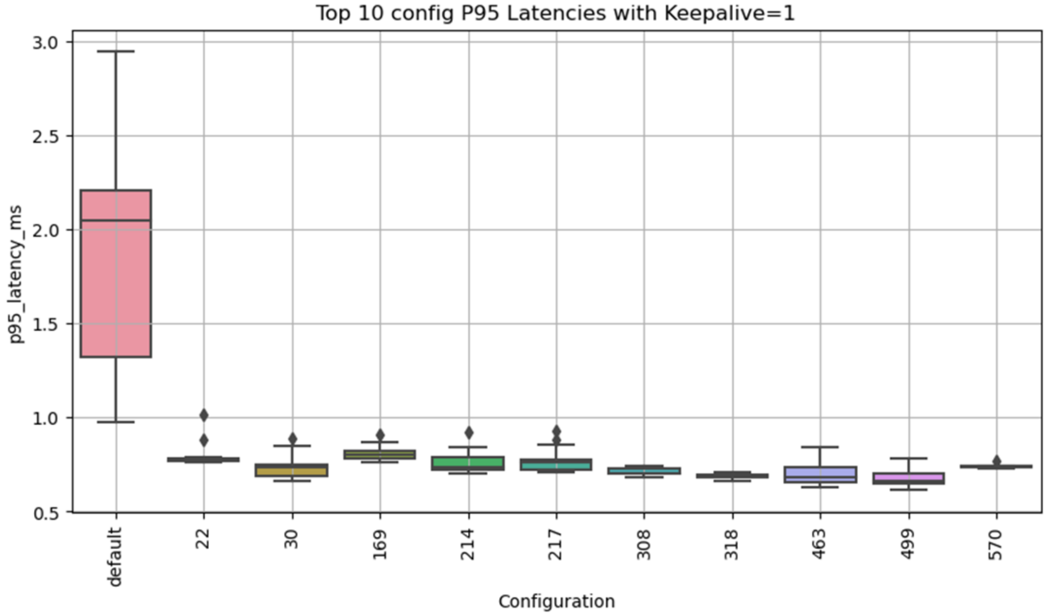


(a) SHAP vs Manual (YCSB-A) **(b) Top-8 YCSB-A to TPC-C**

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

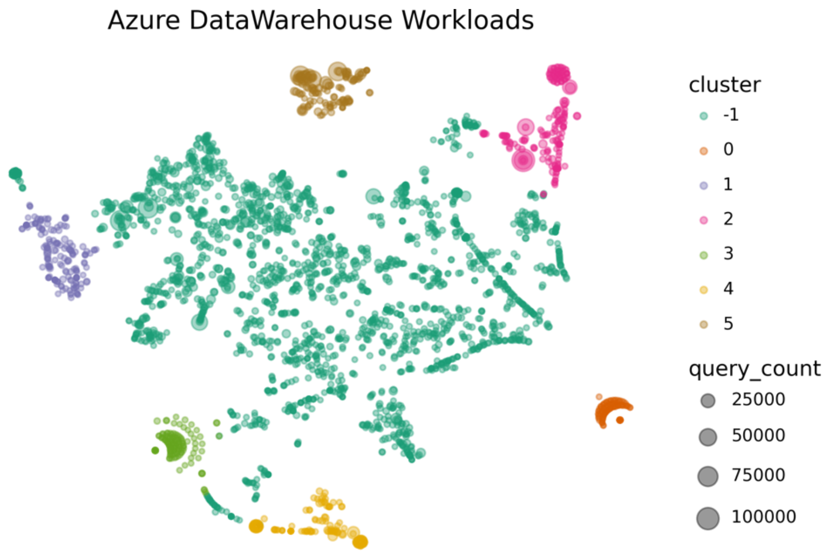
[LlamaTune: Sample-Efficient DBMS Configuration Tuning (VLDB 2022)](#)

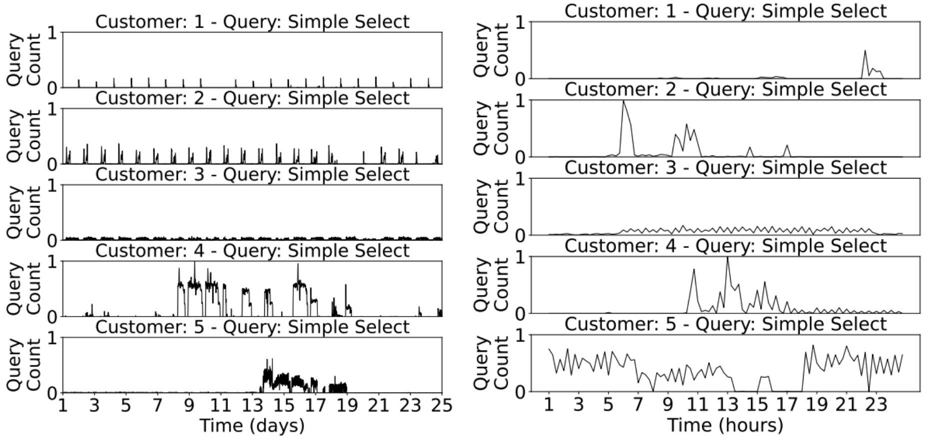
- Better user experience 🤩
- Lower costs 💰
- Fewer machines, CAPEX, OPEX, ***power*** 🏭
- …



Why *Autotune*?

- Cloud Scale
  - Growing # of HW/workload
  - Expectation of “automagic”
  - Not so many DBAs or Sysadmins





(a) January 2025. **(b) January 15 2025.**

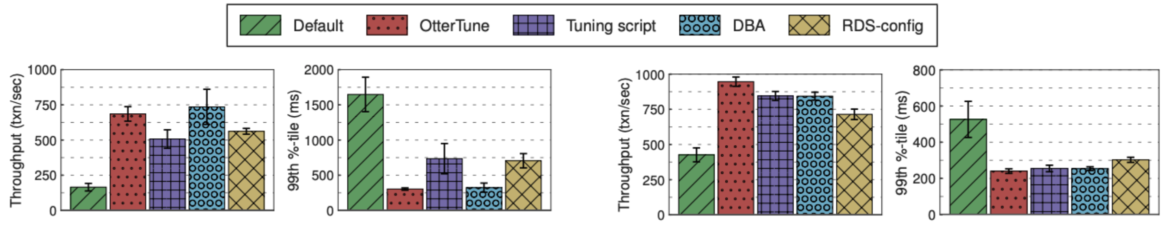
Figure 3: Arrivals of simple select queries over a month and a day for five customers. Query count (y-axis) is min-max normalized separately for monthly and daily arrivals.

Advancing Workload Management with Foundational Models: Challenges in Time Series Similarity and Interpretability. Bang et al., MIDAS 2025

[CDBTune](#) (SIGMOD 2019)
[OtterTune](#) (SIGMOD 2017)

Why is Autotuning Hard?

- Large, and *increasing* # of parameters
- Complex system interactions affect performance
- Not easy, takes time, even for experts!

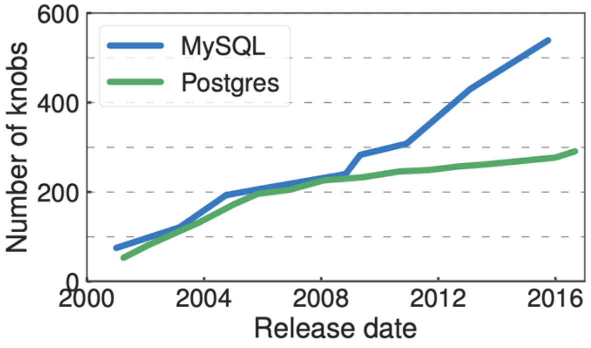


(a) TPC-C (Throughput) (b) TPC-C (99%-tile Latency)

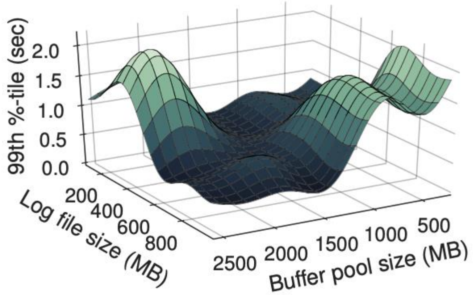
Figure 10: Efficacy Comparison (MySQL) – Throughput and latency measurements for the TPC-C benchmark using the (1) default configuration, (2) OtterTune configuration, (3) tuning script configuration, (4) Lithuanian DBA configuration, and (5) Amazon RDS configuration.

(a) TPC-C (Throughput) (b) TPC-C (99%-tile Latency)

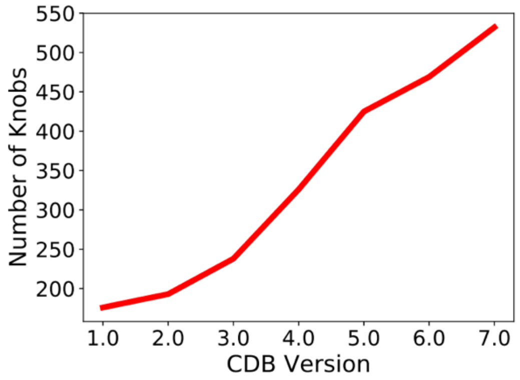
Figure 11: Efficacy Comparison (Postgres) – Throughput and latency measurements for the TPC-C benchmark using the (1) default configuration, (2) OtterTune configuration, (3) tuning script configuration, (4) expert DBA configuration, and (5) Amazon RDS configuration.



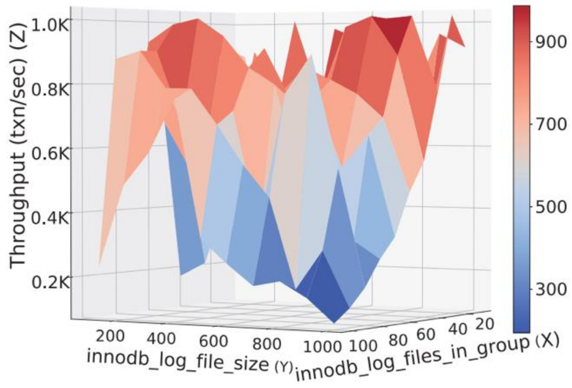
(d) Tuning Complexity



(a) Dependencies



(c) Knobs Increase



(d) Performance surface

Motivating Example: Spark Tuning Game

- testautotune2.azurewebsites.net/app3

Exercise (to do now):

1. Manually optimize TPC-H Q1 runtime
2. Limit 5min and 100 tries
3. Download Data and upload in chat
4. Post your best perf #

Aside: why even have parameters?

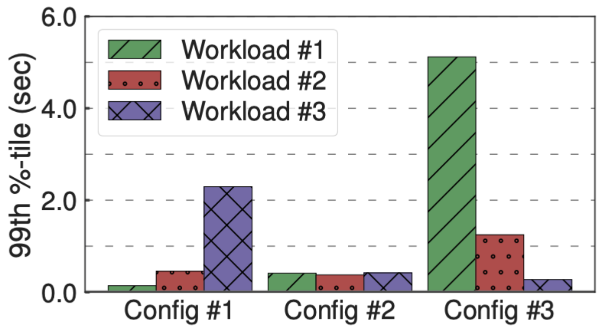
- Build SW to be *adaptive*?
  - Examples:
    - Network Protocols - TCP
    - Autoscaling
    - Load shedding/backpressure
    - DB Index Cracking (Idreos CIDR 2007)
    - Adaptive Query Processing
    - Self Driving Databases (Pavlo CIDR 2017)
  - Cost/complexity to rebuild

- *Still have tunables*
  - Internal vs. External (more later)
  - Different policies/techniques
    - TCP: tahoe, reno, vegas, cubic, BBR
    - AQP: Eddies, SIP, etc.
  - Threshold to kick in
  - Rate to change
  - Backoff delay
  - …

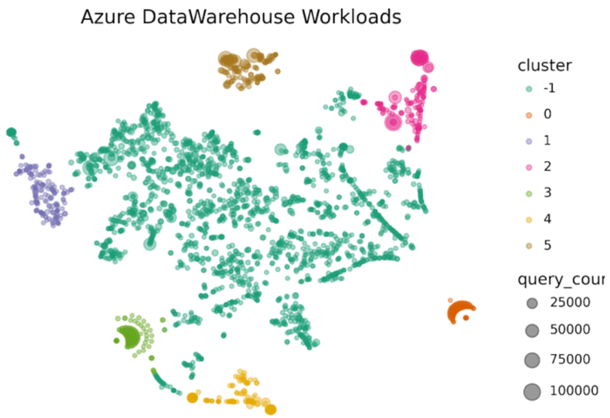
Complementary approaches

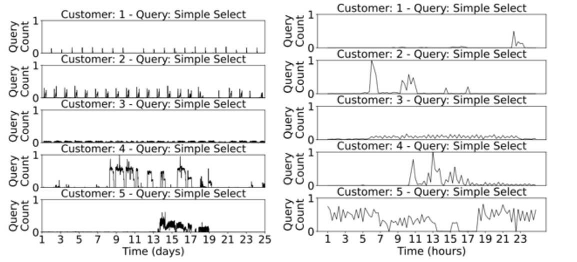
Why is Autotuning Hard? - Workloads

- No “one config to rule them all”
- One workload may *change* over time
- Many, many workloads in the Cloud
- Lack of representative benchmarks
- Not clear how to match them
  - Workload ID: more on this later



(c) Non-Reusable Configurations





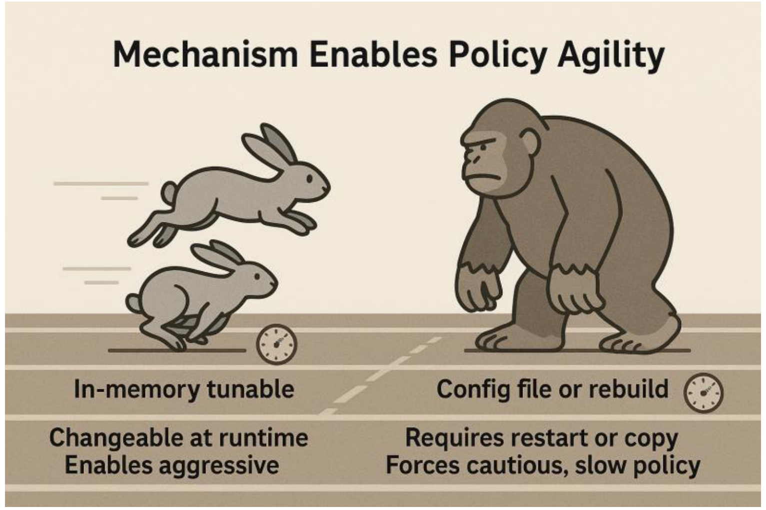
(a) January 2025. (b) January 15 2025.

Figure 3: Arrivals of simple select queries over a month and a day for five customers. Query count (y-axis) is min-max normalized separately for monthly and daily arrivals.

Autotuning in Practice: How to Deploy?

- *Depends on the tunables*:
  - Regularly ***runtime*** (online) adjustable?
    - E.g., join buffer size?
      - `ALTER SYSTEM CONFIGURATION SET tunable=new-val;`
    - Is there some lag before it takes affect?
  - Only at ***startup*** time?
    - E.g., PG `shared_buffers` size
      - `update_config_file new-tunables.json`
      - `systemctl restart postgres.service`
    - Is it expensive to restart?
      - E.g., do you lose buffer pool or cache contents?
    - May need to do this infrequently
  - Only at ***build** or **provision*** time?
    - E.g., FS choice or block size
      - Size of data operation to change
        - E.g., `mkfs && rsync`
    - Maybe just pick better defaults

- Classic "**Policy vs Mechanism**" system challenge



- N.B., in *some* cases, can work to make changing the tunable more adaptable.
  - Orthogonal engineering effort
    - improves mechanism → enable better policy

Deployment Oriented View

- Split the world into:

1. Offline Tuning
   - Two Phases
     1. Explore in a controlled "lab" environment
     2. Deploy "best" config to production
        - Key Issue: When? How?
   - \+ More flexible, expansive (though may crash)
   - \+ Parallel Exploration
   - \+ Easy to explain, rollback
   - \- What workload?
2. Online Tuning
   - Use an "agent" to continually observe and adjust the system
   - \+ Any workload
   - \- Safety? Explainability?
   - \- Generalizability?

- Somewhat artificial separation
- Can use both!
  - E.g., start from better "defaults" using offline
  - Fine-tune from there
- Online can also "pre-train" in an offline "gym"
- Common Challenges/Approaches
  - Size of search space
  - Predictability
  - Noise

Outline

- ~~Overview (15 mins)~~
- Offline Tuning (45 mins)
  - Basic Architectural Overview
  - Running Example
  - Optimization
    - Classic Search
    - Bayesian Optimization
  - Systems Challenges
- Online Tuning (20 mins)
  - Basic Architectural Overview
  - Optimization
    - Reinforcement Learning (RL)
    - Genetic Algorithms (GA)
  - Systems Challenges
- Future Directions (10 mins)
  - Workload Identification

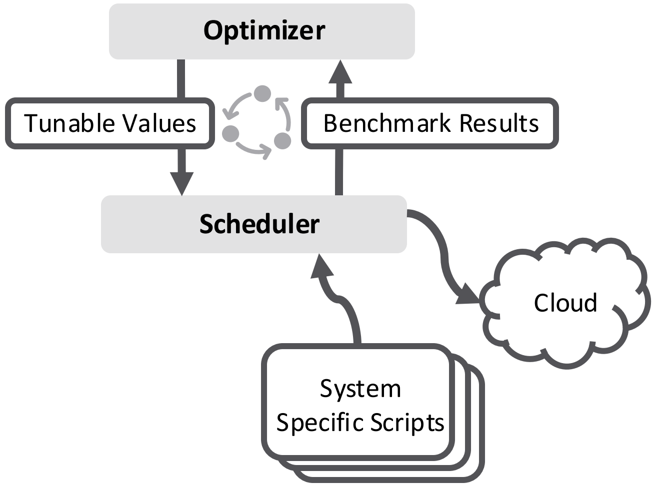
Offline Optimization

SERGIY MATUSEVYCH

Motivating Example

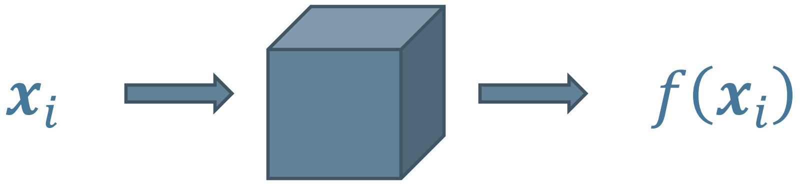
- **System to optimize**: Redis on Linux
- **Goal**: minimize tail latency
- **Benchmark**: Redis benchmark
- **Tunable parameter**: `/proc/sys/kernel/sched_migration_cost_ns`

- Note:
  - We optimize the OS for the benefit of 1 application (and workload)
  - All other configuration parameters fixed (e.g., VM size)
  - We already see the benefits of benchmark automatization



Problem Statement

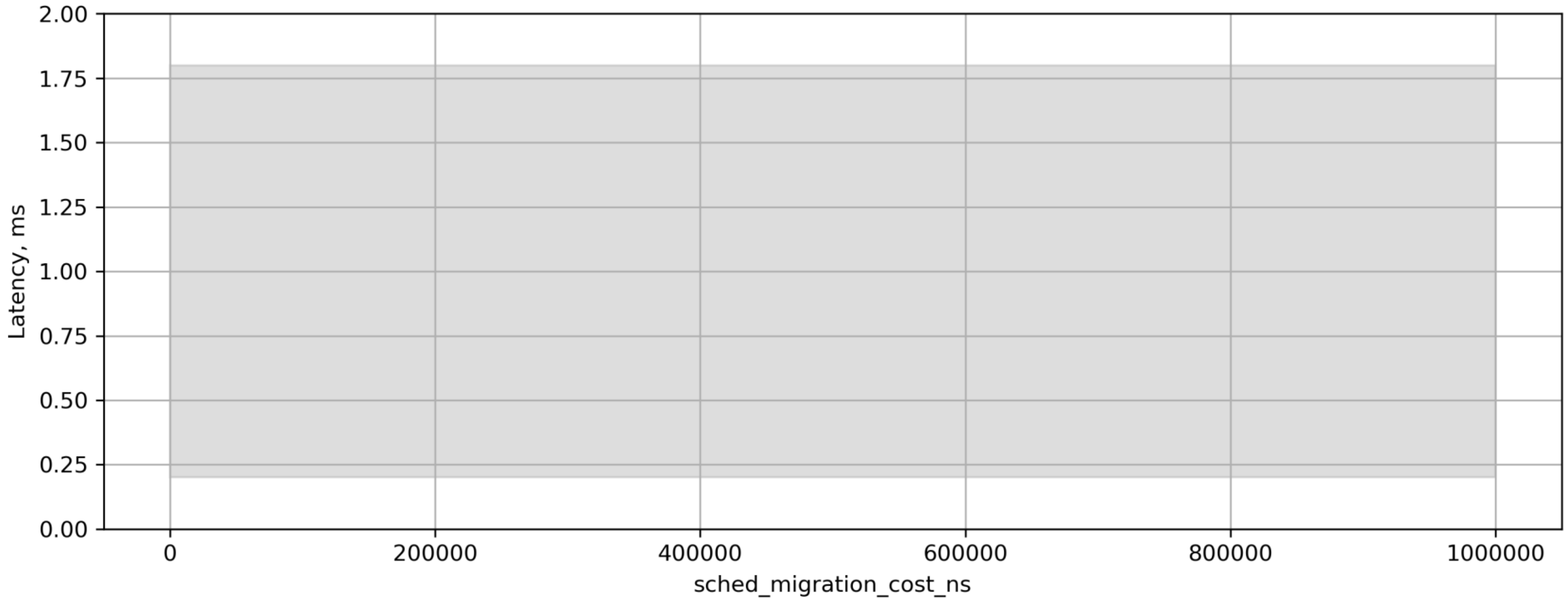
- Optimize expensive **black-box** function in a sample-efficient manner:



$$\boldsymbol{x}^* = \underset{\boldsymbol{x} \in \mathcal{X}}{\operatorname{argmin}} f(\boldsymbol{x})$$

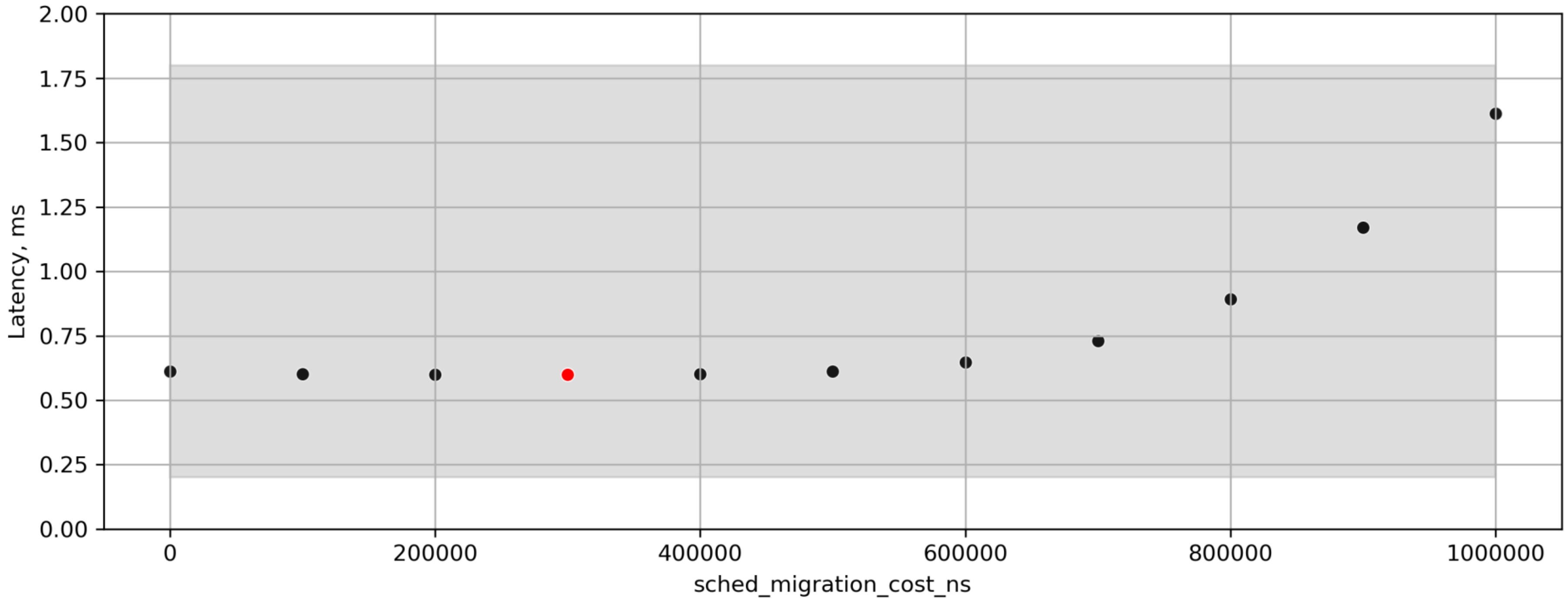
Configuration Space

- Use prior knowledge about the system:
  - Latency ≈ 1.0ms, `sched_migration_cost_ns` ∈ [0 .. 1 000 000]



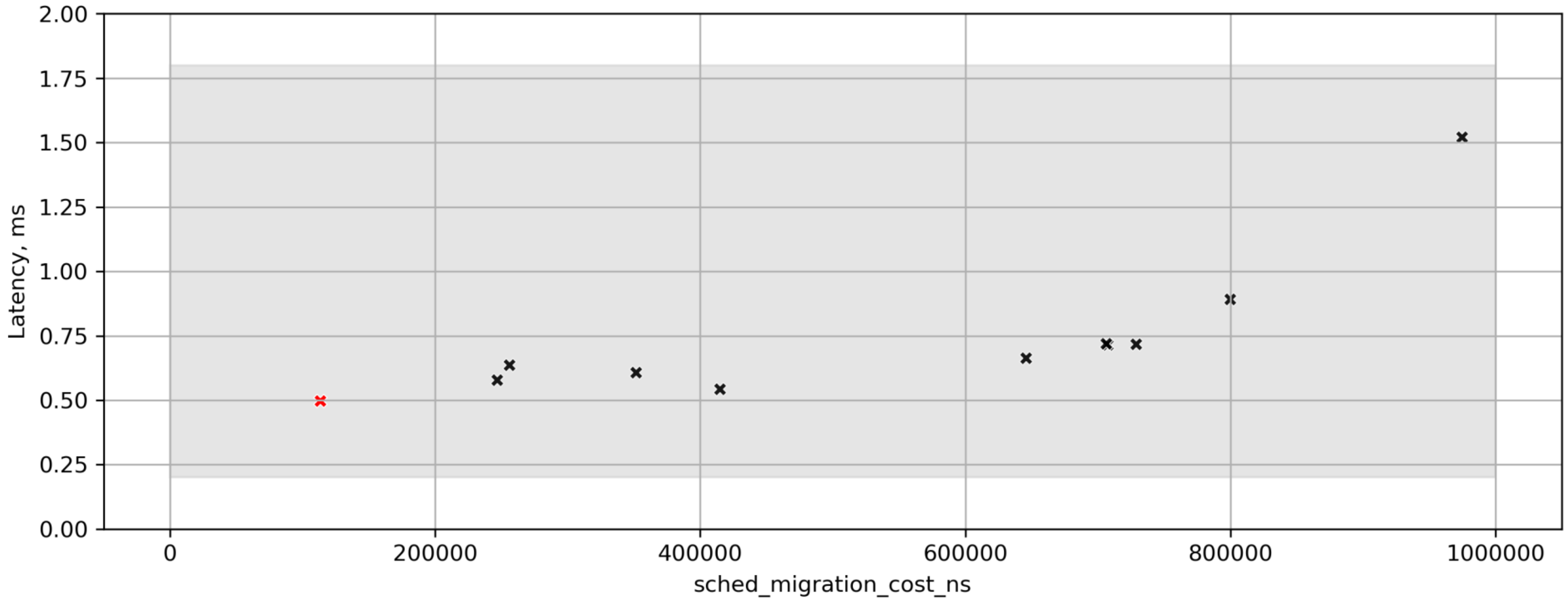
[Not so] Naïve Approach: Grid Search

- Idea: Fixed trial budget, pick values at **even intervals**
  - Try all configs, pick the best



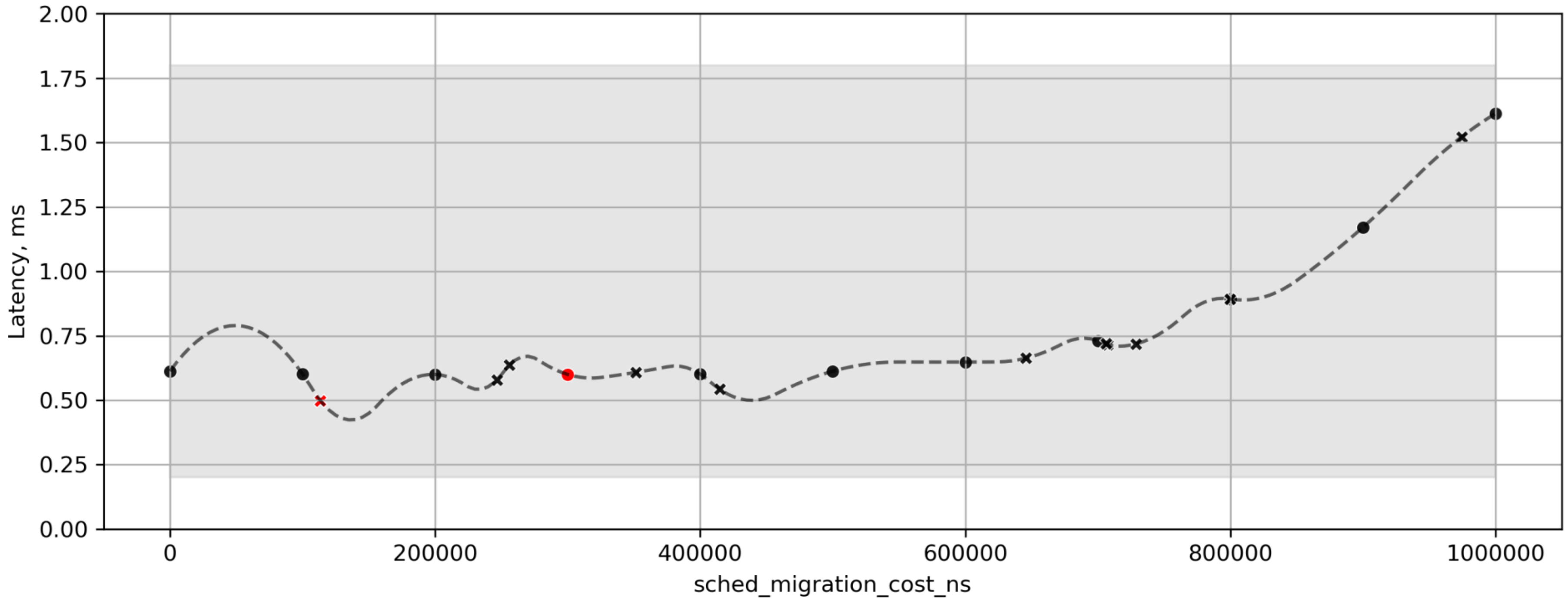
Variation: Random Search

- Idea: Fixed trial budget, pick configuration values at **random**
  - Try all configs, pick the best



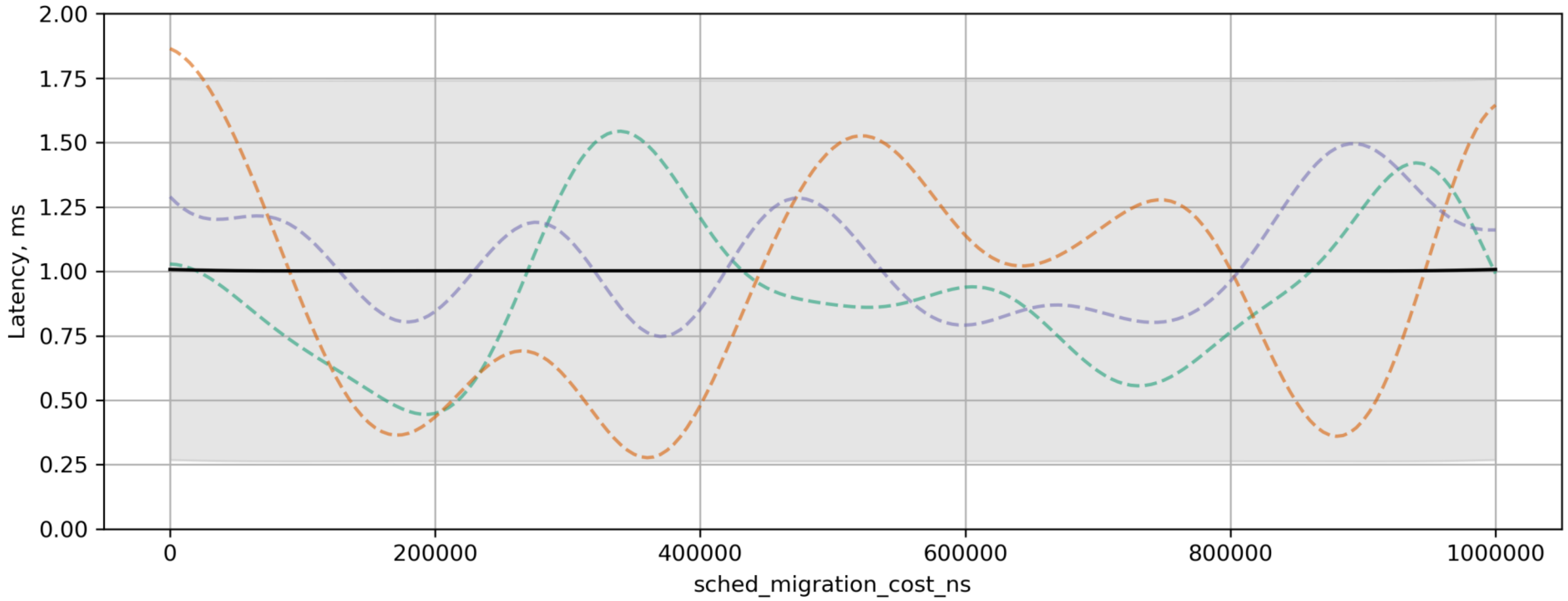
Problem: Sample Efficiency

- Idea: use the information from **previous** trials to pick the **next** configuration
  - Can we do it in a principled way?

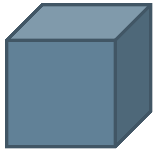


Bayesian Optimization

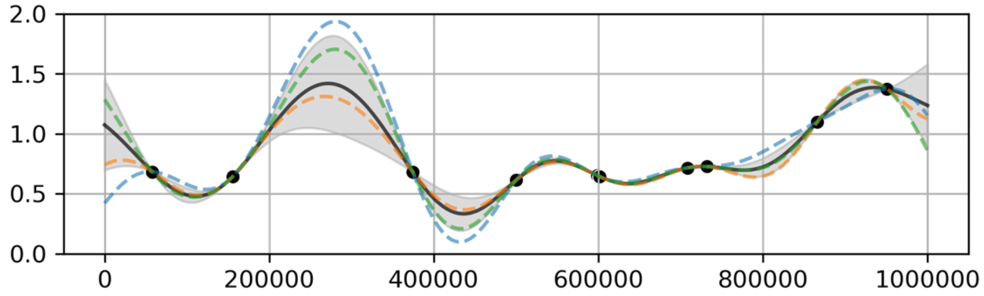
- **Idea:** instead of finding x^* given f, find best **model** of f, given the observations



Sequential Model-Based Optimization

1. Evaluate the expensive function: $\boldsymbol{x}_i \longrightarrow$  $\longrightarrow f(\boldsymbol{x}_i)$

2. Use $f(\boldsymbol{x}_i)$ to update the statistical model M:

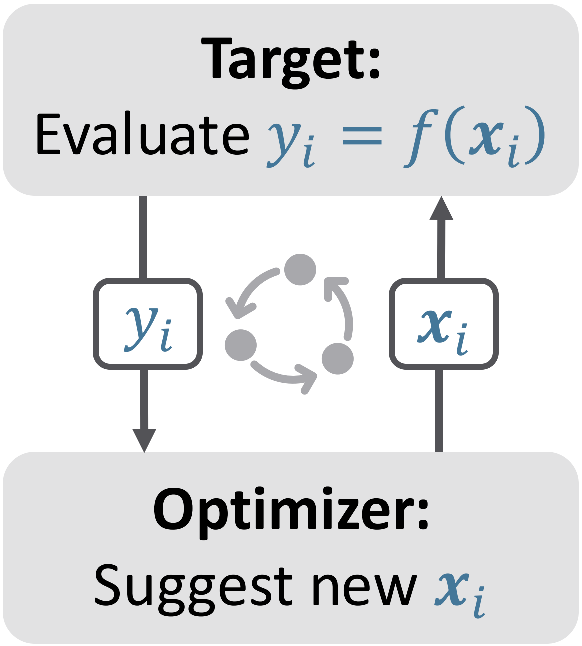


3. Optimize the **Acquisition Function**: $\boldsymbol{x}_{i+1} = \underset{\boldsymbol{x} \in \mathcal{X}}{\operatorname{argmax}} AF(M, \boldsymbol{x})$

4. $++i$; Repeat

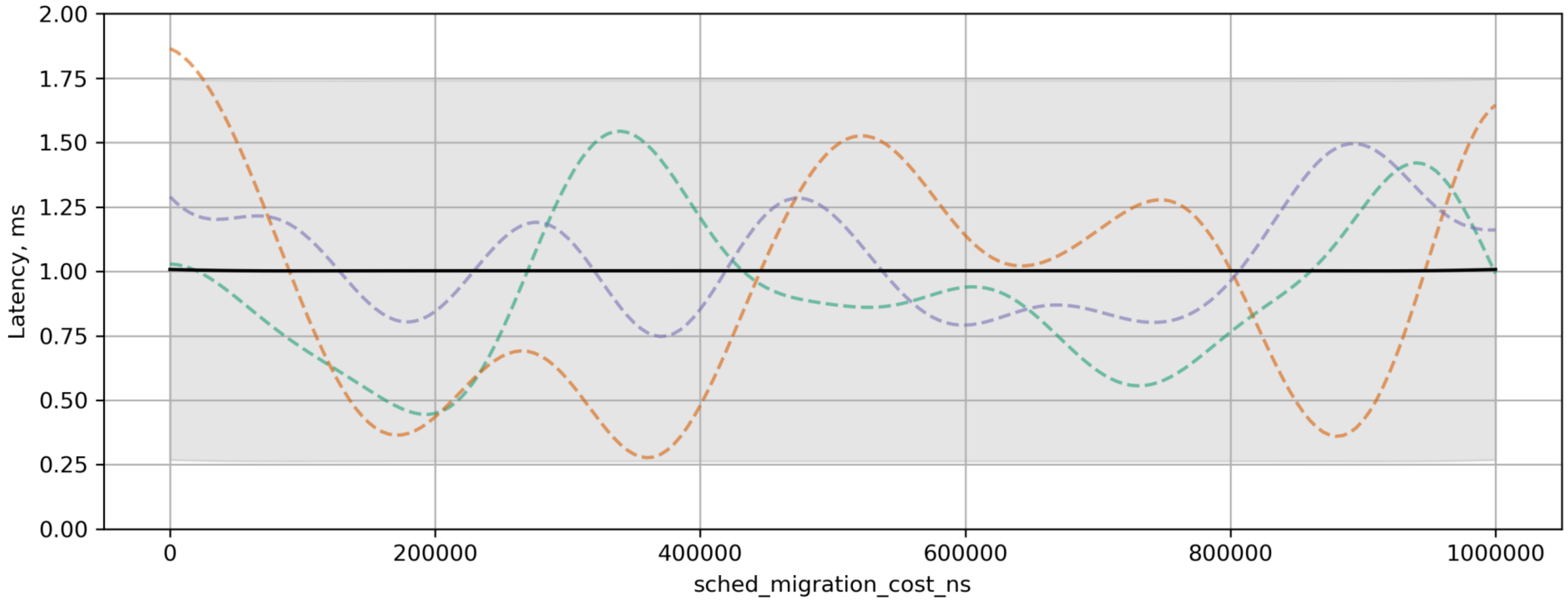
Optimizer as a Black Box

- Target function is a black box to the optimizer
- Optimizer is a black box to the target function
  - TF does not care where the suggestions come from
- One can build an elegant tuning framework



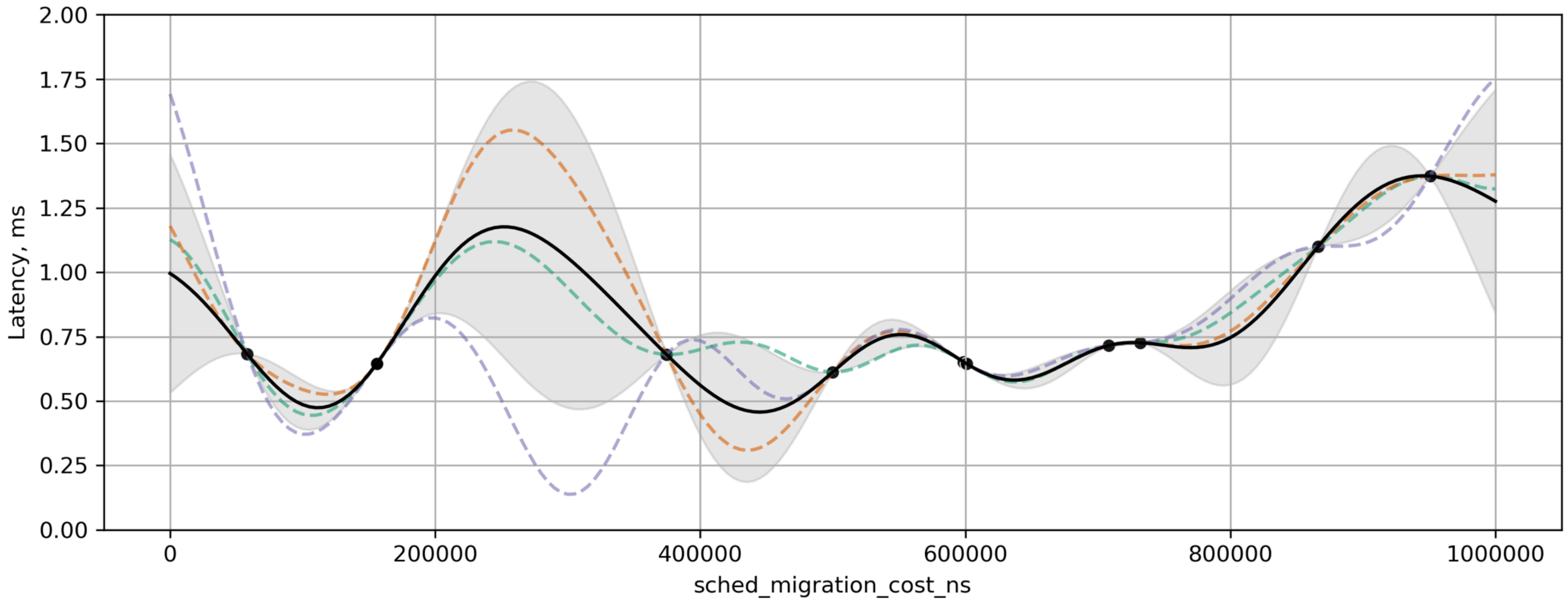
Model M: Gaussian Process

- Model **random functions**: $\hat{f} \sim \mathcal{GP}(\mu(\boldsymbol{x}), \Sigma(\boldsymbol{x}, \boldsymbol{x}'))$



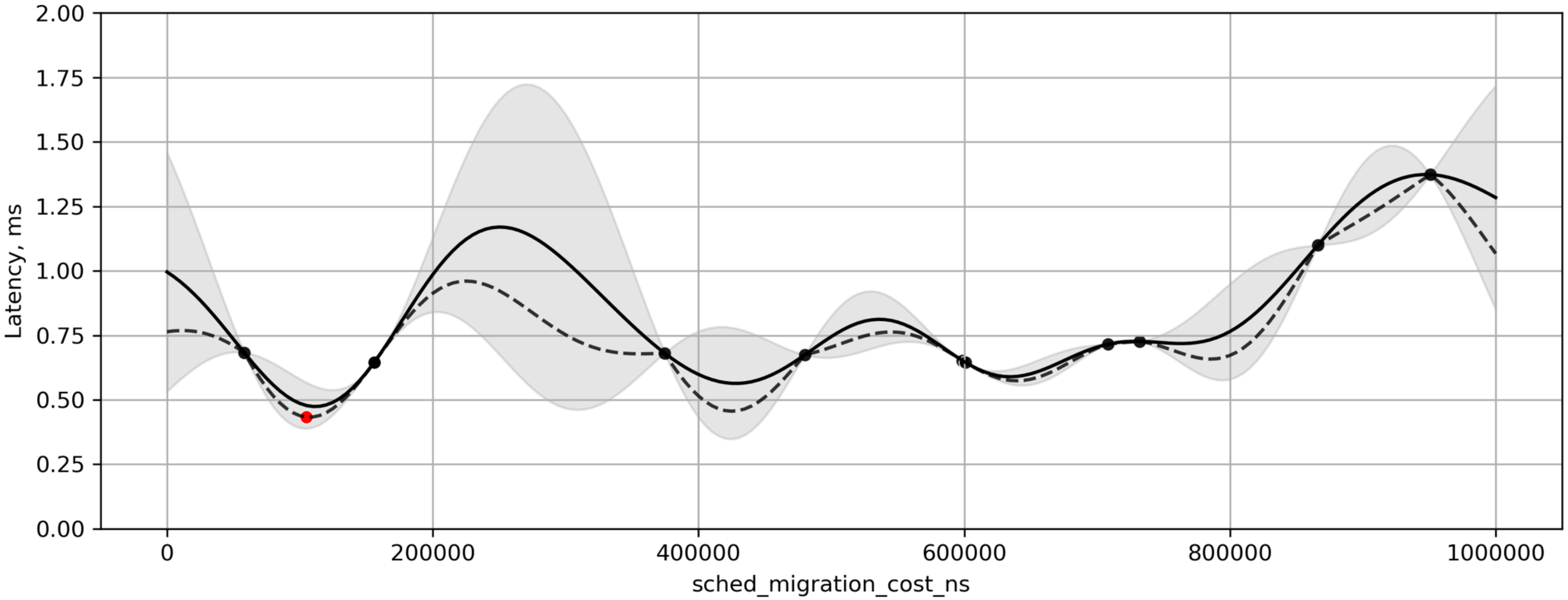
Bayesian Optimization

- Condition on observed points
- Extract the expected function and confidence interval

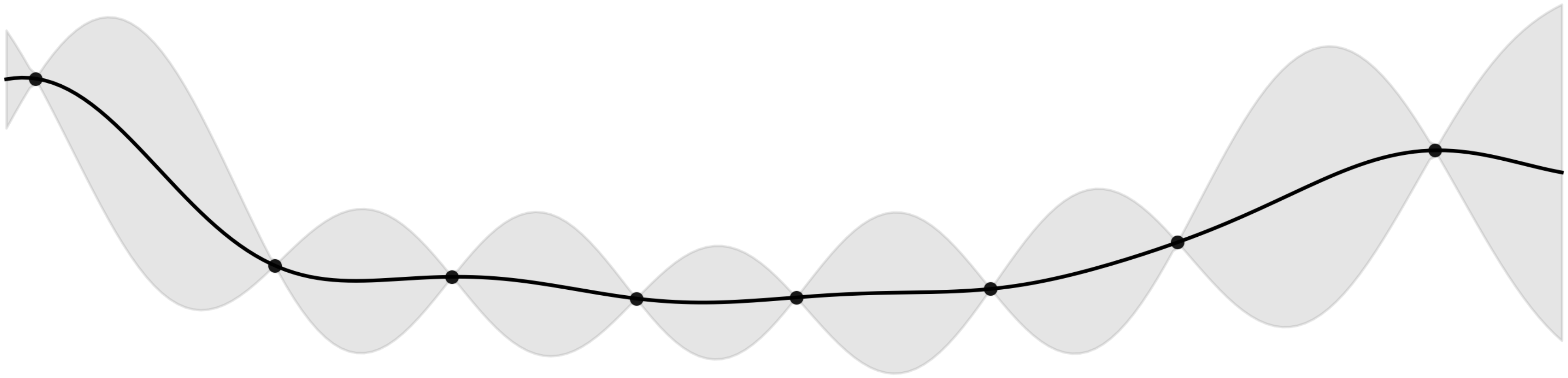


Bayesian Optimization

- **Surrogate function:** our best guess (so far) about the system behavior
- **Acquisition function:** pick the most “interesting” point to evaluate



How the Sausage is Made



Definitions

- **Stochastic Process:** An indexed sequence of random variables
- **Gaussian Process:** Model $\hat{f}(\boldsymbol{x})$ s.t. $\forall[\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots]$: $\left[\hat{f}(\boldsymbol{x}_1), \hat{f}(\boldsymbol{x}_2), \ldots\right] \sim \mathcal{N}(\mu, \Sigma)$

- Why Gaussian?
  - Normal distribution is closed under marginalization and conditioning
  - Leads to elegant closed-form solutions for optimization

Marginalization

- **Marginalization:** $\begin{bmatrix} Y_{obs} \\ Y_{mis} \end{bmatrix} \sim \mathcal{N}(\mu, \Sigma) = \mathcal{N}\left(\begin{bmatrix} \mu_{obs} \\ \mu_{mis} \end{bmatrix}, \begin{bmatrix} \Sigma_{obs,obs} & \Sigma_{obs,mis} \\ \Sigma_{mis,obs} & \Sigma_{mis,mis} \end{bmatrix}\right)$
- Missing data does not impact the inference:

$$\mathrm{P}(Y_{obs}) = \int \mathrm{P}(Y_{obs}, Y_{mis})\, dY_{mis} = \int \mathrm{P}(Y_{obs} | Y_{mis})\, \mathrm{P}(Y_{mis})\, dY_{mis}$$

- We can <u>update the model</u> with the new data points Y_{obs}!

Conditioning

- **Conditioning:**

$$Y_m | Y_o \sim \mathcal{N}\left(\mu_m + \Sigma_{m,o}\, \Sigma_{o,o}^{-1}\, (Y_o - \mu_o), \Sigma_{m,m} - \Sigma_{m,o}\, \Sigma_{o,o}^{-1}\, \Sigma_{o,m}\right)$$

- Probabilistic model for missing points given the observations!

From Distribution to Process

- **Gaussian Process:** distribution over functions $\hat{f} \sim \mathcal{GP}(m(\boldsymbol{x}), K(\boldsymbol{x}, \boldsymbol{x}'))$
- $\mathcal{GP}$ defined by:
  - **Mean function** $m(\boldsymbol{x})$: assigns to each $\boldsymbol{x}$ the expected value $\mathbb{E}[\hat{f}(\boldsymbol{x})]$
  - **Kernel function** $K(\boldsymbol{x}, \boldsymbol{x}')$: assigns to each pair $(\boldsymbol{x}, \boldsymbol{x}')$ covariance between $\hat{f}(\boldsymbol{x})$ and $\hat{f}(\boldsymbol{x}')$

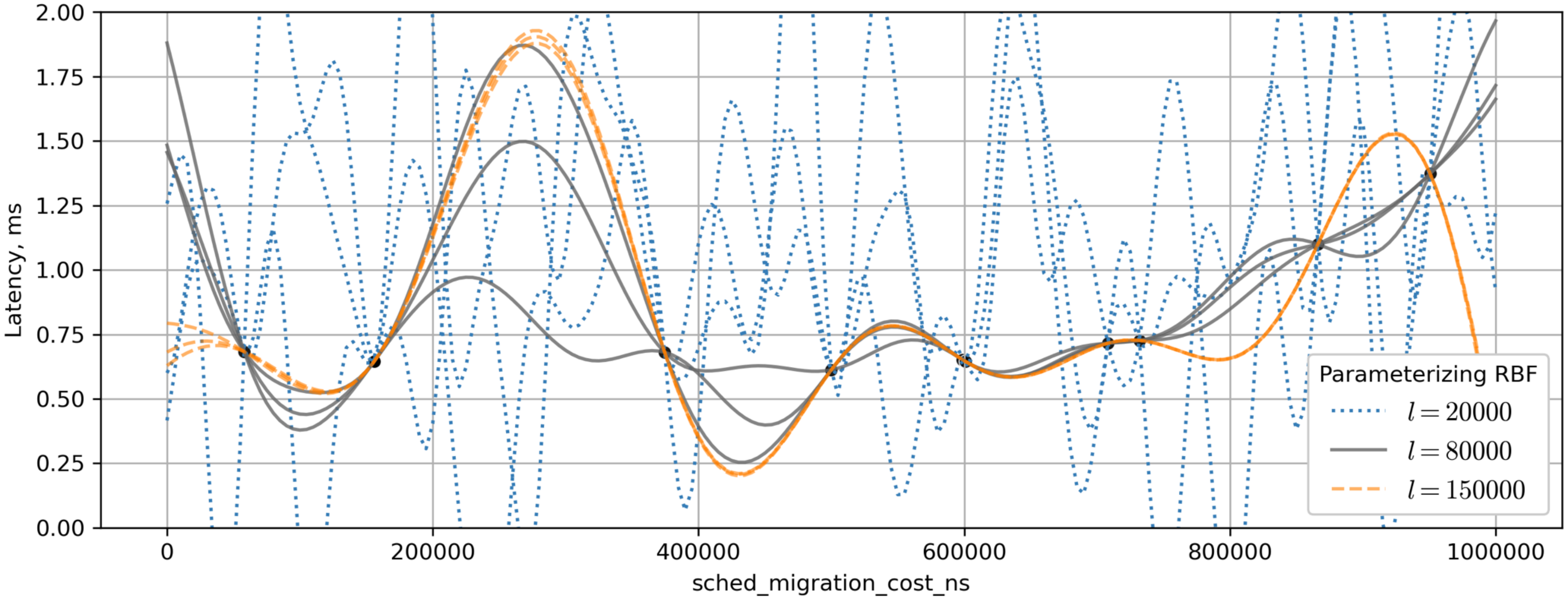
Kernel Functions

- Radial Basis Function (**RBF**): $\exp\left(-\frac{d^2}{2l^2}\right)$
  - scikit-learn default
- **Matérn**: $\frac{2^{1-\nu}}{\Gamma(\nu)}\left(\sqrt{2\nu}\frac{d}{l}\right)^{\nu} K_\nu\left(\sqrt{2\nu}\frac{d}{l}\right)$
  - Most popular kernel nowadays
  - Two parameters to control smoothness: l and ν
  - Becomes RBF at $\nu \to \infty$
- Many others exist: **Constant**, **Linear**, **Periodic**, etc.
  - Kernels can be combined

- d: distance between $\boldsymbol{x}$ and $\boldsymbol{x}'$
- d is usually Euclidean: $d = \|\boldsymbol{x} - \boldsymbol{x}'\|_2$
- $\Gamma(\nu)$: gamma function
- K_ν: modified Bessel function of order ν

Kernel Functions: RBF

- **Radial Basis Function** (RBF): $K(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{\|\boldsymbol{x}-\boldsymbol{x}'\|_2^2}{2l^2}\right)$
  - l controls the smoothness:

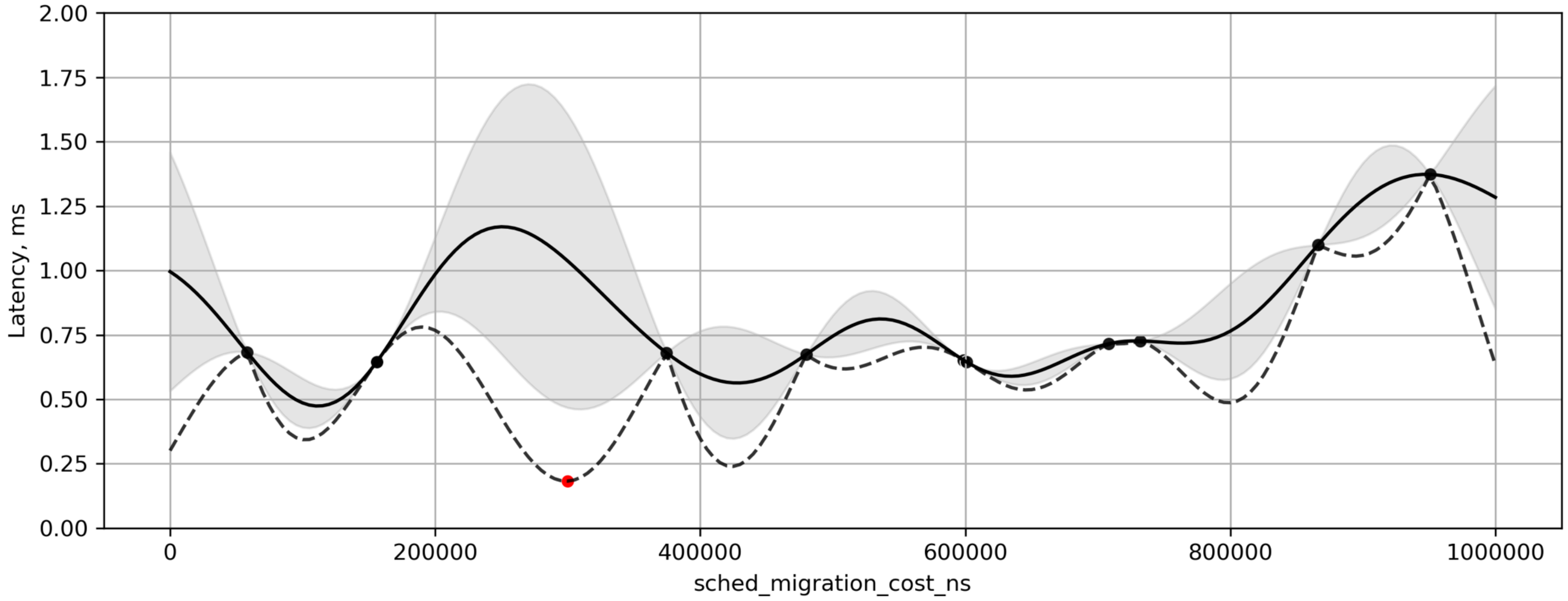


Acquisition Functions

- Probability of improvement: $\mathrm{PI}(\boldsymbol{x}) = \mathrm{P}(f(\boldsymbol{x}) > f(\boldsymbol{x}^*))$
  - Here $\boldsymbol{x}^*$ means the best value so far
- Expected improvement: $\mathrm{EI}(\boldsymbol{x}) = \mathbb{E}[\max(f(\boldsymbol{x}) > f(\boldsymbol{x}^*), 0)]$
  - Takes the **magnitude** of improvement into account!
- Upper Confidence Bound (UCB): $\mathrm{UCB}(\boldsymbol{x}) = m(\boldsymbol{x}) + \beta\sigma(\boldsymbol{x})$
  - $\beta \geq 0$ controls explore/exploit
  - $\sigma(\boldsymbol{x}) = \sqrt{K(\boldsymbol{x}, \boldsymbol{x})}$
- **SOTA:** Information-theoretic approach
  - **MES:** Wang, Z., Jegelka, S. (2017) Max-value entropy search for efficient Bayesian optimization. *ICML*
  - **EHIG:** Neiswanger, W. et al. (2022) Generalizing Bayesian optimization with decision-theoretic entropies. *NeurIPS*

Upper Confidence Bound

- In our case, **<u>Lower</u> Confidence Bound**: $\mathrm{LCB}(\boldsymbol{x}) = m(\boldsymbol{x}) - \beta\sigma(\boldsymbol{x})$



Other Models for Black-Box Optimization

- Random Forest: **SMAC**
  - **Idea**: Learn $\hat{f}(\boldsymbol{x})$ with RF, use regression tree outputs to estimate mean and variance
    - Hutter, F., Hoos, H. H., Leyton-Brown, K. (2010). Sequential model-based optimization for general algorithm configuration. Technical Report TR-2010–10, University of British Columbia.

- Evolutionary algorithms
  - **CMA-ES**: Covariance Matrix Adaptation
    Hansen, N. (2023). The CMA Evolution Strategy: A Tutorial. *arXiv: 1604.00772*
    - Loshchilov, I., Hutter, F. (2016). CMA-ES for Hyperparameter Optimization of Deep Neural Networks. *arXiv: 1604.07269*
  - **PSO**: Particle Swarm Optimization
    - Gad, A. G. (2022). Particle swarm optimization algorithm and its applications: a systematic review. *Archives of computational methods in engineering*, *29*(5), 2531-2561.

Discrete / Hybrid Optimization

- E.g., MySQL parameter `innodb_flush_method` can take values:
  `{fsync, littlesync, nosync, O_DSYNC, O_DIRECT, O_DIRECT_NO_FSYNC}`
- Common approaches:
  - Alternative surrogate models (e.g., Random Forest in **SMAC**)
  - Multi-Armed Bandits (AFs like **UCB** and **EI** do not require sampling from posterior)
  - Adapt features to continuous space (impose order, one-hot, etc.)
    - **MerCBO:** Deshwal, A. et al. (2021). Mercer features for efficient combinatorial Bayesian optimization. *AAAI*. Works with information-theoretic acquisition functions like MES
- **SOTA:** Use NNs to encode features, optimize in latent space
  - **LOL-BO:** Maus, N. et al. (2022) Local latent space Bayesian optimization over structured inputs. *NeurIPS.*

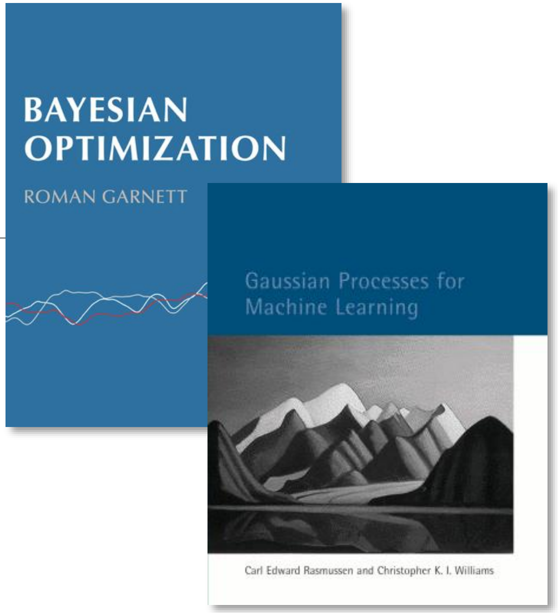
More Fun With Optimization

- **Parallel** Optimization
  - E.g., produce the next 10 configurations to evaluate
- **Constrained** / **Structured Space** / **Causal** Optimization
  - Use / model the parameters' correlations
- **Multi-Fidelity** and **Cost-Based** Optimization
  - Balance the accuracy and <u>cost</u> of measurements
- **Multi-Objective** Optimization
  - Pareto frontier: e.g., an optimal <u>combination</u> of Cost and Throughput
- **Multi-Task** Optimization
  - Efficient config space exploration for finding, e.g., optimal Latency <u>and</u> optimal Throughput

Goal: Efficient Exploration

- **Input:** measurements $\left[\left(\boldsymbol{x}_1, f(\boldsymbol{x}_1)\right), \ldots, (\boldsymbol{x}_n, f(\boldsymbol{x}_n))\right]$ (AKA training data)
- Task: Given the data, ~~find the optimum $\boldsymbol{x}^*$~~?
- **A better task:** Given the data, produce $[\boldsymbol{x}_{n+1}, \ldots]$ that maximize the information gain about the optimum of the unknown function f
- **Sample Efficiency:** minimize the number of trials to achieve desired accuracy

References

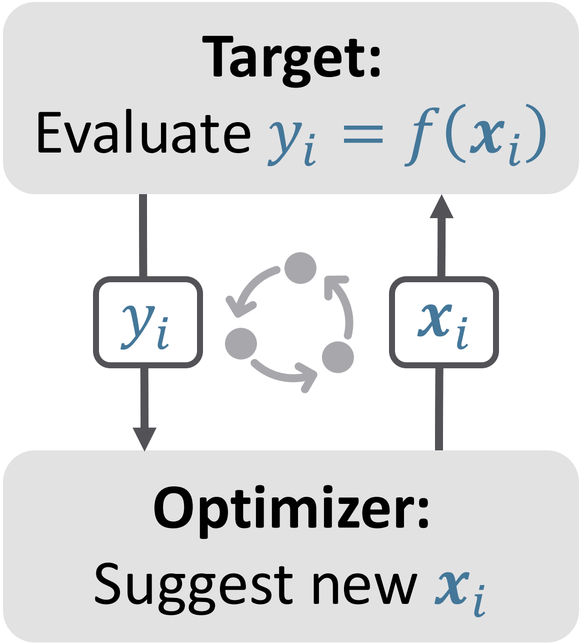


- Books (available online):
  - Rasmussen, C. E., Williams, C. (2006). Gaussian Processes for Machine Learning. MIT Press.
  - Garnett, R. (2023). Bayesian Optimization. Cambridge University Press.
- Tutorials:
  - Frazier, P. I. (2018). A tutorial on Bayesian optimization. *arXiv:1807.02811*.
  - Greenhill, S. et al. (2020). Bayesian Optimization for Adaptive Experimental Design: A Review. *IEEE Access*, vol. 8.
  - Deshwal, A., Belakaria, S., Doppa, J. R. (2023). Recent Advances in Bayesian Optimization, *AAAI*. ← Great bibliography!
  - BoTorch Tutorials. ← Many SOTA algorithms implemented in BoTorch.
- Videos:
  - Doppa, J. R., Aglietti, V., Gardner, J. (2022). Advances in Bayesian Optimization, *NeurIPS Tutorial*.
  - Alvarez, M. et al. (organizers) (2024). Gaussian Process Summer School. University of Manchester.

Is It That Simple?

- As in:
  - Let the optimizer suggest new configurations
  - Evaluate them
  - Repeat

- Yes, **<u>if</u>** configuration space is small… **<u>and</u>** trials are cheap… **<u>and</u>** noise-free… **<u>and</u>** workload is fixed… **<u>and</u>** …



Challenges and Strategies

Challenges

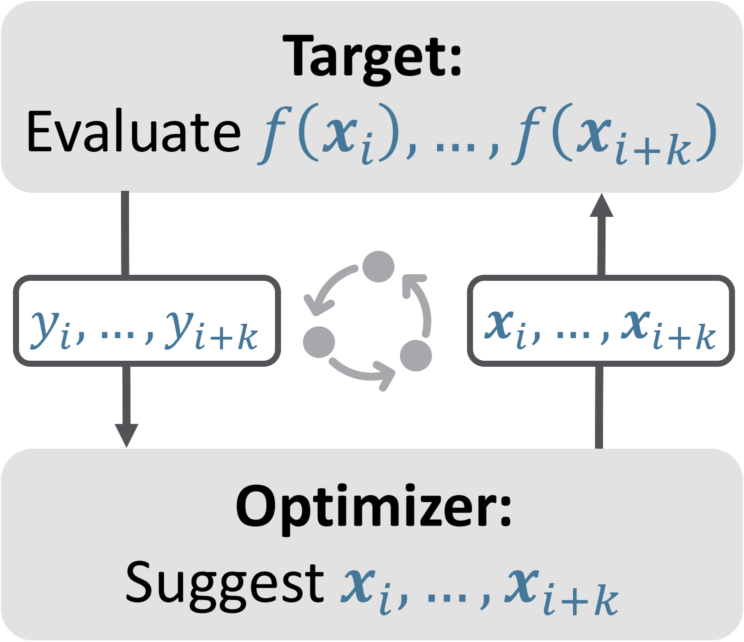
- **Systems**:
  - Execution costs
  - Repeatable experiments
  - Non-representative benchmarks
  - Noise!
- **Optimization**:
  - Curse of dimensionality
  - Parallel / Multi-Task / Multi-Objective opt.
  - Noise!

Strategies

- **Systems:**
  - Make trials faster / cheaper
  - Parallelize
- **Noise:**
  - Collect more data
- **Optimization:**
  - Reduce (focus) the search space
  - Use (more) (noisy) features

Parallel Optimization

- Optimizer suggests many configurations at once
  - **Synchronous:** always suggest k points, batch execute trials
  - **Asynchronous:** suggest 1 point at a time, track up to k in-progress configurations
- **Problem:** maintain the diversity of configurations



- Rebolledo, M., Rehbach, F. et al. (2020). Parallelized Bayesian optimization for problems with expensive evaluation functions. *GECCO 2020*, 231–232.
- Wang, J., Clark, S. C., Liu, E., & Frazier, P. I. (2020). Parallel Bayesian global optimization of expensive functions. *Operations Research*, 68(6), 1850-1865.
- See also: **CMA-ES**

Multi-Objective Optimization

- **Problem:** $\min_{\boldsymbol{x}\in\mathcal{X}}\left(f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \dots, f_k(\boldsymbol{x})\right)$ (e.g., Latency <u>and</u> Cost)
  - Typically, no $\boldsymbol{x}^*$ to optimize all functions simultaneously
- **Pareto frontier:** a set of solutions $\{\boldsymbol{x}^*\}$ not dominated by any other solutions
  - i.e., no objective can be improved without degrading some other objective
- **Scalarization:** Reduce to 1d: $\underset{\boldsymbol{x}\in\mathcal{X}_\theta}{\operatorname{argmin}}\, g_\theta\left(f_1(\boldsymbol{x}), \dots, f_k(\boldsymbol{x})\right)$ where $g_\theta\colon \mathbb{R}^k \to \mathbb{R}$
  - **Linear:** $\min_{\boldsymbol{x}\in\mathcal{X}} \sum_{i=1}^{k} \theta_i f_i(\boldsymbol{x})$ where $\theta_i > 0$ weights for objectives
  - **ParEGO:** Knowles, J. (2006). **ParEGO**: a hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. *IEEE Transactions on Evolutionary Computation, vol. 10, no. 1*

MOMF: Irshad, F., Karsch, S., Döpp, A. (2021). Leveraging Trust for Joint Multi-Objective and Multi-Fidelity Optimization. *arXiv: 2112.13901*

Multi-Target Optimization

- **Problem:** $\min_{\boldsymbol{x}\in\mathcal{X}} f_1(\boldsymbol{x}), \min_{\boldsymbol{x}\in\mathcal{X}} f_2(\boldsymbol{x}), \dots, \min_{\boldsymbol{x}\in\mathcal{X}} f_k(\boldsymbol{x})$ <u>simultaneously</u>
  - Can we reuse the data collected while optimizing $f_1(\boldsymbol{x})$ when optimizing $f_2(\boldsymbol{x})$ etc.? **Yes!**

- **Idea:** <u>exploit the correlations</u> between $f_1(\boldsymbol{x}), \dots, f_k(\boldsymbol{x})$
  - Separable <u>multi-output kernels</u>: $\mathrm{K}\big((i, \boldsymbol{x}), (j, \boldsymbol{x}')\big) = \mathrm{cov}\Big(f_i(\boldsymbol{x}), f_j(\boldsymbol{x}')\Big) = \mathrm{K}_t(i, j)\ \mathrm{K}_x(\boldsymbol{x}, \boldsymbol{x}')$
    - Alvarez, M. et al. (2011). Kernels for Vector-Valued Functions: A Review. Foundations and Trends in Machine Learning.
    - **Video:** Alvarez, M. (2017). Multi-Output Gaussian Processes. *GP Summer School.*
  - Multi-task with <u>common mean</u>: $y_i = \mu_0 + f_i + \epsilon_i$ where each component is a GP
    - Leroy, A. et al. (2023) Cluster-Specific Predictions with Multi-Task Gaussian Processes. *JMLR* 24(5):1−49.

Constraining the Search Space

- **Marginal** Constraints
  - Range limits, quantization, log scale, specifying priors / histograms for individual tunables.
  - E.g., on system with 8GB of RAM MySQL parameter `innodb_buffer_pool_size` likely should be at 6..7GB
- **Constrained** Optimization
  - Constraints can involve multiple tunables and/or be black-box.
  - E.g., MySQL configuration has constraints like:
    `innodb_buffer_pool_chunk_size <= innodb_buffer_pool_size / innodb_buffer_pool_instances`
    - **SCBO:** Eriksson, D., Poloczek, M. (2021). Scalable constrained Bayesian optimization. *AISTATS.* Supports black-box constraints!

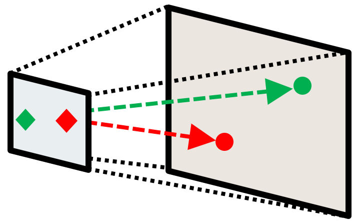
Constraining the Search Space

- **Structured Search Space** Optimization
  - Exploit the independence structure of the tunable parameters
  - E.g., if PostgreSQL config parameter `jit=off`, then ignore JIT parameters `jit_expressions`, `jit_above_cost`, `jit_tuple_deforming`, etc.
    - Jenatton, R. et al. (2017). Bayesian optimization with tree-structured dependencies. *ICML.*
      Idea: Use a mixture of GPs + linear model for a decision tree to capture the dependencies.

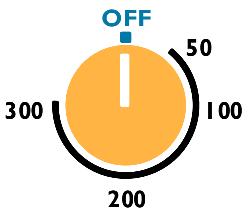
- **Causal** Bayesian Optimization
  - Learn the parameters' independence structure
    - Aglietti, V. et al. (2020). Causal Bayesian optimization. *AISTATS*.

Dimensionality Reduction

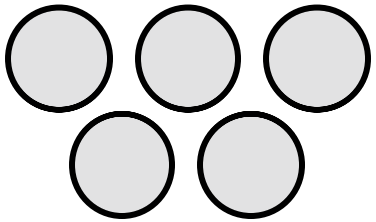
- **LlamaTune:** Use random projection to reduce the search space
  - Many config parameters are correlated ⇒ Replace them with random linear combinations
  - Reduces PG configuration *evaluations* by up to **11x** ; up to **21%** higher *throughput*



Low-Dimensional Search Space Tuning



Special Knob Values Handling



Knob Values Bucketization

- **LlamaTune:** Kanellis, K. et al. (2022) LlamaTune: Sample-Efficient DBMS Configuration Tuning. *VLDB*
- **HesBO:** Nayebi, A., Munteanu, A., Poloczek, M. (2019) A framework for Bayesian optimization in embedded subspaces. *ICML*

LLMs for Parameter Discovery

LLMs are good at **extraction and summarization** of human knowledge from multiple sources (manuals, documentation, source code, StackOverflow, etc.)

- **DBBert:** Identify important tuning knobs and biased ranges with BERT.
  - Trummer, I. (2022). DB-BERT: a Database Tuning Tool that "Reads the Manual". *SIGMOD*.
- **GPTuner:** Discover parameters with LLM, tune with BO.
  - Lao, J. et al. (2024). GPTuner: A manual-reading database tuning system via GPT-guided Bayesian optimization. *VLDB*.

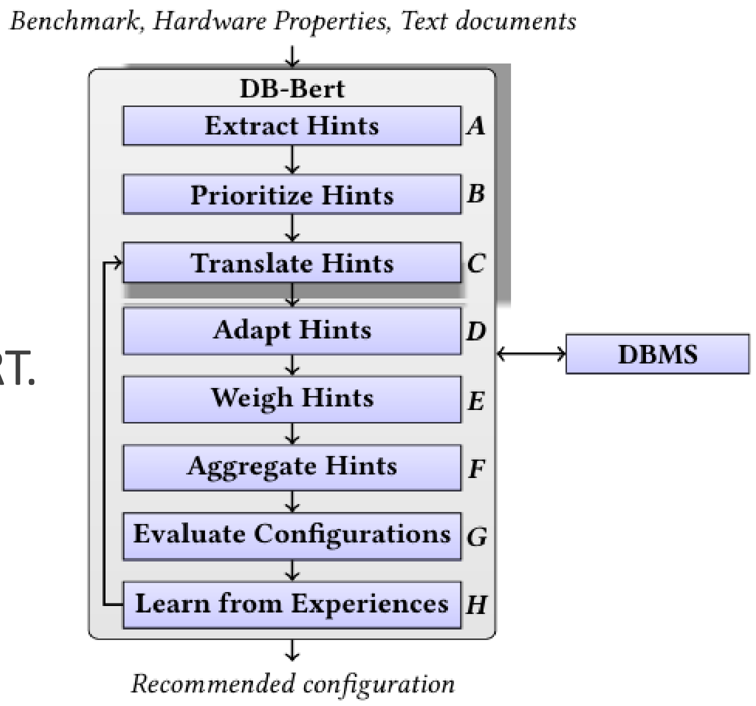


Figure 1: Overview of DB-BERT system: we exploit tuning hints, extracted from text documents, to find optimal DBMS knob settings for a given workload.

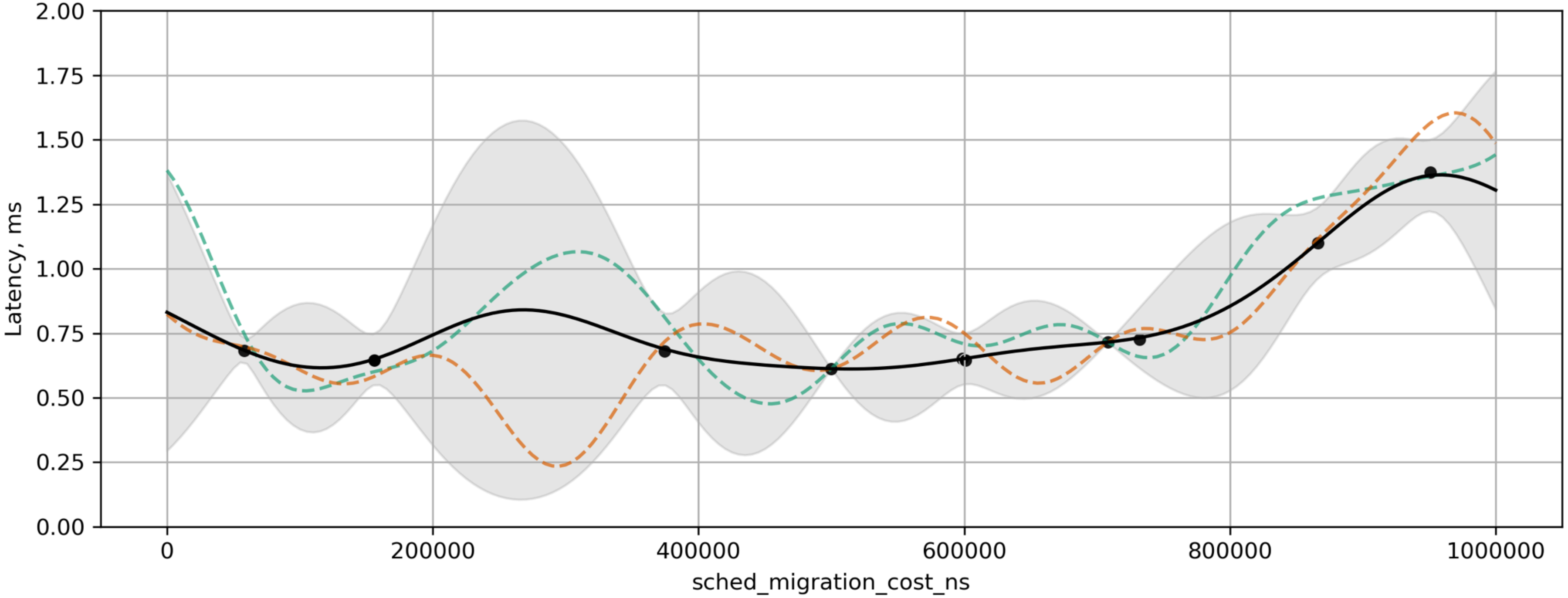
LLMs for Optimization

Next step: Use LLMs to suggest configurations / estimate performance.

- ***λ*-Tune:** Identify the tunables with LLM, then use LLM to generate scripts for *k* configurations and evaluate the most feasible ones.
  - Giannakouris, V., Trummer, I. (2025). λ-Tune: Harnessing Large Language Models for Automated Database System Tuning. *SIGMOD*.
- **LATuner:** Similar to *λ*-Tune to warm-up the optimizer, then use Thompson sampling to select between GP and LLM-based surrogates.
  - Fan, C. et al. (2024). LATuner: An LLM-Enhanced Database Tuning System Based on Adaptive Surrogate Model. *ECML PKDD*.

Multi-Fidelity Optimization

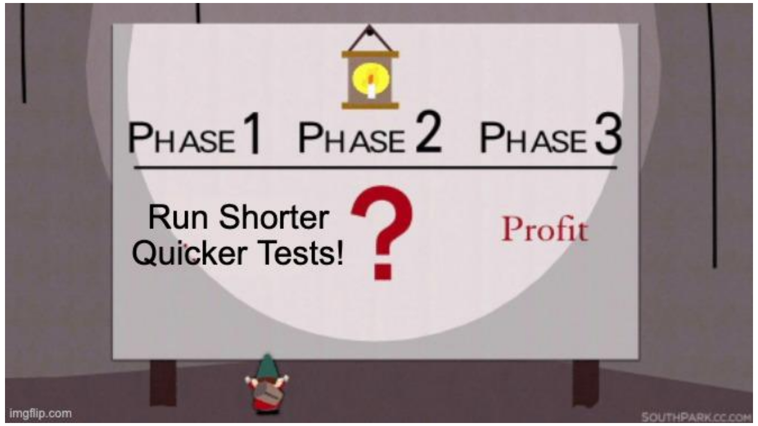
- Combine expensive more accurate measurements and cheaper less accurate ones
  - Use cost-adjusted utility functions, e.g., cost-adjusted Expected Improvement
    - Do, B., Zhang, R. (2023). [Multi-fidelity Bayesian Optimization in Engineering Design](). *CoRR.*



Systems Challenges of Multi-Fidelity

- Remember Goal: reduce cost to find improved config

- Multi-Fidelity Idea: *run cheaper tests!*
  - E.g., Run TPC-H SF1 (seconds), not SF100 (minutes)
  - Alt: TPC-C for 1 minute vs. 20 minutes
  - Sample more points in the same amount of time!



- Is the knowledge gained transferable?
  - E.g., TPC-H SF 1 everything fits in memory, don't need to explore I/O settings
  - TPC-C for 1 minute won't stress the BP or I/O
  - Not as simple as applying a scalar
  - Similar for change in VM size
  - *But,* can score it with "lower confidence"

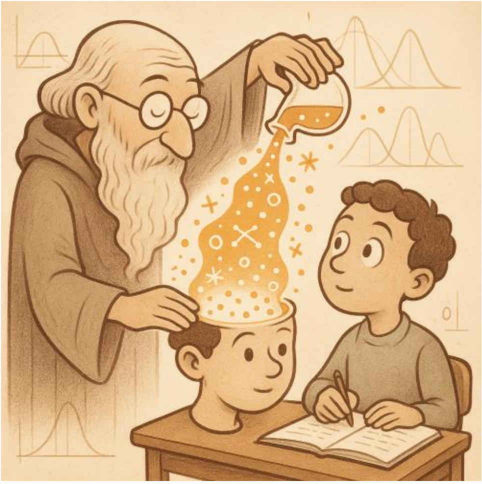
- Important Takeaways:
  - Knowledge Transfer (next)
  - Benchmark Importance
  - Knob Importance

Knowledge Transfer

- Idea: Re-use prior samples
  "warm start" a new optimization
  - i.e., make it cheaper
  - E.g., OpAdvisor (VLDB 2023), Amortized AutoTuning
- Policy:
  1. Good samples: reuse results from "*similar*" workloads
  2. Poor samples: unclear – could be good in this case?
     - Keep exploring these
  3. Bad samples: reuse everywhere
     - Idea: if it crashes the system, probably always does
     - Helps inform the optimizer don't search there again
- How?
  1. Good: keep the score
  2. Bad: no score (e.g., crashed)?
     - Make it up!
     - *N * {worst_score_measured}*

- Assumes "compatible" *context:*
  - Hardware
  - VM Size
  - OS
  - Workload

- What about VM Size Changes?
  - E.g., 2 vCPU 8GB → 4 vCPU 16 GB
  - Just 2x everything? Maybe not.
    - Caches, OK
    - Join or sort buffers?
      Depends on the workload.
    - Threads?



Focus on the Important Knobs!

- Previously:
  - Use LLM to inform which parameters to *focus* on
    - Crowd sourcing a "human expert"
  - LlamaTune to narrow search space
  - Multi-fidelity workload change may impact knob sensitivity
- Related:
  - DBSeer (SIGMOD 2013):
    - Uses models of specific resources to try and diagnose performance bottlenecks
    - Can be used for tuning
  - OtterTune (SIGMOD 2017):
    - Uses Lasso with system metrics and prior configuration runs to identify important knobs
  - More recent work use SHAP (NIPS 2017) values
    - Framework for "explainable AI"
    - Also useful for "knob importance" ranking
    - Still need to have historical values to work from

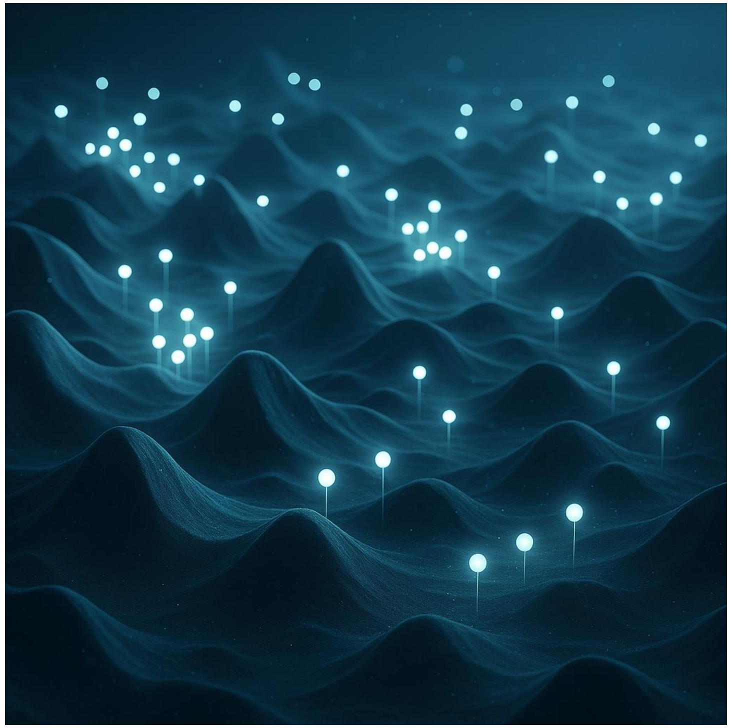
- PGO or FDO (Diniz PLDI 1997): Concept from compilers:
  - Use stack profiles captured from real runs to focus compiler optimizations in "the right places"
- *Could* do similar for other systems tuning:
  - Run workload
  - Capture stack traces
  - Identify Hotspots
  - Search surrounding code for "tunables" (*non-trivial*)
  - Prioritize tuning those
- Reverse: design a workload to exercise certain/all code paths and tune for that "*general case*"?
  - E.g., QO
- Opportunity: *to our knowledge no system currently does this*

To Learn More … Run More Trials!

- Previously:
  - Multi-Fidelity: learn from cheaper trials?

- Parallel Execution
  - In the cloud! Just Run more.
  - Ignores the $$ and WHr cost …
  - Also, see Parallel Optimization issues
  - However, with “async trials” we also have the infra to augment other signals (e.g., additional cloud metrics)

- Alternatively: Early Abort
  - Report bad score sooner
  - Works well for “elapsed time based” benchmarks
    - E.g., TPC-H



To Learn More … Get Stable!

Cloud is noisy
- Despite systems improvements
- Unstable performance, *w/o* config tuning
- Slows rate of learning
- Can have non-transferrable configs (undeployable)

What to do?
- Naïve: run N times, take aggregate (avg, median)
  - Costly
- Alt: measure current resource performance
  - Microbenchmarks
- Throw out outlier machines?
  - No – may be stuck deployed to those later
- Learn noise adjusted performance score?

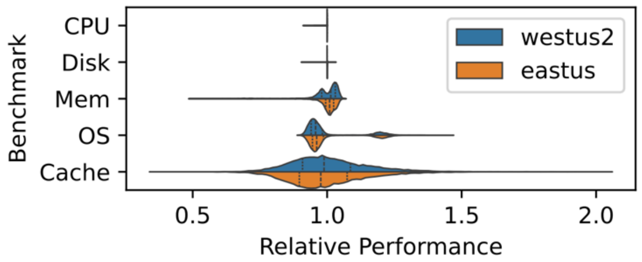


Figure 4. The variance of benchmarks targeting CPU, Disk, Memory, the OS, and CPU cache. Relative performance is relative to the mean performance seen. Higher is better.

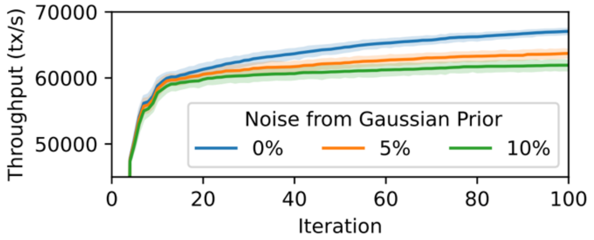
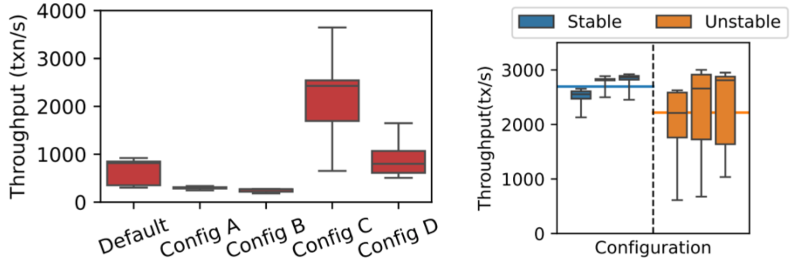


Figure 3. Optimizer Rate of Convergence for epinions workload, running on PostgreSQL 16.1 at various levels of noise.



(a) Throughput for the 5 configurations of the initialization set, when evaluated on the same 30 nodes.

(b) A subset of best-performing configs. when deployed on new nodes.

[TUNA: Tuning Unstable and Noisy Cloud Applications. Eurosys 2025]

You can TUNA Duet!

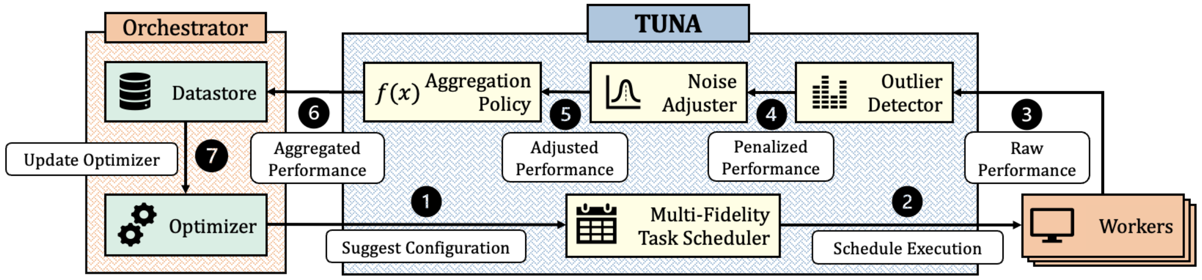
Duet Benchmarking (ICPE 2020)

- “Lean in” to the noise
- Run both default and trial config side by side
- Both should be subject to same noise
- Report *normalized relative* difference
- Originally intended for CI perf regressions

TUNA (Eurosys 2025)

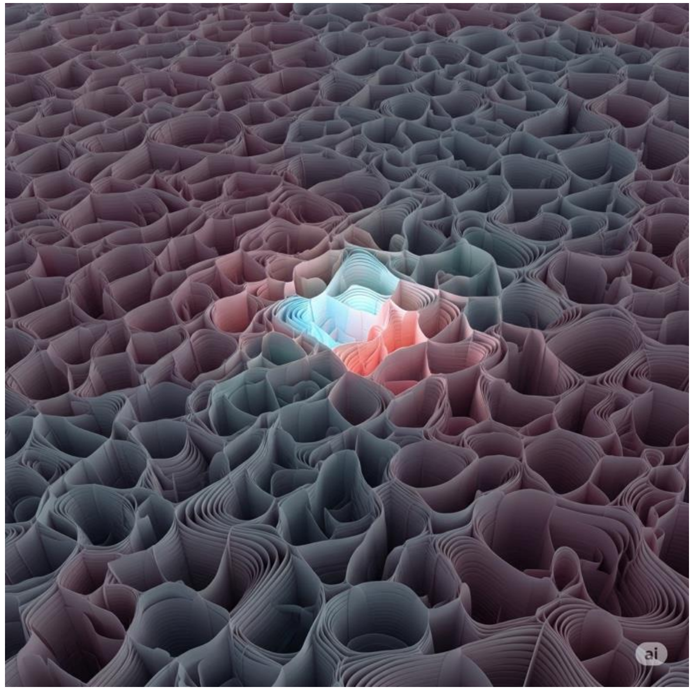
- Successive Halving
  - Progressively run on multiple VMs *iff* the config looks good
  - Samples noise across a cluster/region
- Eliminate outliers and unstable configs
- Use a sideband signals and a model to register more “stable” scores with Optimizer
- Results in faster learning *and* more robust configs

TUNA



Systems References

- OtterTune (SIGMOD 2017, VLDB 2018, VLDB 2021)
- BestConfig (SoCC 2017)
- HyperMapper (MASCOTS 20127)
- CDBTune (SIGMOD 2019)
- QTune (VLDB 2019)
- OnlineTune (SIGMOD 2022)
- LOCAT (SIGMOD 2022)
- DBSeer (SIGMOD 2013)
- Bao for Scope (SIGMOD 2021)
- LlamaTune (VLDB 2022)
- Duet Benchmarking (ICPE 2020)
- TUNA (Eurosys 2025)
- MLOS (VLDB 2024)
- …



Deploying Configs Tuned Offline

Problem: Tuned for TPC-C or YCSB or … ,
but what is my customer running?

Which config should I recommend?
Are any of them "close"?

Alt: They *were* running TPC-C, but now they're doing something else?

- When/how to re-evaluate?
- Timeseries …

Customers want "predicted" improvement

- BO can't even say config is optimal!
- Can't replay their workload (side effects)
- Can't look at it (privacy)

Future Work

- Need some notion of "Similarity" for Workloads
- Create new synthetic benchmarks from just metrics?
  - Stitcher (EDBT 2019)

Alternatively …

Outline

- ~~Overview (15 mins)~~
- ~~Offline Tuning (45 mins)~~
  - ~~Basic Architectural Overview~~
  - ~~Running Example~~
  - ~~Optimization~~
    - ~~Classic Search~~
    - ~~Bayesian Optimization~~
  - ~~Systems Challenges~~
- Online Tuning (20 mins)
  - Basic Architectural Overview
  - Optimization
    - Reinforcement Learning (RL)
    - Genetic Algorithms (GA)
  - Systems Challenges
- Future Directions (10 mins)
  - Workload Identification

Online Optimization

YIWEN ZHU

Online Optimization

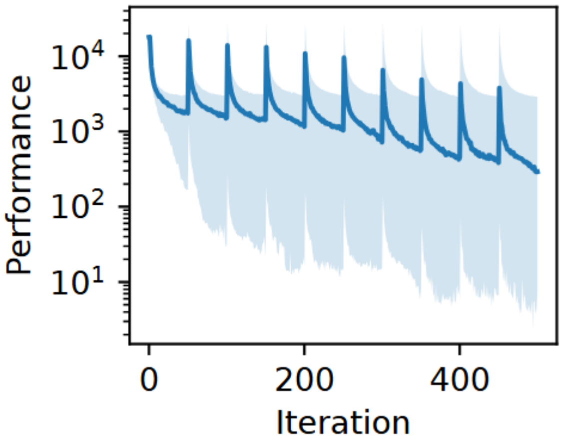
Changing Environment

Workload Shifting

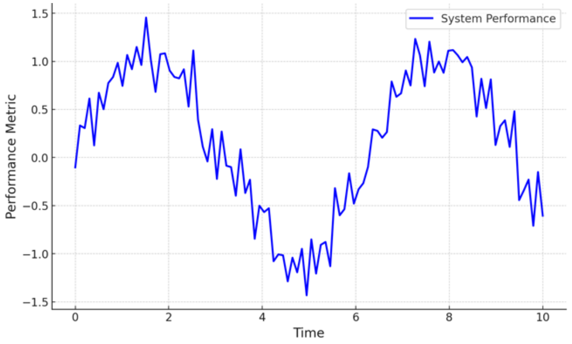
Learning in **real-time** and in **production** environment.

Challenges

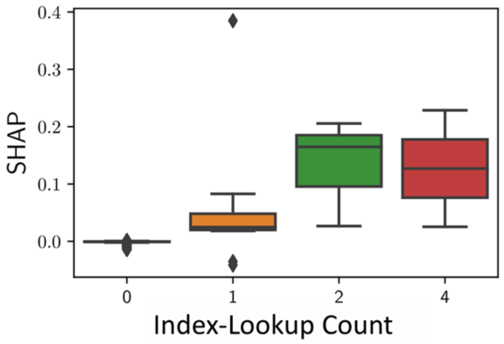
Workload Shifting



Performance Regression / Guardrail



Explainability



Noisy Data

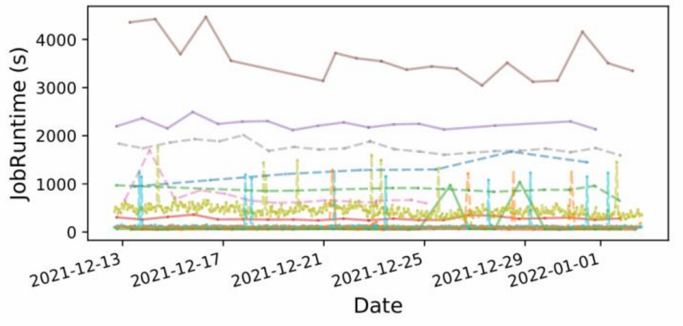


Fig. 1. Recurring jobs with runtime variation.

Online Tuning Architectures

External

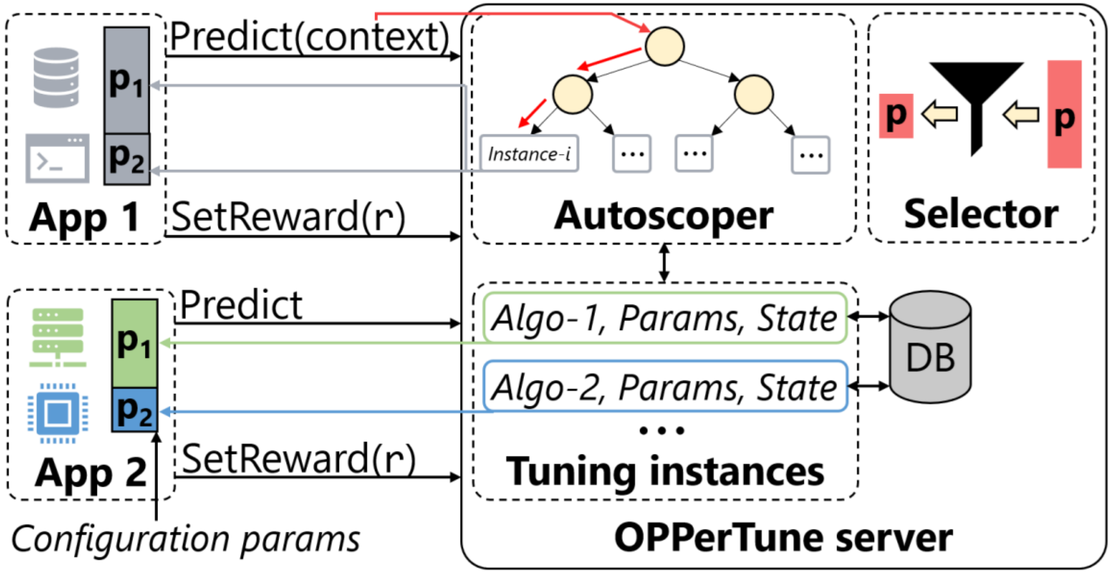
- Use a “side-car” to monitor and adjust the target from the outside
  - Need to expose hooks to outside agent (already done?)
  - Restricted

Internal

- Application contains agent embedded in it to monitor and adjust target from inside
  - More invasive changes, costly to run

Both

- Internal agent monitors, calls out to external service for actions
  - E.g., SelfTune, OPPerTune (NSDI 2024) →



Online Tuning Algorithms: Reinforcement Learning

- Q-Learning:
  - Q Values, **Q(s,a)**: the expected reward when taking the action **a**, given at a state **s**
- Actor-Critic:
  - Policy Function, **π(s,a)**: the probability to take the action **a**, given at state **s** given the current policy
  - Value Function, **V(s)**: the expected future rewards from state **s**

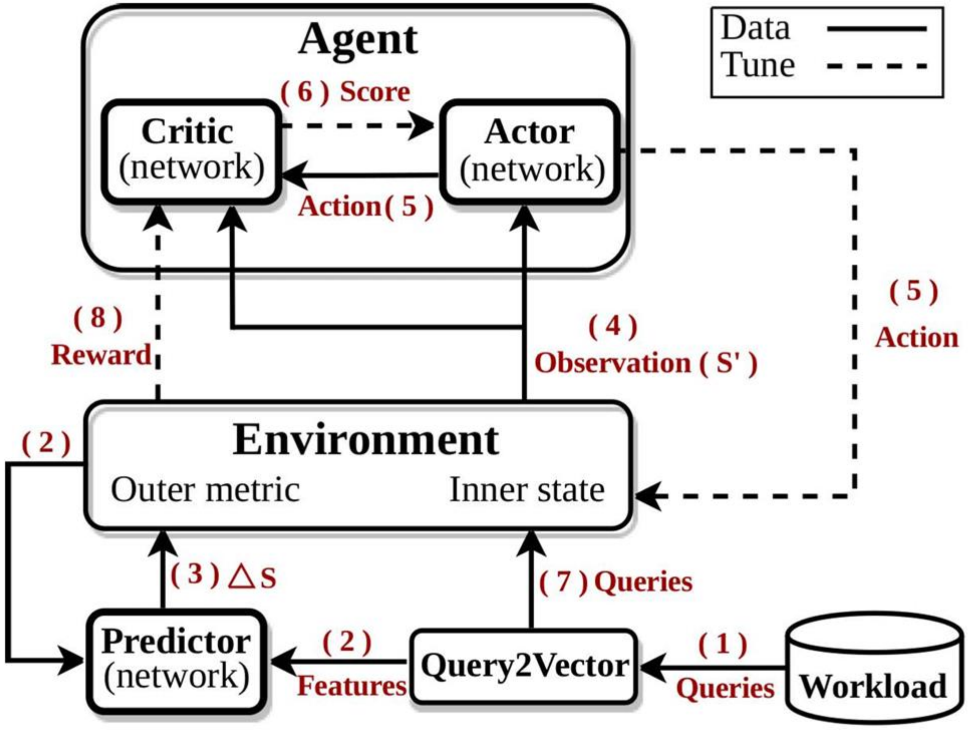


Figure 4: The DS-DDPG Model

•**[28]** Guoliang Li, Xuanhe Zhou, Shifu Li, and Bo Gao. 2019. *QTune: A query-aware database tuning system with deep reinforcement learning. Proc. VLDB Endow.* 12, 12, 2118–2130. DOI: 10.14778/3352063.3352129

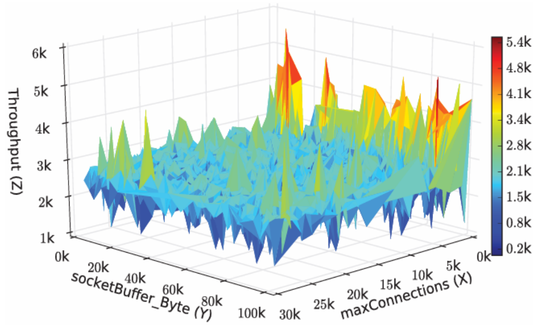
Online Tuning Algorithms: Reinforcement Learning

- Q-Learning:
  - Q Values, **Q(s,a)**: the expected reward when taking the action **a**, given at a state **s**
- Actor-Critic:
  - Policy Function, **π(s,a)**: the probability to take the action **a**, given at state **s** given the current policy
  - Value Function, **V(s)**: the expected future rewards from state **s**

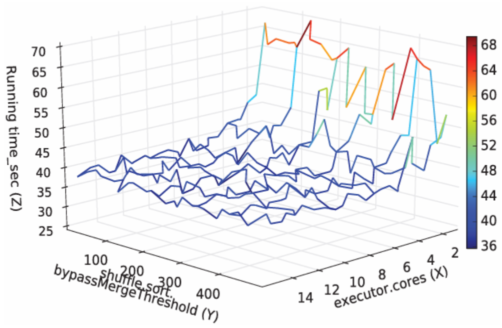
•**[21]** Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. 1996. *Reinforcement learning: A survey*. Journal of Artificial Intelligence Research 4, 237–285.
•**[56]** Ji Zhang, Yu Liu, Ke Zhou, Guoliang Li, Zhili Xiao, Bin Cheng, Jiashu Xing, Yangtao Wang, Tianheng Cheng, Li Liu, et al. 2019. *An end-to-end automatic cloud database tuning system using deep reinforcement learning*. In *Proceedings of the 2019 International Conference on Management of Data*, 415–432.
•**[28]** Guoliang Li, Xuanhe Zhou, Shifu Li, and Bo Gao. 2019. *QTune: A query-aware database tuning system with deep reinforcement learning*. *Proc. VLDB Endow.* 12, 12, 2118–2130. DOI: 10.14778/3352063.3352129
•**[57]** William Zhang, Wan Shen Lim, Matthew Butrovich, and Andrew Pavlo. 2024. *The Holon Approach for Simultaneously Tuning Multiple Components in a Self-Driving Database Management System with Machine Learning via Synthesized Proto-Actions*. *Proc. VLDB Endow.* 17, 11, 3373–3387.
•**[52]** Junxiong Wang, Immanuel Trummer, and Debabrota Basu. 2021. *UDO: universal database optimization using reinforcement learning*. *Proc. VLDB Endow.* 14, 13, 3402–3414. DOI: 10.14778/3484224.3484236
•[New] Microsoft. Self-Tune. microsoft/SelfTune

Online Tuning Algorithms

- Genetic Algorithm [HUNTER, DAC, RFHOC]
- Greedy Search [Auto-Steer]
- HybridBandits [OPPerTune]
- Multi-Objective Optimization [MOO]
- Divide-and-conquer search [BestConfig]
- Adaptive Modeling [LITE]



(b) Tomcat under webpage navigation workload



(c) Spark under HiBench-KMeans workload

•**[6]** Baoqing Cai, Yu Liu, Ce Zhang, Guangyu Zhang, Ke Zhou, Li Liu, Chunhua Li, Bin Cheng, Jie Yang, and Jiashu Xing. 2022. *HUNTER: An Online Cloud Database Hybrid Tuning System for Personalized Requirements.* In *Proceedings of the 2022 International Conference on Management of Data (SIGMOD '22)*, 646–659. DOI: 10.1145/3514221.3517882

•**[30]** Chen Lin, Junqing Zhuang, Jiadong Feng, Hui Li, Xuanhe Zhou, and Guoliang Li. 2022. *Adaptive Code Learning for Spark Configuration Tuning*. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, IEEE, 1995–2007.

•**[54]** Zhibin Yu, Zhendong Bei, and Xuehai Qian. 2018. *Datasize-Aware High Dimensional Configurations Auto-Tuning of In-Memory Cluster Computing. SIGPLAN Not.* 53, 2 (March 2018), 564–577. DOI: 10.1145/3296957.3173187

•**[3]** Zhendong Bei, Zhibin Yu, Huiling Zhang, Wen Xiong, Chengzhong Xu, Lieven Eeckhout, and Shengzhong Feng. 2016. *RFHOC: A Random-Forest Approach to Auto-Tuning Hadoop's Configuration. IEEE Transactions on Parallel and Distributed Systems* 27, 5 (2016), 1470–1483. DOI: 10.1109/TPDS.2015.2449299

•**[2]** Christoph Anneser, Nesime Tatbul, David Cohen, Zhenggang Xu, Prithviraj Pandian, Nikolay Laptev, and Ryan Marcus. 2023. *AutoSteer: Learned Query Optimization for Any SQL Database. Proc. VLDB Endow.* 16, 12 (Aug. 2023), 3515–3527. DOI: 10.14778/3611540.3611544

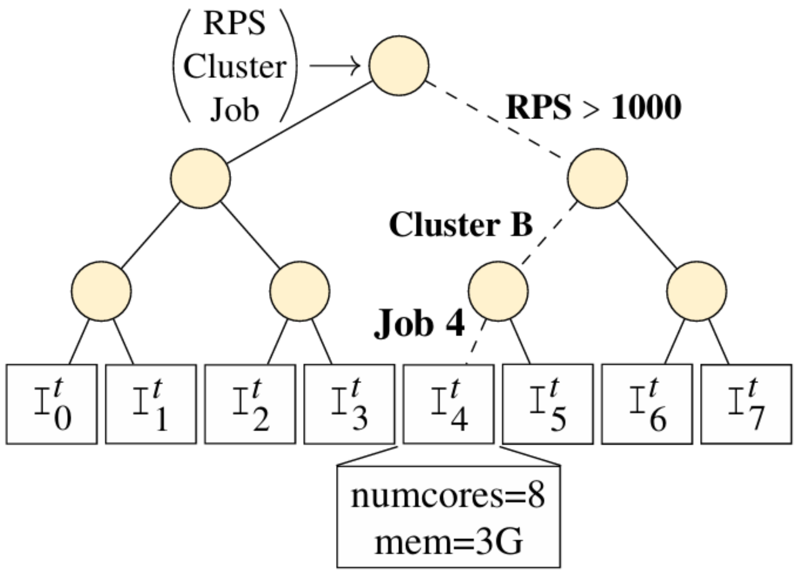
•**[46]** Gagan Somashekar, Karan Tandon, Anush Kini, Chieh-Chun Chang, Petr Husak, Ranjita Bhagwan, Mayukh Das, Anshul Gandhi, and Nagarajan Natarajan. 2024. *OPPerTune: Post-Deployment Configuration Tuning of Services Made Easy*. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, 1101–1120.

•**[31]** Chenghao Lyu, Qi Fan, Philippe Guyard, and Yanlei Diao. 2024. *A Spark Optimizer for Adaptive, Fine-Grained Parameter Tuning. Proc. VLDB Endow.* 17, 11 (Aug. 2024), 3565–3579. DOI: 10.14778/3681954.3682021

•**[67]** Yuqing Zhu, Jianxun Liu, Mengying Guo, Yungang Bao, Wenlong Ma, Zhuoyue Liu, Kunpeng Song, and Yingchun Yang. 2017. *BestConfig: Tapping the Performance Potential of Systems via Automatic Configuration Tuning*. In *Proceedings of the 2017 Symposium on Cloud Computing (SoCC '17)*, 338–350. DOI: 10.1145/3127479.3128605

Challenge: Workload Shifting

- OnlineTune: Dynamically adapts to workload changes by **embedding contextual features** (e.g., data size, query plans) into a Bayesian Optimization framework.
- OPPerTune: Uses AutoScoper, which integrates **job type & RPS** into a **Hybrid Bandit algorithm**, selecting optimal tuning strategies via a **decision tree model**.
- Rockhopper: Generate workload embedding based on the execution plan of each query [SIGMOD Industry 4].



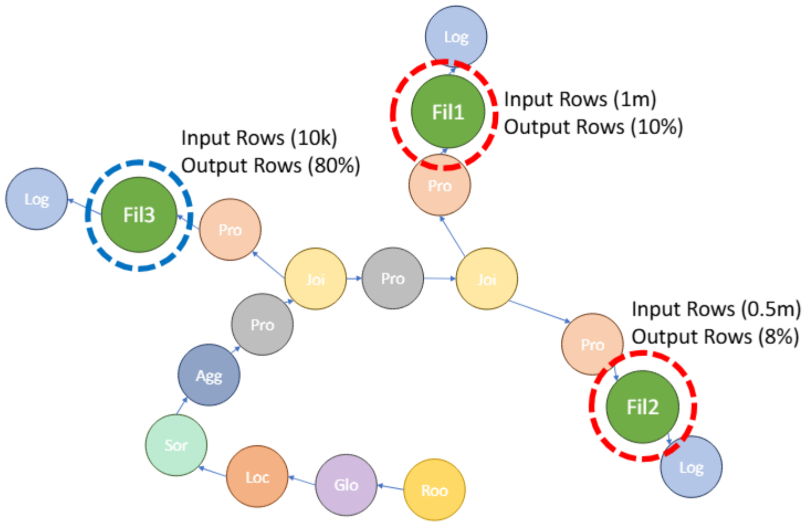


Figure 4: Virtual operator.

Challenge: Workload Shifting

- OnlineTune: Dynamically adapts to workload changes by **embedding contextual features** (e.g., data size, query plans) into a Bayesian Optimization framework.
- OPPerTune: Uses AutoScoper, which integrates **job type & RPS** into a **Hybrid Bandit algorithm**, selecting optimal tuning strategies via a **decision tree model**.
- Rockhopper: Generate workload embedding based on the execution plan of each query [SIGMOD Industry 4].

[1] Xinyi Zhang, Hong Wu, Yang Li, Jian Tan, Feifei Li, and Bin Cui. 2022. TowardsDynamic and Safe Configuration Tuning for Cloud Databases. In Proceedingsof the 2022 International Conference on Management of Data (Philadelphia, PA,USA) (SIGMOD ’22). Association for Computing Machinery, New York, NY, USA,631–645. https://doi.org/10.1145/3514221.3526176

[2] Gagan Somashekar, Karan Tandon, Anush Kini, Chieh-Chun Chang, Petr Husak,Ranjita Bhagwan, Mayukh Das, Anshul Gandhi, and Nagarajan Natarajan. 2024.{OPPerTune}:{Post-Deployment} Configuration Tuning of Services Made Easy.In 21st USENIX Symposium on Networked Systems Design and Implementation(NSDI 24). 1101–1120.

[3] Yiwen Zhu, Rathijit Sen, Brian Kroth, Sergiy Matusevych, Andreas Mueller, Tengfei Huang, Rahul Challapalli, Weihan Tang, Xin He, Mo Liu, Estera Kot, Sule Kahraman, Arshdeep Sekhon, Dario Bernal, Aditya Lakra, Shaily Fozdar, Dhruv Relwani, Rui Fang, Long Tian, Karuna Krishna, Ashit Gosalia, Carlo Curino, and Subru Krishnan. 2025. *Rockhopper: A Robust Optimizer for Spark Configuration Tuning in Production Environment*. In *Companion of the 2025 International Conference on Management of Data (SIGMOD-Companion '25)*. ACM, New York, NY, USA. https://doi.org/10.1145/3722212.3724451

Challenge: Avoid Performance Regression

- OnlineTune: Iteratively optimizes **subspaces around the best-known** configuration, ensuring gradual convergence and assessing safety via lower-bound estimates.
- LOCAT: Uses **Safe Bayesian Optimization** to tune Spark SQL while minimizing performance regressions.
- AutoSteer: Applies **greedy search** to incrementally improve configurations, balancing exploration & exploitation.
- HUNTER: Uses **cloned Cloud Databases** (CDBs) to test configurations without impacting production, acting as a hybrid online-offline approach.
- OPPerTune: Integrates **contextual bandits** with a probabilistic model to safely explore configurations, limiting risk but trading off optimality.
- [29]: Defines a **safe exploration** region using Gaussian Process models, ensuring configurations meet performance constraints (e.g., runtime limits).

Common Strategies

SERGIY MATUSEVYCH

Online vs. Offline

Online:

+ Adapts to individual system instances

+ Dynamically adjusts to workload changes

But:

- Runtime overhead
- Higher integration costs
- Harder to generalize to other systems
- Conservative / can get stuck in local optimum

Offline:

+ Better config space exploration / parallel

+ Cheap and easy to deploy/rollback/maintain

+ Zero runtime costs in prod

But:

- Configurations are static / not adaptable
- Benchmarks may be not representative
- Workload ID challenges

Strategies

- Combine Online *and* Offline optimization
  - **Warm-up** Online with Offline data
- Reuse optimized configs on similar systems
  - Models for **Workload Identification**
    - Pavlo, A. et al. (2017). Self-Driving Database Management Systems. *CIDR* (Vol. 4, p. 1).
- Zero-shot ML models to produce optimal configs
  - North Star: **WorkFM** – Workload Foundation Models

Workload Identification

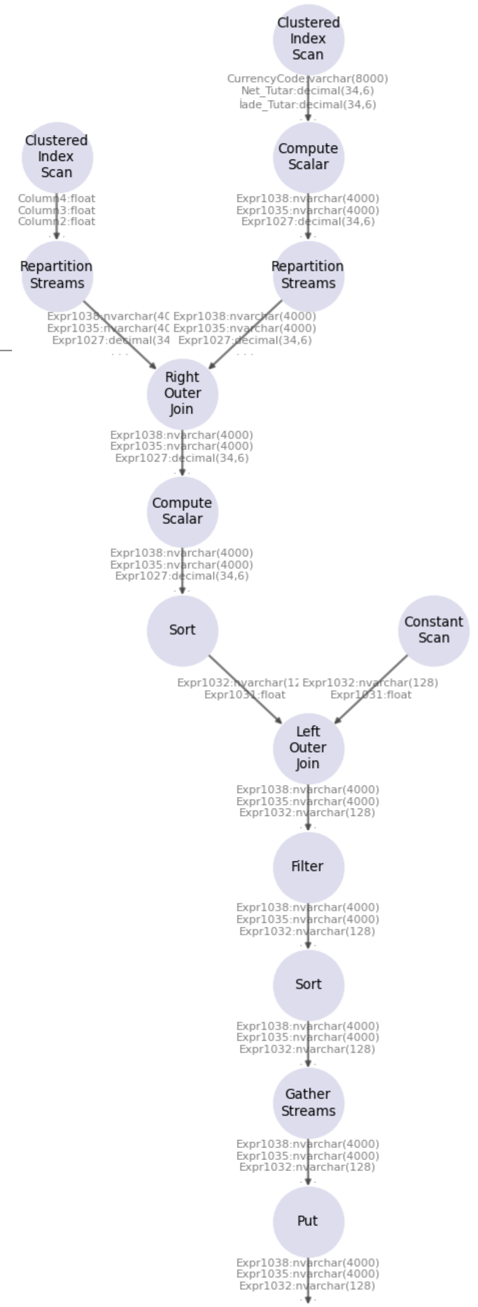
- **Idea:** Systems with *similar workloads* can benefit from the same optimal config
  - Optimize one system
  - Identify other similar systems
  - Reuse the optimized configuration on that set
- **Problem:** How to determine what systems/workloads are similar?
  - Easy if we have labels: e.g., MySQL + Wordpress
  - In general: need a distance / similarity metric between workloads

Workload Embedding

- **Idea:** Build ML model to capture the representation of workloads
  - Map each workload to a multi-dimensional vector (embedding)
  - Kernel function: measure distance between two points in multi-dimensional space
- **Benefits of Embeddings:**
  - Compact representation of large number of heterogeneous features
  - Comparison of not-exactly-alike workloads
  - Clustering / other kernel-based methods
  - Input to other ML models (query optimization, anomaly detection, scheduling, etc.)

Data to Embed

- **Telemetry:** Time Series
  - E.g., CPU load, Memory utilization, Disk and Network I/O, etc.
  - Some app-specific data available (# of inserts/updates/selects, InnoDB stats…)
  - Easy to collect / access; typically, not sensitive
  - Noisy!
- **Query Logs:** Graph
  - Query Logs / Query Plans available (or can be sampled) on *some* systems
  - Can be sensitive; may require anonymization
- **User Data:** Tabular
  - Access requires user consent; eyes-off training possible (maybe)



Building the Embeddings

- **Time Series**
  - Telemetry *alone* can capture irrelevant information about the system
  - Foundation models for time series is an active area of research:
    - **MOIRAI:** Woo, G. et al. (2024). Unified training of universal time series forecasting transformers. *ICML*.
    - **Chronos:** Ansari, A.F. et al. (2024). Chronos: Learning the language of time series. *TMLR*.
    - Liang, Y. et al. (2024). Foundation models for time series analysis: A tutorial and survey. *KDD*.
- **Graph Data**
  - Query data captures *most* of the information about the workload (but not all!)
  - Modeling query workloads with GNNs looks very promising
    - Paul, D., Cao, J., Li, F., Srikumar, V. (2021). Database workload characterization with query plan encoders. *VLDB*.
    - Zhao, Y. et al. (2022). QueryFormer: A tree transformer model for query plan representation. *VLDB*.

Applications

- **Knowledge Transfer**
  - Apply optimized configurations to other similar systems
  - Warm-up optimizations for systems not-so-similar
- **Workload Shift Detection**
  - Identify changes in workload over time
- **Synthetic Benchmark Generation**
  - Generate the optimal mixture of queries to mimic the workload in production
  - Offline optimize the system for that new synthetic benchmark
  - Use the optimized config on system in prod

Future Work

- Two orthogonal / complementing tasks:
  - Build **better embeddings** for workloads
  - Build **better models** that use these embeddings
- **Multi-modal learning:**
  - Combine time series and graph data
- **North Star: WorkFM**
  - Workload Foundation Models
    - Wehrstein, J. et al. (2025). Towards Foundation Database Models. *CIDR best paper award.*

Thank you!

- Ping us: Brian Kroth, Sergiy Matusevych, Yiwen Zhu
- Our team and projects: Microsoft Gray Systems Lab (GSL) https://aka.ms/gsl
- Meet us at the Microsoft booth! Microsoft

- **Questions?**