Set It and Forget It: Zero-Mod ML Magic for Linux Tuning

Georgios Liargkovas
Columbia University
g.liargkovas@cs.columbia.edu

Prabhpreet Singh Sodhi
Columbia University
pss2161@cs.columbia.edu

Kostis Kaffes
Columbia University
kkaffes@cs.columbia.edu

Abstract

Machine learning can turbocharge OS optimization—if one is willing to reinvent the whole stack. Recent work pushes exotic instrumentation or new OS designs that break real-world constraints, demanding app metrics nobody can (or wants to) provide. The alternative—naively optimizing for simple system proxies like IPC—is just as flawed, leading to misleading results that fail to generalize across real-world workloads. Our framework sidesteps this dilemma by learning to optimize without direct visibility. Instead of building brittle models to predict absolute performance, we reframe the problem to learn the relative ranking of system configurations, using a diversified performance signature built from the system counters the OS already has. The outcome is a scalable, robust, and ML-driven performance boost for real applications—delivered without demanding radical shifts in the OS landscape.

CCS Concepts: • **Software and its engineering → Operating systems**; • **Computing methodologies → Machine learning**.

Keywords: Operating systems, machine learning, OS autotuning, preference learning

ACM Reference Format:
Georgios Liargkovas, Prabhpreet Singh Sodhi, and Kostis Kaffes. 2025. Set It and Forget It: Zero-Mod ML Magic for Linux Tuning. In *Practical Adoption Challenges of ML for Systems (PACMI '25), October 13–16, 2025, Seoul, Republic of Korea.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3766882.3767175

1 Introduction

Machine learning (ML) offers a significant opportunity to optimize the OS, improving performance, efficiency, and adaptability [32]. Applied correctly, ML can enable real-time tuning of critical OS components such as CPU scheduling [9], memory management [26, 28, 35], and I/O prioritization [19]. However, practical deployment is challenging. Current ML-based OS optimizers typically depend on application-level metrics (e.g., request latency) [6, 7, 11, 13, 20, 22, 24, 25, 31, 37, 43], creating a gap between what the OS can see and what the optimizer needs. The OS has access to low-level system counters but lacks visibility into high-level application performance.

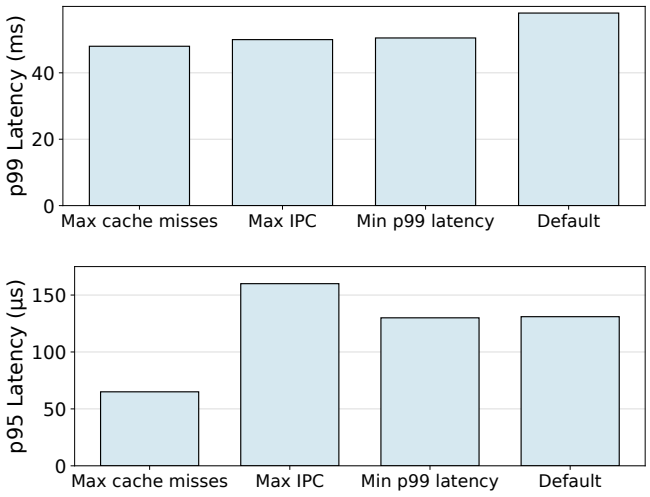

Figure 1. Tail latency for TPC-C (top) and sysbench OLTP-RW (bottom) when using Bayesian Optimization to tune scheduler parameters based on different metrics.

Intrusive solutions, such as modifying applications to export metrics or migrating to entirely new, ML-centric operating systems [42], are non-starters for most production environments. The only universally viable signals are the system-level metrics the OS already collects. However, naively using simple proxies like Instructions per Cycle (IPC) to guide optimization offers limited gains and can be misleading, as shown in Figure 1 (top) vs (bottom). The best-performing system proxy for one workload can be ineffective or even counterproductive for another or when noise is introduced. This shows that such simple approaches are brittle and fail to generalize to the diverse and dynamic workloads found in modern data centers.

PACMI '25, October 13–16, 2025, Seoul, Republic of Korea

ACM ISBN 979-8-4007-2205-9/25/10
https://doi.org/10.1145/3766882.3767175

This paper argues for a pragmatic, non-intrusive framework that bridges this semantic gap. We make three main contributions. First, we empirically demonstrate the pitfalls of relying on simple *system metrics as performance proxies* and *absolute performance prediction models*, showing they are brittle and fail to generalize. Second, we propose a robust workaround: a framework that uses *relative ranking* models and clustering to build specialized, noise-resistant models. Finally, we chart a path toward truly adaptive systems by proposing causal modeling as a mechanism to handle complex environmental variation. Together, these techniques provide a practical path toward a self-optimizing Linux for real-world, unmodified applications.

Our proposed system requires zero modifications to both the application and the OS: no app instrumentation or code changes, and no kernel patches or new kernel subsystems. Our deployment is a userspace agent that (i) samples existing kernel surfaces (`perf_event_open`, `/proc`, `/sys`) and (ii) applies settings via standard knobs (`sysfs`, `sysctl`).

2 Constraints for Practical ML-based OS Optimization

The lack of visibility between the OS and applications imposes a set of practical constraints on any realistic ML-based optimizer.

The Black-Box Application: A core constraint is that the OS must treat applications as black boxes, which restricts gathering insights about their internal state. This opacity stems from 1) software complexity, 2) a lack of standardized performance interfaces, and 3) ubiquitous legacy applications without instrumentation [21, 31, 33]. Unlike specialized domains like databases that have built-in metric reporting, general-purpose OS components like the scheduler cannot assume access to application-specific metrics (*e.g.*, , request latency).

Even when APIs exist [16, 22], using them requires significant engineering effort, introduces dependencies, and adds overhead. This makes approaches that require application changes or migrating to a new OS entirely [42] non-starters for most production environments. The problem is amplified further in virtualized data centers, as the host OS has no insight into guest VMs, and breaching this isolation would violate security and privacy guarantees. Therefore, solutions that require modifying applications are often infeasible due to legal, technical, and risk-management reasons.

Data Scarcity and Quality: The black-box nature of applications leads to scarce, context-specific training data. While some ML domains benefit from large, standardized datasets, performance data is closely tied to its originating environment. Consequently, any model trained to map system metrics to application performance is implicitly specialized to the specific context in which the data was collected. This context includes: **a) The Application:** A model trained on an OLTP workload like TPCC has little relevance for an OLAP workload like TPCH. **b) The Workload:** Performance characteristics change dramatically with the request mix, load intensity, or the presence of noisy antagonists. **c) The System:** The underlying hardware, kernel version, and software dependencies all influence performance.

Even large, public workload traces [10, 18, 29, 34, 38, 39], while valuable, lack direct application performance metrics and contain only a limited set of system-level counters (often just resource utilization), making them unsuitable for training fine-grained optimization models [4].

We therefore reach a generalization barrier. A provider might train a model on a benchmark workload, but it will be deployed on systems running different applications under variable conditions. The model is therefore almost certain to fail, not because the algorithm is flawed, but because the rules it learned from one context do not apply to another. This problem is the primary motivation for our case study (§3), where we demonstrate it empirically.

3 Case Study: Challenges Exposed via CPU Scheduler Tuning

The constraints outlined above manifest as concrete challenges for any non-intrusive optimizer. To illustrate this, we demonstrate two resulting problems: 1) single metrics fail to generalize across workloads, and 2) even complex models built from these metrics are brittle to changing conditions.

Experimental Setup: We conducted a series of experiments on a machine equipped with 2×Intel Xeon Silver 4114 10-core CPUs at 2.20 GHz, 192GB of RAM, and a 480 GB 6G SATA SSD, hosted on CloudLab [14]. We tuned the Linux Completely Fair Scheduler (CFS) [36] for two distinct workloads: TPC-C implemented on Benchbase [12], and the sysbench OLTP-RW benchmark [23], both hosted on PostgresSQL 14.18 with 16 workers pinned to two CPU cores. The system was kept under high load, $\sim$ 90% CPU utilization.

In each experiment, we used Bayesian Optimization (via SMAC3 [27]) to tune CFS's `min_granularity_ns`, which controls the minimum time a task runs on a CPU before being preempted. Low values can improve responsiveness for I/O-bound tasks at the cost of higher context-switching overhead. High values reduce this overhead by allowing tasks longer, uninterrupted run times, which benefits CPU-bound computations. We implemented a tuner that collects application and system-wide measurements every 10 seconds, runs Bayesian Optimization, and applies the new proposed parameters based on the observed metrics. We ran the experiment over 100 iterations. The goal for both experiments was to minimize the application's tail latency.

Challenge 1: Single Metrics Fail to Generalize: A common workaround for the lack of application metrics is to optimize for a single, easy-to-measure system proxy like Instructions Per Cycle (IPC). However, the utility of such a

proxy is not universal. For TPC-C, IPC appears to be a reasonable proxy as it has a significant Pearson correlation of $r = 0.61$ with tail latency. As the top plot in Figure 1 shows, using BO to maximize IPC yields a 17.3% improvement in p99 latency over the default scheduler configuration. However, this relationship completely breaks down for sysbench—IPC has a near-zero correlation with latency ($r = 0.007$) and targeting it actually increases p95 latency by 18.6%.

More surprisingly, targeting a seemingly "bad" metric—maximizing cache misses—yielded even better results. For TPC-C, this improved median latency by 16.6%, while for sysbench it produced the best configuration overall, reducing latency by a massive 55.6% over the default. This counterintuitive outcome occurs because the optimizer discovers scheduler settings that reduce queueing delays by forcing frequent context switches. While this behavior pollutes the cache (increasing misses) and lowers CPU efficiency (reducing IPC), it prevents any single thread from waiting for too long and ensures all workers make progress, which is critical for minimizing tail latency in a contended system. This proves that the relationship between system metrics and application performance is complex and often non-linear, making the choice of a single proxy or even a static combination of proxies not just unreliable, but potentially counterproductive.

Challenge 2: Complex Models Are Brittle: The failure of single-metric proxies suggests that a more robust approach is needed. The logical next step is to move beyond any single counter and instead use a feature vector of system-level metrics to capture a more complete performance signature.

One might argue that a complex model trained on such a rich feature vector—using feature engineering or clustering to identify informative signals [37]—would be more robust.

To test this, we built a gradient boosting regression model to predict the absolute p99 latency for TPC-C using a wide array of system-wide metrics. As shown in Table 1, when trained and evaluated on a stable TPC-C workload, the model was effective, achieving an R^2 of 0.72. However, the model's performance proved brittle. When we applied this exact same trained model to a TPC-C workload co-located with a background "antagonist" process, the model failed completely, yielding a negative R^2. This means its predictions were worse than simply guessing the mean, as it had overfit to specific relationships that did not hold under noisy conditions.

This failure occurs even though some individual metrics, like IPC, maintain a superficially strong correlation with latency in both environments (Table 1). A high correlation for a single metric is not enough; the model's failure stems from its reliance on the entire set of relationships between features, which are unstable. For instance, the model learned to heavily weigh context switches, a metric whose relationship to latency changed dramatically in the presence of noise. Even complex predictive models are too fragile for real-world deployment, where environmental conditions are never static.

4 A Framework for Non-Intrusive OS Optimization

Our Goal: For real-world impact, ML-based OS improvements must target widely-used systems like Linux, requiring absolutely no application modifications and operating under the constraints of visibility and data availability.

The challenges demonstrated in our case study necessitate a new approach. Our proposed framework, detailed in this section, provides one by sidestepping absolute performance predictions and instead focusing on learning the more robust signal of relative performance (see Figure 2).

Our framework has three stages: we (i) define a diversified performance signature (§4.1), (ii) train archetype-specific pairwise rankers offline (§4.2), and (iii) use gated selection with a global ranker to drive a preferential online tuner (see Algorithm 1).

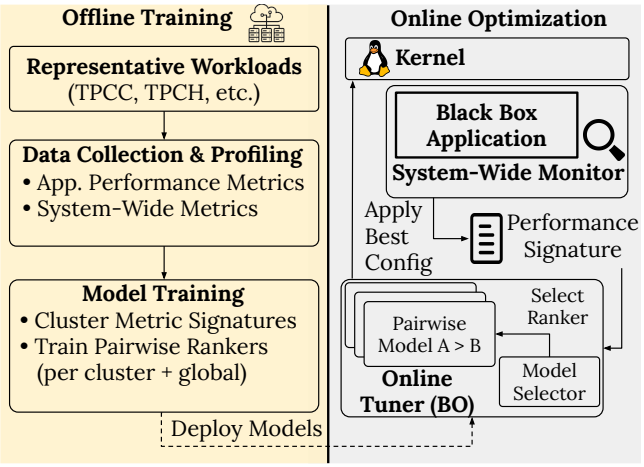

Figure 2. A non-intrusive OS optimization framework. **Offline Training (left):** Representative workloads are profiled to train a suite of specialized pairwise ranking models, one for each application archetype identified via clustering, and a global model used when the online performance signature does not match an existing archetype. **Online Optimization (right):** On a production server, a system monitor observes the black-box application's metric signature to select the appropriate pre-trained ranker, which is used as a pairwise oracle for an online tuner (e.g., BO), guiding it toward the optimal OS configuration using only system-level metrics.

4.1 Performance Signature and Archetype Selection

Signature: The framework builds a diversified performance signature x over 10 s intervals from Linux's existing surfaces (`perf stat`, `/proc`, `/sys`). Categories include CPU core (cycles, instructions, IPC, stalls), scheduler (context switches, runqueue length, migrations, load averages), memory/VM (faults, reclaim, NUMA hit/miss, active/inactive), cache/LLC

Algorithm 1 Online tuner with archetype gating and guardrails

Require: initial config c_0, tuner state $\mathcal{T}$, thresholds τ, δ

$c \leftarrow c_0$ ▷ Initialize incumbent configuration
loop
  $x \leftarrow$ GetCurrentSignature() ▷ Sample OS state
  $k \leftarrow$ NearestArchetype(x)
  if IsNovel(x, τ) **then**
    $\mathcal{R} \leftarrow$ GLOBAL
  else
    $\mathcal{R} \leftarrow$ RANKER[k]
  end if
  $c' \leftarrow$ Propose($\mathcal{T}, \mathcal{R}, x$) ▷ Propose challenger
  $m \leftarrow \mathcal{R}$.Score(x, c') − $\mathcal{R}$.Score(x, c)
  if $m < \delta$ **or** ProxyFlags(x, c') **then**
    $c_{next} \leftarrow c$ ▷ Reject and keep incumbent
  else
    ApplyConfig(c')
    ObservePreference($\mathcal{T}$, winner = c', loser = c)
    $c_{next} \leftarrow c'$ ▷ Promote challenger
  end if
  $c \leftarrow c_{next}$ ▷ Update incumbent for next loop
  Sleep(interval)
end loop

(references/misses), disk I/O (`/proc/diskstats`), network (`/proc/net/snmp`), and CPU utilization.

Archetypes: In the offline phase, we collect signatures from a comprehensive set of representative applications that includes most of the application types we expect to see in the cloud today. The signatures are then clustered (e.g., using k-means on a PCA embedding) to define application archetypes. A pairwise ranker is then trained for each archetype, plus a lightweight global ranker on the union of all data. In the online phase, the agent uses k-NN with Mahalanobis distance to select the best model; if the signature's distance to the nearest centroid exceeds a novelty threshold τ, it falls back to the global ranker. The chosen ranker then acts as the pairwise oracle for the online tuner.

4.2 Simplifying Prediction with Relative Ranking

The Path: The brittleness of both single-metric proxies and complex predictive models motivates our core proposal: we must reframe the problem from predicting absolute performance to the more robust and generalizable task of learning relative rankings.

The model's inability to generalize is a critical barrier as seen in (§3). Even with sophisticated feature engineering, building a general-purpose model to predict absolute performance across diverse workloads remains exceptionally difficult. Our framework's core idea is to sidestep the intractable problem of absolute performance prediction and instead focus on learning relative rankings, which is extensively studied in domains like search and recommendation [8]. Many OS optimizations do not need precise absolute predictions; it often suffices to know if one configuration is better than another. This approach proved far more robust to the challenges of noise and environmental variation demonstrated in our case study.

Table 1. Comparison of predictive approaches on Stable vs. Noisy TPC-C.

Predictive Approach	Stable	Noisy
Single Metric Pearson r with p99 Latency		
IPC	0.61	0.62
Context Switches	-0.19	-0.61
Cache Misses	0.41	0.57
Absolute Prediction Model (Trained on Stable)		
R^2 Score	0.72	<0
Pairwise Ranking Model (Trained on Stable)		
Accuracy	81.7%	70.2%
AUC	0.90	0.79

From Prediction to Ranking: While our absolute prediction model showed promise on a stable TPC-C workload, its performance proved brittle when conditions changed. Table 1 contrasts the different predictive approaches. The complex absolute prediction model achieves a strong R^2 of 0.72 on stable data, but its performance collapses to a negative R^2 under noisy conditions, rendering it worse than useless.

In contrast, a pairwise classification model demonstrates far greater robustness. Its accuracy only drops from 81.7% on stable data to 70.2% under noise, and it continues to provide a strong predictive signal where the absolute model failed. The table also reveals why the absolute model is so brittle: its predictions relied heavily on context switches (41.7% feature importance), a metric whose relationship with latency is unstable—its weak negative correlation ($r = -0.19$) on the stable workload becomes a stronger ($r = -0.61$) under noise.

The pairwise model succeeds because it learns a diversified performance signature from many features, with its top feature contributing less than 8%. By relying on a broader set of signals rather than overfitting to precise, unstable relationships, the diversified model is inherently more robust to the kind of environmental variation that causes single-metric proxies and monolithic prediction models to fail.

Who Manages the Data Collection? In practice, neither end users nor individual developers can run the extensive experiments required for our data-driven approach. Instead, cloud providers and large-scale on-premise operators have both the incentive and the authority to do so. From a business standpoint, providers benefit from more efficient resource

utilization and improved service quality. They can systematically deploy representative benchmarks (*e.g.*, DCPerf [30], Fleetbench [2]) under different OS configurations, recording both hardware counters and application performance to build the rich datasets necessary for model training.

Who Builds the Models at Scale? The provider centrally trains pairwise ranking models using ground-truth data from representative benchmarks. These ranking models are transferable because they learn the generalizable, directional relationship between a configuration change and its performance impact, unlike absolute models which overfit to the specific performance values. The provider then distributes the models, sparing developers the complexity of this process.

Second, for the online optimizer itself (the BO agent), federated learning is a natural fit for continuous improvement. Each server in the fleet can learn locally from its own tuning experience, identifying which regions of the configuration space are promising. Knowledge can then be aggregated into a global model without sharing any raw, potentially sensitive user data, allowing the entire fleet to benefit from the collective experience. This can be done with privacy preserved through techniques like differential privacy [1, 15] and metric encryption [41].

4.3 Tackling Environmental Variation with Causal Models

The second major challenge—performance variability due to noise, antagonists, and changing workload phases—is fundamentally harder to address. Correlation-based ML models, including the ranking models discussed above, struggle with this because they often learn spurious correlations that do not hold when conditions change.

Causal Modeling to Enhance Data Realism: We propose exploring causal modeling as a promising, albeit challenging, research direction to build models that are robust to environmental variation. Approaches like CausalSim [3] attempt to learn cause-effect relationships between system interventions (*e.g.*, scheduling decisions) and performance outcomes. By disentangling the true impact of a configuration change from confounding environmental factors, causal models have the potential to generalize far more effectively than their correlation-based counterparts. A naive simulator may confuse hardware changes with improved scheduling policies, but by isolating causal factors, these models can ensure that performance gains are attributed to the correct policy. While developing a complete and efficient causal engine for the OS is a very hard, open problem, it represents a critical frontier for creating truly adaptive systems.

5 Open Challenges and The Path Forward

A non-intrusive, ML-driven OS tuner must be safe under uncertainty, cheap to run, and easy to operate. We summarize the main challenges and the most promising paths forward.

5.1 Operational challenges

Unseen workloads and representativeness: Archetype rankers are only as good as the offline corpus. Our gating and global ranker mitigates mismatch, but cannot eliminate it. Building broad, periodically refreshed corpora is essential.

Detecting application class online: Signature-space gating must be accurate and low-overhead. Beyond k-NN, lightweight density models and online drift detectors can improve assignment and trigger re-gating when phases change.

Drift and non-stationarity: Workloads, interference, and kernel versions evolve, while the hardware varies. The online tuner should include conservative guardrails (incumbent protection, margin δ, proxy sanity checks) and support safe exploration under drift.

5.2 The Path Forward

The future involves three key research directions:

Building Pairwise-driven Online Tuners: The immediate work is to build practical online agents that leverage our pre-trained pairwise models. This involves integrating the rankers with lightweight online tuning frameworks. Promising directions include Preferential Bayesian Optimization [17], which can model a latent performance landscape from ‘A > B‘ comparisons, or simpler approaches based on multi-armed bandits [5], and variants designed for pairwise comparisons like Dueling Bandits [40].

Validating and Generalizing Ranking Models: The next step is to systematically build and validate a "model zoo" of pairwise rankers. This requires large-scale data collection campaigns—likely managed by cloud providers—to train models that are robust across a wide variety of applications (databases, web servers, HPC), hardware (different CPU generations, memory hierarchies), and software stacks (kernel versions, libraries). The goal is to create a library of specialized models that an OS can select from at runtime based on an application's observed metric signature.

Moving from Correlation to Causality: While ranking models are more robust to noise, they are still correlational. The ultimate frontier is to build a causal inference engine for the OS. Such an engine would allow the system to understand the true impact of its actions, distinguishing performance changes caused by a tuning decision from those caused by external factors like workload shifts or noisy neighbors. This is a very hard problem, but it represents a necessary transition from reactive tuning to proactive, reason-based optimization.

Acknowledgments

This work is partly sponsored by the Columbia–Dream Sports AI Innovation Center.

References

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. 308–318.

[2] Andreas Abel, Yuying Li, Richard O'Grady, Chris Kennelly, and Darryl Gove. 2024. A Profiling-Based Benchmark Suite for Warehouse-Scale Computers. In *2024 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 325–327. doi:10.1109/ISPASS61541.2024.00046

[3] Abdullah Alomar, Pouya Hamadanian, Arash Nasr-Esfahany, Anish Agarwal, Mohammad Alizadeh, and Devavrat Shah. 2023. {CausalSim}: A Causal Framework for Unbiased {Trace-Driven} Simulation. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 1115–1147.

[4] George Amvrosiadis, Jun Woo Park, Gregory R Ganger, Garth A Gibson, Elisabeth Baseman, and Nathan DeBardeleben. 2018. On the diversity of cluster workloads and its impact on research results. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. 533–546.

[5] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine learning* 47, 2 (2002), 235–256.

[6] Romil Bhardwaj, Kirthevasan Kandasamy, Asim Biswal, Wenshuo Guo, Benjamin Hindman, Joseph Gonzalez, Michael Jordan, and Ion Stoica. 2023. Cilantro:{Performance-Aware} resource allocation for general objectives via online feedback. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*. 623–643.

[7] Manaf Bin-Yahya, Yifei Zhao, Hossein Shafieirad, Anthony Ho, Shijun Yin, Fanzhao Wang, and Geng Li. 2024. {Config-Snob}: Tuning for the Best Configurations of Networking Protocol Stack. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*. 749–765.

[8] Christopher Burges. 2010. From ranknet to lambdarank to lambdamart: An overview. *Learning* 11 (01 2010).

[9] Jingde Chen, Subho S Banerjee, Zbigniew T Kalbarczyk, and Ravishankar K Iyer. 2020. Machine learning for load balancing in the linux kernel. In *Proceedings of the 11th ACM SIGOPS Asia-Pacific Workshop on Systems*. 67–74.

[10] Eli Cortez, Anand Bonde, Alexandre Muzio, Mark Russinovich, Marcus Fontoura, and Ricardo Bianchini. 2017. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In *Proceedings of the 26th Symposium on Operating Systems Principles*. 153–167.

[11] Carlo Curino, Neha Godwal, Brian Kroth, Sergiy Kuryata, Greg Lapinski, Siqi Liu, Slava Oks, Olga Poppe, Adam Smiechowski, Ed Thayer, et al. 2020. MLOS: An infrastructure for automated software performance engineering. In *Proceedings of the Fourth International Workshop on Data Management for End-to-End Machine Learning*. 1–5.

[12] Djellel Eddine Difallah, Andrew Pavlo, Carlo Curino, and Philippe Cudré-Mauroux. 2013. OLTP-Bench: An Extensible Testbed for Benchmarking Relational Databases. *PVLDB* 7, 4 (2013), 277–288. http://www.vldb.org/pvldb/vol7/p277-difallah.pdf

[13] Songyun Duan, Vamsidhar Thummala, and Shivnath Babu. 2009. Tuning database configuration parameters with iTuned. *Proc. VLDB Endow.* 2, 1 (Aug. 2009), 1246–1257. doi:10.14778/1687627.1687767

[14] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. 2019. The Design and Operation of CloudLab. In *Proceedings of the USENIX Annual Technical Conference (ATC)*. 1–14. https://www.flux.utah.edu/paper/duplyakin-atc19

[15] Ahmed El Ouadrhiri and Ahmed Abdelhadi. 2022. Differential privacy for deep and federated learning: A survey. *IEEE access* 10 (2022), 22359–22380.

[16] Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and David Sculley. 2017. Google vizier: A service for black-box optimization. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*. 1487–1495.

[17] Javier González, Zhenwen Dai, Andreas Damianou, and Neil D Lawrence. 2017. Preferential bayesian optimization. In *International Conference on Machine Learning*. PMLR, 1282–1291.

[18] Alibaba Group. 2018. Alibaba Cluster Data. https://github.com/alibaba/clusterdata.

[19] Mingzhe Hao, Levent Toksoz, Nanqinqin Li, Edward Edberg Halim, Henry Hoffmann, and Haryadi S Gunawi. 2020. {LinnOS}: Predictability on unpredictable flash storage with a light neural network. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. 173–190.

[20] Lao Jiale, Wang Jianping, Chen Wanghu, Wang Yibo, Zhang Yunjia, Tang Mingjie, Li Yufei, Cheng Zhiyuan, and Wang Jianguo. 2024. GPTuner: A Manual-Reading Database Tuning System via GPT-Guided Bayesian Optimization. *Proceedings of the VLDB Endowment* 17, 8 (2024), 1939–1952.

[21] Kostis Kaffes, Dragos Sbirlea, Yiyan Lin, David Lo, and Christos Kozyrakis. 2020. Leveraging application classes to save power in highly-utilized data centers. In *Proceedings of the 11th ACM Symposium on Cloud Computing (SoCC '20)*. Association for Computing Machinery, Virtual Event, USA, 134–149. doi:10.1145/3419111.3421274

[22] Ajaykrishna Karthikeyan, Nagarajan Natarajan, Gagan Somashekar, Lei Zhao, Ranjita Bhagwan, Rodrigo Fonseca, Tatiana Racheva, and Yogesh Bansal. 2023. {SelfTune}: Tuning Cluster Managers. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 1097–1114.

[23] Alexey Kopytov, Peter Zaitsev, and the Sysbench Project Contributors. [n. d.]. *sysbench: Scriptable database and system performance benchmark*. https://github.com/akopytov/sysbench

[24] Brian Kroth, Sergiy Matusevych, Rana Alotaibi, Yiwen Zhu, Anja Gruenheid, and Yuanyuan Tian. 2024. MLOS in Action: Bridging the Gap Between Experimentation and Auto-Tuning in the Cloud. *Proceedings of the VLDB Endowment* 17, 12 (2024), 4269–4272.

[25] Benjamin Letham, Brian Karrer, Guilherme Ottoni, and Eytan Bakshy. 2019. Constrained Bayesian optimization with noisy experiments. (2019).

[26] Yu Liang, Riwei Pan, Tianyu Ren, Yufei Cui, Rachata Ausavarungnirun, Xianzhang Chen, Changlong Li, Tei-Wei Kuo, and Chun Jason Xue. 2022. {CacheSifter}: Sifting Cache Files for Boosted Mobile Performance and Lifetime. In *20th USENIX Conference on File and Storage Technologies (FAST 22)*. 445–459.

[27] Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhkopf, René Sass, and Frank Hutter. 2022. SMAC3: A Versatile Bayesian Optimization Package for Hyperparameter Optimization. *Journal of Machine Learning Research* 23, 54 (2022), 1–9. http://jmlr.org/papers/v23/21-0888.html

[28] Evan Liu, Milad Hashemi, Kevin Swersky, Parthasarathy Ranganathan, and Junwhan Ahn. 2020. An imitation learning approach for cache replacement. In *International Conference on Machine Learning*. PMLR, 6237–6247.

[29] Charles Reiss, John Wilkes, and Joseph L Hellerstein. 2011. Google cluster-usage traces: format+ schema. *Google Inc., White Paper* 1 (2011), 1–14.

[30] Facebook Research. 2023. DCPerf: Data Center Performance Benchmarking Tool. https://github.com/facebookresearch/DCPerf. Accessed: 2024-01-15.

[31] Krzysztof Rzadca, Pawel Findeisen, Jacek Swiderski, Przemyslaw Zych, Przemyslaw Broniek, Jarek Kusmierek, Pawel Nowak, Beata Strack, Piotr Witusowski, Steven Hand, et al. 2020. Autopilot: workload autoscaling at google. In *Proceedings of the Fifteenth European Conference*

on Computer Systems. 1–16.

[32] Divyanshu Saxena, Nihal Sharma, Donghyun Kim, Rohit Dwivedula, Jiayi Chen, Chenxi Yang, Sriram Ravula, Zichao Hu, Aditya Akella, Sebastian Angel, et al. 2023. On a Foundation Model for Operating Systems. *arXiv preprint arXiv:2312.07813* (2023).

[33] Robert C Seacord, Daniel Plakosh, and Grace A Lewis. 2003. *Modernizing legacy systems: software technologies, engineering processes, and business practices*. Addison-Wesley Professional.

[34] Mohammad Shahrad, Rodrigo Fonseca, Inigo Goiri, Gohar Chaudhry, Paul Batum, Jason Cooke, Eduardo Laureano, Colby Tresness, Mark Russinovich, and Ricardo Bianchini. 2020. Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider. In *Proceedings of the 2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, Boston, MA.

[35] Zhenyu Song, Daniel S Berger, Kai Li, Anees Shaikh, Wyatt Lloyd, Soudeh Ghorbani, Changhoon Kim, Aditya Akella, Arvind Krishnamurthy, Emmett Witchel, et al. 2020. Learning relaxed belady for content distribution network caching. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. 529–544.

[36] The Linux Kernel Development Community. 2024. *Completely Fair Scheduler (CFS)*. https://docs.kernel.org/scheduler/sched-design-CFS.html

[37] Dana Van Aken, Andrew Pavlo, Geoffrey J Gordon, and Bohan Zhang. 2017. Automatic database management system tuning through large-scale machine learning. In *Proceedings of the 2017 ACM international conference on management of data*. 1009–1024.

[38] John Wilkes. 2011. More Google cluster data. Google research blog. Posted at http://googleresearch.blogspot.com/2011/11/more-google-cluster-data.html..

[39] John Wilkes. 2020. Yet more Google compute cluster trace data. Google research blog. Posted at https://ai.googleblog.com/2020/04/yet-more-google-compute-cluster-trace.html..

[40] Yisong Yue, Josef Broder, Robert Kleinberg, and Thorsten Joachims. 2012. The k-armed dueling bandits problem. *J. Comput. System Sci.* 78, 5 (2012), 1538–1556.

[41] Chengliang Zhang, Suyi Li, Junzhe Xia, Wei Wang, Feng Yan, and Yang Liu. 2020. {BatchCrypt}: Efficient homomorphic encryption for {Cross-Silo} federated learning. In *2020 USENIX annual technical conference (USENIX ATC 20)*. 493–506.

[42] Yiying Zhang and Yutong Huang. 2019. " Learned" Operating Systems. *ACM SIGOPS Operating Systems Review* 53, 1 (2019), 40–45.

[43] 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* (Santa Clara, California) *(SoCC '17)*. Association for Computing Machinery, New York, NY, USA, 338–350. doi:10.1145/3127479.3128605