Machine Learning for Cloud Data Systems: the Progress so far and the Path Forward

Alekh Jindal
Microsoft
alekh.jindal@microsoft.com

Matteo Interlandi
Microsoft
mainterl@microsoft.com

ABSTRACT

The goal of this tutorial is to educate the audience about the state of the art in ML for cloud data systems, both in research and in practice. The tutorial is divided in two parts: the progress, and the path forward.

Part I covers the recent successes in deploying machine learning solutions for cloud data systems. We will discuss the practical considerations taken into account and the progress made at various levels. The goal is to compare and contrast the promise of ML for systems with the ground actually covered in industry.

Finally, Part II discusses practical issues of machine learning in the enterprise covering the generation of explanations, model debugging, model deployment, model management, constraints on eyes-on data usage and anonymization, and a discussion of the technical debt that can accrue through machine learning and models in the enterprise.

PVLDB Reference Format:
Alekh Jindal and Matteo Interlandi. Machine Learning for Cloud Data Systems: the Progress so far and the Path Forward. PVLDB, 14(12): 3202-3205, 2021.
doi:10.14778/3476311.3476408

1 INTRODUCTION

Modern cloud has democratized access to sophisticated and scalable data processing systems. In contrast to the days of long hardware and software procurement cycles, before any data processing could be done, the modern cloud has transformed data processing capabilities into commodities of instant gratification — the state of the art data processing stack is available at one's disposal in a matter of few clicks. In addition to quick provisioning, cloud also offers managed data services where many of the operational tasks are taken care by the cloud provider. These could include security, updates, backups, scaling up or down, reliability, tracking, and support for various systems, among others, thus making the lives of system users much easier. Finally, there is a newer trend for serverless data processing infrastructures which further relieves users from deciding and paying for a fixed set of resource configurations. Instead, the cloud provider takes care of allocating resources for each of the user tasks and they pay only for the actual processing incurred. All of this has resulted in cloud becoming the destination when it comes to scale and complexity in data processing.

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit https://creativecommons.org/licenses/by-nc-nd/4.0/ to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.
Proceedings of the VLDB Endowment, Vol. 14, No. 12 ISSN 2150-8097.
doi:10.14778/3476311.3476408

Unfortunately, the shift to modern cloud has also surfaced newer pain. Today, while even non-expert users can quickly compose various cloud data services into highly complex application workflows, they quickly realize the challenges in stitching, configuring, tuning, debugging, analyzing, or just getting the most out of their data services. This is because cloud users often do not have the expertise or domain knowledge, or access to database administrators (DBAs) that were traditionally hired on-premise, or even the control to the low level system components in managed services. This is painful not just for the end users but also for cloud providers since the onus is now on them to provide a good user experience. Furthermore, cloud data systems also end up having too many moving parts, with more layers of virtualization and abstraction, which makes them much harder to manage and tune, leaving users and/or cloud providers to deal with infeasible decision space, settle with sub-optimal performance, and yet end up with much higher operational costs. All of these become critical areas for the cloud to cover as it embraces the next wave of digital transformation for businesses to stay relevant and competitive, along with the next level of customer expectations that include higher quality of service (QoS) and lower total cost of ownership (TCO).

Luckily, cloud data systems also have an unfair advantage: they have visibility to massive amount of workloads that capture the heterogeneity across many different users and applications, as well as the changes over time. This global visibility coupled with advances in machine learning (ML) tools and libraries, end-to-end system control, and faster release cycles has several implications. First, there is a push to make cloud data systems data-driven, i.e., move from intuition and guesstimates to quantifiable insights. Second, there is an opportunity to introduce self-tuning feedback-loops in cloud data systems, i.e., learn from how things went in the past and apply them to future workloads. And third, there is a need to evolve the cloud data systems to newer user requirements, workload types, hardware designs, and other things over time, i.e., adapt to the newer trends and realities. In fact, leveraging cloud observability to constantly learn and improve the system behavior is fast emerging as a design principle for modern cloud data systems and presents enormous potential to rebuild the cloud that was promised, i.e., one which is simpler, faster, and cheaper.

In the remaining of this proposal, we outline the two parts of the tutorial, namely the progress, and the path forward.

2 THE PROGRESS

The newfound excitement in ML for systems started with the case for learned indexes [32]. Afterwards, there has been has been a deluge of ideas from the academic and research community on how to integrate ML into pretty much any system layer. In industry however, the pace is more regulated as those ideas get hardened into

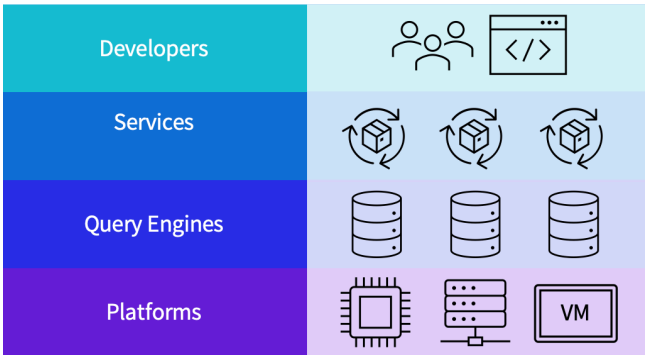

Figure 1: Layers of the cloud data systems that will be covered during the tutorial.

products. In this tutorial we illustrate a snapshot of the progress made in bringing ML for systems closer to practice in cloud data systems. In order to achieve this, we will represent the cloud data systems as layered into four different levels (Figure 1) sitting one on each other.

- At the bottom level we have *cloud platforms* (Section 2.1). ML solutions in this level tries to address questions such as *What are the right machine SKUs for my workload? How do we meet VM provisioning SLAs given the current load?*
- Query engines (Section 2.2) runs on top of cloud platforms. At this level there are several interesting components that can be tuned using ML. In general, at this level we are interested in knowing *What can we learn from past query workloads? How can we improve query plans in the future?*
- Higher level cloud services (Section 2.3) can be employed on the query engines. At this level, cloud customers are interested in solutions regarding questions such as *What are the peak and low load intervals? When do we backup? What is the optimal scaling strategy given the current workload?*
- Finally, developers (Section 2.4) are end users for all the above levels. At this level we will be exploring works answering questions such as *How can I inject data science in my project? What is the end-to-end experience when ML is part of my software? How can I exploit ML to be more productive?*

Next we will double click on each of the levels starting bottom-up from the cloud platforms.

2.1 Platforms

Cloud platforms are typically large and often unwieldy. Therefore, frameworks designed specifically for tuning the hardware (SKU design, power capping) and software (containers per machine and other software configurations) are starting to emerge. For example, one can model performance with various VM-level characteristics such as storage type, network, CPU, etc., [49], select the best cloud configurations [7], or choose the flash storage [6]. The next step is to study cloud traces and provide recommendations based on previous workloads [30]. In fact, cloud workloads are often recurrent [25], whereby past traces are good approximations for future loads. At this point applying ML is the natural next step. Indeed, ML applied to cloud platform and resource management is still in its infancy [10] but the premises are appealing, and the trend is there. Cluster-level tuning has become part of Cosmos [67], improving resource management with machine learning in now in Azure [11], SKU recommendations are available when migrating SQL Server from on-premise to cloud [40], and similar tuning and migration services are also available in AWS and GCP.

2.2 Query Engines

In the query engine domain, there has been tremendous excitement and interest in the last few years, especially in academia. [1] In general, the most common targets for ML for system approaches over query engines are *indexes and layout [22, 22, 32], cardinality [17, 23, 64], cost modeling [5, 20, 34], query planner and optimizer [36, 37], query resources [35, 61, 66], and self-tuning [14, 18].*

Nevertheless, there is a collective realization of the challenges involved in bringing ML for query engines to industry. For instance, bringing learned indexes into BigTable required a completely new approach [2]. Similarly, recent discussion on the practical implications of learned cardinality [63] concluded that is better to narrow down the focus to improving cardinality where it really matters [46]. This brought the first industry deployment of learned cardinality in Cosmos [28]. Interesting enough, the same infrastructure used for learned cardinality [28], can be reused for cost modeling [59], as well as views [26, 29], automatic scaling [56], and steering the query optimizer using learned query hints [47]. This brought to the realization that what is really needed for injecting ML into query engines at cloud scale is a broader infrastructure whereby learned components can be plugged when appropriate. On this respect, Cosmos currently embraces the Microlearning [28] architecture and follows the workload optimization patterns of [27]. To prove the generality of the approach, a similar infrastructure is currently under development over the Spark stack as well [54].

Finally, optimizing resources is critical on the Cloud. Automatic resource management is becoming more practical. Examples are resource optimizations such as memory grant feedback in SQL Server [41] or degree of parallelism in big data workloads [8, 52, 56]. Auto-tuning of cloud-scale query engines looks like a far-away dream, although commercial databases are already providing such functionalities [38, 50].

2.3 Services

Cloud data platforms are often accompanied by a number of low-level tasks and services to keep it functional. Several prior work have looked into how cloud services can be managed at scale without human intervention. Examples include auto-scaling [19], mitigating slow instances [62], auto-sharding [3], load prediction and backup [53], among others. Even in this space, lately we are witnessing a new set of ML-driven approaches to proactively adjust the services given observed workloads. For example, predictive auto-scaling is now available in many cloud data services such as HDInsight [39] and EC2 [9]. Likewise, predictive backups are now also possible [53], as well ML-driven checkpointing decisions [66].

[1] An extensive compilation of recent papers and developments can be found here [1].

2.4 Developers

Finally, developers are at the core of data systems, and so improving developer experience is paramount. More advanced ML-driven developer experiences are, for instance, now available in Visual Studio [43], VSCode, and Github (e.g., copilot [21]). While a large amount of tooling has been developed for application developers, there is also a push to improve the lives of system developers by introducing newer ways to build ML infused software that is easy to track, debug, evolve, and performance engineered over time [13]. This is also connected to the Software 2.0 trend [31].

3 THE PATH FORWARD

We now list many of the open problems that we see going forward as ML for systems becomes more mainstream in industry and practice.

Experimentation. Getting good accuracy on train/test/validation datasets is just the beginning for ML for systems. The major steps in the workflow include testing the model over large production workloads or on canary settings. This requires an experimentation framework (e.g., Diametrics [15], SCOPE Flighting tool) for extensive A/B testing, performance monitoring over diverse metrics, storage and retrieval of historical runs, as well as anonymization. A good experimentation frameworks allows the product support team to build trust that the machine learning component will not create problems over time.

Model Serving. Once the product team trusts the ML model, it can be deployed in several flavors [33]: one can use containers (e.g., as in Clipper [12]) or out of process execution; an alternative is to use in-process execution by importing the models as a libraries (e.g., as in ML.NET [4] or Scikit-Learn [51]), or by converting the model into a target format that then can be called at runtime (e.g., achieved by using ONNX [44] or Hummingbird [45]). Furthermore, predictions can be served online (e.g., one-at-a-time) or batched. The former is preferred when predictions are on the critical path (e.g., cardinality estimation is used by the query optimizer and therefore only few milliseconds are allowed to render the prediction) while the latter approach is commonly used for achieving better throughput and taking advantage of modern hardware accelerators.

Model Management Often multiple ML models have to be deployed within the same ML-enabled-feature, e.g., one model for operator, or because models are updated over time (e.g., every day, week, month) in order to avoid staleness and follow the workload trend. Having multiple models concurrently in the system, requires not only a proper model management infrastructure [60] allowing fast model retrieval, versioning, and update [55], but also a unified API because different models could be implemented using different frameworks (e.g., Scikit-Learn and PyTorch models could live together in the same deployment because the former provides a large set of "traditional" ML models such as tree-ensemble models, while the latter allows the deployment of state of the art DNNs). MLFlow [65] is an example of standard API for model inference commonly adopted by any major cloud vendor.

Performance Regressions. The most important requirement when deploying ML models in cloud data system is minimization of performance regressions: given the massive scale of cloud products, any performance regression will likely generate a ticket that must be manually addressed by a support team—any increase in the number of tickets generated could be a blocker. To address this concern, proper guards need to be put in place for making sure that models behave accordingly [16, 24].

Debugging and Root Cause Analysis. When some performance regressions or errors are generate by a ML-enhanced cloud data system, the infrastructure should provide extensive logging, as well as means to debug the predictions [48, 57]. Making direct use of cloud ML offerings such as Azure ML, SageMaker or Google Cloud AI allows to take advantage of their logging and debugging tools, instead of having to build another system, specific for the application.

Privacy and GDPR. Since models are likely trained over datasets generated from customer workloads, any sensible information must be properly anonymized. Furthermore, ML-for-systems product must be built with GDPR (or CCPA) requirements in mind in order to avoid regulatory and performance implications [58]. Finally, customer may request some data to be deleted, whereby updatable models [55] are a new interesting technique.

Common Platforms. All of the above aspects are relevant for every cloud data system that we want to enhance with ML. Major cloud provides such as Amazon, Google and Microsoft have several managed data systems products, and building a new ML for systems infrastructure each time is prohibitively expensive and redundant. Consolidating data products over a unified service and infrastructure whereby common ML components can be built once and reused across different products is therefore preferred. Azure Synapse [42] is a step forward in this direction.

4 PRESENTERS

Alekh Jindal is a Principle Scientist at Gray Systems Lab (GSL), Microsoft and manages the Redmond site of the lab. His research focuses on improving the performance of large-scale data-intensive systems. Earlier, he was a postdoc associate in the Database Group at MIT CSAIL. Alekh received his PhD from Saarland University, working on flexible and scalable data storage for traditional databases as well as for MapReduce. In the past 10 years, Alekh has served as a chair, PC member and reviewer at top-tier conferences in the field including SIGMOD, VLDB, ICDE, and SOCC. He received best paper awards at VLDB 2014 and CIDR 2011, and an honorable mention at SIGMOD 2021.

Matteo Interlandi is a Senior Scientist in the Gray Systems Lab (GSL) at Microsoft, working on scalable Machine Learning Systems. Before Microsoft, he was a Postdoctoral Scholar at the University of California, Los Angeles. Prior to joining UCLA, he was Research Associate at the Qatar Computing Research Institute and at the Institute for Human and Machine Cognition. Matteo's work have received an honorable mention at SIGMOD 2021 and was featured in the "Best of VLDB".

REFERENCES

[1] 2021. GitHub - LumingSun/ML4DB-paper-list: Papers for database systems powered by artificial intelligence (machine learning for database). https://github.com/LumingSun/ML4DB-paper-list. (Accessed on 03/20/2021).

[2] Hussam Abu-Libdeh, Deniz Altinbüken, Alex Beutel, Ed H. Chi, Lyric Doshi, Tim Kraska, Xiaozhou Li, Andy Ly, and Christopher Olston. 2020. Learned

Indexes for a Google-scale Disk-based Database. *CoRR* abs/2012.12501 (2020). arXiv:2012.12501 https://arxiv.org/abs/2012.12501
[3] Atul Adya and et al. 2016. Slicer: Auto-Sharding for Datacenter Applications.
[4] Zeeshan Ahmed and et al. 2019. Machine Learning at Microsoft with ML.NET. In *SIGKDD*.
[5] Mert Akdere and et al. 2012. Learning-based query performance modeling and prediction. In *ICDE*. 390–401.
[6] Christoph Albrecht and et al. 2013. Janus: Optimal Flash Provisioning for Cloud Storage Workloads. In *ATC*. 91–102.
[7] Omid Alipourfard and et al. 2017. Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics. In *NSDI*. 469–482.
[8] Malay Bag and et al. 2020. Towards Plan-aware Resource Allocation in Serverless Query Processing. In *HotCloud*.
[9] Jeff Barr. 2018. New – Predictive Scaling for EC2, Powered by Machine Learning. https://aws.amazon.com/blogs/aws/new-predictive-scaling-for-ec2-powered-by-machine-learning/.
[10] Ricardo Bianchini and et al. 2020. Toward ml-centric cloud platforms. *CACM* 63, 2 (2020), 50–59.
[11] Eli Cortez and et al. 2017. Resource Central: Understanding and Predicting Workloads for Improved Resource Management in Large Cloud Platforms. In *SOSP*. 153–167.
[12] Daniel Crankshaw and et al. 2017. Clipper: A Low-Latency Online Prediction Serving System. In *NSDI*.
[13] Carlo Curino and et al. 2020. MLOS: An Infrastructure for Automated Software Performance Engineering. In *DEEM*.
[14] Geoff Gordon Dana Van Aken and Andy Pavlo. 2017. Tuning Your DBMS Automatically with Machine Learning. https://aws.amazon.com/blogs/machine-learning/tuning-your-dbms-automatically-with-machine-learning/.
[15] Shaleen Deep and et al. 2020. DIAMetrics: Benchmarking Query Engines at Scale. *PVLDB* 13, 12 (Aug. 2020), 3285–3298. https://doi.org/10.14778/3415478.3415551
[16] Bailu Ding and et al. 2019. Ai meets ai: Leveraging query executions to improve index recommendations. In *SIGMOD*. 1241–1258.
[17] Anshuman Dutt and et al. 2019. Selectivity estimation for range predicates using lightweight models. *PVLDB* 12, 9 (2019), 1044–1057.
[18] Ayat Fekry and et al. 2020. To Tune or Not to Tune? In Search of Optimal Configurations for Data Analytics. In *SIGKDD*. 2494–2504.
[19] Avrilia Floratou and et al. 2017. Dhalion: Self-Regulating Stream Processing in Heron. *PVLDB* 10, 12 (2017), 1825–1836.
[20] Archana Ganapathi and et al. 2009. Predicting multiple metrics for queries: Better decisions enabled by machine learning. In *ICDE*. IEEE, 592–603.
[21] Github. 2021. GitHub Copilot. https://copilot.github.com/. (Accessed on 07/20/2021).
[22] Ali Hadian and Thomas Heinis. 2019. Considerations for Handling Updates in Learned Index Structures. In *aiDM*. Article 3.
[23] Shohedul Hasan and et al. 2019. Multi-attribute selectivity estimation using deep learning. arXiv:1903.09999
[24] H. Hossain and et al. 2020. PerfGuard: Deploying ML-for-Systems without Performance Regressions.
[25] Virajith Jalaparti, P. Bodík, I. Menache, Sriram Rao, K. Makarychev, and Matthew C. Caesar. 2015. Network-Aware Scheduling for Data-Parallel Jobs: Plan When You Can. *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication* (2015).
[26] Alekh Jindal and et. al. 2018. Computation reuse in analytics job service at microsoft. In *ICDE*. 191–203.
[27] Alekh Jindal and et al. 2019. Peregrine: Workload Optimization for Cloud Query Engines. In *SOCC*. 416–427.
[28] Alekh Jindal and et al. 2021. Microlearner: A fine-grained Learning Optimizer for Big Data Workloads at Microsoft. In *ICDE*.
[29] Alekh Jindal and et al. 2021. Production Experiences from Computation Reuse at Microsoft. In *EDBT*. 623–634. https://doi.org/10.5441/002/edbt.2021.72
[30] Sangeetha Abdu Jyothi and et al. 2016. Morpheus: Towards automated slos for enterprise clusters. In *OSDI*. 117–134.
[31] Andrej Karpathy. 2017. Software 2.0. https://medium.com/@karpathy/.
[32] Tim Kraska and et al. 2018. The Case for Learned Index Structures. In *SIGMOD*. ACM, 489–504. https://doi.org/10.1145/3183713.3196909
[33] Yunseong Lee and et al. 2018. From the Edge to the Cloud: Model Serving in ML.NET. *DEB* 41, 4 (2018).
[34] Jiexing Li and et al. 2012. Robust estimation of resource consumption for sql queries using statistical techniques. *PVLDB* 5, 11 (2012), 1555–1566.
[35] Hongzi Mao and et al. 2019. Learning scheduling algorithms for data processing clusters. In *SIGCOM*. 270–288.
[36] Ryan Marcus and et al. 2019. Neo: A Learned Query Optimizer. *PVLDB* 12 (2019), 1705–1718.
[37] Ryan Marcus and et al. 2020. Bao: Learning to Steer Query Optimizers. *ArXiv* abs/2004.03814 (2020).
[38] Microsoft. [n.d.]. SQL Server Auto-Tune. https://docs.microsoft.com/en-us/sql/relational-databases/automatic-tuning/automatic-tuning?view=sql-server-ver15. Accessed on 07/20/2021.
[39] Microsoft. 2019. Azure HDInsight—Autoscale is now generally available. https://azure.microsoft.com/en-us/updates/autoscale-for-azure-hdinsight-is-now-general-available/.
[40] Microsoft. 2019. Identify the right Azure SQL Database or SQL Managed Instance SKU for your on-premises database. https://docs.microsoft.com/en-us/sql/dma/dma-sku-recommend-sql-db?view=sql-server-2017.
[41] Microsoft. 2019. Intelligent query processing in SQL databases. https://docs.microsoft.com/en-us/sql/relational-databases/performance/intelligent-query-processing?view=sql-server-ver15.
[42] Microsoft. 2021. Azure Synapse Analytics. https://azure.microsoft.com/en-us/services/synapse-analytics/.
[43] Microsoft. 2021. Visual Studio IntelliCode. https://visualstudio.microsoft.com/services/intellicode/. (Accessed on 07/20/2021).
[44] Microsoft and Facebook. 2018. Open Neural Network Exchange (ONNX). https://onnx.ai.
[45] Supun Nakandala and et al. 2020. A Tensor Compiler for Unified Machine Learning Prediction Serving. In *OSDI*.
[46] Parimarjan Negi and et al. 2021. Flow-Loss: Learning Cardinality Estimates That Matter. *arXiv:2101.04964* (2021).
[47] Parimarjan Negi and et al. 2021. Steering Query Optimizers: A Practical Take on Big Data Workloads. In *SIGMOD*.
[48] Harsha Nori and et al. 2019. InterpretML: A Unified Framework for Machine Learning Interpretability. *CoRR* abs/1909.09223 (2019).
[49] Dejan Novaković, Nedeljko Vasić, Stanko Novaković, Dejan Kostić, and Ricardo Bianchini. 2013. DeepDive: Transparently Identifying and Managing Performance Interference in Virtualized Environments. In *2013 USENIX Annual Technical Conference (USENIX ATC 13)*. USENIX Association, San Jose, CA, 219–230. https://www.usenix.org/conference/atc13/technical-sessions/presentation/novakovi{ć}
[50] Oracle. [n.d.]. Oracle Auto-Tune. https://docs.oracle.com/cd/B19306_01/server.102/b14211/sql_tune.htm#i36217. Accessed on 07/20/2021.
[51] F. Pedregosa and et al. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.
[52] Anish Pimpley and et al. 2021. Optimal Resource Allocation for Serverless Queries. *(Under Submission)* (2021).
[53] Olga Poppe and et al. 2020. Seagull: An Infrastructure for Load Prediction and Optimized Resource Allocation. *PVLDB* 14, 2 (2020).
[54] Abhishek Roy, Alekh Jindal, Hiren Patel, Ashit Gosalia, Subru Krishnan, and Carlo Curino. 2019. SparkCruise: Handsfree Computation Reuse in Spark. *Proc. VLDB Endow.* 12, 12 (2019), 1850–1853. https://doi.org/10.14778/3352063.3352082
[55] Ted Dunning Sebastian Schelter, Stefan Grafberger. 2021. HedgeCut: Maintaining Randomized Trees for Low-Latency Machine Unlearning. In *SIGMOD*.
[56] Rathijit Sen and et al. 2020. AutoToken: Predicting peak parallelism for big data analytics at Microsoft. *PVLDB* 13, 12 (2020), 3326–3339.
[57] Liqun Shao, Yiwen Zhu, Siqi Liu, Abhiram Eswaran, Kristin Lieber, Janhavi Mahajan, Minsoo Thigpen, Sudhir Darbha, Subru Krishnan, Soundar Srinivasan, et al. 2019. Griffon: Reasoning about Job Anomalies with Unlabeled Data in Cloud-based Platforms. In *SOCC*.
[58] Supreeth Shastri and et al. 2020. Understanding and Benchmarking the Impact of GDPR on Database Systems. *PVLDB* 13, 7 (March 2020), 1064–1077.
[59] Tarique Siddiqui and et al. 2020. Cost models for big data query processing: Learning, retrofitting, and our findings. In *SIGMOD*. 99–113.
[60] Manasi Vartak, Harihar Subramanyam, Wei-En Lee, Srinidhi Viswanathan, Saadiyah Husnoo, Samuel Madden, and Matei Zaharia. 2016. ModelDB: A System for Machine Learning Model Management *(HILDA)*.
[61] L. Viswanathan, A. Jindal, and K. Karanasos. 2018. Query and Resource Optimization: Bridging the Gap. In *ICDE*. 1384–1387.
[62] Ke Wang and et al. 2020. Spur: Mitigating Slow Instances in Large-Scale Streaming Pipelines. In *SIGMOD*. 2271–2285.
[63] Xiaoying Wang, Changbo Qu, Weiyuan Wu, Jiannan Wang, and Qingqing Zhou. 2020. Are We Ready For Learned Cardinality Estimation? arXiv:2012.06743
[64] Chenggang Wu, Alekh Jindal, Saeed Amizadeh, Hiren Patel, Wangchao Le, Shi Qiao, and Sriram Rao. 2018. Towards a Learning Optimizer for Shared Clouds. *PVLDB* 12, 3 (2018), 210–222.
[65] M. Zaharia and et al. 2018. Accelerating the Machine Learning Lifecycle with MLflow. *DEB* 41 (2018), 39–45.
[66] Yiwen Zhu, Krishnadhan Das, Matteo Interlandi, Abhishek Roy, and et al. 2021. Phoebe: A Learning-based Checkpoint Optimizer. *(Under Submission)* (2021).
[67] Yiwen Zhu, Subru Krishnan, Konstantinos Karanasos, Isha Tarte, and et al. 2021. KEA: Tuning an Exabyte-Scale Data Infrastructure. In *SIGMOD*.