Towards Building Autonomous Data Services on Azure

Yiwen Zhu
Microsoft, USA
yiwzh@microsoft.com

Yuanyuan Tian
Microsoft, USA
yuanyuantian@microsoft.com

Joyce Cahoon
Microsoft, USA
jcahoon@microsoft.com

Subru Krishnan
Microsoft, USA
subru@microsoft.com

Ankita Agarwal
Microsoft, USA
ankiagar@microsoft.com

Rana Alotaibi
Microsoft, USA
ranaalotaibi@microsoft.com

Jesús Camacho-Rodríguez
Microsoft, USA
jesusca@microsoft.com

Bibin Chundatt
Microsoft, India
bibin.chundatt@microsoft.com

Andrew Chung
Microsoft, China
andchung@microsoft.com

Niharika Dutta
Microsoft, USA
nidutta@microsoft.com

Andrew Fogarty
Microsoft, USA
anfog@microsoft.com

Anja Gruenheid
Microsoft, USA
agruenheid@microsoft.com

Brandon Haynes
Microsoft, USA
brhaynes@microsoft.com

Matteo Interlandi
Microsoft, USA
mainterl@microsoft.com

Minu Iyer
Microsoft, USA
minu.iyer@microsoft.com

Nick Jurgens
Microsoft, USA
nicholas.jurgens@microsoft.com

Sumeet Khushalani
Microsoft, USA
sukhusha@microsoft.com

Brian Kroth
Microsoft, USA
bpkroth@microsoft.com

Manoj Kumar
Microsoft, India
manok@microsoft.com

Jyoti Leeka
Microsoft, USA
Jyoti.Leeka@microsoft.com

Sergiy Matusevych
Microsoft, USA
sergiym@microsoft.com

Minni Mittal
Microsoft, India
minni.mittal@microsoft.com

Andreas Mueller
Microsoft, USA
amueller@microsoft.com

Kartheek Muthyala
Microsoft, USA
kamuth@microsoft.com

Harsha Nagulapalli
Microsoft, USA
hanagula@microsoft.com

Yoonjae Park
Microsoft, USA
yoonjae.park@microsoft.com

Hiren Patel
Microsoft, USA
hirenp@microsoft.com

Anna Pavlenko
Microsoft, USA
anna.pavlenko@microsoft.com

Olga Poppe
Microsoft, USA
olpoppe@microsoft.com

Santhosh Ravindran
Microsoft, USA
santhosh.ravindran@microsoft.com

Karla Saur
Microsoft, USA
kasaur@microsoft.com

Rathijit Sen
Microsoft, USA
rathijit.sen@microsoft.com

Steve Suh
Microsoft, USA
stsuh@microsoft.com

Arijit Tarafdar
Microsoft, USA
arijitt@microsoft.com

Kunal Waghray
Microsoft, USA
kunalwaghray@microsoft.com

Demin Wang
Microsoft, USA
deminw@microsoft.com

Carlo Curino
Microsoft, USA
ccurino@microsoft.com

Raghu Ramakrishnan
Microsoft, USA
raghu@microsoft.com

ABSTRACT

Modern cloud has turned data services into easily accessible commodities. With just a few clicks, users are now able to access a catalog of data processing systems for a wide range of tasks. However, the cloud brings in both complexity and opportunity. While cloud users can quickly start an application by using various data services, it can be difficult to configure and optimize these services to gain the most value from them. For cloud providers, managing

SIGMOD-Companion '23, June 18–23, 2023, Seattle, WA, USA

ACM ISBN 978-1-4503-9507-6/23/06…$15.00
https://doi.org/10.1145/3555041.3589674

every aspect of an ever-increasing set of data services, while meeting customer SLAs and minimizing operational cost is becoming more challenging. Cloud technology enables the collection of significant amounts of workload traces and system telemetry. With the progress in data science (DS) and machine learning (ML), it is feasible and desirable to utilize a data-driven, ML-based approach to automate various aspects of data services, resulting in the creation of *autonomous data services*. This paper presents our perspectives and insights on creating autonomous data services on Azure. It also covers the future endeavors we plan to undertake and unresolved issues that still need attention.

CCS CONCEPTS

• **Information systems** → **Autonomous database administration**; • **Computer systems organization** → *Cloud computing*.

KEYWORDS

cloud, data service, autonomous data service, self-driving data service, artificial intelligence (AI), machine learning (ML), data science (DS), ML for system, cloud infrastructure, query engine

ACM Reference Format:

Yiwen Zhu, Yuanyuan Tian, Joyce Cahoon, Subru Krishnan, Ankita Agarwal, Rana Alotaibi, Jesús Camacho-Rodríguez, Bibin Chundatt, Andrew Chung, Niharika Dutta, Andrew Fogarty, Anja Gruenheid, Brandon Haynes, Matteo Interlandi, Minu Iyer, Nick Jurgens, Sumeet Khushalani, Brian Kroth, Manoj Kumar, Jyoti Leeka, Sergiy Matusevych, Minni Mittal, Andreas Mueller, Kartheek Muthyala, Harsha Nagulapalli, Yoonjae Park, Hiren Patel, Anna Pavlenko, Olga Poppe, Santhosh Ravindran, Karla Saur, Rathijit Sen, Steve Suh, Arijit Tarafdar, Kunal Waghray, Demin Wang, Carlo Curino, and Raghu Ramakrishnan. 2023. Towards Building Autonomous Data Services on Azure. In *Companion of the 2023 International Conference on Management of Data (SIGMOD-Companion '23), June 18–23, 2023, Seattle, WA, USA.* ACM, Seattle, WA, USA, 8 pages. https://doi.org/10.1145/3555041.3589674

1 INTRODUCTION

Modern cloud has made access to various data processing systems easier than ever. Today, Azure—Microsoft's public cloud offering—provides a large range of data services to customers, including SQL databases (e.g. SQL Server, PostgreSQL, and MySQL), NoSQL databases (e.g. CosmosDB), analytics (e.g. Synapse SQL and Synapse Spark), big data (e.g. Apache Kafka and Apache Storm), and BI (e.g. PowerBI Analysis Service). While cloud providers benefit from the economy of scale, migration to cloud also brings with it complexity. As the number and the complexity of data services continue to grow, cloud providers are facing increasing difficulties in managing all aspects of a service, such as resource provisioning, scheduling, query optimization, query execution and service tuning, while still satisfying customer SLAs and reducing operational expenses. For cloud users, on the other hand, it is non-trivial to extract the maximum benefit from these data services, with each service exposing many configurations and performance knobs to tune. The recent trend of *serverless computing* seeks to relieve users from the burden of choice. However, this product line simply transfers the problem from cloud users back to cloud providers. Automating data services thus is an integral part of the cloud to operate at scale.

While the cloud brings with it complexity, it presents massive opportunities. We have never before had access to such detailed workload traces and system telemetries, collected across millions of users and applications. More instrumentation is continuously added to the cloud for better tracing and monitoring. The combination of the recent advances in data science (DS) and machine learning (ML), sophisticated telemetry, and shortage of data experts make now an ideal time for the development and adoption of *autonomous data services*. Prior research on self-adaptive [17], self-tuning [7], and self-managing [37] databases has been ongoing for decades, but it is only with the advent of cloud technology that the practicality of autonomous data services has emerged. Oracle announced the "World's First Self-Driving Database" in 2017 [12] suggesting that ML will replace DBA, followed by many efforts on autonomous databases from industry and academia [38, 50]. Furthermore, there is a wealth of research focused on utilizing ML to improve or substitute various components of database engines, such as the cardinality estimator, cost model, query planner, and indexer [23–25]. We are witnessing an explosion of DS/ML-for-Systems innovations applied in the area of autonomous databases.

The vision described in the paper is the distillation of multiple research efforts led by applied researchers and data scientists from the Gray Systems Lab (GSL) [32], in close collaboration with engineers from various departments within Azure Data. The set of research initiatives seek to enhance and automate different facets of Azure Data services, which have yielded significant COGS (cost of goods sold) saving for Azure. In this paper, we present our perspectives on the development of autonomous cloud data services, including the challenges involved, the progress we have made, the lessons we have learned, the future directions we are pursuing, and the outstanding questions that require further investigation.

2 OUR VIEWPOINTS

We first present our viewpoints on building autonomous data services in the cloud and explain the rationals behind them.

Viewpoint 1: *The economic scale that has driven the adoption of cloud technology has also necessitated the development of autonomous data services. However, we contend that true autonomous data services can only be achieved in the cloud, meaning that the cloud is a necessary precondition for the attainment of autonomy in data services.*

Gaining knowledge from past experiences, which may span multiple users and applications, is a critical step towards achieving autonomy for data services. The cloud platform provides extensive visibility into a vast array of system metrics and workloads from numerous users and applications over time. Due to its expansive range of services and customer base, the cloud amortize the cost of advanced quality-of-service (QoS) features, making it more financially viable to invest in ML-based solutions. Although a 1% improvement in on-premise systems for an individual customer may seem insignificant, when applied across millions of cloud users, it can have a substantial impact. Additionally, as customer workloads continue to evolve, the learning process must adapt accordingly, which the cloud facilitates through the rapid deployment of updates, often without requiring end-user involvement.

Viewpoint 2: *Autonomy spans all layers of data services: cloud infrastructure layer, query engine layer, and service layer.*

Developing an autonomous data service on the cloud requires leveraging ML to improve or replace more than just the work of DBAs and individual engine components. To illustrate, let us consider the life cycle of a query on a serverless big data service. These services offer a range of adjustable knobs that can impact system performance. For example, Spark requires the user to specify the number of executors and resources (cores and memory) allocated to them. Since this is a serverless service, all such decisions must be made automatically at the *service layer*. Prior to running the query, the service must be operational. If this is the first time the customer has used the service, then VM or container resources must be provisioned at the *cloud infrastructure layer*. This raises a number of questions, such as which VM or container SKU to select, what the software configuration should be, and whether proactive resource provisioning is necessary to meet SLA for the customer. If so, what SKUs should be proactively provisioned? Once the service is running, the query must be optimized using an accurate cardinality, the correct cost model, and a reliable query planner. The query is then executed efficiently, potentially utilizing indexes and materialized views recommended based on the workloads. It is evident that all layers of the cloud stack, including the cloud infrastructure layer, query engine layer, and service layer, as well as the interactions among them, must be taken into consideration when creating autonomous data services. This level of complexity can be daunting for many institutions.

Viewpoint 3: *The objectives of autonomous data services are: improving ease of use, optimizing performance, reducing costs, and maintaining data privacy.*

Autonomy is not the ultimate goal of cloud services, but rather a means to achieving simpler, faster, and more cost-effective services for users, while also prioritizing data privacy. Simplicity or ease of use is an important aspect, whereby users should not have to worry about resource allocation, query optimization, or excessive configuration and tuning decisions to use a data service. Achieving optimized performance is a shared goal among cloud users and providers. To meet this goal, the cloud infrastructure layer needs to offer fast and intelligent resource provisioning and scheduling, queries must be optimized and executed efficiently, and services should be appropriately tuned. Cost savings benefit both users and providers. Interestingly, as we will show later, many times performance and cost saving can be achieved simultaneously, but sometimes they are at odds with each other and requires a trade-off between these two goals. Finally, preserving privacy must be a fundamental requirement when pursuing the other three objectives.

3 THE VANTAGE POINT

As an applied research organization under Azure Data, GSL is situated at the intersection of research and product development. Our viewpoints on building autonomous cloud data services are therefore shaped by the rich research and our first-hand experience working with various product groups in Azure Data.

Research in Autonomous Data Services. Our perspectives and progress in creating autonomous data services are built upon a strong foundation of research from the academic community. In the interest of space, we focus on the major trends and provide examples of influential works, although a comprehensive survey of research in this area is beyond the scope of this discussion. As pointed out by [39], the work on autonomous data systems dates back more than four decades, with the development of self-adaptive databases [17]. In the early 2000s, projects such as Microsoft's AutoAdmin [4] and IBM's DB2 design advisor [54] marked the era of self-tuning databases [7]. Oracle subsequently introduced the self-managing database [37]. While these earlier works aimed to alleviate various administrative tasks for DBAs, such as memory allocation, index recommendations, and materialized views, they did not utilize DS&ML techniques. However, the cloud and the advancements in DS&ML technologies have accelerated progress towards autonomous data services. In 2017, Oracle announced the "World's First Self-Driving Database" [12]. In academia, several efforts have focused on autonomous or self-driving databases [38, 50], which aim to automatically tune database configurations or optimize databases for predicted future workloads. In a related line of research, many studies have applied ML to improve database engine components, often referred to as learned components. These include learned indexes [24], learned cardinality estimation [23], learned query optimizer [25], and learned checkpoint [52]. All of these efforts occur either inside the database engine or on top of the engine in the service layer. The efforts in optimized infrastructure support [10, 11] for cloud data services are fewer in comparison.

First-Hand Product Experience. Over the years, we have worked on a large number of ML-for-Systems projects, and successfully delivered many new or improved features in making the corresponding Azure data services more autonomous. Horizontally, we have worked on Cosmos [42] (an internal cloud data service in Microsoft), Azure SQL Database [30], Azure Synapse SQL [31], Synapse Spark [26], HDInsight [28], etc. Vertically, our work has touched all three layers of data services. We have proposed novel techniques as well as adapted existing state-of-the-art research ideas to address practical concerns such as explainability, debuggability, and cost management. In Section 4, we will showcase some of these projects.

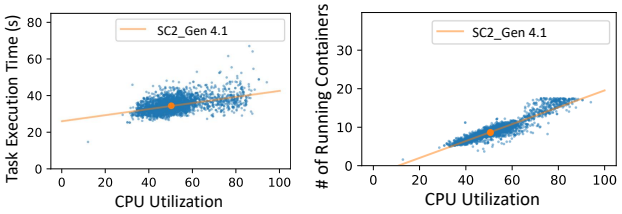
4 CHALLENGES AND PROGRESS

In this section, we discuss the challenges in automating each layer of the cloud stack and report on the progress we have achieved.

4.1 Cloud Infrastructure Layer

The cloud infrastructure manages all hardware and software resources for the life cycle of data services. Significant technical and research efforts have been made to enhance it, including resource provisioning [16], job scheduling [3, 15], container imaging [5], and autoscaling [13, 48]. However, these components heavily depend on the manual adjustments by experts in the field, with fixed parameters dispersed throughout the code base in configuration files. With the emergence of advanced analytical tools and abundant telemetry data, new opportunities for automation arises. Our solutions in this layer were built based on our findings on the predictability of system behaviors and user behaviors.

Modeling system behaviors based on domain knowledge and system metrics. Training models for autonomous data services requires a substantial amount of data from various system tunables. While existing observational data may suffice in scenarios



(a) CPU vs Task Exec Time (b) CPU vs Runinng Containers

Figure 1: Models to predict machine behavior [53]

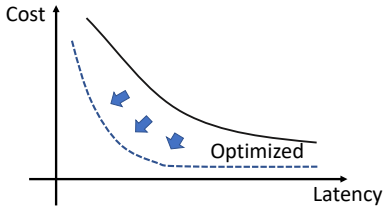


Figure 2: Pareto curve depicting the trade-offs between the QoS (x-axis) and the cost (y-axis)

with inherent volatility [53], additional data is often necessary. However, gathering such data through rounds of trials on the production infrastructure is impractical due to potential service disruptions. As a result, we must either devise ways to minimize the number of experiment runs or gather system metrics and use ML to "emulate" system dynamics. In both cases, domain knowledge is crucial to comprehend the causal links among different components and establish trustworthy models of the complex system.

As an example, in [53], we employed multiple linear models to predict machine behavior, such as CPU utilization versus task execution time or the number of running containers (see Figure 1). These models were then integrated into an optimizer to balance workloads by tuning Cosmos scheduler configurations, such as the maximum running containers for each SKU. Similar methods were used to determine the hardware/software configuration, such as RAM/SSD size and the mapping of logical drives to physical media, and to set power limits on Cosmos racks. For Azure Synapse Spark [26], we developed a simulator to mimic the cluster initialization process and derived the optimal policy for sending requests, reducing its tail latency. As another example, by using ML to predict the throughput and latency of benchmark workloads on VMs with various kernel parameters, developed on MLOS [9], we refined the parameters of the Azure VM that runs Redis workloads.

Modeling user behaviors for better trade-offs between quality of service (QoS) and cost. Cloud operators face a continuous challenge in managing resources, striking a balance between QoS, such as low latency, and operational costs. To fulfill customer SLAs, cloud operators often need to proactively provision resources, which can lead to additional expenses. This interdependence is illustrated by the Pareto curve in Figure 2. By utilizing ML, these trade-offs can be measured, and the Pareto curve can be globally optimized. In [41], we demonstrated that 77% of Azure SQL Database Serverless usage is predictable and used ML forecasts to pause/resume databases proactively. Another instance is proactive cluster provisioning based on expected user cluster creation demand to reduce wait time for cluster initialization on Azure Synapse Spark, optimizing both COGS and performance.

4.2 Query Engine Layer

Despite the significant amount of research conducted on utilizing ML techniques to enhance or replace parts of the query engine [23–25], there is still reluctance within the industry to apply these advanced methods to actual production systems. This reluctance can be attributed to several factors. Firstly, real production systems are often more intricate than the academic prototypes used in research papers. For instance, the start-of-the-art learned optimizer, Bao [25], which provides rule hints to steer the optimizer towards better plans, only takes into account 48 rule configurations, whereas the SCOPE query engine [42] used in Cosmos has 256 rules in the query optimizer, which leads to 2^{256} rule configurations that need to be considered. Secondly, while sophisticated ML algorithms have demonstrated superior performance over current engine components, production engineers prioritize the interpretability and debuggability of the models, as every new feature introduced may generate new incident tickets for on-call engineers to resolve. Thirdly, workload patterns change over time due to data or concept drift, and regression is a genuine concern. Finally, the cost of training, especially for deep neural networks, becomes as another obstacle to their adoption.

This subsection outlines our efforts to automate various aspects of query engines in production environments and address the aforementioned challenges. The fundamental principle underlying our work is to **learn from the past to improve the future**. Our work is based on the observation that in actual production workloads, **queries and jobs are often recurrent and similar**. For instance, in SCOPE, over 60% of jobs are recurring (involving periodic runs of scripts with the same operations but different predicate values [51]), and nearly 40% of daily jobs share common subexpressions with at least one other job [22]. This highlights the potential benefits of insights learned from the past workloads to improve the efficiency of future workloads.

Workload Analysis. To automate query engines, we start from workload analysis [20]. There are several pieces of information that are crucial for learning: meta data, query logs, and run time statistics (such as execution time and actual cardinality). However, these data sources are frequently dispersed in different locations. Consequently, our first step is to combine this information. To facilitate various applications of the workload data, queries or subexpressions of queries are categorized into templates based on their recurrence and similarity, and the dependencies of queries/jobs (where the output of one job serves as the input of another) in pipelines are captured [20]. Furthermore, workloads evolve over time, and as such, we also learn the evolving nature of the historical workloads to forecast future workloads.

Query Optimization. The optimizer serves as the brain of a query engine, and decades of research and development have been invested in improving this component for any data engine on Azure. Our guiding principle is to **minimize changes to the existing**

optimizer and supplement it with learned components. Specifically, we *externalize* the learned components and add simple extensions to the optimizer to leverage these external services. For cardinality estimation, we utilize the templates generated by workload analysis and train per-template micromodels [49]. We reduce the number of micromodels by retaining only those that would actually improve performance. Consequently, the optimizer can employ more precise cardinalities for queries or subexpressions with corresponding models while reverting to the default cardinalities for others. We adopt the same micromodel approach for learned cost models [46] and introduce a meta ensemble model that corrects and combines predictions from individual models to increase coverage. To enhance optimizer plans using rule hints, we have made notable progress in applying state-of-the-art research ideas from Bao [25] to production settings. However, we had to make significant adjustments for the production system, including limiting steering to small incremental steps for better interpretability and debuggability, minimizing pre-production experimentation costs using a contextual bandit model, and guarding against regression with a validation model [35, 51].

Query Execution. In query engines of big data services like Cosmos and Spark, a job is compiled into a Direct Acyclic Graph (DAG) of stages that are executed in parallel. In the case of Cosmos, we have observed an increase in the job complexity over the years, with some jobs containing thousands of stages [52]. During runtime, these large jobs can lead to machine hotspots that run out of local temporary storage space, longer restarting times in case of failures, and suboptimal performance due to compounding errors from poor optimizer estimates. In [52], we trained models to estimate the execution time, output size, and start/end time of each stage taking into account of the inter-stage dependency, then applied a linear programming algorithm to introduce checkpoint "cut(s)" of the query DAG. With this checkpoint optimizer, we were able to free the temporary storage on hotspots by more than 70% and restart failed jobs 68% faster on average with minimal impact on Cosmos performance.

Computation Reuse. With the large portion of recurrent and overlapping queries observed in real production workloads, there is a great opportunity to reuse past computations for future queries. CloudViews [21, 43] was developed to detect and reuse common computations on Cosmos and Spark. It relies on a lightweight subexpression hash, called a signature, for scalable materialized view selection and efficient view matching. Deployed on Cosmos, we have observed 34% improvement on the accumulative job latency, and 37% reduced total processing time [21]. We have worked on improvements of CloudViews on several fronts, including extending the reuse from the *syntactically* equivalent subexpressions detected by the signatures to *semantically* equivalent and contained subexpressions while still maintaining the efficiency and scalability of the detection process, as well as enabling a query to *partially* take advantage of a view with the remaining results computed on the base tables.

Pipeline Optimization. Production workloads not only have many recurrent queries, but also many recurrent query pipelines, where queries are interconnected by their outputs and inputs. For example, 70% of daily SCOPE jobs have inter-job dependencies. We analyzed the interdependency to facilitate job scheduling [8] and developed a pipeline optimizer to optimize these recurrent pipelines [14], including collecting pipeline-aware statistics and pushing common subexpressions across consumer jobs to their producer job.

4.3 Service Layer

DS&ML solutions impact how customers engage with a system at the service level. The primary goal of the autonomous cloud services is to automate as many customer-facing decisions and options as possible while also providing highly customizable solutions. DS&ML tools allow for the automation of various decisions by studying customer and application profiles. We can develop models with different levels of granularity: 1) a global model that is broad but may not be precise, 2) a segment model that groups similar customers or applications and shares insights within the group, and 3) an individual model for each customer or application that requires sufficient data observations.

Individual models are more accurate when there is enough data. To automate the scheduling of backups for PostgreSQL and MySQL servers, we used ML models to forecast user load for each specific server [40]. The system identifies low load windows with 99% accuracy, and the solution has been deployed for tens of thousands of PostgreSQL and MySQL servers across all Azure regions.

Segment models or global models are deployed jointly to transfer learning across customers/applications. To automate the SKU suggestion for migrating from on-premise SQL Server to the cloud, we proposed a profiling model that compares new customers to existing segments of Azure customers. This enables new customers to benefit from the decisions made by customers with similar characteristics. We achieved a recommendation accuracy of over 95% by combining the segment-wise knowledge with a per-customer price-performance curve that offers a customized rank of all SKU options [6]. Another example involves auto-tuning configurations for Spark, built on top of the resource usage predictor [45]. We use iterative tuning algorithms to replace the manual process for customers. We start with a global model trained using data from multiple benchmark queries. While the global model may not be highly accurate, it serves as a reasonable starting point and is fine-tuned for each application as more observational data becomes available.

5 LESSONS LEARNED

Given our experience developing and integrating DS&ML solutions at the cloud infrastructure, query engine and data service layers for various cloud services across Microsoft (e.g., SCOPE [42], Synapse DW [31] as well as Spark [26, 28]), common patterns arose across our engagements. Here we list some key lessons that we believe underlie our production successes and the speed at which we generate value for our product partners.

Insight 1: *Simplicity rules.*

The common pattern across all our engagements is that simple heuristics tend to overrule ML and simple ML models, like linear models and tree-based models, tend to overrule complex deep learning models. This is particularly true for new engagements, with teams that have yet to adopt ML within production. For example,

in [40], for PostgreSQL or MySQL servers that follow a stable daily or a weekly pattern, a simple heuristic that predicts the load of a server based on that of the previous day was already sufficient to generate 96% accuracy. There are many projects in which a linear regression was most appropriate [45, 53]. Simplicity helps with:

Cost. While there is growing consensus on the positive impact of ML in automating and optimizing cloud services, production deployments have to evaluate trade-offs with the increase in COGS to enable it. Consequently, algorithms are selected, not only by performance, but also by other factors, like training and inference cost, dependency on specialized hardware (GPU, FPGAs), etc..

Scalability. For production systems, we need the metrics to seamlessly scale in training time, re-train frequency and data/parameter handling. Most sophisticated machine learning techniques do not satisfy this fundamental requirement, e.g., reinforcement learning requires substantial training data before outperforming traditional approaches. Moreover, when inference is on the critical path (which impacts the customer-experienced latency), latency becomes crucial for the design of the infrastructure which prunes the solutions space considerably.

Manageability. Manageability is important in two dimensions — debuggability and upgrades/rollbacks. ML models are sometimes notorious for their difficulty in debugging. In a production environment, when encountering regression, a complex data lineage *across a multitude of systems and language* is needed for a close investigation from data ingestion to model (deployed) inference[34]. Debuggability needs to be well-supported with tracking/versioning through MLOps [2] for continuous integration.

Explainability. For customer-facing solutions, the expectation is that the reasoning of a choice made under the hood by any algorithm has a succinct and ideally intuitive rationale. In this sense, an explainable solution, which in turn translates to simplicity such as [6], is very much preferable, while also improving the manageability as mentioned before.

Insight 2: *One size does not fit all.*

One global (macro) model that functions reasonably well for all scenarios can typically be traded off against several specific (micro) models that are tailored for individual customers, as discussed in Section 4.3. Identifying and crafting a single global model is generally difficult, as data heterogeneity necessitates considerable feature construction and model hyperparameter tuning for optimal performance. Micro models, however, go against simplicity due to the challenges in managing the large number of models. A happy middle ground can be achieved by identifying natural ways to stratify the data, and building micro models for each cluster as done in the SKU recommendation framework [6] that recommends right-sized Azure SQL SKU to migrate on-premise databases.

Insight 3: *Feedback loop is indispensable.*

It is universally accepted that all ML solutions undergo extensive testing before being deployed into production, including backtesting, flighting [53] or A/B testing (potentially with a smaller group). The dynamic nature of cloud data services, however, necessitates ongoing improvement of even "well-tested" solutions in order to maintain performance, which leads to requirements for (1) a thorough monitoring system to spot potential changes in real-time, continually assess, and initiate fine-tuning of the model, and (2) a rollback mechanism that reacts fast and avoids regression.

6 FUTURE DIRECTIONS

In this section, we discuss some of the future directions that we are currently pursuing while also highlighting the challenges.

Direction 1: *Reuse, reuse and reuse!*

Despite the differences between distinct data services on Azure, they all face a set of similar issues. For example, at the infrastructure level, many services need efficient cluster provisioning and auto-scaling. At the engine level, many require improvement in cardinality estimation, query planning, and computation reuse. At the service level, auto-tuning is highly sought after for many services. Working on similar issues with multiple Azure data services over time, we came to the realization that **a common reusable solution is highly desirable to efficiently leverage the similar technologies and software artifacts among multiple services**.

However, reality presents a lot of challenges to reusability. Distinct services collect different service-specific telemetries and workload traces, store them in different places (e.g., Kusto[27], SQL server, etc.), and have different preferences on the infrastructure for model deployment (e.g. AML[29], Synapse ML[33], etc.).

So, can we reach the holy grail of reusable ML solutions? Although we don't have a complete answer, we can perhaps try to tackle this problem at different granularities of reusability.

Function Level Reuse. In the finest granularity, we can reuse pieces of code modules that implement specific functions, e.g., time series analysis of OS performance counter data. Our proposal is to create a *AlgorithmStore (analogous to a GitHub for models)*, which is a project gallery with predefined algorithm templates. The previously developed algorithm can be discovered and adapted to address new scenarios quickly. For this type of algorithm catalog, it is required to have: (1) an easy search interface to discover similar pre-existing solutions; (2) good API design to support extensibility and customizations; (3) clean modularized functions; (4) significant coverage of common use cases; (5) code quality to allow robust reuse; and (6) better documentation.

Component Level Reuse. At the component level, the question of reusability pertains to whether we can establish a shared infrastructure that supports similar or related system components across various data services. For example, can we develop a common infrastructure that facilitates auto-scaling for all services or query optimization for all data engines? This task becomes increasingly difficult due to the aforementioned differences among distinct services. Nevertheless, we have made some strides on this front. The Peregrine workload optimization platform [20] represents a common infrastructure for a set of related engine problems, such as cardinality estimation, cost models, and computation reuse. It has been implemented for both Cosmos and Spark. Peregrine consists of an engine-agnostic workload representation, workload categorization based on patterns, and a workload feedback mechanism that enables query engines to respond to workload feedback.

System-for-ML Support Level Reuse. At the highest level of granularity, all ML-for-Systems projects require System-for-ML support, from data ingestion, featurization, model training and

tuning, model deployment, to model tracking. In GSL, we have a large collection of System-for-ML projects towards building such a common infrastructure. A summary and vision of our efforts in this area is provided in [2].

Direction 2: *Standardization.*

Standardization is critical for developing reusable infrastructure across data services. It begins with telemetry. In addition to structuring the collected telemetry similarly across platforms and data services through initiatives such as OpenTelemetry [36], we are also exploring the use of semantic information from telemetry to enhance reusability across platforms and services (e.g. CPU utilization metrics on Windows and Linux VMs possess the same meaning even though they may have different names). At the query engine level, we require standardization for representing workloads and query plans. We have made some initial efforts on an engine-agnostic workload representation as part of the Peregrine workload optimization platform [20]. We are now exploring the use of cross-language query plan specification, such as Substrait [47], as a standard plan representation across our engines. To simplify the reuse of models for deployment within a common infrastructure, we also adopt standard representations for ML models, such as ONNX [1]. Furthermore, we package an ML model (along with any additional required code and libraries) into a standard generic container that can be efficiently reused across systems [44], making it portable across all of our model-serving capabilities at Microsoft.

Direction 3: *Optimization across components jointly.*

In many projects, the primary focus is typically on optimizing a single component of the entire system since it is owned by a specific product team. For example, VM provisioning is owned by the cluster service team, while cardinality estimates are owned by the query optimizer team, and so on. However, sequentially optimizing each individual component is unlikely to yield optimal overall performance. Conversely, for a complex cloud service, especially at scale, it is impractical to create a massive optimization problem that simultaneously optimizes all components while accurately capturing interactions across different components. Ongoing efforts continue to jointly optimize a selection of components and synchronize the deployment of changes so that the observational data reflects the latest deployed configuration. This approach enables us to focus on optimizing related components that work together in a coordinated manner. By improving the joint optimization of these components, we can improve the overall system performance.

Direction 4: *Responsible AI (RAI)*

ML cannot be applied without risks [19], e.g., over-indexing on a particular customer or workload, and bias is an inherent problem that we continually encounter. We introduce guardrails to protect customers from expensive solutions and from performance regressions, and we regularly check that our ML-driven decisions serve all customers fairly. We have a responsibility to ensure that customers, big or small, do not get marginalized from autonomous decisions.

At Microsoft, we are operationalizing the Responsible AI (RAI) at scale to protect privacy and security, improve fairness, inclusiveness, reliability, safety, transparency, and accountability. For the ML-related projects, we perform a comprehensive RAI assessment which is for now a manual and prolonged process by domain experts. Several automation tools were developed (e.g., [18]), however, ad-hoc solutions are still required for many cases.

7 CONCLUSION AND CALL TO ACTION

We are living in fascinating and rapidly evolving times where technology is advancing at a breakneck pace. Cloud and AI are among the most transformative technologies of our era. The intersection of these two revolutionary technologies can be witnessed in the progress made towards autonomous data services on cloud. In this paper, we showcased some of our progress in automating data services on Azure. However, challenges remain to be overcome as highlighted in the previous section. We believe that the database community has a vital role to play in shaping the future of cloud data services. We welcome other researchers to join us in this exciting journey.

ACKNOWLEDGEMENTS

We thank past team members, interns, and MAIDAP collaborators for their contribution to our progress.

REFERENCES

[1] 2017. Open Neural Network Exchange (ONNX). https://onnx.ai/.
[2] Ashvin Agrawal, Rony Chatterjee, Carlo Curino, Avrilia Floratou, Neha Godwal, Matteo Interlandi, Alekh Jindal, Konstantinos Karanasos, Subru Krishnan, Brian Kroth, Jyoti Leeka, Kwanghyun Park, Hiren Patel, Olga Poppe, Fotis Psallidas, Raghu Ramakrishnan, Abhishek Roy, Karla Saur, Rathijit Sen, Markus Weimer, Travis Wright, and Yiwen Zhu. 2020. Cloudy with high chance of DBMS: a 10-year prediction for Enterprise-Grade ML. In *CIDR*.
[3] Eric Boutin, Jaliya Ekanayake, Wei Lin, Bing Shi, Jingren Zhou, Zhengping Qian, Ming Wu, and Lidong Zhou. 2014. Apollo: Scalable and Coordinated Scheduling for {Cloud-Scale} Computing. In *OSDI*. 285–300.
[4] Nicolas Bruno, Surajit Chaudhuri, Arnd Christian König, Vivek R. Narasayya, Ravishankar Ramamurthy, and Manoj Syamala. 2011. AutoAdmin Project at Microsoft Research: Lessons Learned. *IEEE Data Eng. Bull.* 34, 4 (2011), 12–19.
[5] cachelot.io. [n.d.]. Memcached Performance Tuning. https://cachelot.io/blog/2015/04/20/Speed-up-your-application-by-fine-tuning-Memcached.html.
[6] Joyce Cahoon, Wenjing Wang, Yiwen Zhu, Katherine Lin, Sean Liu, Raymond Truong, Neetu Singh, Chengcheng Wan, Alexandra M Ciortea, Sreraman Narasimhan, and Subru Krishnan. 2022. Doppler: Automated SKU Recommendation in Migrating SQL Workloads to the Cloud. *PVLDB* 15, 12 (2022).
[7] Surajit Chaudhuri and Vivek Narasayya. 2007. Self-Tuning Database Systems: A Decade of Progress. In *VLDB '07*. 3–14.
[8] Andrew Chung, Subru Krishnan, Konstantinos Karanasos, Carlo Curino, and Gregory R. Ganger. 2020. Unearthing inter-job dependencies for better cluster scheduling. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 1205–1223. https://www.usenix.org/conference/osdi20/presentation/chung
[9] 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 *DEEM*. 1–5.
[10] Sudipto Das, Divyakant Agrawal, and Amr El Abbadi. 2013. ElasTraS: An Elastic, Scalable, and Self-Managing Transactional Database for the Cloud. *TODS* 38, 1, Article 5 (apr 2013), 45 pages.
[11] Sudipto Das, Feng Li, Vivek R. Narasayya, and Arnd Christian König. 2016. Automated Demand-Driven Resource Scaling in Relational Database-as-a-Service. In *SIGMOD*. 1923–1934.
[12] Edgar Haren. 2017. Oracle Revolutionizes Cloud with the World's First Self-Driving Database. https://blogs.oracle.com/database/post/oracle-revolutionizes-cloud-with-the-worlds-first-self-driving-database.
[13] Avrilia Floratou, Ashvin Agrawal, Bill Graham, Sriram Rao, and Karthik Ramasamy. 2017. Dhalion: self-regulating stream processing in heron. *PVLDB* 10, 12 (2017), 1825–1836.
[14] Sunny Gakhar, Joyce Cahoon, Wangchao Le, Xiangnan Li, Kaushik Ravichandran, Hiren Patel, Marc Friedman, Brandon Haynes, Shi Qiao, Alekh Jindal, and Jyoti Leeka. 2022. Pipemizer: An Optimizer for Analytics Data Pipelines. *PVLDB* (2022).

[15] Robert Grandl, Srikanth Kandula, Sriram Rao, Aditya Akella, and Janardhan Kulkarni. 2016. {GRAPHENE}: Packing and {Dependency-Aware} Scheduling for {Data-Parallel} Clusters. In *OSDI*. 81–97.

[16] Ori Hadary, Luke Marshall, Ishai Menache, Abhisek Pan, Esaias E Greeff, David Dion, Star Dorminey, Shailesh Joshi, Yang Chen, Mark Russinovich, et al. 2020. Protean:{VM} allocation service at scale. In *OSDI*. 845–861.

[17] Michael Hammer and Arvola Chan. 1976. Index Selection in a Self-Adaptive Data Base Management System. In *SIGMOD*. 1–8.

[18] Kenneth Holstein, Jennifer Wortman Vaughan, Hal Daumé III, Miro Dudik, and Hanna Wallach. 2019. Improving fairness in machine learning systems: What do industry practitioners need?. In *CHI*. 1–16.

[19] Alekh Jindal and Jyoti Leeka. 2022. Query Optimizer as a Service: An Idea Whose Time Has Come! *SIGMOD Record* (2022).

[20] Alekh Jindal, Hiren Patel, Abhishek Roy, Shi Qiao, Zhicheng Yin, Rathijit Sen, and Subru Krishnan. 2019. Peregrine: Workload Optimization for Cloud Query Engines. In *SoCC*. 416–427.

[21] Alekh Jindal, Shi Qiao, Hiren Patel, Abhishek Roy, Jyoti Leeka, and Brandon Haynes. 2021. Production Experiences from Computation Reuse at Microsoft.. In *EDBT*. 623–634.

[22] Alekh Jindal, Shi Qiao, Hiren Patel, Zhicheng Yin, Jieming Di, Malay Bag, Marc Friedman, Yifung Lin, Konstantinos Karanasos, and Sriram Rao. 2018. Computation Reuse in Analytics Job Service at Microsoft. In *SIGMOD*. 191–203.

[23] Kyoungmin Kim, Jisung Jung, In Seo, Wook-Shin Han, Kangwoo Choi, and Jaehyok Chong. 2022. Learned Cardinality Estimation: An In-Depth Study. In *SIGMOD*. 1214–1227.

[24] Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The Case for Learned Index Structures. In *SIGMOD*, Gautam Das, Christopher M. Jermaine, and Philip A. Bernstein (Eds.). 489–504.

[25] Ryan C. Marcus, Parimarjan Negi, Hongzi Mao, Chi Zhang, Mohammad Alizadeh, Tim Kraska, Olga Papaemmanouil, and Nesime Tatbul. 2019. Neo: A Learned Query Optimizer. *PVLDB* 12, 11 (2019), 1705–1718.

[26] Microsoft. [n.d.]. Apache Spark in Azure Synapse Analytics. https://docs.microsoft.com/en-us/azure/synapse-analytics/spark/apache-spark-overview.

[27] Microsoft. [n.d.]. Azure Data Explorer. https://docs.microsoft.com/en-us/azure/data-explorer/.

[28] Microsoft. [n.d.]. Azure HDInsight. https://azure.microsoft.com/en-us/services/hdinsight/.

[29] Microsoft. [n.d.]. Azure Machine Learning. https://azure.microsoft.com/en-us/services/machine-learning/.

[30] Microsoft. [n.d.]. Azure SQL Database. https://azure.microsoft.com/en-us/products/azure-sql/database.

[31] Microsoft. [n.d.]. Azure Synapse SQL architecture. https://docs.microsoft.com/en-us/azure/synapse-analytics/sql/overview-architecture.

[32] Microsoft. [n.d.]. Gray Systems Lab. https://www.microsoft.com/en-us/research/group/gray-systems-lab/.

[33] Microsoft. [n.d.]. SynapseML. https://microsoft.github.io/SynapseML/.

[34] Mohammad Hossein Namaki, Avrilia Floratou, Fotios Psallidas, Subru Krishnan, Ashvin Agrawal, Yinghui Wu, Yiwen Zhu, and Markus Weimer. 2020. Vamsa: Automated Provenance Tracking in Data Science Scripts. In *KDD*. 1542–1551. https://doi.org/10.1145/3394486.3403205

[35] Parimarjan Negi, Matteo Interlandi, Ryan Marcus, Mohammad Alizadeh, Tim Kraska, Marc Friedman, and Alekh Jindal. 2021. Steering Query Optimizers: A Practical Take on Big Data Workloads. In *SIGMOD*. 2557–2569.

[36] OpenTelemetry. [n.d.]. OpenTelemetry. https://substrait.io/.

[37] Oracle. 2006. *Oracle Database 10g Release 2: The Self-Managing Database.* Technical Report. Oracle.

[38] Andrew Pavlo, Gustavo Angulo, Joy Arulraj, Haibin Lin, Jiexi Lin, Lin Ma, Prashanth Menon, Todd C. Mowry, Matthew Perron, Ian Quah, Siddharth Santurkar, Anthony Tomasic, Skye Toor, Dana Van Aken, Ziqi Wang, Yingjun Wu, Ran Xian, and Tieying Zhang. 2017. Self-Driving Database Management Systems. In *CIDR*.

[39] Andrew Pavlo, Matthew Butrovich, Lin Ma, Prashanth Menon, Wan Shen Lim, Dana Van Aken, and William Zhang. 2021. Make Your Database System Dream of Electric Sheep: Towards Self-Driving Operation. *PVLDB* 14, 12 (2021), 3211–3221.

[40] Olga Poppe, Tayo Amuneke, Dalitso Banda, Aritra De, Ari Green, Manon Knoertzer, Ehi Nosakhare, Karthik Rajendran, Deepak Shankargouda, Meina Wang, Alan Au, Carlo Curino, Qun Guo, Alekh Jindal, Ajay Kalhan, Morgan Oslake, Sonia Parchani, Vijay Ramani, Raj Sellappan, Saikat Sen, Sheetal Shrotri, Soundararajan Srinivasan, Ping Xia, Shize Xu, Alicia Yang, and Yiwen Zhu. 2020. Seagull: An Infrastructure for Load Prediction and Optimized Resource Allocation. In *PVLDB*. VLDB Endowment, 154–162.

[41] Olga Poppe, Qun Guo, Willis Lang, Pankaj Arora, Morgan Oslake, Shize Xu, and Ajay Kalhan. 2022. Moneyball: proactive auto-scaling in Microsoft Azure SQL database serverless. *PVLDB* 15, 6 (2022), 1279–1287.

[42] Conor Power, Hiren Patel, Alekh Jindal, Jyoti Leeka, Bob Jenkins, Michael Rys, Ed Triou, Dexin Zhu, Lucky Katahanas, Chakrapani Bhat Talapady, Joshua Rowe, Fan Zhang, Rich Draves, Marc Friedman, Ivan Santa Maria Filho, and Amrish Kumar. 2021. The Cosmos Big Data Platform at Microsoft: Over a Decade of Progress and a Decade to Look Forward. *PVLDB* 14, 12 (2021).

[43] Abhishek Roy, Alekh Jindal, Priyanka Gomatam, Xiating Ouyang, Ashit Gosalia, Nishkam Ravi, Swinky Mann, and Prakhar Jain. 2021. SparkCruise: Workload Optimization in Managed Spark Clusters at Microsoft. *PVLDB* 14, 12 (2021), 3122–3134.

[44] Karla Saur, Tara Mirmira, Konstantinos Karanasos, and Jesús Camacho-Rodríguez. 2022. Containerized Execution of UDFs: An Experimental Evaluation. *PVLDB* 15, 11 (2022), 3158 – 3171. https://doi.org/doi:10.14778/3551793.3551860

[45] Rathijit Sen, Alekh Jindal, Hiren Patel, and Shi Qiao. 2020. AutoToken: Predicting peak parallelism for Big Data analytics at Microsoft. *PVLDB* 13, 12 (2020), 3326–3339.

[46] Tarique Siddiqui, Alekh Jindal, Shi Qiao, Hiren Patel, and Wangchao Le. 2020. Cost Models for Big Data Query Processing: Learning, Retrofitting, and Our Findings. In *SIGMOD*. 99–113.

[47] Substrait. [n.d.]. Substrait: Cross-Language Serialization for Relational Algebra. https://substrait.io/.

[48] Muhammad Tirmazi, Adam Barker, Nan Deng, Md E Haque, Zhijing Gene Qin, Steven Hand, Mor Harchol-Balter, and John Wilkes. 2020. Borg: the next generation. In *EuroSys*. 1–14.

[49] 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 (nov 2018), 210–222.

[50] Bohan Zhang, Dana Van Aken, Justin Wang, Tao Dai, Shuli Jiang, Jacky Lao, Siyuan Sheng, Andrew Pavlo, and Geoffrey J. Gordon. 2018. A Demonstration of the OtterTune Automatic Database Management System Tuning Service. *PVLDB* 11, 12 (2018), 1910–1913.

[51] Wangda Zhang, Matteo Interlandi, Paul Mineiro, Shi Qiao, Nasim Ghazanfari, Karlen Lie, Marc Friedman, Rafah Hosn, Hiren Patel, and Alekh Jindal. 2022. Deploying a Steered Query Optimizer in Production at Microsoft. In *SIGMOD*. 2299–2311.

[52] Yiwen Zhu, Matteo Interlandi, Abhishek Roy, Krishnadhan Das, Hiren Patel, Malay Bag, Hitesh Sharma, and Alekh Jindal. 2021. Phoebe: A Learning-Based Checkpoint Optimizer. *PVLDB* 14, 11 (jul 2021), 2505–2518. https://doi.org/10.14778/3476249.3476298

[53] Yiwen Zhu, Subru Krishnan, Konstantinos Karanasos, Isha Tarte, Conor Power, Abhishek Modi, Manoj Kumar, Deli Zhang, Kartheek Muthyala, Nick Jurgens, et al. 2021. KEA: Tuning an Exabyte-Scale Data Infrastructure. In *SIGMOD*. 2667–2680.

[54] Daniel C. Zilio, Jun Rao, Sam Lightstone, Guy Lohman, Adam Storm, Christian Garcia-Arellano, and Scott Fadden. 2004. DB2 Design Advisor: Integrated Automatic Physical Database Design. In *VLDB '04*. 1087–1097.