I research machine learning and optimization techniques for data systems, with a focus on **instance-optimization**, a new design paradigm for building data systems that can *automatically self-optimize* to achieve the best performance for any specific application or use case.

Modern enterprises are collecting, analyzing, and capitalizing on exponentially increasing troves of data. Therefore, the data systems that power the business-critical applications of these enterprises must continue to push the boundaries of **performance**, in order to keep pace with the prolific growth of data volumes, while simultaneously powering a wide variety of increasingly **diverse use cases**, ranging from maintaining real-time inventory metrics for e-commerce companies to supporting business intelligence tools that enable data-driven decision making.

With the end of Moore's Law, advances in hardware can no longer keep pace with data growth. Therefore, we must overcome performance bottlenecks through advancements in software and system design. However, it is challenging to build data systems that support diverse use cases while simultaneously achieving high performance. Widely-used general-purpose data systems achieve adequate performance across a wide variety of use cases, but do not achieve optimal performance for any specific use case. On the other hand, organizations can build custom-tailored data systems for achieving high performance on specific applications, but this can take years of engineering effort and is too labor-intensive for all but the largest organizations.

Instance-optimized data systems are an emerging class of data systems that use machine learning and optimization techniques to automatically achieve the best performance for each use case. In my research, I have leveraged instance-optimization to introduce novel designs for database **indexes** and **data storage layouts** that outperform existing state-of-the-art techniques by orders of magnitude. I also demonstrated how to incorporate multiple instance-optimized database components into an end-to-end system that outperforms a well-tuned commercial cloud-based analytics system by up to 3×. To prove that these ideas work in practice, I have spent the past year productionizing my research at AWS, culminating in a new instance-optimized feature called Multidimensional Data Layouts[1] which has already been released to customers in Amazon Redshift, a widely-used commercial database system. Looking towards the future, I strive to continue conducting collaborative research at the intersection of machine learning and data systems that balances innovation with practicality.

Instance-Optimization

Data Layouts. Workloads in analytic data systems are typically composed of queries that scan and filter large amounts of data. An appropriately-configured *data layout* (i.e., the way in which data records are ordered and clustered in physical storage) reduces the amount of I/O required for query processing by enabling the system to avoid accessing data that is not relevant to the query's filter. However, filters vary widely in content and complexity, both between different workloads and within the same workload, so data layouts must be reconfigured for every workload in order to achieve optimal performance.

Existing data layout techniques—such as sorting all data records in a table by a "sort key" column, specialized multi-column sort orders, and multi-dimensional indexes—are difficult to configure and their performance is inconsistent for different workloads. To address these shortcomings, I introduced three new *instance-optimized* data layout techniques, which build upon each other:

- Flood [1] (SIGMOD '20) is a multi-dimensional grid-based data layout, which is a generalization of existing techniques such as multi-column (compound) sort keys and Z-order (interleaved) sort keys. We use machine learning to automatically configure the grid layout to achieve the best performance for a given dataset and workload, producing up to three orders of magnitude faster query runtime than existing state-of-the-art layout techniques.

[1] `https://aws.amazon.com/blogs/big-data/improve-performance-of-workloads-containing-repetitive-scan-filters-with-multidimensional-data-layout-sort-keys-in-amazon-redshift/`

- Tsunami [2] (VLDB '21) improves upon Flood by addressing two key weaknesses which are commonly surfaced in real applications: the presence of correlated data (e.g., distance and price of a ride are correlated in taxi data) and skewed query workloads (e.g., the workload filters more frequently for recent data than stale data). Tsunami achieves high performance in these cases by introducing new tree-based data structures to augment Flood's existing grid-based layout, and unifying these layouts under a ML-based optimization algorithm based on a new formal definition of query skew. Tsunami pushed performance gains by a further order of magnitude compared to Flood.

- MTO [3] (SIGMOD '21) is a Multi-Table Optimizer that extends instance-optimized data layouts to cloud-based analytic applications, in which queries typically use diverse join patterns over multiple tables and data is stored on disk or in cloud objects stores. MTO is the first instance-optimized storage layout for jointly optimizing the storage layouts of all tables in disk-based or cloud-based multi-table datasets, using the key idea of *join-induced predicates*, which are used to pass information through joins, to jointly optimize the layout for all tables simultaneously. Through a collaboration with Microsoft Research, I implemented MTO in Azure Synapse Analytics and showed that it achieves up to 75% reduction in end-to-end query times compared to state-of-the-art data layout strategies.

Indexes. Flood, Tsunami, and MTO minimize I/O by adjusting the data layout. However, there are situations in which either it is not possible to adjust the data layout or data layouts are ineffective because the workload requires precise retrievals of individual data items without scanning large amounts of data. In these situations, data systems often rely on auxiliary data structures called indexes to improve the speed of data retrieval. One of the most fundamental database indexes is the B+ Tree, which is general-purpose and has been used in commercial databases for many decades. However, the generality of the B+ Tree means that it does not maximize performance for each use case.

The idea of *learned* indexes is to take advantage of knowledge of the specific dataset being indexed by replacing the generic B+ Tree structure with a ML model that is trained over the indexed data distribution, which speeds up retrievals over that specific dataset while also reducing the index's storage overhead. However, existing learned indexes have key limitations—such as lack of support for data updates and persistent data—that hinder more widespread adoption. My research tackles these limitations and brings learned indexes closer to reality:

- ALEX [4] (SIGMOD '20) is one of the first *updatable* learned indexes. ALEX supports data modification operations like inserts and updates while simultaneously improving read performance by introducing a dynamic tree structure with online reorganization driven by a cost model. ALEX has been open-sourced[2] and has garnered great interaction from the research community, having been cited or used as a state-of-the-art baseline in over 50 publications from top-tier database conferences (SIGMOD, VLDB, ICDE), as well as the developer community, with over 600 stars on Github.

- I collaborated with colleagues from Microsoft Research, Simon Fraser University, and CUHK on APEX [5] (VLDB '22), which builds on ALEX and is one of the first learned indexes to operate over not only in-memory data, but also persistent data, specifically on persistent memory, which is becoming a favored form of high-performance persistence in modern hardware.

Learned indexes have broad applications, even beyond data systems. I advised an undergraduate research project on applying learned indexes to genomic sequencing, a $10 billion industry in which computational efficiency is key. In collaboration with Intel Labs and the Broad Institute of MIT and Harvard, we introduced LISA [6], which uses learned indexes to improve the performance of the FM-index, which is the state-of-the-art technique widely deployed in genomics tools, by 10×.

[2] `https://github.com/microsoft/ALEX`

Systems. While instance-optimized database components such as data layouts and learned indexes are impressive in isolation, the real test for instance-optimization is how these techniques would perform in an end-to-end system. I made two contributions on this front:

- SageDB [7] (VLDB '23) is, to the best of our knowledge, the first end-to-end data system built with instance-optimization as a foundational design principle. SageDB is a case study in synthesizing the rich space of research on instance-optimized data systems, combining two carefully selected components: (1) replicated data layouts and (2) partial materialized views, which are a generalization of traditional materialized views with more degrees of freedom. Combined with a global optimization algorithm to automatically and simultaneously configure both instance-optimized components, our SageDB prototype outperforms a commercial cloud-based analytics system by up to 3× on end-to-end query workloads and up to 250× on individual queries. The SageDB codebase has been licensed by Amazon.
- In an ongoing research project, I am collaborating with colleagues from MIT and Intel on Self-Organizing Data Containers [8] (CIDR '22), which synthesizes our work on instance-optimized replicated data layouts into an *open table format* that enables high-performance data processing on data lakes.

From Ideas to Reality

While the research community has produced a rich body of work on instance-optimization over the past years, the impact of instance-optimization on real production systems has been more muted. In collaboration with Microsoft Research, I first had the opportunity to implement my data layouts research in a production system as a prototype in Azure Synapse Analytics, achieving impressive performance results on standardized benchmarks [3]. After I graduated from my PhD, I joined AWS as an applied scientist, with the goal of fully productionizing my research. Building on my experiences with Azure Synapse, I have spent the past year implementing and integrating a new instance-optimized feature in Redshift—Amazon's cloud data warehouse with tens of thousands of customers—called Multidimensional Data Layouts (MDDL) [9], which has already been deployed to preview customers and will roll out as a general-availability feature in the coming months. From the outset, I made data-driven decisions to strike a balance between design simplicity and performance improvements while eschewing flashy but ultimately ineffective embellishments. Internal telemetry indicates that across *all* customers, MDDL improves performance on 25% of tables, with over 2× improvement over existing data layouts for the top 5% of database instances.

My experience with implementing and deploying research ideas in real systems has solidified my research agenda: to produce research that is not only novel and interesting, but is also grounded in reality and has a bent towards practicality and simplicity. Systems research is at its core an applied field, and industry adoption is the real measure of an idea's staying power.

Future Directions

Instance-optimized systems is a rich and active area of research. In the short-term future, I aim to make the field of instance-optimized data systems more mature—and continue to push for their adoption in real systems—by addressing two key limitations in the existing literature:

Dynamic workloads and datasets. Instance-optimization is defined in the context of a certain use case, which means that the system must undergo a potentially expensive re-optimization process when the use case (i.e., the data and workload) changes. I plan to research re-optimization mechanisms and policies for instance-optimized systems, which must consider not only the performance benefits of a new configuration but also the cost of the re-optimization itself, while taking advantage of cost-saving opportunities such as *partial* re-optimization when different components of the existing configuration are affected by use case changes to different degrees.

I also believe that instance-optimized systems must ultimately shift from using *reactive* policies, which aim to quickly detect and respond to changes after they have already started to occur, to using *proactive* policies, which predict how the data and workload will change in the future and directly optimize for that future. For proactive techniques to be effective, we require more accurate forecasting of the future data and workload. We also need to explore multi-step planning techniques, e.g., instead of optimizing a single data layout for a single static dataset and workload, we plan a sequence of data layouts and times at which we transition from each layout to the next. Thus, my work has significant overlap with reinforcement learning and operational planning, and I would be eager to collaborate with experts in those areas.

Storage meets compute. SageDB [7] is currently an instance-optimized database system that incorporates multiple instance-optimized components from the data storage tier. I plan to pursue the outstanding challenge of integrating an instance-optimized data storage tier with an instance-optimized query processing tier, potentially in collaboration with colleagues at MIT and AWS. For example, we can incorporate the extensive work on learned query optimizers, learned cardinality estimation, and learned cost models. The main challenge will be to keep components from these two tiers synchronized: for example, if we optimize a data layout under the assumption that it is used in conjunction with a learned optimizer, but the optimizer subsequently changes due to re-optimization, then the data layout may become obsolete because the new optimizer never decides to use it.

My research vision for the long-term future has two parts:

Beyond relational databases. The ideas of instance-optimization are applicable to other types of database systems, such as NoSQL databases, graph databases, time-series databases, vector databases, or even data lakes composed of data in diverse formats. This requires a new analysis of the most impactful components to instance-optimize. For example, the way in which a graph is physically stored can be optimized for the access patterns. An auto-tuner might decide between using a sparse or dense representation, but an instance-optimized graph database might be able to materialize any design on the sparsity spectrum. In general, any performance-sensitive component that is typically designed based on static heuristics and best practices is an excellent candidate for instance-optimization.

We have the answer, but what is the question? The premise of instance-optimization is to automatically maximize performance. So far, the community has typically assumed performance to refer to query execution time, but the reality is more nuanced. Ultimately, users do not care about the performance of *queries*; users care about the performance of their *applications*. Users may care only about the performance of an application-critical subset of their workload, or they may only care that their performance hits a certain threshold but do not care about any further improvement. Furthermore, in the modern cloud era, users often care less about maximizing performance for a fixed amount of on-premise hardware and more about minimizing cloud costs for a fixed amount of performance.

My vision is to develop instance-optimized systems in which users can intuitively and precisely express their optimization goal. This will likely transcend typical low-level SLAs on query execution time and move towards higher-level SLAs for end-to-end data pipelines. As a result, we must consider how instance-optimization can be applied holistically, not only over a single data system, but an entire data mesh. This expanded scope raises a rich array of research questions spanning systems (e.g., how to enforce fair scheduling and work sharing among components and users in distributed and decentralized environments), programming languages (e.g., formulating DSLs for expressing optimization goals, leveraging program synthesis to produce more efficient data structures), and machine learning (e.g., incorporating new techniques in reinforcement learning to achieve optimization goals, making instance-optimized systems robust to adversarial agents), and I look forward to collaborating with current and future colleagues on these directions.

[1] V. Nathan*, J. Ding*, M. Alizadeh, T. Kraska. "Learning Multi-dimensional Indexes." SIGMOD 2020. (*equal contribution)
[2] J. Ding, V. Nathan, M. Alizadeh, T. Kraska. "Tsunami: A Learned Multi-dimensional Index for Correlated Data and Skewed Workloads." VLDB 2021.
[3] J. Ding, U.F. Minhas, B. Chandramouli, C. Wang, Y. Li, Y. Li, D. Kossmann, J. Gehrke, T. Kraska. "Instance-Optimized Data Layouts for Cloud Analytics Workloads." SIGMOD 2021.
[4] J. Ding, U.F. Minhas, J. Yu, C. Wang, J. Do, H. Zhang, Y. Li, B. Chandramouli, J. Gehrke, D. Kossmann, D. Lomet, T. Kraska. "ALEX: An Updatable Adapative Learned Index." SIGMOD 2020.
[5] B. Lu, J. Ding, E. Lo, U.F. Minhas, T. Wang. "APEX: A High-Performance Learned Index on Persistent Memory." VLDB 2022.
[6] D. Ho, J. Ding, S. Misra, N. Tatbul, V. Nathan, V. Md, T. Kraska. "LISA: Towards Learned DNA Sequence Search." System for ML Workshop at NeurIPS 2019.
[7] J. Ding, R. Marcus, A. Kipf, V. Nathan, A. Nrusimha, K. Vaidya, A. Renen, T. Kraska. "SageDB: An Instance-optimized Data Analytics System." VLDB 2023.
[8] S. Madden, J. Ding, T. Kraska, S. Sudhir, D. Cohen, T. Mattson, N. Tatbul. "Self-Organizing Data Containers." CIDR 2022.
[9] J. Ding, et al. "Automated Multidimensional Data Layouts in Amazon Redshift." Under submission to SIGMOD 2024 Industrial Track.