I research **machine learning for systems, especially database management systems**. I believe that modern machine learning techniques can be used to build the next generation of computer systems. These next-generation systems could automatically understand user intention, adapt to changes in workload or hardware, and autonomously invent new ways to best meet the user's requirements. While I have investigated a wide range of systems spanning many subfields, my primary focus has been databases, especially workload management, query optimization, and indexing.

Machine Learning for Database Systems

Learned Query Optimization. Query optimization is the translation of SQL (a declarative language) into execution plans (programs). Current optimizers are, arguably, a large pile of hand-crafted heuristics that must be maintained and manually tuned on a system-by-system basis. Machine learning provides an opportunity not only to improve query optimizers by automatically adapting to the user's workload and hardware, but also to make query optimizers easier to maintain. During my postdoc, I developed two systems for learned query optimization, Neo and Bao.

- Neo [1] uses deep reinforcement learning to entirely replace a traditional query optimizer, discovering entirely new query execution strategies. Neo outperforms state-of-the-art query optimizers with less than 24 hours of training on several industrial workloads. However, Neo can struggle with workloads that are highly dynamic.
- Bao [2] (SIGMOD 2021 best paper) sits on top of a traditional query optimizer, and uses a deep-learning powered multi-armed bandit approach to "steer" the underlying query optimizer. Compared to Neo, Bao learns significantly faster (< 1 hour), and can handle dynamic workloads and datasets. However, on stable workloads and datasets, Bao does not achieve the same performance as Neo. Bao has seen significant adoption, including at Microsoft [3].

My work on learned query optimization is open-source.[1]

Learned Index Structures. Index structures, like B-Trees, accelerate queries for particular data items or ranges of data items. Although index structures are almost as old as database systems themselves, recent work has shown how machine learning techniques can both shrink and accelerate traditional index structures. Unfortunately, the evaluation datasets and initial prototypes of learned index structures were never released. During my postdoc, I developed and released the first efficient open-source implementation of a learned index structure, along with a benchmark for learned indexes and an automatic tuning tool.

- The learned index benchmark [4] contained the first open-source implementation of several learned index structures, along with four real-world datasets for evaluation. Additionally, the benchmark study shed light on *why* learned index structures outperformed their traditional counterparts; while previous work had attributed gains to branch prediction or data compression, the study revealed that branch misses, cache misses, and instruction counts all played a statistically-significant role in the performance of learned indexes. This work is open-source.[2]
- In order to maximize the performance of a learned index, one must tune it to the underlying dataset. CDFShop [5] is the first entirely-automated learned index structure tuning tool. CDFShop works by computing properties of the data's cumulative distribution function (CDF) and mapping those properties to parameters of learned indexes. CDFShop can achieve super-human tunings for our test datasets in under an hour. This work is open-source.[3]

Learned Cloud Workload Management. Cloud database systems, compared to traditional database systems, promise both increased performance and decreased costs for many workloads. But obtaining these benefits requires navigating significant complexity, such as cluster sizing, query scheduling, and data placement. During my Ph.D., I developed WiSeDB and NashDB to address this need.

- WiSeDB [6], [7] transforms a user's workload description (e.g., types of queries) and performance requirements (e.g., certain queries have specific deadlines) into a customized workload management policy that handles cluster sizing and query scheduling. For batch workloads, WiSeDB uses a supervised learning regime that extracts

1 https://learned.systems/qo
2 https://learned.systems/sosd
3 https://learned.systems/rmi

patterns from a generated training set of optimal solutions to small problems. For online workloads, WiSeDB uses a network of multi-armed bandits, which are optimized globally. We found that WiSeDB could lower cloud costs by up to 2x while increasing total workload performance by 20%. WiSeDB is open-source.[4]

- NashDB [8] is an adaptive data layout system for cloud databases. In order to achieve low query latency on a budget, users must partition and replicate their data. Additional replicas can increase query performance, but increase costs. Thus, users should partition their database so that frequently-accessed or high-priority data can be highly replicated. NashDB solves this problem by deeply integrating with the underlying economics of the cloud: by allowing the user to assign an economic value to each query, NashDB efficiently finds data layouts that are in Nash equilibrium. While such layouts may not be optimal (finding an optimal layout is NP-Hard), they are provably superior to any non-equilibrium solution. NashDB is open-source.[5]

Machine Learning for Other Systems

The potential for machine learning to improve systems spans far outside of data management. In addition to my primary research projects in database systems, I have also lead and collaborated on efforts in other subfields.

- **Learned Garbage Collection.** Languages like Python, JavaScript, and Go use runtime garbage collection (GC) systems to free the user from manual memory management. While these runtime systems are often entirely transparent to the user, GC can become a performance bottleneck in advanced applications. I led an effort resulting in a learned approach to garbage collection [9]: by using a modified variant of Q-learning, this GC can automatically learn when an application's performance (defined as a metric specified by the user) will be least impacted by GC, and when GC will be most beneficial to the application's memory footprint. The learned GC matched or outperformed expert-tuned GC policies for several test applications. This work is open-source.[6]
- **Learned Video Compression.** Low-bitrate video compression is critical for streaming video over poor or sporadic Internet connections. Traditionally, low-bitrate compression has made use of several intuitive assumptions about motion, but these assumptions often do not line up with reality. In a collaborative effort [10], I contributed to a low-bitrate compression technique that learns a model of motion over a video and compresses the video by encoding the learned model's parameters. Decompression is then performed via model inference. I encourage you to look at the "Ours / AVC" video on the project's webpage,[7] which shows the same video encoded with our technique (left) and the current state-of-the-art (right) at identical bitrates.

Future directions & the next big thing

My next big challenge will be building fully-autonomous and self-tailoring data analytics systems. Imagine a data system that automatically invents new algorithms optimized for their (quickly-changing) underlying hardware, while simultaneously adapting their physical design to the user's workload. Further, imagine that such a system has a rich semantic understanding of user intent, deeply integrating recommendations and domain-knowledge throughout. Such systems would free practitioners from the drudgery of maintaining indexes, tuning layouts, materializing view, etc., but I believe that such systems could go far beyond reducing costs and speeding up reports. Learned systems will enable practitioners to use their data in ways that were previously infeasible or unknown to them, and will greatly reduce the cost of applying data analysis techniques to new domains.

To me, there are few things in life more satisfying than the feeling of cracking a difficult problem with a team of collaborators. With the expertise I have developed in data management and machine learning, I am eager to investigate a number of new challenges. There is obvious potential for learned approaches in almost every field of systems, but also within algorithms, compilers, networks, HPC, and software engineering. I would be excited to collaborate with other experts to determine to investigate the unifying properties of such problems, and to discover what generic and reusable tools can be built to make a given solution "learned."

4 https://git.io/wisedb
5 https://git.io/nashdb
6 https://github.com/bobbyluig/cpython-ml
7 https://rm.cab/semvid1

[1] R. Marcus *et al.*, “Neo: A Learned Query Optimizer,” *PVLDB*, vol. 12, no. 11, pp. 1705–1718, 2019.
[2] R. Marcus, P. Negi, H. Mao, N. Tatbul, M. Alizadeh, and T. Kraska, “Bao: Making Learned Query Optimization Practical,” presented at the SIGMOD ’21, China, Jun. 2021. doi: 10.1145/3448016.3452838.
[3] P. Negi *et al.*, “Steering Query Optimizers: A Practical Take on Big Data Workloads,” in *Proceedings of the 2021 International Conference on Management of Data*, Virtual Event China, Jun. 2021, pp. 2557–2569. doi: 10.1145/3448016.3457568.
[4] R. Marcus *et al.*, “Benchmarking Learned Indexes,” *PVLDB*, vol. 14, no. 1, pp. 1–13, Sep. 2020.
[5] R. Marcus, E. Zhang, and T. Kraska, “CDFShop: Exploring and Optimizing Learned Index Structures,” presented at the SIGMOD, Portland, OR, Jun. 2020. doi: https://doi.org/10.1145/3318464.3384706.
[6] R. Marcus and O. Papaemmanouil, “WiSeDB: A Learning-based Workload Management Advisor for Cloud Databases,” *PVLDB*, vol. 9, no. 10, pp. 780--791, 2016, doi: 10.14778/2977797.2977804.
[7] R. Marcus and O. Papaemmanouil, “Releasing Cloud Databases from the Chains of Performance Prediction Models,” presented at the CIDR, San Jose, CA, 2017.
[8] R. Marcus, O. Papaemmanouil, S. Semenova, and S. Garber, “NashDB: An End-to-End Economic Method for Elastic Database Fragmentation, Replication, and Provisioning,” Houston, TX, 2018. doi: https://doi.org/10.1145/3183713.3196935.
[9] L. Cen, R. Marcus, H. Mao, J. Gottschlich, M. Alizadeh, and T. Kraska, “Learned Garbage Collection,” presented at the SIGPLAN, 2020. doi: https://doi.org/10.1145/3394450.3397469.
[10] S. Garber, R. Marcus, A. DiLillo, and J. Storer, “Low Bitrate Compression of Video with Dynamic Backgrounds,” Snowbird, Utah, 2020.

I have been blessed with good teachers. In grade school, I had teachers who encouraged me to investigate and play with the concepts in class (as opposed to rote memorization), and to make discoveries by interrogating my own assumptions. While at the time I frequently wanted my teachers to "just give me the answer," in retrospect this type of education – a mix of the Socratic method and creating room for exploration – was absolutely critical for the development of my intellectual curiosity. Computer science especially lends itself to exploratory learning, as acquiring a basic computer is much easier than acquiring, for example, a particle accelerator or a chemistry lab.

While this method of teaching does not work for everyone, the results are exactly what I want for my own students: fostering not only mastery of the subject matter, but curiosity and deep engagement. In my view, sparking interest and equipping students with the tools to understand computer science literature and perform their own investigations is equally important to mastering any particular course's subject matter.

Teaching Experience

- I was the sole instructor and lecturer for a senior-level database course during my last semester at Brandeis. I developed lesson plans and lecture slides, along with homework assignments and programming assignments. I also designed the course's final project, which had students develop a working column-store database that could process simple join queries. Code was graded based on performance on an anonymized public leaderboard. My course evaluations (available upon request) placed my course in the top 1% of courses at Brandeis.
- I developed an entirely new curriculum, programming assignments, and tests for a 2nd semester introductory programming course (you can see the programming assignments here on GitHub). The course was co-taught by myself and a faculty member.
- I worked as a TA for various instructors for 8 semesters (3 semesters were required), which included grading, one-on-one mentoring, and occasional lecturing. I won the Brandeis Outstanding Teaching Fellow award.
- While at MIT, I advised two undergraduates on separate research projects. Both projects resulted in a publication.

Teaching Philosophy

My teaching philosophy is to *ensure competency while pushing for mastery.* The courses I have taught have all contained four components: in-class interaction, written homework assignments, programming assignments, and exams. I structure each component strategically to facilitate student learning.

In-class interactions. While I believe that college-level students are both capable and responsible for making decisions that facilitate their learning, I also believe that incentives can help. Students that attend and interact with lectures tend to do better than their peers. Part of this is correlation (e.g., stronger students may know the answer the questions posed to the class), but I believe there is a causal component as well (e.g., an engaged listener is more likely to retain information). In-class interactions – like daily quizzes, cold-calling, or just trying to be an entertaining lecturer – both create an incentive to attend lecture and opportunities to interact with the material.

Written homework assignments. Highly-motivated students will further investigate the material presented in a course with no prompting, and this investigation is often the most fruitful part of a student's education. I view written homework assignments as an opportunity to pose interesting questions that go beyond the lecture's content. This helps direct those highly-motivated students in a productive direction, as well as encouraging less-motivated students to engage beyond the lecture (admittedly, this engagement is often surface-level, but the exceptions make it worthwhile). Written homework assignments also provide a critical indicator to me as an instructor, allowing me to see which students are bored (and need more challenging material) and which students are struggling (and need help catching up).

Programming assignments. Not everyone "learns by doing," but everyone can demonstrate competency by doing. I design my programming assignments so that a student with an acceptable understanding of the material can receive an acceptable grade, but getting an exceptional grade requires an exceptional understanding of the material beyond the lectures. This means designing programming assignments that can be completed in a reasonable time by a particularly

busy student, but can also be “dug into” by a highly motivated student. Performance or accuracy-driven code provide easy opportunities for this, but it is possible even when the only requirements are functional-correctness (e.g., additional behavior).

Exams. I view exams as a forcing-function for studying, specifically for those students who will avoid learning the course material until right before an exam deadline. While exam scores serve as a somewhat-accurate measure of competence, this is only a secondary goal of giving an exam. The primary purpose is to cause students to engage with and study the material. To facilitate this, I usually release “supersets” of my exams as practice exams beforehand. For example, I will release a practice exam with 50 questions, and tell the students that the exam will consist of 5 of those questions. This is a lot of work, but is extremely effective – some students will do all the work required to perfectly answer all 50 questions, significantly more studying than they would have done if I had kept the 5 exam questions a secret. This also gives students a sense that the exam “was fair” (even if some of the sample questions required the student to learn far beyond the course material!).

Combined together, I believe I effectively use these four tools to deliver instruction that is effective for both students who may struggle with the material and for students who are seeking to excel.

Courses I Can Teach Immediately I am prepared to teach introductory or advanced courses on database systems, machine / deep learning (especially computer vision and reinforcement learning), and cloud computing. I can also teach introductory computer science courses, introductory material in computational theory, and introductory material in data structures and algorithms.

I believe that obvious and objective injustices demand that those with power engage in a life-long process of continuous learning and strategic engagement. The myriad of institutionalized harms – including, but absolutely not limited to, classism, sexism, racism, ableism, nationalism, xenophobia, homophobia, and Imperial colonialism – are complex issues that entire academic disciplines actively study. In this context, my job is two-fold:

1. To continuously practice empathy and actively pursue knowledge and understanding about social hierarchy.
2. To strategically and intentionally encourage organization and coalition-building against these evils.

I will not wax poetic about the harms that contemporary social hierarchy have brought upon vulnerable populations because (a) everything I say would be better stated by an expert, and (b) while the intricacies of these hierarchies are complex, the magnitude of their harm is obvious (especially within CS communities). Instead, I will try to summarize the path I've taken, what I still have to learn, and how I think I can make myself part of a strategic solution.

What I've Learned. When I enrolled the University of Arizona, I enrolled in a "History of Feminism" class because I thought it would be easy. It wasn't easy, but it did fundamentally change the way I viewed the world. I added a gender studies minor. I was dumbfounded not just by what I was learning about gender, but by *what I thought I knew about gender that was absolutely and objectively wrong.* Three years later, I was writing term papers about Judith Butler or Martin Heidegger and starting to convince myself I knew what I was talking about. I thought I had read my way out of patriarchy. Obviously, this isn't possible. Hopefully such hubris is excusable in a 20-year old.

Of course, studying race, gender, class, sexuality, etc. are critical to understanding the current social climate. But the most important takeaway from me wasn't a formal theory, statistic, or syllogism. What I learned was that the most important thing to become a better person is empathy. When someone gains the courage to speak about harm they have experienced, empathy does not just mean listening to their testimony and nodding along. It means elevating their testimony to the level of theory. It means *reading* and *internalizing* their testimony. It means treating the lessons behind their testimony with the same importance as we treat an optimality proof. It is critical to treat these voices not just as individual testimonies, but as *theory in the flesh*.

What I Still Need to Learn. Despite some formal training and a decade of effort, I still have a lot to learn. First, while it is impossible for me to ever fully understand the experiences of many marginalized groups because I simply will never have the same experiences they do, I still need to better myself by listening to and actively seeking out their voices. Second, engagement with theory, while secondary to critically listening to the experiences of others, is important to help contextualize testimony, as well as linking that testimony to a larger picture.

What I Can Do. As a potential faculty member, it is important to me that my efforts reach far beyond increasing the contrast of the University's recruitment flyer photo. I want to help build an environment of "deep inclusion." In an academic environment, "deep inclusion" means, at minimum, an environment where everyone feels not only included, but safe enough to challenge the ideas of others and to have their ideas challenged in return. While this is good for scientific progress, such efforts are also demanded by justice.

I can personally contribute to such an environment in two ways. First, to intentionally train empathy in myself and to encourage it in others. Like a muscle, empathy needs exercise and frequent use in order to grow (or remain) strong. When we are empathetic, we are both less likely to harm others, and more likely to understand and recognize when harm is inflicted on others. Empathy is how we make someone feel "heard." "Heard" not just in the sense that I parsed the sentence they spoke, but that I have internalized a small part of the experience behind their words, and will take appropriate action in the future. I believe that such empathy must be developed continuously over a lifetime.

Second, I can support and encourage coalition-building and organizing for the purpose of challenging forms of institutionalized harm. While individuals can make a difference, the power of an organized group, and the collective consciousness it can bring, is unmatched. In my opinion, committing time and effort to such organizations is the most effective way to create change. Contributing positively to such coalition-building sometimes requires loud advocacy, but frequently requires supportive allyship.

While I still have a lot to learn, I am confident in my ability to better myself and my surrounding community.