I research **machine learning for systems, especially database management systems**. From database query optimizers to garbage collectors, today's computer systems are composed of hand-tuned heuristics. These off-the-shelf heuristics are convenient and freely available, but are optimized for a "general purpose" case, and thus do not maximize system performance for most actual applications. Often, for a specific applications, learned instance-optimized policies can outperform the standard heuristics by orders of magnitude. These gains might come from specific properties of the user's workload (that are not true in general), or from properties of the underlying hardware (e.g., larger or smaller cache sizes), or from complex interactions between the two. Application engineers are forced to choose between investing massive engineering effort into manually tuning heuristics for each specific application, or accepting greatly diminished performance.

For example, consider the data warehousing needs of a university. In order to speed up analysis performed by administrators about tuition payments, the system may physically organize its data by student graduation year, scholarship status, or payment plan enrollment. Such a layout would be great for those administrators, but won't help professors looking up course feedback information, analyzing which courses lead to student success, or even querying historical grade data. These queries may be slow due to reading far more data than necessary. The people in charge of the data warehouse must manually balance these constraints, finding physical layouts that work for all clients, or manually employing techniques like replication or materialization to make up for poor data layout. This is a grueling task, and doing it correctly would essentially require writing an entirely new data warehousing platform. The people running the data warehouse generally have more pressing concerns, so often one type of requests is simply left with poor performance or relegated to a secondary, lower-priority system.

I believe machine learning techniques are the key to *automatically* achieving the performance of a custom-tailored system with same conveniences as off-the-shelf data tools. With machine learning, next-generation systems will adapt to changes in workload or hardware, autonomously invent new ways to best meet the user's requirements, and even understand user intention. Unfortunately, building such next-generation systems is not as simple as taking an existing ML tool and applying it to systems problems. In my experience, this work requires a deep understanding of both machine learning and systems techniques, resulting in entirely new algorithms, data structures, training regimes, etc. Personally, I have investigated a wide range of systems spanning many subfields, with my primary focus on 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). A good query plan might execute a complex query in seconds, while a poor plan may require hours of execution time to perform the same logical computation. 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 at MIT, I developed two systems for learned query optimization, Neo and Bao.

- Neo [1] is based on our discovery of a deep connection between the dynamic programming algorithm typically used in query optimization and value iteration. Armed with this knowledge, we developed a tree convolution neural network and search technique specific to database query plans. The result was the first deep reinforcement learning powered query optimizer, capable of 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 '21 best paper) sits on top of a traditional optimizer, and uses a version of Neo's value model to power a multi-armed bandit approach that "steers" the traditional optimizer using the optimizer's own configuration parameters. Bao thus combines the stability and reliability of a traditional optimizer with the adaptiveness of a learned system. Compared to Neo, Bao learns significantly faster (< 1 hour), and can handle dynamic workloads and datasets. But, on stable workloads, Bao does not achieve the same performance as Neo,

but still outperforms traditional optimizers. Bao also includes a slew of user tools to aid users in incremental adoption, a key practical barrier.

Neo was, to the best of my knowledge, the first system to fully "close the loop" between the query optimizer and query execution using deep reinforcement learning, answering a long-outstanding question within the database community. With Bao, we took the lessons learned from Neo and figured out how to build a highly-practical learned query optimizer. Bao is actively being integrated at Intel, Facebook, Google, and Microsoft; Microsoft has published a study showing Bao's significant impact on their internal data warehouse [3].

Learned Index Structures. Index structures, like B-Trees, accelerate queries for particular data items or ranges of data items. B-Trees are so powerful, they have become ubiquitous, and you can find a B-Tree implementation in nearly every database management system. And for good reason: for highly selective queries, a B-Tree can perform the same computation as a full table scan with significantly fewer disk reads. 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 a 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 only a combination of branch misses, cache misses, and instruction counts provided a statistically-significant explanation of the performance of learned indexes.
- 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.

The learned index benchmark and the CDFShop tuner have seen significant adoption in the community. Since publication, there have been 31 papers about learned index structures at major database conferences and their workshops (VLDB, SIGMOD, ICDE). Of them, 28 of them either used our benchmark and/or our datasets.

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., query templates) and performance requirements (e.g., certain queries have deadlines) into a customized workload management policy that handles cluster sizing and query scheduling. For batch workloads (queries arrive all at once), WiSeDB uses a supervised learning regime that extracts patterns from a generated training set of optimal solutions to small problems. For online workloads (queries arrive one by one), 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%.
- NashDB [8] is an adaptive data layout system for cloud databases. In order to achieve low query latency on a monetary 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 cost structure 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.

Open source. I believe researchers, especially systems researchers, should open source their research artifacts whenever possible. This is a great way to ensure research is reproducible, and open sourcing also helps accelerate fair and scientific comparisons between systems. More on my open source philosophy and a listing of artifacts I have made open source (which includes all of the above) can be found on my website: https://rmarcus.info/blog/code

Other interests, 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. I led an effort that created a **reinforcement learning powered garbage collector** for Python that custom-tailored a GC policy to the user's metric of interest (e.g., requests per second) [9] . I collaborated on a **low-bitrate video compression** technique [10] that learns a model of motion over a video and compresses the video by encoding the learned model's parameters – check out the project's webpage for examples. I also contributed to **Park, an AI gym for systems problems** (a suite of benchmark problems) [11], which makes a wide array of systems problems – from networking to database systems – accessible to ML researchers.

Future directions & the next big thing

My next big challenge will be building fully-autonomous and self-tailoring data analytics systems. This goes way beyond simple knob-tuning: imagine a data system that automatically invents new algorithms optimized for its (quickly-changing) underlying hardware, while simultaneously adapting its physical design to the user's workload. Imagine that such a system has a rich semantic understanding of user intent, deeply integrating recommendations and domain-knowledge throughout – possibly even suggesting new explorations or analyses to perform. Such systems would free practitioners from the drudgery of maintaining indexes, tuning layouts, materializing views, etc., but I believe that we could go far beyond reducing costs and speeding up reports. Instance-optimized 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. Achieving a truly autonomous instance-optimized learned data analytics system would be like putting the power of an entire team of data engineers into the hands of every practitioner, for free.

Concretely, my next step towards achieving such a system will be combining my work on query optimization with recent results in learned data layout to create a *learned execution engine* that simultaneously learns how to layout data and how to query that data efficiently. Of course, such a learned execution engine would only be the first step towards realizing the vision of instance-optimized data systems – I plan to continue my current collaborations with MIT and Brandeis to connect the learned engine I develop to broader efforts.

Of course, along the way, there will be plenty of "lessons learned" from navigating the complexities of integrating machine learning techniques into data systems. There will be many opportunities for shorter-term wins, such as integrating ML techniques into queuing systems or distributed caches like memcached.

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."

[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," demo 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,” presented at the 37th ACM SIGMOD, 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 MAPS workshop at 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,” presented at the Data Compression Conference (DCC), Snowbird, Utah, 2020.

[11] H. Mao *et al.*, “Park: An Open Platform for Learning-Augmented Computer Systems,” in *Advances in Neural Information Processing Systems 32*, 2019, pp. 2490–2502. Accessed: Dec. 19, 2019. [Online]. Available: http://papers.nips.cc/paper/8519-park-an-open-platform-for-learning-augmented-computer-systems.pdf

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 (see CV) 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.