

CSE 132C

Database System Implementation

Arun Kumar

Topic 7: ML for RDBMSs

Optional; NOT included in the syllabus for Final Exam

ML for Systems

Q: Why bother applying ML to well-studied systems issues?

- ❖ Jeff Dean's rationales (from NIPS MLSys'17 keynote):
 - ❖ Hand-crafted heuristics are pervasive but not very adaptive; data-driven ML can improve system metrics
 - ❖ User-tunable knobs have exploded and are painful
 - ❖ Hardware has caught up with ML/DL demands; cloud resources are cheap and widely available
 - ❖ Automated ML simplifies use of ML for systems
- ❖ Also, cynically: "ML for Systems" is a hot/ controversial topic for publications! May get a lot of (not all wanted) attention! :)

ML for Systems

Anywhere We're Using Heuristics To Make a Decision!

Compilers: instruction scheduling, register allocation, loop nest parallelization strategies, ...

Networking: TCP window size decisions, backoff for retransmits, data compression, ...

Operating systems: process scheduling, buffer cache insertion/replacement, file system prefetching, ...

Job scheduling systems: which tasks/VMs to co-locate on same machine, which tasks to pre-empt, ...

ASIC design: physical circuit layout, test case selection, ...

ML for Systems

Anywhere We've Punted to a User-Tunable Performance Option!

Many programs have huge numbers of tunable command-line flags, usually not changed from their defaults

```
--eventmanager_threads=16  
--bigtable_scheduler_batch_size=8  
--mapreduce_merge_memory=134217728  
--lexicon_cache_size=1048576  
--storage_server_rpc_freelist_size=128  
...
```

ML for Systems

Keys for Success in These Settings

- (1) Having a numeric metric to measure and optimize
- (2) Having a clean interface to easily integrate learning into all of these kinds of systems

Current work: exploring APIs and implementations

Basic ideas:

- Make a sequence of choices in some context

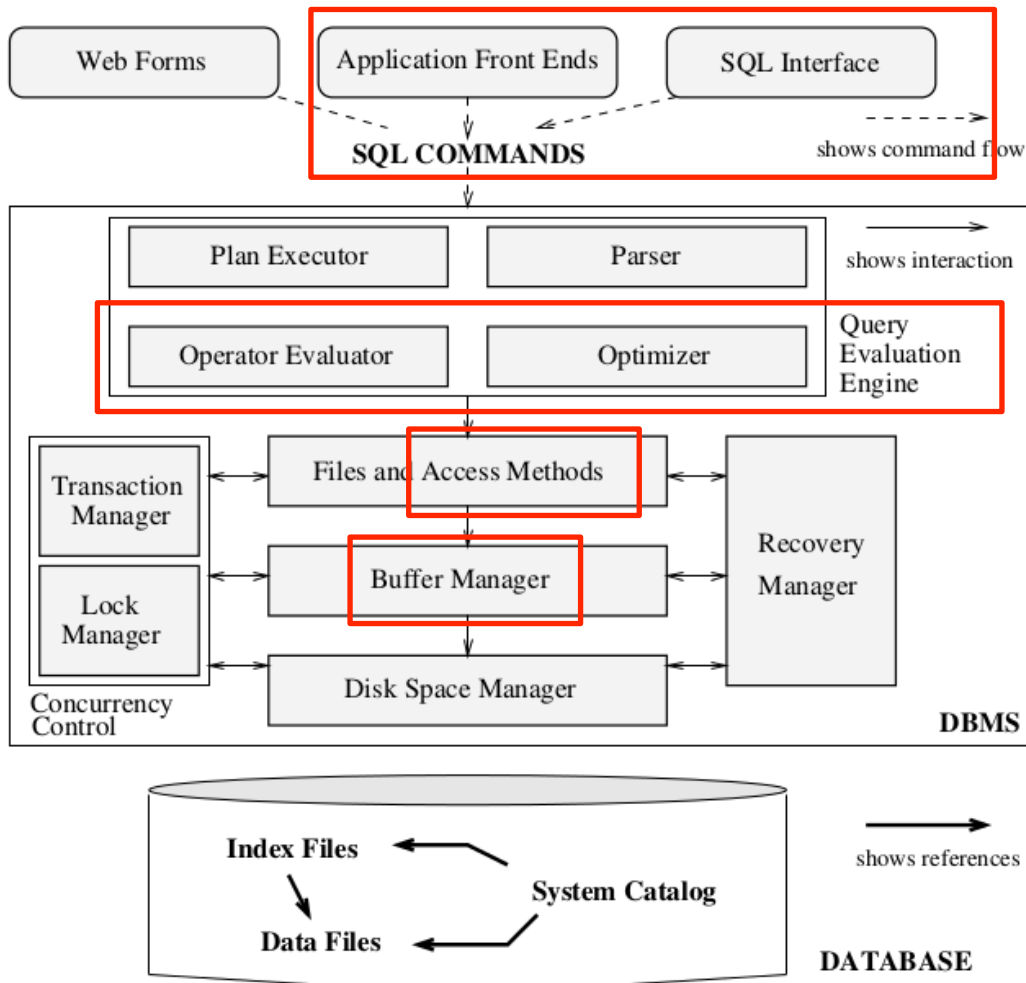
- Eventually get feedback about those choices

- Make this all work with very low overhead, even in distributed settings

- Support many implementations of core interfaces

ML for an RDBMS

Q: Where is ML potentially helpful in an RDBMS?



Natural language interfaces (NLIs)

Learned Query Processing and Query Optimization

Learned Access Methods

Learned Caching and Scheduling Policies

ML for Knob Tuning and Resource Management

ML for Knob Tuning/Resource Mgmt

- ❖ **Motivation:** Modern RDBMSs have 100s of config parameters (buffers for EMS, degree of parallelism, etc.)
 - ❖ Mixture of continuous and discrete parameters
 - ❖ Effects on query latency, etc. can be non-monotonic
 - ❖ Optimal settings highly dependent on schema properties, database instance, hardware, auxiliary data structures, and query workload properties
 - ❖ Impossible for DBAs to keep up, esp. cloud
- ❖ **Why ML?** Adapt quickly to instance/query workload/etc.; target flexibility (latency/utilization/etc.); can be more accurate
- ❖ “Autonomous”/“Self-driving” are the industry buzzwords

Example

Automatic Database Management System Tuning Through Large-scale Machine Learning

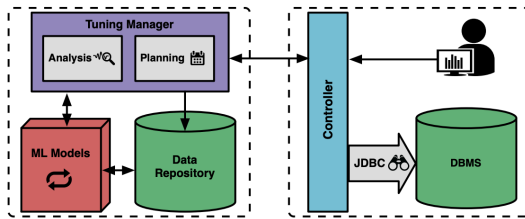


Figure 2: OtterTune Architecture – An overview of the components in the OtterTune system. The controller connects to the DBMS and collects information about the performance of the system. This information is then sent to the tuning manager where it is stored in its repository. It then builds models that are used to select an optimal configuration for the DBMS.

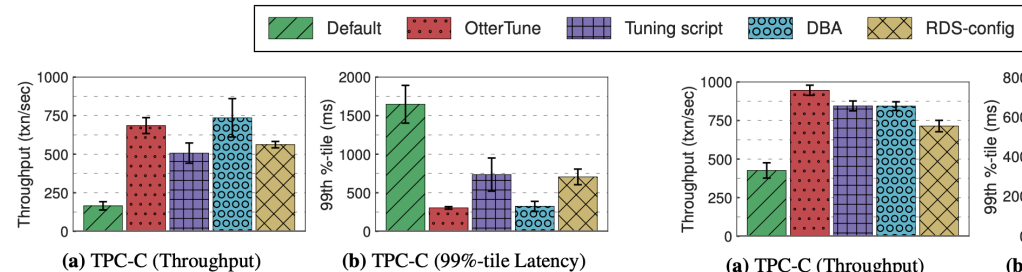


Figure 10: Efficacy Comparison (MySQL) – Throughput and latency measurements for the TPC-C benchmark using the (1) default configuration, (2) OtterTune configuration, (3) tuning script configuration, (4) Lithuanian DBA configuration, and (5) Amazon RDS configuration.

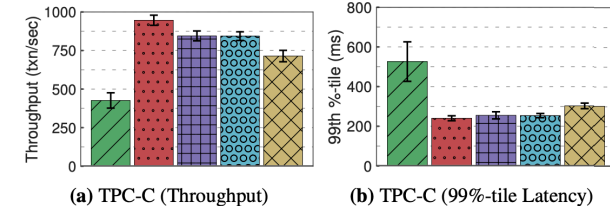


Figure 11: Efficacy Comparison (Postgres) – Throughput and latency measurements for the TPC-C benchmark using the (1) default configuration, (2) OtterTune configuration, (3) tuning script configuration, (4) expert DBA configuration, and (5) Amazon RDS configuration.

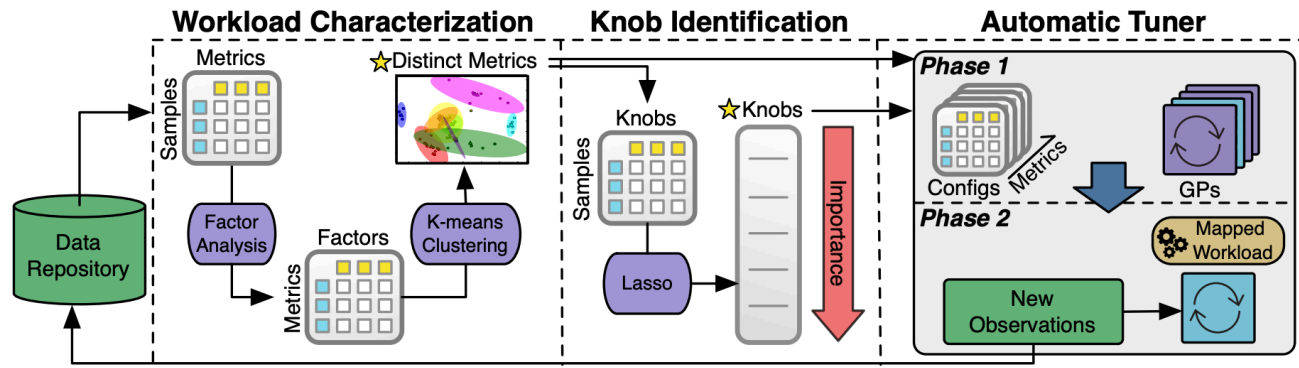


Figure 3: OtterTune Machine Learning Pipeline – This diagram shows the processing path of data in OtterTune. All previous observations reside in its repository. This data is first then passed into the **Workload Characterization** (Sect. 4) component that identifies the most distinguishing DBMS metrics. Next, the **Knob Identification** (Sect. 5) component generates a ranked list of the most important knobs. All of this information then fed into the **Automatic Tuner** (Sect. 6) component where it maps the target DBMS's workload to a previously seen workload and generates better configurations.

Natural Language Interfaces (NLIs)

- ❖ **Motivation:** SQL is too hard for non-technical business users (sales, marketing, etc.) and lay public
 - ❖ NLIs allow more people to exploit relational databases
 - ❖ No need to learn complex syntax or even schema details
 - ❖ Regular conversational style interactions
- ❖ **Why ML?** State-of-the-art in natural language processing (NLP) is DL-based; pure parsing/rule-based is too brittle
 - ❖ Extremely challenging to automatically infer both structure and literals from NL query to translate to proper SQL!
 - ❖ AFAIK, no robust open-domain commercial system today

Example

DBPal: An End-to-end Neural Natural Language Interface for Databases

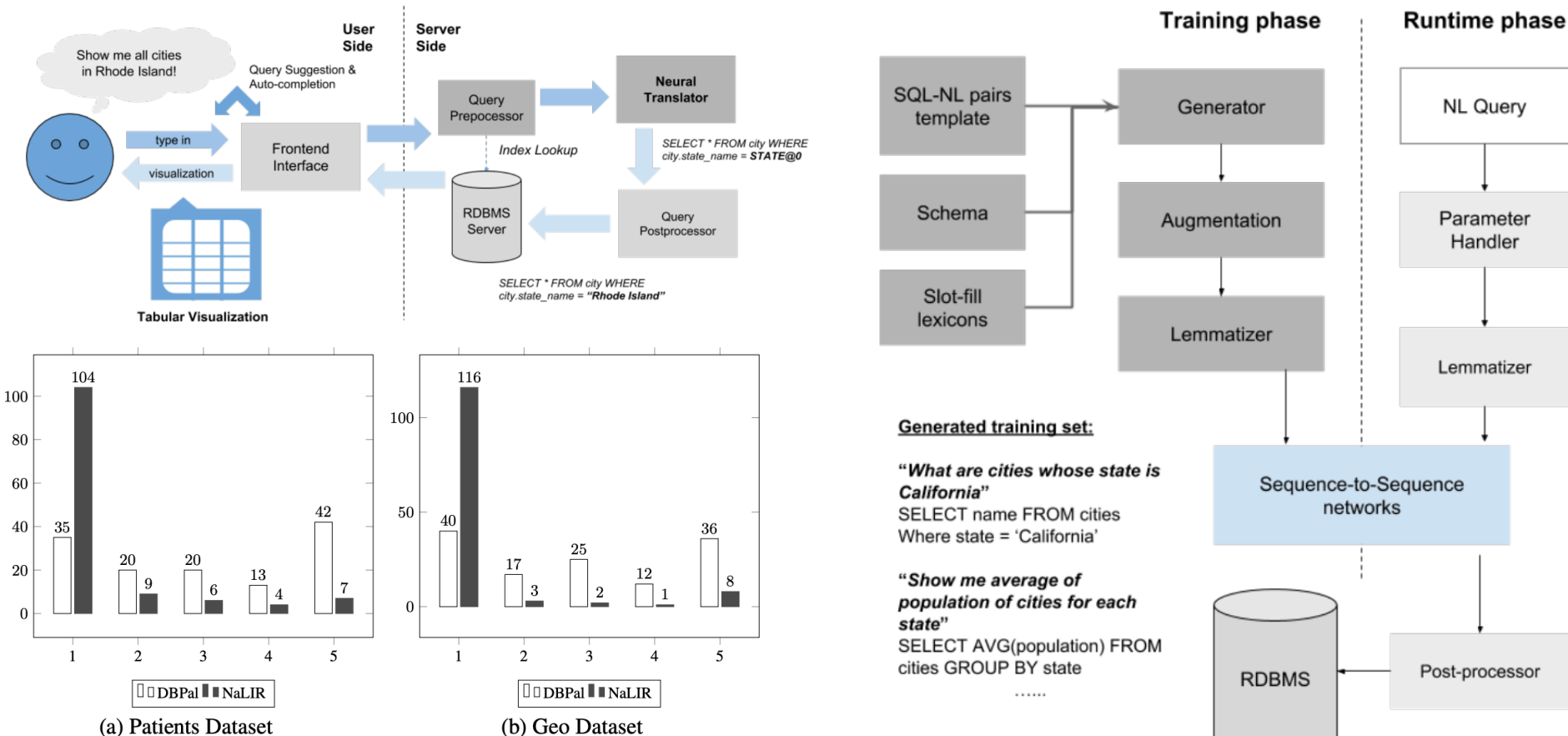


Figure 5: User ratings for different data sets on DBPal and NaLIR (higher is better)

Learned Scheduling/Caching Policies

- ❖ **Motivation:** Existing heuristic policies may not exploit data/query distributions well and thus waste runtime
- ❖ **Why ML?** By learning the underlying data/workload distributions, ML can help reduce runtimes/resource wastage
 - ❖ Learned schedulers: better load balancing to reduce worker idle times to improve utilization and/or latency
 - ❖ Learned caching/buffering: better retention and eviction decisions to increase cache hits and reduce latency

Examples

SageDB: A Learned Database System

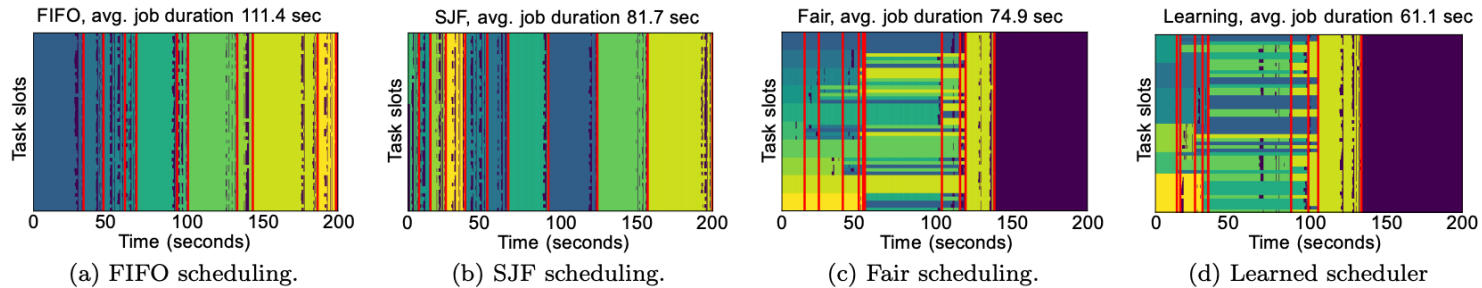


Figure 8: Our learned scheduler improves average job completion time of 10 randomly-sampled TPC-H queries by 45% over Spark's default FIFO scheduler, and by 19% over a fair scheduler on a cluster with 50 task slots (executors). Different queries are shown in different colors; vertical red lines show job completions.

Automating Distributed Tiered Storage Management in Cluster Computing

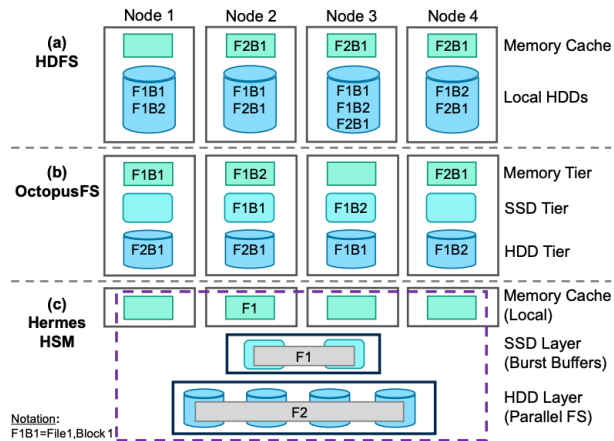


Figure 1: Data placement for three tiered storage systems

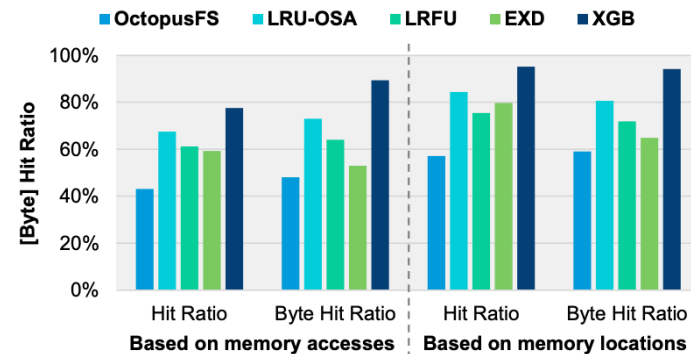


Figure 9: Hit Ratio and Byte Hit Ratio for the FB workload based on memory accesses and locations

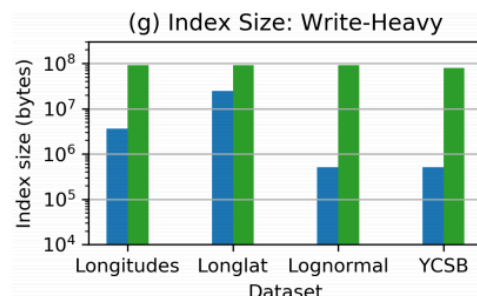
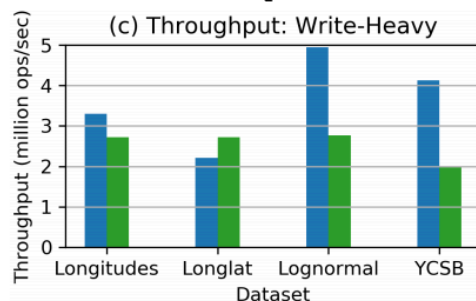
Learned Access Methods

- ❖ **Motivation:** Existing access methods may be wasting some system resources (memory, storage, runtime, etc.) because they do not exploit database instance distributions
- ❖ **Why ML?** By learning/approximating the underlying data distributions, ML can help reduce resource demands
 - ❖ Resource reduction target depends on use-case
 - ❖ Learned index structures: reduce memory/storage footprint of index, while maintaining or reducing query latency
 - ❖ Learned compression formats: reduce memory/storage footprint and file I/O time

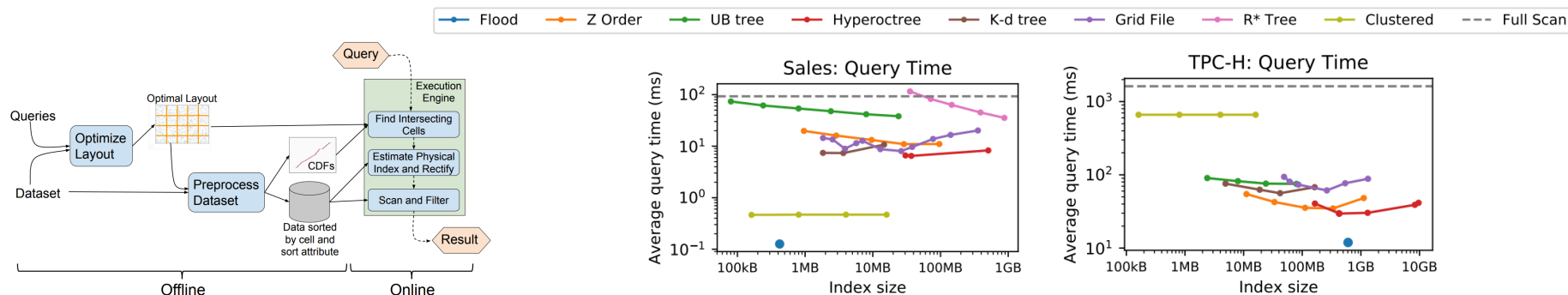
Examples

The Case for Learned Index Structures

ALEX: An Updatable Adaptive Learned Index



Learning Multi-dimensional Indexes



DeepZip: Lossless Data Compression Using Recurrent Neural Networks

https://www.cl.cam.ac.uk/~ey204/teaching/ACS/R244_2018_2019/papers/Kraska_SIGMOD_2018.pdf

<https://arxiv.org/pdf/1905.08898.pdf> ; <https://arxiv.org/pdf/1912.01668.pdf>

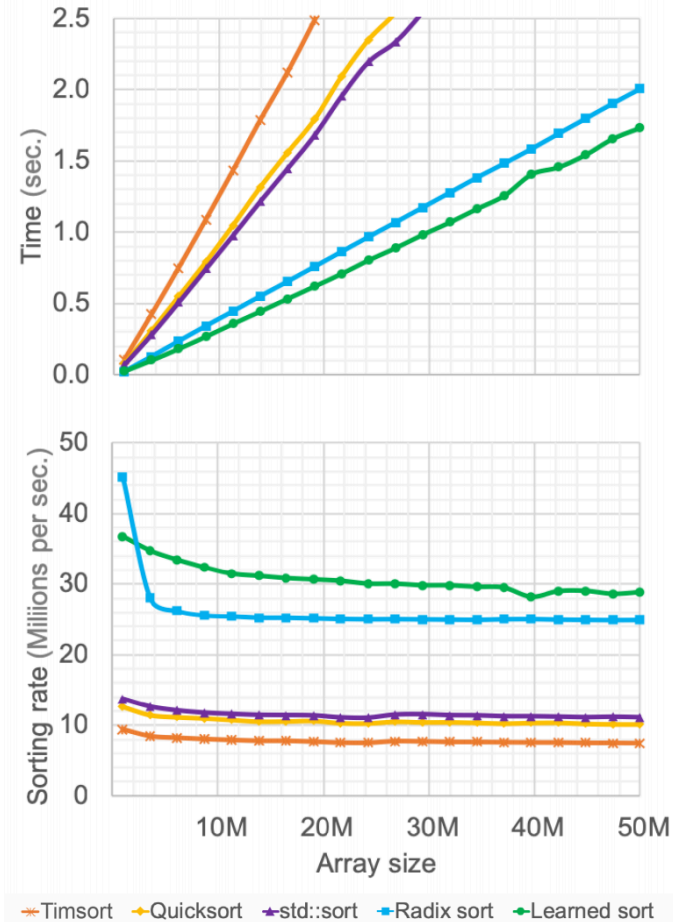
<https://ieeexplore.ieee.org/document/8712659>

Learned Query Processing

- ❖ **Motivation:** Existing phy. op. impl. are not exploiting database instance distributions well; can save some runtime or improve runtime predictability by doing so
- ❖ **Why ML?** By learning/approximating the underlying data distributions, ML can reduce runtimes/improve accuracy
 - ❖ Learned sorting: the closer the distribution is to pre-sorted, the less time we can spend on sorting
 - ❖ Learned joins: learn the distribution and location of the join attributes to reduce hash look up and/or sorting needs
 - ❖ Learned query plans: Improve runtime predictability

Examples

SageDB: A Learned Database System



Plan-Structured Deep Neural Network Models for Query Performance Prediction

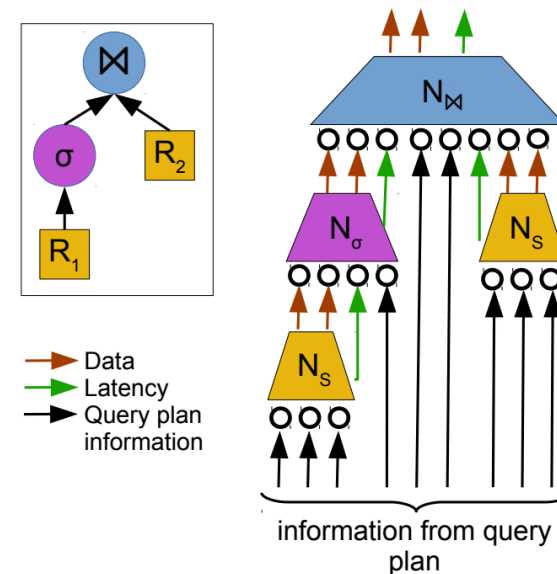


Figure 4: General neural network for latency prediction

Learned Query Optimizers

- ❖ **Motivation:** Existing optimizers have many heuristics (join orders, plan selection, cardinality estimation, etc.)
- ❖ **Why ML?** By learning/approximating the underlying data distributions, ML can reduce runtimes for final plan
 - ❖ Learned join order: Use join attribute distribution info and reinforcement learning to figure better join orders
 - ❖ Learned plan rewrites: Use database instance properties and attribute distributions to rewrite plans

Examples

Neo: A Learned Query Optimizer

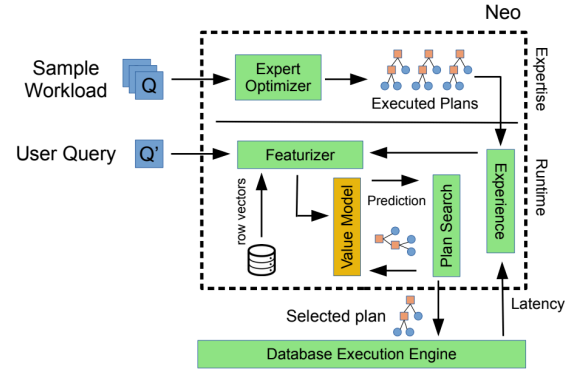


Figure 1: Neo system model

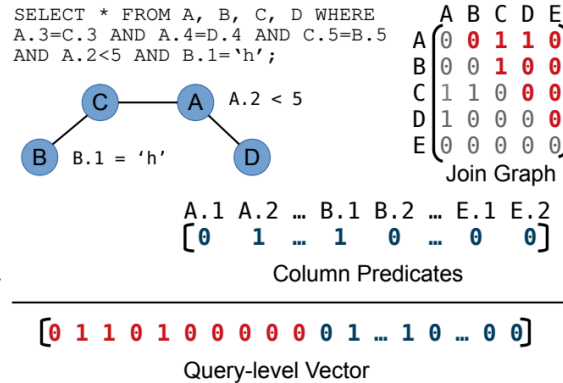


Figure 3: Query-level encoding

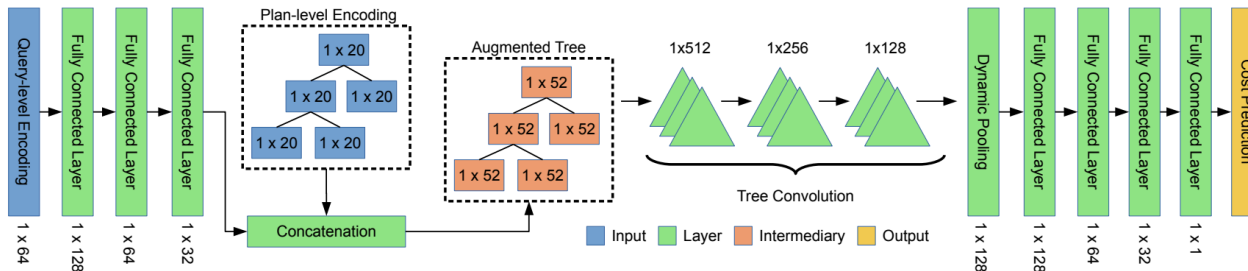


Figure 5: Value network architecture

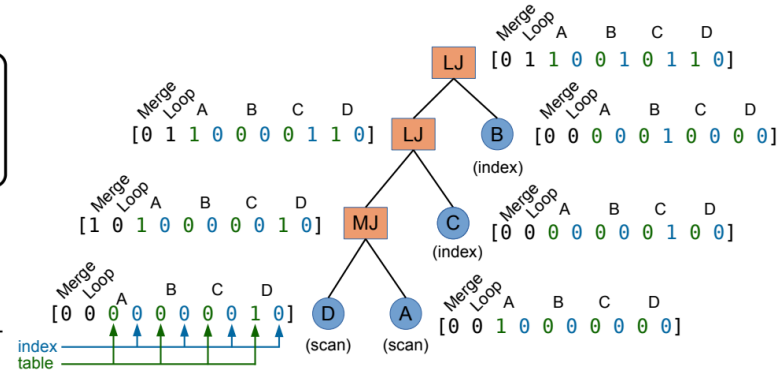


Figure 4: Plan-level encoding

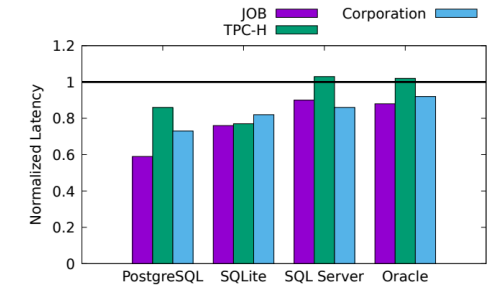
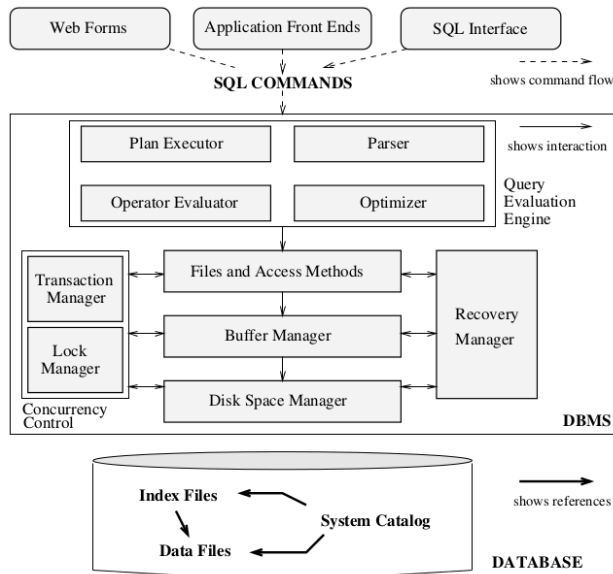


Figure 9: Latency of test query plans created by Neo after 100 episodes of training, normalized to plans created by the target system's corresponding optimizer for different workloads.

Learning to Optimize Join Queries With Deep Reinforcement Learning

Takeaways: ML for RDBMSs

Many parts of the RDBMS stack may benefit from ML/DL



ML for Knob Tuning and Resource Management

Natural language interfaces (NLIs)

Learned Caching/Scheduling Policies

Learned Access Methods

Learned Query Processing/Optimization

...

Apart from above, note that ML is already common in other data systems settings: data integration, data cleaning, etc.

Data systems will keep evolving due to evolution of hardware, cloud, and ML capabilities; stay informed of latest research!