Machine Learning for Data Management: A System View

Guoliang Li, Xuanhe Zhou
Department of Computer Science,
Tsinghua University, Beijing, China
Revisit database systems: What are the critical requirements

- **File System**
  - Relations: Easy to manipulate data

- **Relational Model, SQL**
  - Transactions: Atomicity, Consistency, Isolation, Durability
  - OLTP: rollbacks, triggers, locking, logging, …
  - Data analysis: High Quality

- **OLAP**
  - JIT compilation, vectorized execution, …

- **NoSQL**
  - Transactions + big data: High Scalability; ACID
  - Data models, indexing, partitioning, …

- **Distributed OLTP**
  - 2PC, Paxos, Distributed SQL…
  - DBaaS: Flexibility, Cost-Saving

- **Cloud-Native**
  - Compute-storage disaggregation, …

Cost Saving
(resource, DBAs, …)

Adaptivity
(applications, hardware, data, query, …)

High SLAs
(throughput, latency, scalability, …)

Learned Database
New Opportunities: What benefits can ML bring for databases?

- **Cost Saving: Manual → Autonomous**
  - Auto Knob Tuner: ↓ Maintenance cost
  - Auto Index Advisor: ↓ Optimization latency

- **High SLAs: Heuristic → Intelligent**
  - Intelligent Optimizer: ↓ Query plan costs
  - Intelligent Scheduler: ↑ Workload performance

- **Adaptivity: Empirical → Data-Driven**
  - Learned Index: ↑ Data access efficiency
  - Learned Layout: ↑ Data manipulation efficiency
New Opportunities: Why Now?

- **Cost Saving:** Manual → Autonomous
  - Auto Knob Tuner: ↓ Maintenance cost
  - Auto Index Advisor: ↓ Optimization latency

- **High SLAs:** Heuristic → Intelligent
  - Intelligent Optimizer: ↓ Query plan costs
  - Intelligent Scheduler: ↑ Workload performance

- **Adaptivity:** Empirical → Data-Driven
  - Learned Index: ↑ Data access efficiency
  - Learned Layout: ↑ Data manipulation efficiency

---

- On-Premise → Cloud
  (maintenance, setting, ⋯)
- Experience → Data-driven
  (various applications, hardware, data, query, ⋯)
- Heuristic → Intelligent
  (database design, data layout)

**Learned Database**
Double-Edged Sword: What are the challenges?

- Cost Saving: **Manual → Autonomous**
  - Auto Knob Tuner: ↓ Maintenance cost
  - Auto Index Advisor: ↓ Optimization latency

- High SLAs: **Heuristic → Intelligent**
  - Intelligent Optimizer: ↓ Query plan costs
  - Intelligent Scheduler: ↑ Workload performance

- Adaptivity: **Empirical → Data-Driven**
  - Learned Index: ↑ Data access efficiency
  - Learned Layout: ↑ Data manipulation efficiency

Challenges:
- **Feature Selection**: Pick relevant features from numerous query / database / os metrics;
- **Model Selection**: Design ML models to solve different database problems;
- **Diverse Targets**: Meet the SLA requirements under different scenarios;
- **Training Data**
- **Adaptivity**
ML4DB: An Overview

- **Automatic Advisor**
  - Knob Tuner
  - Index/View Advisor
  - Partitioner/Scheduler

- **Learned Generator**
  - SQL Generator
  - Adaptive Benchmark

- **Intelligent Optimizer**
  - Query Rewriter
  - Plan Enumerator
  - Cost Estimator

- **Learned Designer**
  - Learned Index
  - Learned Data Layout

- **Autonomous Databases**

---

**Automatic Advisor**

- Self-Configuration
- Self-Optimization
- Self-Organization

**Intelligent Optimizer**

- Query Rewriter
- Plan Enumerator
- Cost Estimator
- End-to-End Optimizer

**Learned Generator**

- Adaptive Benchmark
- SQL Generator

**Learned Designer**

- Learned Indexes
- Learned Layout

**Autonomous Data Management System**

- Paloton
- SageDB
- openGauss
- …
ML4DB: An Overview

- **Automatic Advisor**
  - Knob Tuner
  - Index/View Advisor
  - Partitioner/Scheduler

- **Learned Generator**
  - SQL Generator
  - Adaptive Benchmark

- **Intelligent Optimizer**
  - Query Rewriter
  - Plan Enumerator
  - Cost Estimator

- **Learned Designer**
  - Learned Index
  - Learned Data Layout

- **Autonomous Databases**
Automatic Advisor: Technique Development

- Performance (tuning quality, overhead, benefit estimation)
- Adaptivity (queries/codes, datasets, instances)

Knob Tuner

- Utilize Historical Data
  - Gaussian Process [sigmod, 2017]
- Tuning Effect Estimation
  - Deep Learning [vldb, 2019]
- Optimize Tuning Quality
  - Reinforcement Learning [sigmod, 2019] [vldb, 2019]
- Reduce Tuning Overhead
  - Delayed RL [vldb, 2021]
- Improve Adaptivity
  - Meta Learning [sigmod, 2021]
- Spark Tuning

Index Advisor

- Optimize Index Estimation
  - Deep Neural Networks [sigmod, 2019]
- Improve Index Quality
  - Reinforcement Learning [cikm, 2020] [icde, 2021]
- Support Index Update
  - Monte Carlo Tree Search [icde, 2022]

View Advisor

- Optimize MV Selection
  - Reinforcement Learning [icde, 2020]
- Optimize MV Estimation
  - Encoder-Decoder + RL [icde, 2021]
Automatic Knob Tuning

- Motivation

- Large number of configuration knobs
  - Total > 400
  - 10-15 are most vital for any workloads

- Knobs control nearly every aspect and have complex correlations
  - The relations are non-linear
  - One-knob-at-a-time is inefficient

Hi, list. I've just upgraded psql from 8.3 to 8.4. I've used pgTune before and everything worked fine for me. And now I have ~93% cpu load. Here's changed values of config:

```plaintext
default_statistics_target = 50
maintenance_work_mem = 1GB
constraint_exclusion = on
checkpoint_completion_target = 0.9
effective_cache_size = 22GB
work_mem = 192MB
wal_buffers = 8MB
checkpoint_segments = 16
shared_buffers = 7680MB
max_connections = 80
```
Traditional Knob Tuning Method

- **Sampling-based:** Explore knob-performance relations
  - **Planner:** Adaptively sample some knob settings
  - **Executor:** Get the performance of sampled settings by running workloads
  - **Estimator:** Predict knob-performance relations with Gaussian Process;
  - **Termination:** Terminate if arriving time limit; otherwise repeat above steps

Songyun Duan, Vamsidhar Thummala, Shivnath Babu. Tuning Database Configuration Parameters with iTuned. VLDB, 2009.
Problems in Traditional Knob Tuning

- **Challenges**
  - Sampling configurations from scratch is inefficient
    - Utilize historical configuration data
  - Knob-performance relations are extremely complex
    - Advanced ML techniques (depending on scenarios)
  - Important configuration features are not utilized
    - Inner metrics; query features; data features
Knob Tuner: Technique Development

- Performance (tuning quality, overhead, benefit estimation)
- Adaptivity (queries/codes, datasets, instances)

Knob Tuner

- Utilize Historical Data
  - Gaussian Process [sigmod, 2017]

- Tuning Effect Estimation
  - Deep Learning [vldb, 2019]

- Optimize Tuning Quality
  - Reinforcement Learning [sigmod, 2019] [vldb, 2019]

- Reduce Tuning Overhead
  - Delayed RL [vldb, 2021]
  - DB Manual [sigmod, 2022]

- Improve Adaptivity
  - Meta Learning [sigmod, 2021]

- Spark Tuning
  - Contextual GP [sigmod, 2020]
  - Adaptive Learning [icde, 2022]
Knob Tuner: Utilize Historical Data

- Automatically tune knobs with numerous historical data
  - Characterize workloads with runtime metrics (e.g., #-read-page, #-write-page)
  - Identify important knobs (rank knobs through knob-performance sampling)
  - Generate workload-to-identified-knob-settings correlations (data repository)
  - Given a workload, compute a mapped workload via metric similarity, use corresponding knob settings to initialize GP, explore more settings to get better performance
Knob Tuner: Tuning Effect Estimation

- **Motivation:** Expensive to run workloads to evaluate tuning effects
- **Basic Idea:** Estimate tuning effects without running workloads
- **Challenge:** Many metrics affect the performance
- **Solution:**
  - Collect DB metrics: logical-read, QPS, CPU usage, response time;
  - Initialize a buffer size using historical workloads with similar metrics;
  - Design a neural network to estimate the response time as tuning feedback;
  - Greedily reduce the initialized buffer size until arriving safe response time.

Knob Tuner: Optimize Tuning Quality

- **Motivation:** Traditional methods fall into local optimum
- **Basic Idea:** Use reinforcement learning (exploration-exploitation)
- **Challenge:** Map knob tuning into RL
- **Solution**

<table>
<thead>
<tr>
<th>RL</th>
<th>CDBTune</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agent</td>
<td>The tuning system</td>
</tr>
<tr>
<td>Environment</td>
<td>DB instance</td>
</tr>
<tr>
<td>State</td>
<td>Internal metrics</td>
</tr>
<tr>
<td>Reward</td>
<td>Performance change</td>
</tr>
<tr>
<td>Action</td>
<td>Knob configuration</td>
</tr>
<tr>
<td>Policy</td>
<td>Deep neural network</td>
</tr>
</tbody>
</table>

**Basic Idea:**
- Use reinforcement learning (exploration-exploitation).

**Motivation:**
- Traditional methods fall into local optimum.

**Challenge:**
- Map knob tuning into RL.

**Solution:**
- Use reinforcement learning to optimize tuning quality.

---

Knob Tuner: Optimize Tuning Quality

☐ Select proper RL models

- Many continuous system metrics and knobs

  - Value-based method (DQN)  
    - Replace the Q-table with a neural network
    - Input: state metrics; Output: Q-values for all the actions

  - Policy-based method (DDPG)  
    - (actor) Parameterized policy function: \( a_t = \mu(s_t|\theta^\mu) \)
    - (critic) Score specific action and state: \( Q(s_t, a_t|\theta^Q) \)

Knob Tuner: Optimize Tuning Quality

- Select high performance settings with RL (QTune as an Example)

Guoliang Li, Xuanhe Zhou, Shifu Li, Bo Gao. QTune: A Query-Aware Database Tuning System with Deep Reinforcement Learning. VLDB 2019.
Knob Tuner: Reduce Tuning Overhead

- **Problems in RL**
  - Significant Tuning Overhead
  - Require DBAs (select knobs, select knob ranges)

- **Tuning hints from manual**
  - Extract hints from manual

- **Parameter** = **Value** [*System Property*] [*Constant*]

Given in Text:  
- Set shared_buffers to 25% of RAM and work_mem to 256MB
- Utilize up to 6 workers
- Try setting random_page_cost to 1

Ram/Disk/Cores:  
- DBA.stackexchange.com
- Try setting random_page_cost to 1
- Utilize up to 6 workers
- max_parallel_workers

Immanuel Trummer. DB-BERT: a Database Tuning Tool that" Reads the Manual". SIGMOD, 2022.
Knob Tuner: Reduce Tuning Overhead

- Problems in RL
  - Significant Tuning Overhead
  - Require DBAs (select knobs, select knob ranges)

- Tuning hints from texts
  - Apply collected hints with reinforcement learning
  - $Parameter = Value \left[ * \text{System Property} \right] \left[ * \text{Constant} \right]$
Historical learned models are hard to migrate to new scenarios

- Characterize the common features of workloads
  - Reserved words in the SQLs

- Cluster similar historical workloads
  - Cluster with random forest and learn a base learner for each workload cluster

- Migrate to new workloads with meta learning
  - Given a workload, generate meta-learner based on the weighted sum of the base learners;
  - Fine-tune the meta-learner by running workload;
  - Recommend promising knobs with meta-learner.

Knob Tuner: Improve Tuning Adaptivity

Knob Tuner: Spark Tuning

- Spark tuning needs to consider knobs at different levels
  - Empirically initialize knob values at resource/APP/VM levels
    \[ q^x_2 = \frac{M_i + m_c}{\min(m_o^x, m_c^x)} \]
    - \( x \): Tested knob setting
    - \( M_i \): Code overhead value
    - \( m_c \): Required cache storage
    - \( m_o \): GC settings
  - Guided Gaussian Process
    1. Input both the execution statistics and initialized knob values;
    2. Use GP to fit existing tuning data.

Spark code involves complex semantics, and it is expensive to migrate tuning models across applications

- Sample candidate knob settings based on the data and code features;
- Conduct code instrumentation to enrich code tokens; then encode the code with CNN;
- Predict the tuning performance (NECS model) with encoded code, data, knob, DAG features;
- Generalize the NECS model to new applications using adaptive learning

Knob Tuner: Spark Tuning

- Sample candidate knob settings based on the data and code features;
- Conduct code instrumentation to enrich code tokens; then encode the code with CNN;
- Predict the tuning performance (NECS model) with encoded code, data, knob, DAG features;
- Generalize the NECS model to new applications using adaptive learning
<table>
<thead>
<tr>
<th>Summarization of Learned Knob Tuning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Quality</strong></td>
</tr>
<tr>
<td>Gaussian Process</td>
</tr>
<tr>
<td>Deep Learning</td>
</tr>
<tr>
<td>Reinforcement Learning</td>
</tr>
<tr>
<td>Manual Learning</td>
</tr>
<tr>
<td>Meta Learning</td>
</tr>
<tr>
<td>Spark Tuning (Contextual GP)</td>
</tr>
<tr>
<td>Spark Tuning (MLP+ Adaptive Learning)</td>
</tr>
</tbody>
</table>
Take-aways of Knob Tuning

• Gradient-based method reduces the tuning complexity by filtering out unimportant features. However, it heavily relies on training data, and requires other migration techniques to adapt to new scenarios.

• Deep learning method considers both query performance and resource utilization. And they can significantly reduce the tuning overhead.

• Reinforcement learning methods take long training time, e.g., hours, from scratch. However, it only takes minutes to tune the database after well trained and gains relatively good performance.

• Learning based methods may recommend bad settings when migrated to a new workload. Hence, it is vital to validate the tuning performance.

• Open problems:
  - One tuning model fits multiple databases
  - Natively integrate empirical knowledge
ML4DB: An Overview

- **Automatic Advisor**
  - Knob Tuner
  - Index/View Advisor
  - Partitioner/Scheduler

- **Learned Generator**
  - SQL Generator
  - Adaptive Benchmark

- **Intelligent Optimizer**
  - Query Rewriter
  - Plan Enumerator
  - Cost Estimator

- **Learned Designer**
  - Learned Index
  - Learned Data Layout

- **Autonomous Databases**
  - Paloton
  - SageDB
  - openGauss
  - …
Automatic Index Selection

- **Motivation:**
  - Indexes are essential for efficient execution
    - SELECT c_discount from bmsql_customer where c_w_id = 10;
    - CREATE INDEX on bmsql_customer(c_w_id);
  - Select from numerous indexable columns
    - Columns have different access frequencies, data distribution
  - Indexes may cause negative effects
    - Increase maintenance costs for update/delete operations
    - Performance Degradation \[ T_{(hash-index)} > T_{(full-table-scan)} \]
      \[ T_{(btree-index)} > T_{(full-table-scan)} \]
Automatic Index Selection

- Challenge
  - The index benefit is hard to evaluate
    - Multiple evaluation metrics (e.g., index benefit, space cost)
    - Cost estimation by the optimizer is inaccurate
  - Index selection is an NP-hard problem
    - The set of candidate index combinations is huge

- Index Update
Automatic Index Selection

- Two sub-problems
  - Index selection
    - Select indexes from a large number of possible combinations to maximize the benefit within a budget
  - Benefit estimation
    - Estimate the benefit of creating an index
      - $\text{Cost}(q) - \text{Cost}(q, \text{index})$, $q$ is a query
Traditional Index Advisor (Dynamic Programming)

- Model index selection as a knapsack problem
  - Candidate index scheme as item
  - Index size as item weight
  - Benefit of the item (optimizer) as value

- Use DP to select the highest-benefit indexes

Traditional Index Advisor (what-if estimation)

- Index selection for dynamic workloads

  - Divide a workload into epochs of queries
  
  - Profile candidate indexes for each new query

    - **Index Benefit**: average latency reduction for the queries within the same epoch (time-series)
    
    - Estimate the index benefit through a *what-if call* (assume: *similar queries have similar index benefits*)
    
    - Update the index set and statistics

  - Create indexes with highest index benefit at the end of each epoch

Index Advisor: Technique Development

- Performance (tuning quality, overhead, benefit estimation)
- Adaptivity (queries/codes, datasets, instances)

- Deep Neural Networks [sigmod, 2019]
- Reinforcement Learning [cikm, 2020] [icde, 2021]
- Monte Carlo Tree Search [icde, 2022]
Index Advisor: Optimize Index Estimation

- Critical to estimate index benefits by comparing execution costs of plans with/without created indexes
  - Prepare training data: Workloads + execution feedback from customers
  - Train the evaluation model: Predict the index benefits (1: performance gains; 0: no)
  - Solve Classification Problem: Use the model to create indexes with performance gains

(a) Example query plan.

(b) Feature channels for the plan.

Index Advisor: Optimize Index Selection

- Motivation: Index selection using reinforcement learning
- How to extract candidate indexes?
  - Extract candidate indexes from query predicates with empirical rules
- How to choose from candidate indexes?
  - Map into Markov Decision Process (MDP)
    - State: Info of current built indexes
    - Action: Choose an index to build
    - Reward: Cost reduction ratio after building the index

**Index Advisor: Support Index Update**

- **Motivation:** Indexes need to be updated based on workload changes
- **Core Idea:** Incrementally add/remove indexes with MCTS
  - Generate candidate indexes based on incoming queries
    - Merge similar queries into templates
    - Extract columns from template predicates
    - Combine columns into candidate indexes
  - Update existing indexes with candidate indexes
    - Initialize a policy tree
    - Explore more beneficial index sets on the tree

---

# Summarization of Automatic Index Advisor

<table>
<thead>
<tr>
<th>Optimization Targets</th>
<th>Overhead</th>
<th>Training Data</th>
<th>Adaptive</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Deep Learning</strong></td>
<td>Accurate Estimation</td>
<td>numerous data</td>
<td>much</td>
</tr>
<tr>
<td><strong>Reinforcement Learning</strong></td>
<td>High Performance</td>
<td>high computation costs</td>
<td>no prepared Data</td>
</tr>
<tr>
<td><strong>MCTS</strong></td>
<td>High Performance with index update</td>
<td>trade-off (costs, performance)</td>
<td>a few prepared data</td>
</tr>
</tbody>
</table>
Take-aways of Index Advisor

• RL-based index selection works takes much time for model training (cold start); while MCTS can gain similar performance and better interpretability (or regret bounds)

• Learned estimation models need to be trained periodically for data or workload update

• Open problems:
  - Benefit prediction for future workload
  - Cost for future updates
ML4DB: An Overview

- **Automatic Advisor**
  - Knob Tuner
  - Index/View Advisor
  - Partitioner/Scheduler

- **Learned Generator**
  - SQL Generator
  - Adaptive Benchmark

- **Intelligent Optimizer**
  - Query Rewriter
  - Plan Enumerator
  - Cost Estimator

- **Learned Designer**
  - Learned Index
  - Learned Data Layout

- **Autonomous Databases**

---

Autonomous Data Management System

- Paloton
- SageDB
- openGauss
- ...
Challenges in Heuristic MV Selection

- **Estimate** the utility of view candidates:
  \[ B = t_v - t_{v_{scan}} \]
  - Make no sense when MVs change the query plan drastically; Hard to estimate MV update cost

- **Select** views to materialize
  - Greedy/Genetic/other heuristics, Integer Linear Programming
  - Perform poor when the assumption is not satisfied (e.g. MV with higher cost has higher utility)

- **Update** views based on credits
  - A view’s credit is the sum of future utility and recreation cost
  - Cause delay to measure and estimate the credit value

---

MV Advisor: Technique Development

- Performance (tuning quality, overhead, benefit estimation)
- Adaptivity (queries/codes, datasets, instances)

View Advisor

Optimize MV Selection

- Reinforcement Learning
  [VLDB, 2020]
- Reinforcement Learning
  [ICDE, 2020]

Optimize MV Estimation

- Encoder-Decoder + RL
  [ICDE, 2021]
MV Advisor: Optimize MV Selection

- Numerous candidate MVs ➔ Greedily select MVs
  - ① Generate candidate MVs that balance between conflict queries (merge MVs)
  - ② Enumerate queries, MVs, and estimate the query costs with/without the selected MVs
  - ③ Verify the performance of selected MVs

- MV Update ➔ Predict MV usage frequency with a neural network

MV Advisor: Optimize MV Selection

- Extract candidate MVs from numerous common subqueries
  - Cluster equivalent queries and select the least overhead ones as the candidate;

- Select optimal candidate MVs with RL (under budget)
  - (1) Solve MV Selection with DQN model:
    - (2) Estimate the MV benefits with a deep neural network

$$Z = \{z_j\}$$: $$z_j$$ is a 0/1 variable indicating whether to materialize the subquery $$s_j$$

$$Y = \{y_{ij}\}$$: $$y_{ij}$$ is a 0/1 variable indicating whether to use the view $$v_{s_j}$$ for the query $$q_i$$

MV Advisor: Optimize MV Estimation

- Previous MV estimation cannot capture query-MV correlations
- Capture query-MV correlations with Encoder-Reducer Model
  - Generate query-MV pairs (queries can utilize multiple MVs)
  - Estimate the query-MV benefits with encoder-reducer model
    - Encoder-Reducer Model: Encode various number of queries and views with LSTM network, which captures query-MV correlations with attention
  - Select optimal MV combinations with reinforcement learning

Take-aways of MV Advisor

- Learned MV selection gains higher performance than heuristic methods.
- Learned MV selection works well for read workloads, and cannot efficiently support data update.
- Learned MV utility estimation is more accurate than traditional empirical methods.
- Learned MV utility estimation is also accurate for multiple-MV optimization.
**ML4DB: An Overview**

- **Automatic Advisor**
  - Knob Tuner
  - Index/View Advisor
  - Partitioner/Scheduler

- **Learned Generator**
  - SQL Generator
  - Adaptive Benchmark

- **Intelligent Optimizer**
  - Query Rewriter
  - Plan Enumerator
  - Cost Estimator

- **Learned Designer**
  - Learned Index
  - Learned Data Layout

- **Autonomous Databases**

---

**Autonomous Data Management System**

- Paloton
- SageDB
- openGauss

---

**Automatic Advisor**

- Self-Configuration
  - Knob Tuner

- Self-Optimization
  - Index Advisor
  - MV Advisor

- Self-Organization
  - Partition Advisor
  - Learned Scheduler

**Intelligent Optimizer**

- Query Rewriter
- Plan Enumerator
- Cost Estimator
- End-to-End Optimizer

**Learned Generator**

- Adaptive Benchmark
- SQL Generator

**Learned Designer**

- Learned Indexes
- Learned Layout
Automatic Database Partition

Motivation:

- A vital component in distributed database
  - Place partitions on different nodes to speedup queries
  - Trade-off between data balance & access frequency

- Database partition problem is NP-hard
  - Combinatorial problem: 61 TPC-H columns, 145 query relations, $2.3 \times 10^{18}$ candidate combinations
Traditional Database Partition Method

- Select partition keys from foreign-key relations
  - ↑ Data-locality: for each query, select partition keys with Maximum spanning tree
  - ↓ Data-redundancy: for all the queries, combine selected partition keys and take the optimal combination with DP

Traditional Database Partition Method

- Combine exact and heuristic algorithms to find good partition strategies
  - The partitioning performance is affected by the join queries
  - Build a weighted undirected graph, where the nodes are tables and edges are join relations.
  - Key Selection on the graph is a maximum weight matching problem
  - Provide both exact (i.e., each table uses a column, and turn into a integer programming problem) and heuristic (i.e., select the table columns whose edge weights are maximal) algorithms; and apply the appropriate algorithm under the time budget.

Challenges in Traditional Database Partition

- Rely on foreign-key relations to select partition keys
  - Other vital columns may be ignored, and cause sub-optimum

- Greedily select partition keys without considering the query costs and data distributions

- Cannot learn from historical partitioning data
DRL for Partition-Key Selection

- Typical OLAP Workloads contain complex and recursive queries
  - State Features: [ tables, query frequencies, foreign keys ]

- Select from numerous partition-key combinations and support new queries
  - (1) Use DQN to partition or replicate tables;
  - (2) Pretrain a cluster of RL models to support new queries

Takeaways of Database Partition

- Learned key-selection partition outperforms heuristic partition
- Learned key-selection partition has much higher partition latency for model training

Open Problems:
- Adaptive partition for relational databases
- Partition quality prediction
- Improve partition availability with replicates
ML4DB: An Overview

- **Automatic Advisor**
  - Knob Tuner
  - Index/View Advisor
  - Partitioner/Scheduler

- **Learned Generator**
  - SQL Generator
  - Adaptive Benchmark

- **Intelligent Optimizer**
  - Query Rewriter
  - Plan Enumerator
  - Cost Estimator

- **Learned Designer**
  - Learned Index
  - Learned Data Layout

- **Autonomous Databases**

---

![Diagram of ML4DB components]

**Automatic Advisor**
- Self-Configuration
- Self-Optimization
- Self-Organization

**Intelligent Optimizer**
- Query Rewriter
- Plan Enumerator
- Cost Estimator
- End-to-End Optimizer

**Learned Generator**
- Adaptive Benchmark
- SQL Generator

**Learned Designer**
- Learned Indexes
- Learned Layout

**Autonomous Data Management System**
- Paloton
- SageDB
- openGauss
- …
Automatic Query Scheduling

Motivation

☐ Effective Scheduling can Improve the Performance
  ➢ Minimize conflicts between read queries

☐ Concurrency Control is Challenging
  ➢ #-CPU Cores Increase

☐ Transaction Management Tasks
  ➢ Transaction Prediction
  ➢ Transaction Scheduling
Learned Query Scheduling

- Challenge: Keep the most important blocks cached
- Core Idea: Schedule queries to minimize disk access requests with RL

- Collect requested data blocks (buffer hit) from the buffer pool:
  - State Features: buffer pool size, data block requests;

- Schedule Queries to optimize global performance with Q-learning
Takeaways of Transaction Scheduling

- Learned scheduling can achieve higher performance, but takes intolerable long training time
- Learned scheduling requires detailed caching block information, which may not be available in some scenarios

Open Problems:
- Online workload Scheduling
- Query Trend Prediction
- Support transactions
ML4DB: An Overview

- **Automatic Advisor**
  - Knob Tuner
  - Index/View Advisor
  - Partitioner/Scheduler

- **Learned Generator**
  - SQL Generator
  - Adaptive Benchmark

- **Intelligent Optimizer**
  - Query Rewriter
  - Plan Enumerator
  - Cost Estimator

- **Learned Designer**
  - Learned Index
  - Learned Data Layout

- **Autonomous Databases**

---

**Autonomous Data Management System**

- Paloton
- SageDB
- openGauss
...
Automatic Query Generation

**Motivation**

- Companies generally do not release user queries (out of privacy issues);
- It is vital to generate *synthetical workloads* (in replace of real workloads), and release the synthetical workloads to the public to train the ML models
Automatic Query Generation

How to generate queries that meet **legality**, **diversity**, and **representative**?

**Definition**: Given a scheme and constraints (e.g., cost/cardinality ranges), we generate k SQL queries which can (i) legally execute in the database and (ii) meet the constraints.

**Example**: Generate 1000 TPC-H SQLs whose cardinality equals 1000.

**Challenges & Solutions**:

- It is hard to predict the performance of generated SQLs, i.e., whether they meet the constraints;
- It is hard to generate diverse SQLs;
- Grammar and syntax constraints need to be considered to generate legal queries;
- Construct a LSTM-based critic to predict the long-term benefits of any intermediate queries; utilize actor to explore new tokens;
- Construct a probabilistic model to ensure the diversity of generated queries;
- Construct a FSM to prune illegal tokens for current intermediate queries;

Lixi Zhang, Chengliang Chai, Xuanhe Zhou, Guoliang Li. LearnedSQLGen: Constraint-aware SQL Generation using Reinforcement Learning. SIGMOD 2022.
Automatic Query Generation

Query Legality

➢ SQL Grammar:
  • FSM

Advantage:
  ✓ Easy to add new grammar
  ✓ Customize SQL queries

Semantic Checks:

① Join Relation

② Type Checking
  • Aggregation: Aggregate Function
  • Predicate: WHERE caluse, HAVING clause

③ Operand Restriction
  • “people_name = China” X
ML4DB: An Overview

- **Automatic Advisor**
  - Knob Tuner
  - Index/View Advisor
  - Partitioner/Scheduler

- **Learned Generator**
  - SQL Generator
  - Adaptive Benchmark

- **Intelligent Optimizer**
  - Query Rewriter
  - Plan Enumerator
  - Cost Estimator

- **Learned Designer**
  - Learned Index
  - Learned Data Layout

- **Autonomous Databases**

---

**Automatic Advisor**
- Self-Configuration
- Knob Tuner

**Learned Generator**
- SQL Generator
- Adaptive Benchmark

**Intelligent Optimizer**
- MV Advisor
- Index Advisor
- Learned Schedule

**Learned Designer**
- Learned Indexes
- Learned Layout

**Autonomous Data Management System**
- Paloton
- SageDB
- openGauss
- …
Automatic Training Data Generation

Motivation

- Machine learning is widely adopted in database components
- It is challenging to obtain suitable datasets
  - Training data is rarely available in public
  - It is time-consuming to manually generate samples (e.g., over 6 months for 10,000 jobs with 1T data)
- It is hard to measure the dataset quality
  - The size of training data
  - The quality of extracted features
  - The availability of valuable ground-truth labels
Automatic Training Data Generation

- Challenges in existing workload generators (tpch, sqlsmith)
  - The workloads have low variance
  - Fail to label the workloads (e.g., cost, execution time)

- Core Idea: Label workloads with adaptive learning
  - Input a small query workload and sample data;
  - Create abstract plans (DAGs without actual physical operators) which follow the patterns in the input workload;
  - Instantiate the abstract plans based on the data distribution in the sample data;
  - Label the generated plans via active learning without executing all the plans

Takeaways of Learned Generator

- Generated queries or performance labels are useful to test database functions.

- Sometimes most real queries have similar structures and may not be effective as generated queries.

- Open Problems:
  - Semantic-aware query generation
  - Low overhead query generation
### ML4DB: An Overview

<table>
<thead>
<tr>
<th>Autonomous Advisor</th>
<th>Intelligent Optimizer</th>
<th>Learned Generator</th>
<th>Learned Designer</th>
<th>Autonomous Databases</th>
</tr>
</thead>
<tbody>
<tr>
<td>Knob Tuner</td>
<td>Query Rewriter</td>
<td>SQL Generator</td>
<td>Learned Indexes</td>
<td>Paloton</td>
</tr>
<tr>
<td>Index/View Advisor</td>
<td>Plan Enumerator</td>
<td>Adaptive Benchmark</td>
<td>Learned Layout</td>
<td>SageDB</td>
</tr>
<tr>
<td>Partitioner/Scheduler</td>
<td>Cost Estimator</td>
<td></td>
<td>Learned Generator</td>
<td>openGauss</td>
</tr>
</tbody>
</table>

**Learned Generator**
- SQL Generator
- Adaptive Benchmark

**Intelligent Optimizer**
- Query Rewriter
- Plan Enumerator
- Cost Estimator

**Learned Designer**
- Learned Index
- Learned Data Layout

**Autonomous Databases**
Learned Optimizer: An Overview

- **Query Rewriter:** Efficiently optimize query in logical Level
- **Plan Enumerator:** Powerfully optimize query in physical Level
- **Cost Estimator:** Accurately estimate the plan execution cost
Intelligent Optimizer: Technique Development

- Performance (latency, quality, cost accuracy)
- Adaptivity (queries, datasets)

**Query Rewriter**

- Optimize Predicate Pushdown
  - SMT+Binary Classifier

- Optimize Rewrite Orders
  - Monte Carlo Tree Search

**Join Enumerator**

- Optimize Join Orders
  - Reinforcement Learning

- Dynamic Plan Adjustment
  - Monte Carlo Tree Search

- Optimize Physical Operators
  - Learned Hinter

**Cost Estimator**

- Improve Estimation Quality
  - AR; SPNs
  - LSTMs;

- Support Multi-Table
  - Tree Ensembles
  - Joint AR

- Support String Data
  - Normalizing Flows

- Improve Adaptivity
  - Pre-Trained Models
ML4DB: An Overview

- **Automatic Advisor**
  - Knob Tuner
  - Index/View Advisor
  - Partitioner/Scheduler

- **Learned Generator**
  - SQL Generator
  - Adaptive Benchmark

- **Intelligent Optimizer**
  - Query Rewriter
  - Plan Enumerator
  - Cost Estimator

- **Learned Designer**
  - Learned Index
  - Learned Data Layout

- **Autonomous Databases**

---

**Automatic Advisor**

- Self-Configuration: Knob Tuner
- Self-Optimization: Index Advisor, MV Advisor
- Self-Organization: Partition Advisor, Learned Scheduler

**Intelligent Optimizer**

- Query Rewriter
- Plan Enumerator
- Cost Estimator
- End-to-End Optimizer

**Learned Generator**

- Adaptive Benchmark
- SQL Generator

**Learned Designer**

- Learned Indexes
- Learned Layout

**Autonomous Data Management System**

- Paloton
- SageDB
- openGauss

...
Many queries are poorly-written
- Slow operations (e.g., subqueries/joins, union/union all);
- Looks pretty to humans, but physically inefficient (e.g., take subqueries as temporary tables);

Existing methods are based on heuristic rules
- Top-down order may not be optimal (e.g., remove aggregates before pulling up subqueries)
- Available rules are limited

Trade-off in SQL Rewrite
- Best Performance: Enumerate for the best rewrite order
- Minimal Latency: SQL Rewrite requires low overhead (milliseconds)
Learning-based Query Rewrite

Challenge:

- Optimize existing rewrite rules, or even generate new rules
  - Equivalence verification

- Search rewrite space within time constraints
  - Rewrite within milliseconds;

- Estimate rewrite benefits by multiple factors
  - Case1: Reduced costs by selected rewrites
  - Case2: Future reduced costs by further rewriting the query
Query Rewrite: Technique Development

- Performance (latency, quality, cost accuracy)
- Adaptivity (queries, datasets)

Query Rewriter

- SMT+Binary Classifier
- Monte Carlo Tree Search
Query Rewrite: Optimize Predicate Pushdown

**Motivation:** Traditional predicate-pushdown is less powerful

- **Core Idea:** Predicate $\rightarrow$ Learn a binary classifier to synthesize valid and better predicates

- **Approach:** Generate TRUE/FALSE samples to train the binary classifier
  - Classifier: SVM model over the input columns
  - Each TRUE sample should be accepted by a valid predicate
  - Each FALSE sample should by rejected by an optimal predicate

Query Rewrite: Optimize Rewrite Orders

- The Strategies of applying rewrite rules

Given a slow query and a set of rewrite rules, apply the rewrite rules to the query so as to gain the equivalent one with the minimal cost.

Input SQL Query

```
SELECT
  MAX(DISTINCT L1.col1)
FROM
  lineitem L1
WHERE
  L1.col1 = ANY
  (SELECT
    MAX(C.col1)
  FROM
    customer C,
    lineitem L2
WHERE
    C.col1 = L2.col1
    AND ((C.col2<2
    AND C.col3<2)
    OR ((C.col2<2
    AND L2.col2>5))
GROUP BY
  C.col1);
```

Performance

Planning: 0.341 ms
Execution: > 20 min

Planning: 0.172 ms
Execution: 1.941 s
A slow query may have various rewrite of different benefits

- **(1) Initialize a Policy Tree Model**
  - **Node** $v_i$: any rewritten query
  - $C^\uparrow(v_i)$: previous cost reduction
  - $C^\downarrow(v_i)$: subsequent cost reduction

To select from numerous rewrite orders

- **(2) Policy Tree Search Algorithm**

\[
U(v_i) = (C^\uparrow(v_i) + C^\downarrow(v_i)) + \gamma \sqrt{\ln(F(v_0)) / F(v_i)}
\]

- **(3) Multiple Node Selection**
  - DP Algorithm

——

Xuanhe Zhou, Guoliang Li, Chengliang Chai. A Learned Query Rewrite System using Monte Carlo Tree Search. VLDB, 2022.
Take-aways of Query Rewrite

- Traditional query rewrite method is unaware of rewrite benefits, causing redundant or even negative rewrites.
- Search-based rewrite works better than traditional rewrite for complex queries.
- Rewrite benefit estimation improves the performance of simple search-based rewrite.

Open Problems

- Balance Rewrite Latency & Performance
- Adapt to different rule sets/datasets
- Design new rewrite rules
ML4DB: An Overview

- **Automatic Advisor**
  - Knob Tuner
  - Index/View Advisor
  - Partitioner/Scheduler

- **Learned Generator**
  - SQL Generator
  - Adaptive Benchmark

- **Intelligent Optimizer**
  - Query Rewriter
  - Plan Enumerator
  - Cost Estimator

- **Learned Designer**
  - Learned Index
  - Learned Data Layout

- **Autonomous Databases**

---

**Automatic Advisor**
- Self-Configuration: Knob Tuner
- Self-Optimization: Index Advisor, MV Advisor
- Self-Organization: Partition Advisor, Learned Scheduler

**Intelligent Optimizer**
- Query Rewriter
- Plan Enumerator
- Cost Estimator
- End-to-End Optimizer

**Learned Generator**
- Adaptive Benchmark
- SQL Generator

**Learned Designer**
- Learned Indexes
- Learned Layout

**Autonomous Data Management System**
- Paloton
- SageDB
- openGauss
...
Plan Enumerator

**Motivation:**

- Planning cost is hard to estimate
  - The plan space is huge
- Traditional optimizers have some limitations
  - DP gains high optimization performance, but causes great latency;
  - Random picking has poor optimization ability
- Finetuning existing optimizers can gain higher performance
Plan Enumerator: Technique Development

- Performance (latency, quality, cost accuracy)
- Adaptivity (queries, datasets)

Plan Enumerator

- Join Order Selection before Execution
  - Reinforcement Learning
- Join Order Selection on-the-fly
  - Monte Carlo Tree Search
- Physical Operator Selection
  - Learned Hinter
Join Order Selection: Optimize the Performance

Numerous candidate join orders to select before execution → Model it as RL

- Agent: optimizer
- Action: join
- Environment: Cost model, database
- Reward: Cost, Latency
- State: join order (Neo: encode query structures)

Join Order Selection: Adapt to Schema Changes

- Adaptively assemble the selection model, and adapt to schema changes (e.g., column, table)
  - Encode the operator relations and metadata features of the query;
  - Embed the query features with Tree-LSTM; (the tree structure can adapt to different tables/columns)
  - Decide join orders with RL model

Join Order Selection On-the-Fly

- Update execution orders of tuples on the fly and preserve the execution state

  - Tuples flows into the Eddy from input relations (e.g., R, S, T);
  - Eddy routes tuples to corresponding operators (the order is adaptively selected by the operator costs);
  - Eddy sends tuples to the output only when the tuples have been handled by all the operators.

Join Order Selection On-the-Fly

- Improve runtime plan adjustment performance
  - Join reorder with MCTS
    - Assume executing joins in “depth-first”;
    - Split time slices: 0.001s;
  - Approach
    - In each slice, reserve complete result tuples, and drop under-join intermediate tuples
    - Evaluate the join order benefits by (1) the table coverage and (2) result tuple ratio
    - Remove duplicate result tuples (based on their position vectors)
    - Judiciously select higher-benefit join orders with MCTS (the UCT function)

Physical Operator Selection (Plan Hinter)

- Physical operators can significantly affect the performance
  - E.g., improving performance by deactivating the loop join operator

- Model it as a Multi-armed Bandit Problem
  - Model each hint set $HSet_i$ as a query optimizer

$HSet_i : Q \rightarrow T$

- For a query $q$, it aims to generate optimal plan by selecting proper hint sets, which is dealt as a regret minimization problem:

$$R_q = \left( P(B(q)(q)) - \min_i P(HSet_i(q)) \right)^2$$

Take-aways of Plan Enumerator

- Learning based algorithm usually can find high efficient plans, especially for large queries with multiple joins.
- Offline learning methods use the sampled workload to pretrain the model. It will give good plans for the incoming queries.
- Online-learning methods do not need previous workload and can give good plans. But it needs the *customized engine* and is hard to be applied in existing databases.

**Open Problems**

- Raise the generalization performance of offline learning methods for unseen queries.
- Ensure the plan given by learned model is robust (explicable).
- Speed up the model training time, e.g. transferring previous knowledge.
- Make the model aware of the data update.
ML4DB: An Overview

- **Automatic Advisor**
  - Knob Tuner
  - Index/View Advisor
  - Partitioner/Scheduler

- **Learned Generator**
  - SQL Generator
  - Adaptive Benchmark

- **Intelligent Optimizer**
  - Query Rewriter
  - Plan Enumerator
  - Cost Estimator

- **Learned Designer**
  - Learned Index
  - Learned Data Layout

- **Autonomous Databases**

---

Automatic Advisor:
- Self-Configuration
  - Knob Tuner
- Self-Optimization
  - Index Advisor
  - MV Advisor
- Self-Organization
  - Partition Advisor
  - Learned Scheduler

Intelligent Optimizer:
- Query Rewriter
- Plan Enumerator
- Cost Estimator
- End-to-End Optimizer

Learned Generator:
- Adaptive Benchmark
- SQL Generator

Learned Designer:
- Learned Indexes
- Learned Layout

Autonomous Data Management System:
- Paloton
- SageDB
- openGauss
...
Cost Estimator: Technique Development

- Performance (latency, cost accuracy)
- Adaptivity (queries, datasets)

Cost Estimator

Improve Estimation Quality
- AR; SPNs
- LSTMs;

Support Multi-Table
- Tree Ensembles
- Joint AR

Support String Data
- Normalizing Flows

Improve Adaptivity
- Pre-Trained Models
Automatic Cardinality/Cost Estimation

Motivation:

- One of the most challenging problems in databases
  - Achilles Heel of modern query optimizers

- Traditional methods for cardinality estimation
  - Sampling (on base tables or joins)
  - Kernel-based Methods (Gaussian Model on Samples)
  - Histogram (on single column or multiple columns)

- Traditional cost models
  - Data sketching/data histogram based methods
  - Sampling based methods

Categories of Cardinality Estimation

(1) Supervised Query Methods
- Multi-set Convolutional network
- Tree-based ensemble

(2) Supervised Data Methods
- Gaussian kernel
- Uniform mixture model

(3) Unsupervised Data Methods
- Autoregressive
- Sum product network
Problem Definition

A regression problem: learn the mapping function between query \( Q \) and its actual cardinality.
Query-Driven: Deep Learning for Cardinality Estimation

- Motivation: Traditional estimator makes errors
- Core Idea: Use Multi-set Convolutional Neural Network to support join queries

- Linear Models for different part of SQL (table, joins, predicates)
- Pooling Varying-sized representations (avg pooling)
- Concatenate different parts

Query-Driven: Tree-Ensembling for Cardinality Estimation

- Traditional estimation methods assume column independency, and works bad for multi-dimension range queries
  - Any conjunctive query on columns $C$ can be represented as:
    $$(c_1 \leq l_{b_1} < c_2) \land (c_3 < u_{b_1} \leq c_4) \land (c_5 \leq u_{b_2} \leq c_6)$$
  - Tree-based ensembles: pass query encoding vectors (e.g., encode ‘>’, ‘5’ for ‘a>5’) through the traversal of multiple binary trees
Problem Definition

A density estimation problem: learn a joint data distribution of each data point
Supervised Data-Driven: Kernel-Density

☐ Support point queries on single tables

Training Phase

- Sample tuples from the table and initialize the **bandwidth** (distance from the true distribution) of the kernel density model.

    - Compute the optimal bandwidth via stochastic gradient descent (with labeled queries).

Inference Phase

- Sample some new data tuples

- Estimate the cardinality based on the kernel density model.

---

Supervised Data-Driven: Mixture Model

- **Support Range Queries**
  - **Training Phase**
    - Sample points within each history queries.
    - Generating **subgroups** for the points.
    - Learn the weights of all the Uniformity Mixture Models for range queries.
  - **Inference Phase**
    - Sample tuples within predicate ranges
    - Compute the cardinality by estimating the density of accessed ranges

Yongjoo Park, Shucheng Zhong, and Barzan Mozafari. Quicksel: Quick selectivity learning with mixture models. SIGMOD 2020
Unsupervised Data Methods for Cardinality Estimation

Problem Definition

A regression problem: learn a probability function for each data point

- Data Sampler
- Dataset
- tuples
- Data Model Training
- Model
- Well-Trained Data Model
- Card
- Optimizer
- Query
Unsupervised Data-Driven: Autoregressive Model

- Learn the joint probability distribution over columns for range queries
  - Use Autoregressive Model to fit the joint probability of different columns
  - Support range query with Progressive Sampling (sample from the estimated data distribution)

Unsupervised Data-Driven: Autoregressive for Multiple Tables

- Deep AR models can only handle single tables, and we need to learn from join correlations

  - Learn a single autoregressive model for all the tables (joined)
  - Join Sampler provides correct training data (sampled tuples from join) by using unbiased join counts (reduce sampling time and memory consumption)
  - To estimate for queries with a subset of tables, use fanout scaling to down sample from the joined table

Unsupervised Data-Driven: Sum-Product Network

- Learn a proper data partitioning strategy to accurately estimate the cardinality

  - Split data table into multiple segments and columns in each segment are near independent.

  - SPN: Product for different joins and Sum for different filters.

  - Adjust the data partitioning (e.g., column splits, column region splits) to learn the accurate SPN models.

<table>
<thead>
<tr>
<th>c.id</th>
<th>c.age</th>
<th>c.region</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>80</td>
<td>EU</td>
</tr>
<tr>
<td>2</td>
<td>70</td>
<td>EU</td>
</tr>
<tr>
<td>3</td>
<td>60</td>
<td>ASIA</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td>EU</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>998</td>
<td>20</td>
<td>ASIA</td>
</tr>
<tr>
<td>998</td>
<td>25</td>
<td>EU</td>
</tr>
<tr>
<td>999</td>
<td>30</td>
<td>ASIA</td>
</tr>
<tr>
<td>1000</td>
<td>70</td>
<td>ASIA</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>c.age</th>
<th>c.region</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>EU</td>
</tr>
<tr>
<td>70</td>
<td>EU</td>
</tr>
<tr>
<td>60</td>
<td>ASIA</td>
</tr>
<tr>
<td>20</td>
<td>EU</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>20</td>
<td>ASIA</td>
</tr>
<tr>
<td>25</td>
<td>EU</td>
</tr>
<tr>
<td>30</td>
<td>ASIA</td>
</tr>
<tr>
<td>70</td>
<td>ASIA</td>
</tr>
</tbody>
</table>

(a) Example Table

(b) Learning with Row/Column Clustering

(c) Resulting SPN

(d) Probability of European Customers younger than 30

### Summarization of Learned Cardinality Estimation

<table>
<thead>
<tr>
<th>Method</th>
<th>Quality</th>
<th>Training Overhead</th>
<th>Training Data</th>
<th>Adaptivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deep Neural Network</td>
<td>✓✓</td>
<td>low</td>
<td>many queries</td>
<td>✓✓</td>
</tr>
<tr>
<td>Tree-Ensemble</td>
<td>✓</td>
<td>low</td>
<td>many queries</td>
<td>✓</td>
</tr>
<tr>
<td>Gaussian Kernel</td>
<td>✓</td>
<td>relatively high</td>
<td>Data samples</td>
<td>✓✓</td>
</tr>
<tr>
<td>Mixture Model</td>
<td>✓</td>
<td>relatively high</td>
<td>Data samples</td>
<td>✓✓</td>
</tr>
<tr>
<td>Autoregressive</td>
<td>✓✓</td>
<td>high (join tables)</td>
<td>Data samples</td>
<td>✓✓✓</td>
</tr>
<tr>
<td>Sum-Product Network</td>
<td>✓✓</td>
<td>high</td>
<td>Data samples</td>
<td>✓✓✓</td>
</tr>
</tbody>
</table>
The Relations of Card/Cost Estimation

- **Task Target**
  - Cost estimation is to approximate the execution-time/resource consumption;

- **Correlations**
  - Cost estimation is based on cardinality

- **Estimation Difficulty**
  - Cost is harder to estimate than cardinality, which considers multiple factors (e.g., seq scan cost, cpu usage)
Traditional cost estimation uses estimated card, which is inaccurate without predicate encoding

- **The representation layer** learns an embedding of each subquery (global vector denotes the subquery, local vector denotes the root operator)
- **The estimation layer** outputs cardinality & cost simultaneously
Take-aways

- Data-driven methods are more effective for single tables.
- Query-driven methods are more effective for multiple tables.
- Query-driven methods are more efficient than Data-driven methods.
- Data-driven methods are more robust than Query-driven methods.
- Training queries are vital to Query-driven methods.
- Samples are crucial to Data-driven methods.
- Estimators based on neural network are more accurate than statistic-based estimators.
- Statistic-based query model is the most efficient.
ML4DB: An Overview

- **Automatic Advisor**
  - Knob Tuner
  - Index/View Advisor
  - Partitioner/Scheduler

- **Learned Generator**
  - SQL Generator
  - Adaptive Benchmark

- **Intelligent Optimizer**
  - Query Rewriter
  - Plan Enumerator
  - Cost Estimator

- **Learned Designer**
  - Learned Index
  - Learned Data Layout

- **Autonomous Databases**

---

**Automatic Advisor**

- Self-Configuration
- Knob Tuner

- Self-Optimization
  - Index Advisor
  - MV Advisor

- Self-Organization
  - Partition Advisor
  - Learned Scheduler

**Intelligent Optimizer**

- Query Rewriter
- Plan Enumerator

- Cost Estimator
- End-to-End Optimizer

**Learned Generator**

- Adaptive Benchmark
- SQL Generator

**Learned Designer**

- Learned Indexes
- Learned Layout

**Autonomous Data Management System**

- Paloton
- SageDB
- openGauss
- …
Learned Designer: Technique Development

- **Learned Designer**
  - Adaptivity (data update)
  - Complex Scenarios (e.g., multi-table, concurrency, persistent storage)

### Learned Index
- Learn from Data
  - Learn from Data
- Supprt Index Update
  - Alex
- Support Multi-D Tables
  - Flood
- Support Parallel R/W
  - XIndex
- Support PMem
  - APex
- Index Self-Design
  - Generic

### Learned Layout
- Single-Table Layout
  - qd-tree;
- Multi-Table Layout
  - MTO;
ML4DB: An Overview

- **Automatic Advisor**
  - Knob Tuner
  - Index/View Advisor
  - Partitioner/Scheduler

- **Learned Generator**
  - SQL Generator
  - Adaptive Benchmark

- **Intelligent Optimizer**
  - Query Rewriter
  - Plan Enumerator
  - Cost Estimator

- **Learned Designer**
  - Learned Index
  - Learned Data Layout

- **Autonomous Databases**

---

**Automatic Advisor**

- **Self-Configuration**
  - Knob Tuner

- **Self-Optimization**
  - Index Advisor
  - MV Advisor

- **Self-Organization**
  - Partition Advisor
  - Learned Scheduler

**Intelligent Optimizer**

- **Query Rewriter**
- **Plan Enumerator**

- **Cost Estimator**
- **End-to-End Optimizer**

**Learned Generator**

- Adaptive Benchmark
- SQL Generator

**Learned Designer**

- Learned Indexes
- Learned Layout

**Autonomous Data Management System**

- Paloton
- SageDB
- openGauss
  …
Learned Index: Technique Development

- Learned Designer
  - Adaptivity (data update)
  - Complex Scenarios (e.g., multi-table, concurrency, persistent storage)

Learned Index

- Learn from Data
  - RMI;
- Support Index Update
  - Alex
- Support Multi-D Tables
  - Flood
- Support Parallel R/W
  - XIndex
- Persistence Support
  - APex
- Index Self-Design
  - Generic
Learned Index

Motivation

☐ Indexes are essential for database system
  ➢ Indexes significantly speed up query process
  ➢ Take up unignorable memory in huge data-scale situation

☐ Limitations in Traditional Index
  ➢ Unaware of data features
  ➢ Trade-off between Space and Access Efficiency

☐ Advantages of Learned Index
  ➢ Space efficient, only store several parameters
  ➢ Further speed up data access if learning the data distribution well
1-Dimension Immutable Index

- Model cumulative distribution function (CDF)
  - Range index: predict approximate location as \( p = F(\text{Key}) \times N \), search precise location within error-bounded range
  - Point index: CDF as hash function to reduce conflict
  - Existence index: add a binary classification model to reduce effective keys of bloom filter

Kraska, T., Beutel, A., Chi, et al. The case for learned index structures. SIGMOD 2018
1-Dimension Mutable Index

- Support data inserts and index structure update
  - Use linear model in each node, exponential search in data node
  - Gapped array layout: accelerate inserts (gaps among arrays)
  - Predict the time to update indexes: predict costs of read queries and write queries, expand/split data node if cost deviate (given a threshold) to ensure high efficiency

1-Dimension Concurrent Index

- Handle concurrent updates
  - Two update cases
    - Update in-place,
    - Insert into buffer (delta index)
  - Approach: 2-phase compaction
    - Each group has a delta buffer for insertions
    - First merge a group’s data and buffer into array of pointers;
    - Then copy the value
    - Similar design for hash index which supports non-blocking resize operations

1-Dimension Auto-generated Index

- Use genetic algorithm to optimize indexes from origin data or indexes
  - **Population**: a set of physical indexes
  - **Mutations**: Adjust the data layout (column/row storage) and searching algorithm (binary/hash …) of a data node; merge/split nodes horizontally and vertically
  - **Fitness function**: optimize indexes for the runtime given a specific workload
Multi-d Immutable Index

- Support multi-D index with learned grid index
  - Sorted cells by 1st, 2nd, … column; within cell, points sorted by the last columns
  - Gradient descent to find the optimal number of regions for each column using sample of dataset and workload
  - Learn CDF of each column to predict region and location within cell

ML4DB: An Overview

- **Automatic Advisor**
  - Knob Tuner
  - Index/View Advisor
  - Partitioner/Scheduler

- **Learned Generator**
  - SQL Generator
  - Adaptive Benchmark

- **Intelligent Optimizer**
  - Query Rewriter
  - Plan Enumerator
  - Cost Estimator

- **Learned Designer**
  - Learned Index
  - Learned Data Layout

- **Autonomous Databases**
Motivation

- To reduce the #-data read from disk
  - Split data into data blocks (main-memory, secondary storage)
  - in-memory min-max index for each block

- It is challenging to partition data into data blocks
  - Numerous ways to assign records into blocks
  - Traditional: assign by arrival time; hash/range partition
Learned Data Layout (Qd-tree)

- **Qd-tree**: Learn the branch predicates
  - Root Node: The whole data space
  - Other Nodes: A part of the whole space

- **Approach**
  - **Constructor**: Construct a qd-tree based on the workload and dataset (greedy/RL)
  - **Query Router**: Route access requests based on the constructed qd-tree

---

Learned Data Layout: Consider Join Predicates

- **Motivation**
  - **Traditional**: either provide rare data skipping (zone maps), or require careful manual designs (Z-order)
  - **qd-tree**: only optimize single-table layouts

- **Qd-Trees for the whole datasets**
  - **Step#1**: Learn qd-tree for each table:
    - Extract simple predicates;
    - Create join-induced predicates;
    - Induce relevant tuples based on the simple&join-induced predicates
  - **Step#2**: Skip useless blocks based on the qd-trees

---

Take-aways of Learned Data Designer

- Learned index opens up a novel idea to replace traditional index, and show good performance in small datasets.
- Learned index uses machine learning technology, which provides probability of combining new hardware like NVM with database system in future.
- Though some research has already verified the benefit of learned index, performance in *industrial level data scale* still needs to be studied, especially in *updatable* and *multi-dimension* situation.

Open problems
- Distributed System: **Concurrency Control** algorithms for Learned Index
- Data scale and stability: Make Learned Index applicable to industrial database systems.
ML4DB: An Overview

● **Automatic Advisor**
  ● Knob Tuner
  ● Index/View Advisor
  ● Partitioner/Scheduler

● **Learned Generator**
  ● SQL Generator
  ● Adaptive Benchmark

● **Intelligent Optimizer**
  ● Query Rewriter
  ● Plan Enumerator
  ● Cost Estimator

● **Learned Designer**
  ● Learned Index
  ● Learned Data Layout

● **Autonomous Databases**
Autonomous Database Systems

Motivation

☐ Traditional Database Design is laborious
  ➢ Develop databases based on workload/data features
  ➢ Some general modules may not work well in all the cases

☐ Most AI4DB Works Focus on Single Modules
  ➢ Local optimum with high training overhead

☐ Commercial Practices of AI4DB Works
  ➢ Heavy ML models are hard to implement inside kernel
  ➢ A uniform training platform is required
Peloton

- Adapt optimization actions based on forecasted workloads
  - **Embedded Monitor**: Detect the event stream and extract incoming queries
  - **Workload Forecast**: Cluster queries and forecast for each cluster with RNN
  - **Optimization Actions**: Physical design, data layouts, and config tuning

Customize DB design via learning the data distribution

- Learn Data Distribution by Learned CDF

\[ M_{CDF} = F_{X_1, \ldots, X_m}(x_1, \ldots, x_m) = P(X_1 \leq x_1, \ldots, X_m \leq x_m) \]

- Design components via learned CDFs, and utilize cost-saving ML to replace traditional operators (e.g., seq scans)
  - Query optimization and execution
  - Data layout design
  - Advanced analytics

Implement learned components with model validation

- **Learned Optimizer**
  - Query Rewriter
  - Cost/Card Estimator
  - Plan Enumerator

- **Learned Advisor**
  - Self-Monitoring
  - Self-Diagnosis
  - Self-Configuration
  - Self-Optimization

- **Model Validation**

- **Data/Model Management**

---

Research Challenges
Future Works: Optimization Overhead

- Cold-Start Problems
  - Across datasets / instances / hardware / database types

- Lightweight in-kernel components
  - Efficient ML models; rare-data/compute-dependency;

- Online Optimization

- Workload execution overhead

- Model training overhead
Future Works: Adaptivity

- Significant data changes
  - Migration from small datasets to large datasets

- Completely new instances
  - New dataset, workload, and SLA requirements;

- Incremental DB module update
  - Learned knob tuner for hardware upgrade, learned optimizer for dynamic workloads.
Future Works: Complex Scenarios

- **Hybrid Workloads**
  - HTAP, dynamic streaming tasks

- **Distributed Databases**
  - Distributed plan optimization

- **Cloud Databases**
  - Dynamic environment, serverless optimization
Future Works: Small Training Data

- **Few Training Samples**
  - Few-shot learning

- **Knowledge + Data-driven**
  - Summarize (interpretable) experience from data

- **Pre-Trained Model**
  - Train a model for multiple scenarios
Future Works: SLA Improvement

- Optimize databases under noisy scenarios
  - Training Data Cleaning, Model Robust

- Optimize for extremely complex queries (e.g., nested queries)
  - Adaptive cardinality estimation → efficient query plan

- Optimize for OLTP queries
  - Multiple query optimization
Thanks