Machine Learning for Databases

Guoliang Li
Tsinghua

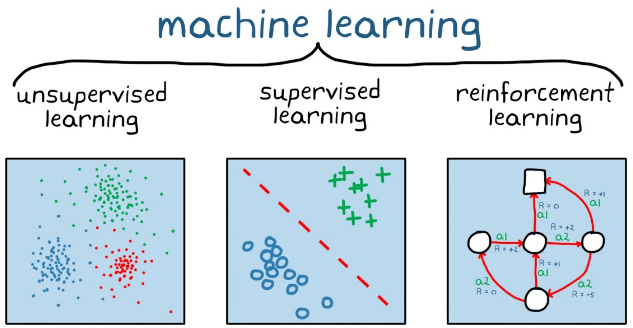
Xuanhe Zhou
Tsinghua

Lei Cao
MIT

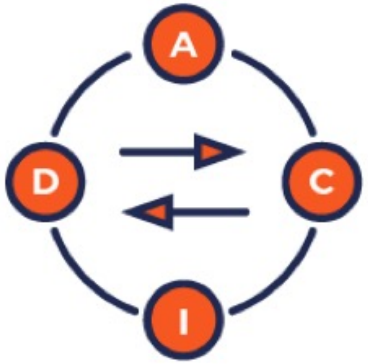
Motivation of ML4DB

- ☐ Machine Learning gets more practical. And empirical databases meet bottlenecks.

- ☐ Various ML models are available. It is challenging to select proper ML models.

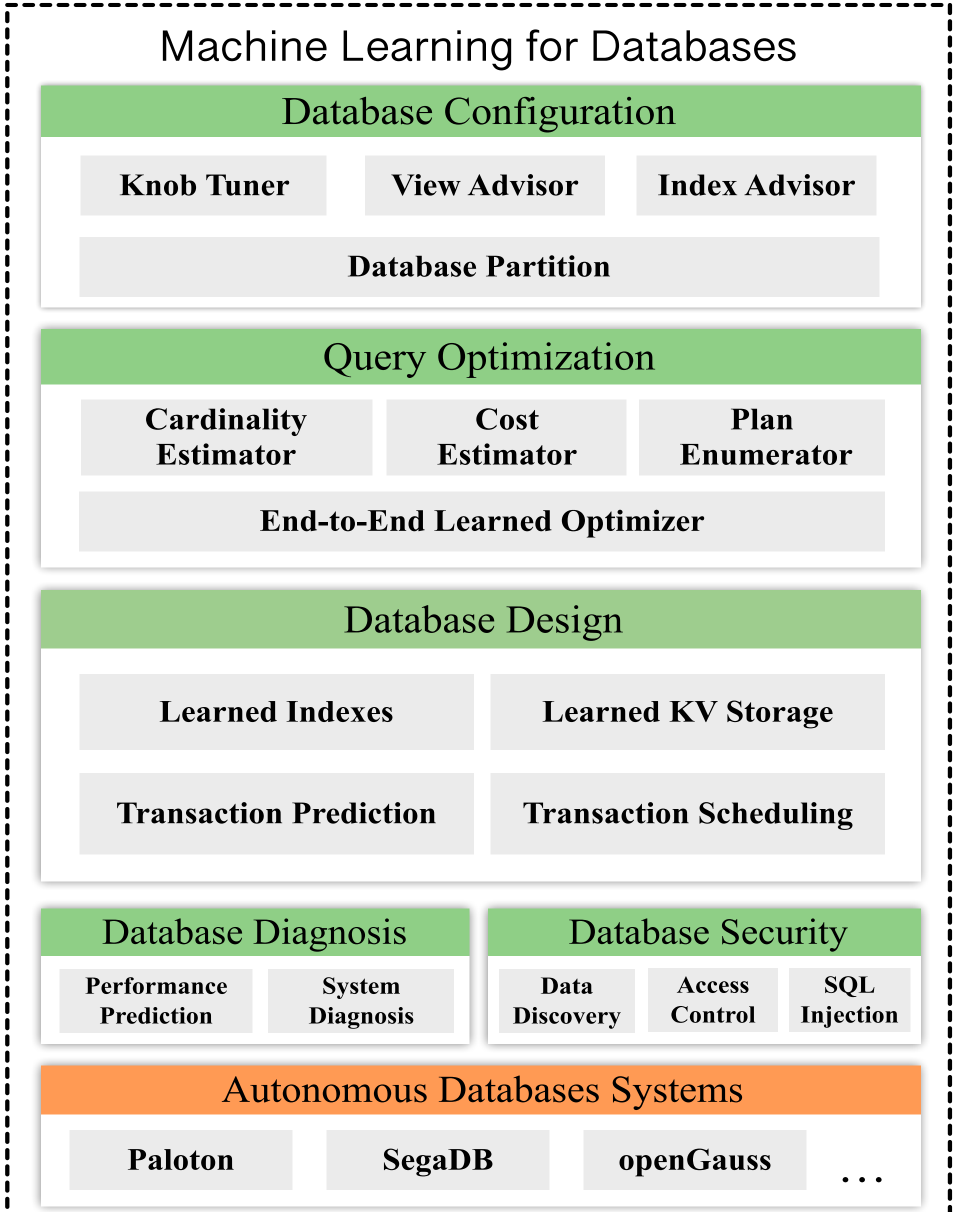


- ☐ Rigorous requirements for ML in databases, e.g., performance, robustness, interpretable.



Database Problems

- Database Core
  - Query Rewrite
  - Cost/Cardinality Estimation
  - Join Order Selection
- Database Configuration
  - Index/View Advisor
  - Knob Tuning
  - Workload Prediction



Overview of ML4DB

Problem	Description	DB Tasks
Offline NP Optimization	Optimize an NP-hard problem with large search space	**Knob Tuning**
		Index/View Selection
		Partition-key Selection
Online NP Optimization	Optimize an NP-hard problem with large search space (instant feedback)	**Query rewrite**
		Plan Enumeration
Regression	Determine the relationship between one dependent variable and a series of other independent variables	**Cost/Cardinality Estimation**
		Index/View Benefit Estimation
		Latency Estimation
Prediction	Forecast the likelihood of a particular outcome	**Trend Forecast**
		Workload Prediction & Scheduling

Overview of NP-hard Problems

	Method	Strategy	Search Space	Training Data
Offline Optimization (knob tuning, view selection, index selection, partition-key selection)	**Gradient based**	Local search	Small	Huge
	Deep Learning (DL)	Continuous space approximation	Large	Huge
	Meta Learning	Share common model weights	Various spaces	Huge
	Reinforcement Learning (RL)	Multi-step search	Large	--
Online Optimization (query rewrite, plan enumeration)	**MCTS(Monte Carlo Tree Search)+DL**	Multi-step search	Large	Huge
	Multi-armed	Multi-step search	Small	Small

Overview of Regression Problems

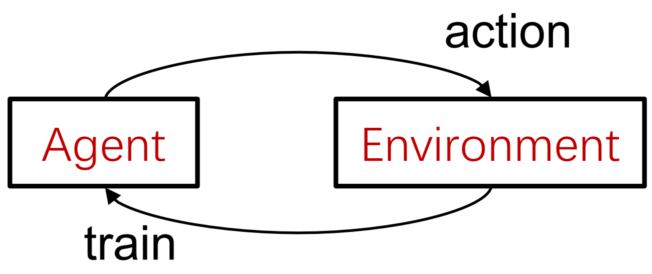
Method	Task	Feature Space	Feature Type	Training Data
Classic ML (e.g., tree-ensemble, gaussian, autoregressive)	cost estimation, view/index benefit estimation	Small	Continuous	Huge
Sum-Product Network	cost estimation	Small	Discrete	Small
Deep Learning	cost estimation, benefit estimation, latency estimation	Large	Continuous	Huge
Graph Embedding	benefit estimation, latency estimation	Large	Continuous	Huge

Overview of Prediction Problems

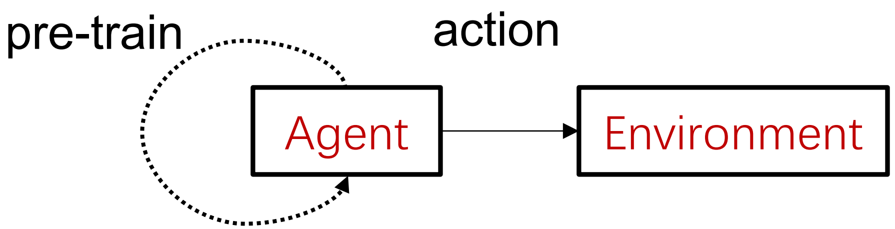
Method	Task	Target	Training Data
Clustering Algorithm	Trend Forecast	High accuracy	Huge
Reinforcement Learning	Workload Scheduling	High performance	--

Optimizing NP-hard Problems

- **Offline Optimization vs Online Optimization**
  - **Model Selection**
    - **E.g., Offline is model-free and online is model-based**
  - **Overhead**
    - **Online requires instant feedback and offline is insensitive**
  - **Performance**
    - **Generally offline has better performance**



Offline Optimization



Online Optimization

Offline Optimizing NP-hard Problems

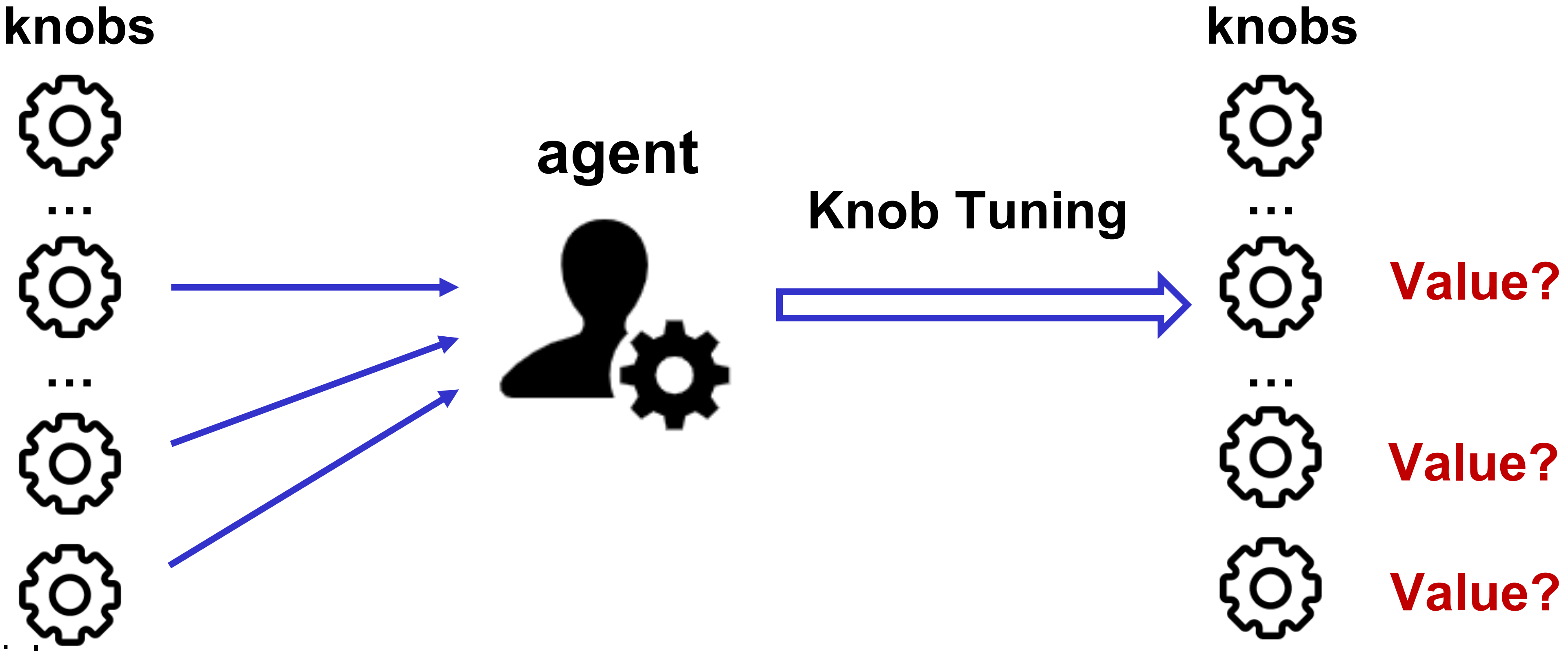
Offline Optimization for Knob Tuning

- **Motivation:**
  - DBMSs have different optimal knob settings, which significantly affect the query performance and resource utilization
  - DBMSs have numerous runtime metrics. Classic ML models cannot efficiently select knobs based on all the metrics
  - DBMSs have numerous system knobs to choose from, which makes it harder to find optimal knobs

Offline Optimization for Knob Tuning

□ Problem Definition

Consider a database with different workloads, the target is to find the optimal knob settings, i.e., satisfying SLA or resource requirements under several constraints (e.g., over 5% throughput improvement).



Offline Optimization for Knob Tuning

- **Existing Works**

  (1) Gradient-based Methods

  [Dana et al. SIGMOD 2017], [Kunjir et al. SIGMOD 2020]

  [Cereda et al. VLDB 2021]

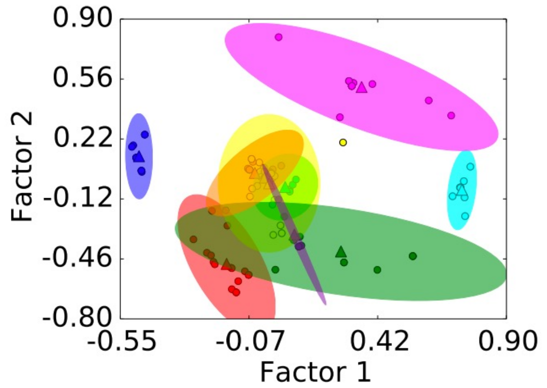
  (2) Deep Learning Method [Tan et al. VLDB 2019]

  (3) Meta Learning Method [Zhang et al. SIGMOD 2021]

  (4) Deep Reinforcement Learning Methods

  [Zhang et al. SIGMOD 2019], [Li et al. VLDB 2019]

(1) Gradient-based Method for Database Tuning



☐ Feature Extraction

- Characterize Workload Behaviors →
- Extract and Prune Runtime Metrics (e.g., #-page-read, #-page-write)
- Identify Important Knobs →
- Estimate the knob correlations by minimizing the square errors

☐ Model Construction

- Search Optimial Knobs based on the Runtime Metrics →
- Gaussian Process: (1) Approximate the knob-performance relations with numerous historical data; (2) Recommend knobs based on the most similar historical workload

Dana Van Aken, Andrew Pavlo, et al. Automatic Database Management System Tuning Trough Large-scale Machine Learning. In SIGMOD, 2017.

(1) Gradient-based Method for Spark Tuning

☐ Feature Extraction

- **Spark tuning considers knobs at different levels →**
- **Empirically estimate execution profiles at resource/APP/VM levels**

E.g., Memory Efficiency: $\boxed{q_2^{\mathbf{x}}} = \dfrac{M_i + m_c}{\min(m_o^{\mathbf{x}}, m_c^{\mathbf{x}})}$

x: Tested knob setting
M_i: Code overhead value
m_c: Required cach storage
m_o: GC settings

☐ Model Construction

- **Gaussian Process is black-box and requires much training data →**
- **Guided Gaussian Process: (1) Enhance tuning with the estimated execution profiles as inputs; (2) Use GP to fit existing tuning data**

Mayuresh Kunjir, Shivnath Babu. Black or White? How to Develop an AutoTuner for Memory-based Analytics. SIGMOD 2020.

(2) Deep Learning for Buffer Tuning

☐ Feature Extraction

- Buffer Pool is critical resource in cloud databases →
- Only tune the buffer_pool_size knob for higher resource utilization

- Many Metrics affect the response time besides buffer_pool_size →
- Database metrics: logical-read, io-read, QPS, CPU usage, historical RT

☐ Model Construction

- Tune buffer size that maximizes resource utilization under SLAs →

- (1) Select buffer sizes for databases with similar database metrics; (2) Design a neural network to estimate SLA as tuning feedback

J. Tan, T. Zhang, F. Li, et al. iBTune: Individualized Buffer Tuning for Large-Scale Cloud Databases. VLDB 2019.

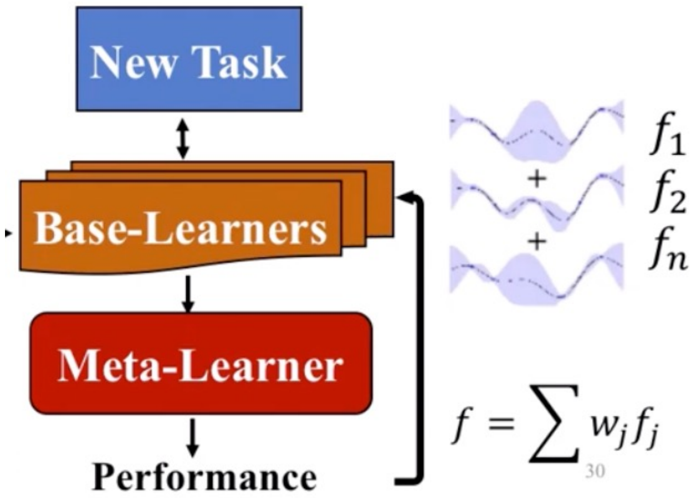
(3) Meta Learning for Knob Tuning

☐ Feature Extraction

- Characterize the common features of workloads →
- Meta-Features: Reserved words in the SQLs
- Cluster historical workloads (random forest) and learn a base learner (meta-features as inputs) for each workload cluster

☐ Model Construction

- Boost tuning for new instances →



- (1) Learn meta-learner based on the weighted sum of the base learners;
(2) Fine-tune the meta-learner on the new instance;
(3) Recommend promising knobs

Xinyi Zhang, Hong Wu, and et al. ResTune: Resource Oriented Tuning Boosted by Meta-Learning for Cloud Databases. SIGMOD, 2021.

(4) Reinforcement-learning for Knob Tuning

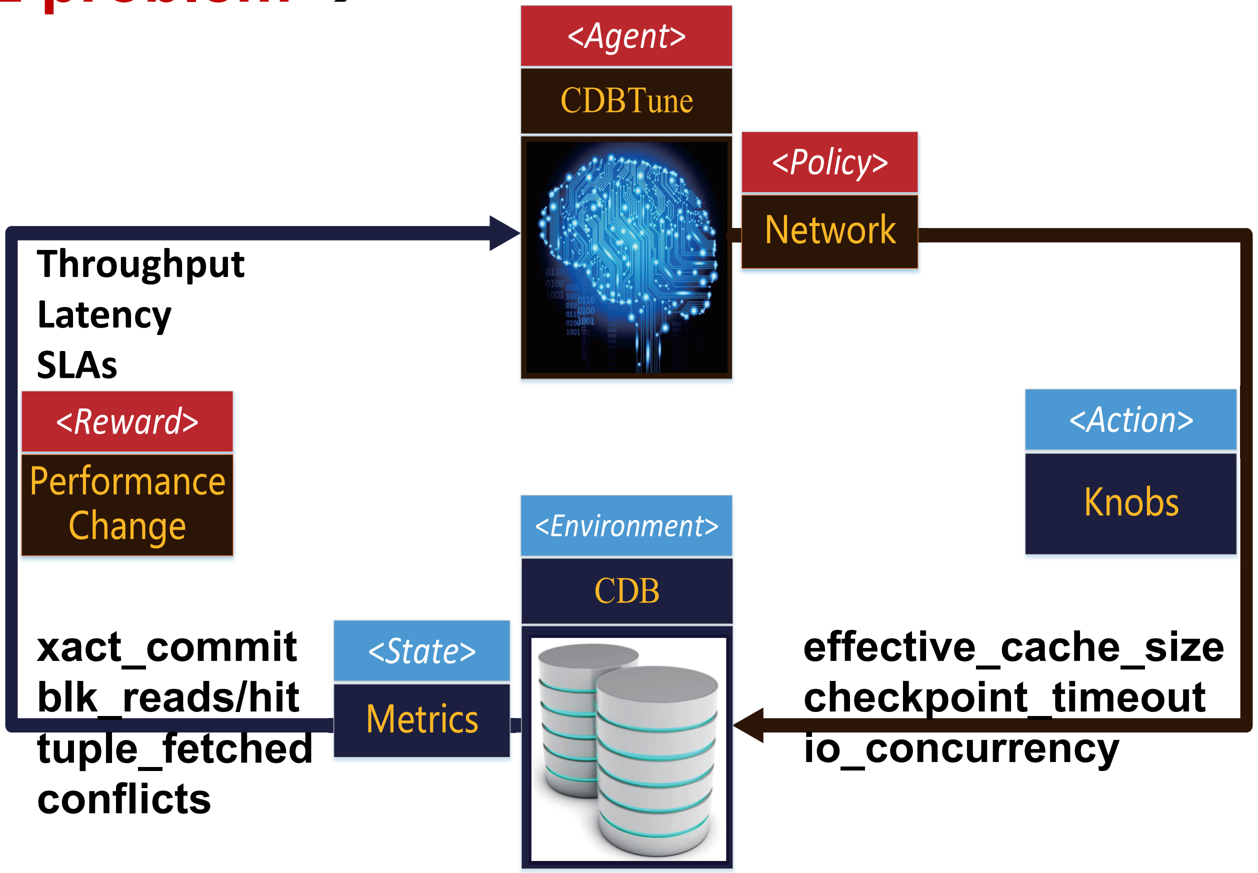
- **Challenge:**
  - **Basic ML models tune a small part of knobs. It is challenging to support more knobs with complex correlations.**
  - **High-quality training samples are hard to obtain, especially in real-world scenarios**

(4) Reinforcement-learning for Knob Tuning

☐ Feature Extraction

- **Map knob tuning into an RL problem →**

RL	CDBTune
Agent	The tuning system
Environment	DB instance
State	Internal metrics
Reward	Performance change
Action	Knob configuration
Policy	Deep neural network



Ji Zhang, Yu Liu, Ke Zhou, Guoliang Li et al. An End-to-End Automatic Cloud Database Tuning System Using Deep Reinforcement Learning. SIGMOD 2019.

(4) Reinforcement-learning for Knob Tuning

☐ Model Construction

- **Many Continuous system metrics and knobs →**
  - **Value-based method (DQN)** **Discrete Action ✕**
    - Replace the Q-table with a neural network
    - **Input:** state metrics; **Output:** Q-values for all the actions
  - **Policy-based method (DDPG)** **Continuous State/Action ✓**
    - (**actor**) Parameterized policy function: $a_t = \mu(s_t|\theta^{\mu})$
    - (**critic**) Score specific action and state: $Q(s_t, a_t|\theta^{Q})$

Ji Zhang, Yu Liu, Ke Zhou, Guoliang Li et al. An End-to-End Automatic Cloud Database Tuning System Using Deep Reinforcement Learning. SIGMOD 2019.

Summarization of Knob Tuning

	Optimizastion Target	Loss/Reward Function	Training Data	Adaptive (workload)
Gradient-based [SIGMOD 2017] [SIGMOD 2020]	Performance	The weighting coefficients are equal to the mean estimates of the target values	High	--
Deep Learning [VLDB 2019]	Resource Utilization	$L : (e, \lambda) \rightarrow \begin{cases} l(e)I(e \geq 0) \\ \lambda l(e)I(e < 0) \end{cases}$ *I(e)*: mean square error; *λ*: Control the impact of overestimating	High	--
Meta Learning [SIGMOD 2021]	Resource Utilization	The loss is the number of misranked pairs the model predicted	High	✓
Deep Reinforcement Learning [SIGMOD 2019] [VLDB 2019]	Performance	$r = \begin{cases} ((1+\Delta_{t\rightarrow 0})^2 - 1)\lvert 1+\Delta_{t\rightarrow t-1}\rvert, \Delta_{t\rightarrow 0} > 0 \\ -((1-\Delta_{t\rightarrow 0})^2 - 1)\lvert 1-\Delta_{t\rightarrow t-1}\rvert, \Delta_{t\rightarrow 0} \leqslant 0 \end{cases}$ *r: the reward;* $\Delta T_{t\rightarrow t-1}$ / $\Delta T_{t\rightarrow 0}$: the performance change	Low (RL does not need prepared data)	✓ (Pre-train a query model)

Take-aways of Knob Tuning

- **Gradient-based method reduces the tuning complexity by filtering out unimporant features.** However, different scenarios may have different key features, which makes it hard to train a generalizable tuning model.
- **Deep learning method considers both query performance and resource utilization.** And they work better for resource-sensitive scenarios.
- **Reinforcement learning methods take longest 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:**
  - Predict workload execution performance for knob tuning
  - One tuning model fits multiple databases
  - Utilize empirical knowledge

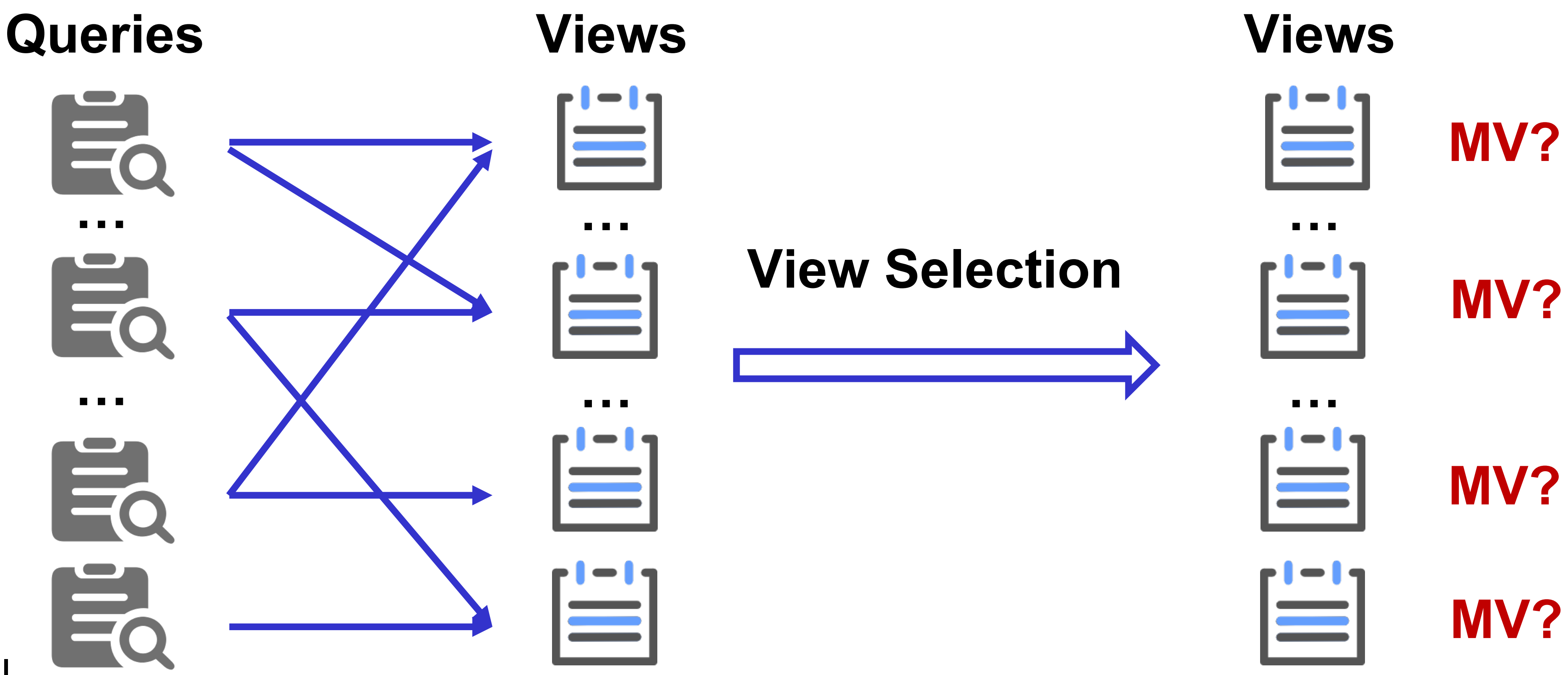
Offline Optimization for View Selection

- **Motivation:**
  - **Materialized Views (MVs) optimize queries**
    - Share common subqueries
  - **Space-for-time trade-off principle**
    - Materialize hot data (MVs) within limited space
    - How to estimate the MV utilities
  - **The number of potential MVs grows exponentially**
    - Greedy/Genetic/other-heuristics work bad

Offline Optimization for View Selection

☐ Problem Definition

Given a workload *Q* and a space budget, select optimal subqueries to materialize (MVs), including (i) candidate MV generation; (ii) MV Selection.



Offline Optimization for View Selection

- Two sub-problems
  - Benefit estimation
    - Estimate the benefit of materializing a view
      - Cost(q) - Cost(q,v), q is a query and v is a view
  - View selection
    - Select views from a large number of possible combinations to maximize the benefit within a budget

Offline Optimization for View Selection

◻ Existing Works

(1) Hybrid View Selection

[Ahmed et al. VLDB 2020]

(2) DRL for View Selection

[Yuan et al. ICDE 2020]

(3) Encoder-Decoder for View Benefit Estimation

[Han et al. ICDE 2021]

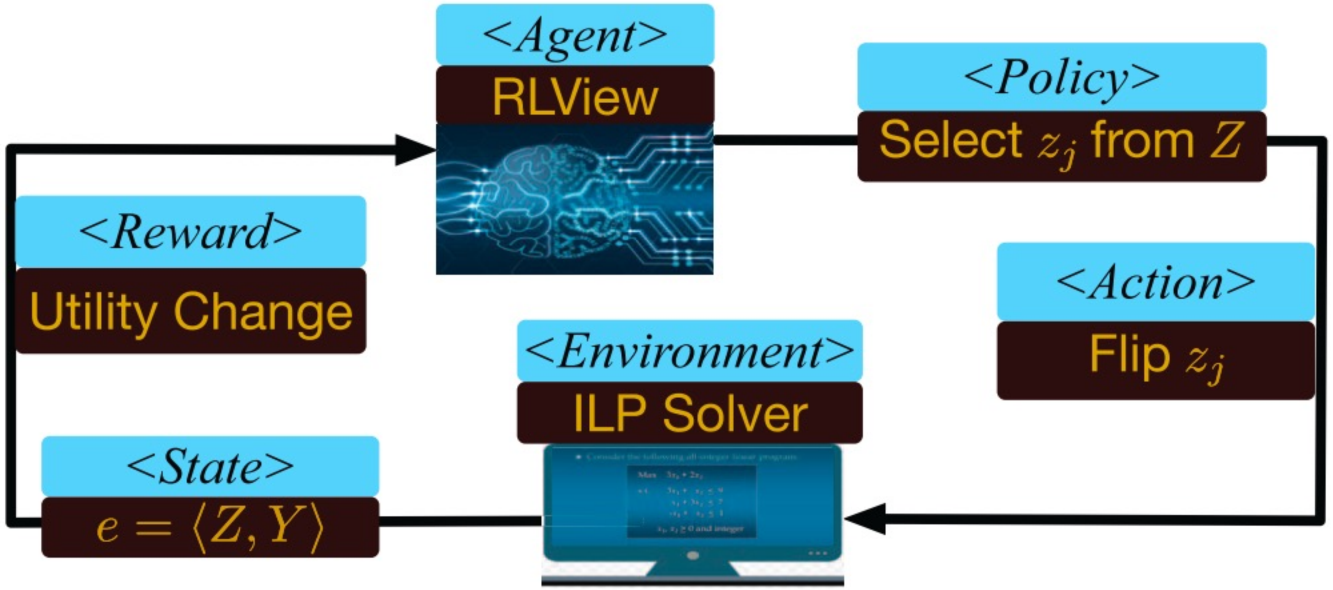
DRL for View Selection

◻ Feature Extraction

- Numerous common subqueries among workload queries →
- Cluster equivalent queries and select the least overhead ones as the candidate;

◻ Model Construction

- Numerous combinations of candidate subqueries →
- (1) Solve MV Selection with Q-learning: (2) Estimate the MV utility 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

H. Yuan, G. Li, L. Feng, and et al. Automatic view generation with deep learning and reinforcement learning. In ICDE, 2020.

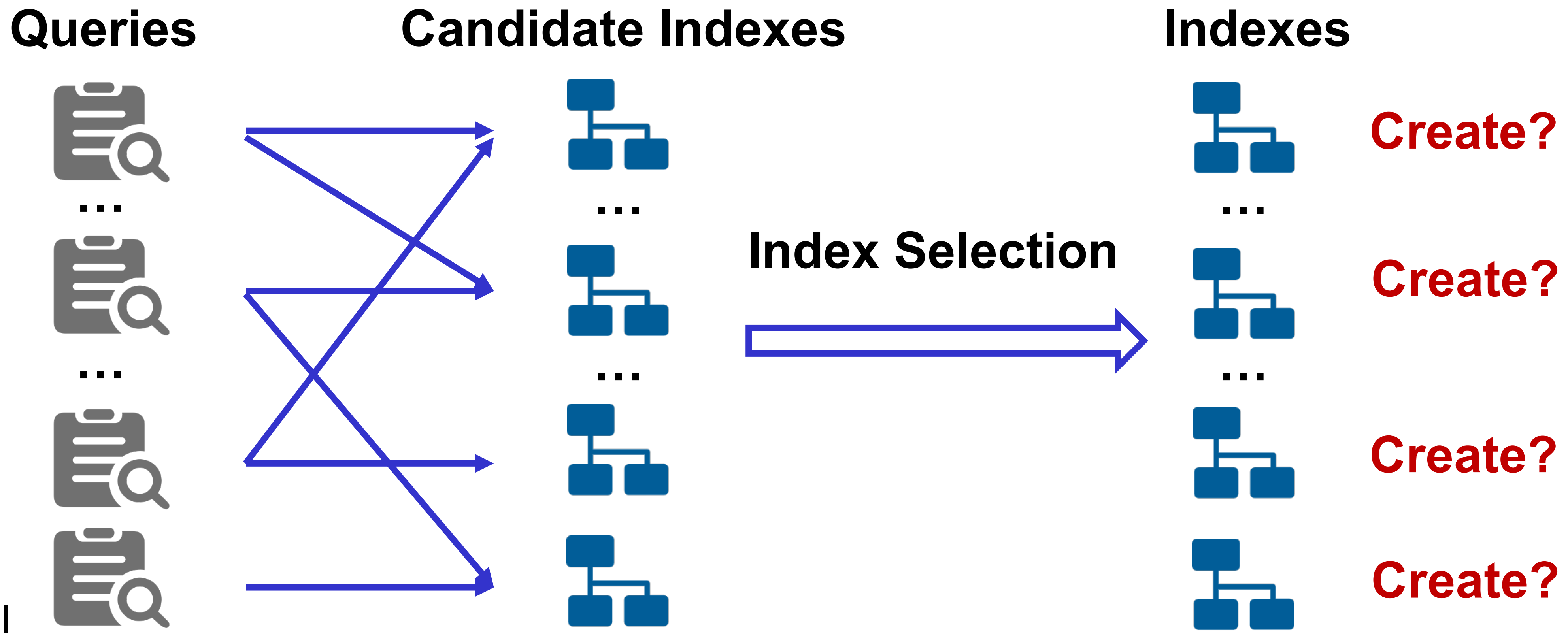
Offline Optimization for Index Selection

- **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
  - **Redundant indexes may cause negative effects**
    - Increase maintenance costs for update/delete operations

Offline Optimization for Index Selection

□ Problem Definition

Given a workload and constraints (e.g., disk limit), find an index set, such that the performance is optimal with the constraint.



Offline Optimization for Index Selection

- Two sub-problems
  - Benefit estimation
    - Estimate the benefit of creating an index
      - Cost(q) - Cost(q, index), q is a query
  - Index selection
    - Select indexes from a large number of possible combinations to maximize the benefit within a budget

DRL for Index Selection

☐ Feature Extraction

- ➢ Map candidate indexes with empirical rules

Rule 1: Construct all single-attribute indexes by using the attributes in J, EQ, RANGE.
Rule 2: When the attributes in O come from the same table, generate the index by using all attributes in O.
Rule 3: If table *a* joins table *b* with multiple attributes, construct indexes by using all join attributes.

☐ Model Construction

- ➢ Map the index selection problem into a reinforcement learning model

State: Information of current built indexes

large space

Action: Choose an index to build

discrete space

DQN Model

Reward: Cost reduction ratio after building the index

H. Lan, Z. Bao, Y. Peng. An Index Advisor Using Deep Reinforcement Learning. CIKM, 2020.

Take-aways of View/Index Selection

- **Learned selection is more robust than heuristics**
- **Learned selection works well in online service, but takes much time for model training (cold start)**
- **Query encoding models need to be trained periodically when data update**
- **Open problems:**
  - Benefit prediction for future workload
  - Cost for future updates
  - Support updates/eviction

Offline Optimization for 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 x 10^{18} candidate combinations

Offline Optimization for Database Partition

☐ Problem Definition

Raw Tables

customer	
c_custkey	**c_nationkey**
1	15
2	13
3	1
4	4

lineitem	
l_orderkey	**l_suppkey**
197	15
69	13
161	1
64	15

Distribute by HASH (**c_custkey**)

Distribute by HASH (**l_orderkey**)

Node 0

customer_ p_00	
2	13
4	4

lineitem_p_00	
64	15

Node 1

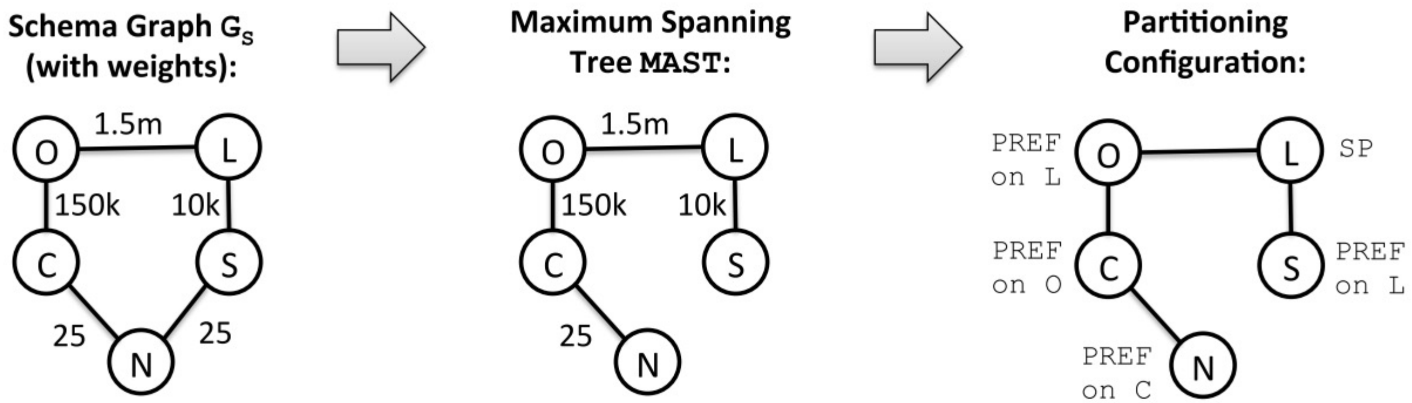
customer_ p_01	
1	15
3	1

lineitem _p_01	
197	15
69	13
161	1

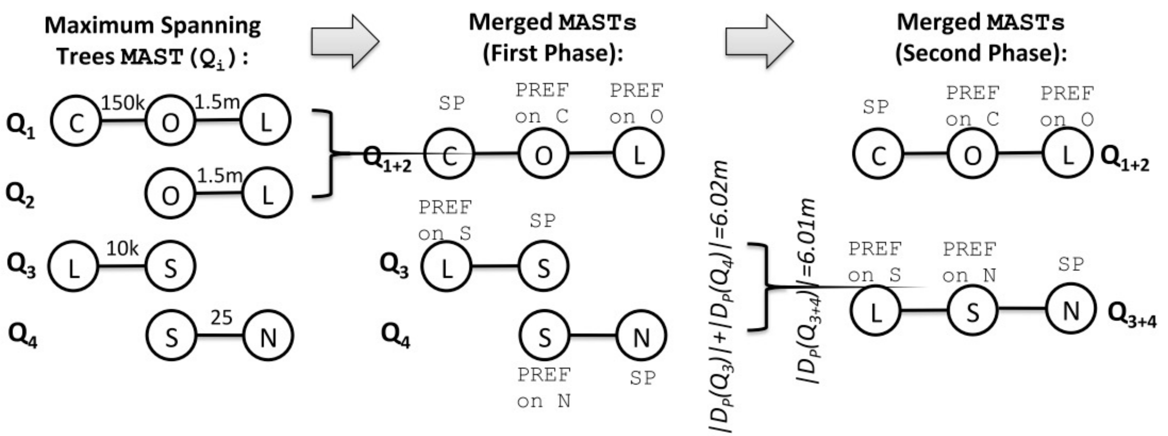
SELECT * FROM customer,lineitem WHERE **c_nationkey** = **l_suppkey** and **c_nationkey** < 4;

Heuristic Method for Offline Co-Partition

- **Select from foreign-key relations between tables →**
  - **↑ Data-locality: Maximum spanning tree for each query**



  - **↓ Data-redundancy: Enumerate selected partitions with DP**



Erfan Zamanian, Carsten Binnig, Abdallah Salama. Locality-aware Partitioning in Parallel Database Systems. SIGMOD 2015.

Hybrid Methods for Partition-Key Selection

- 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.

P. Parchas, et al. Fast and Effective Distribution-Key Recommendation for Amazon Redshift. Proc. VLDB Endow, 2020.

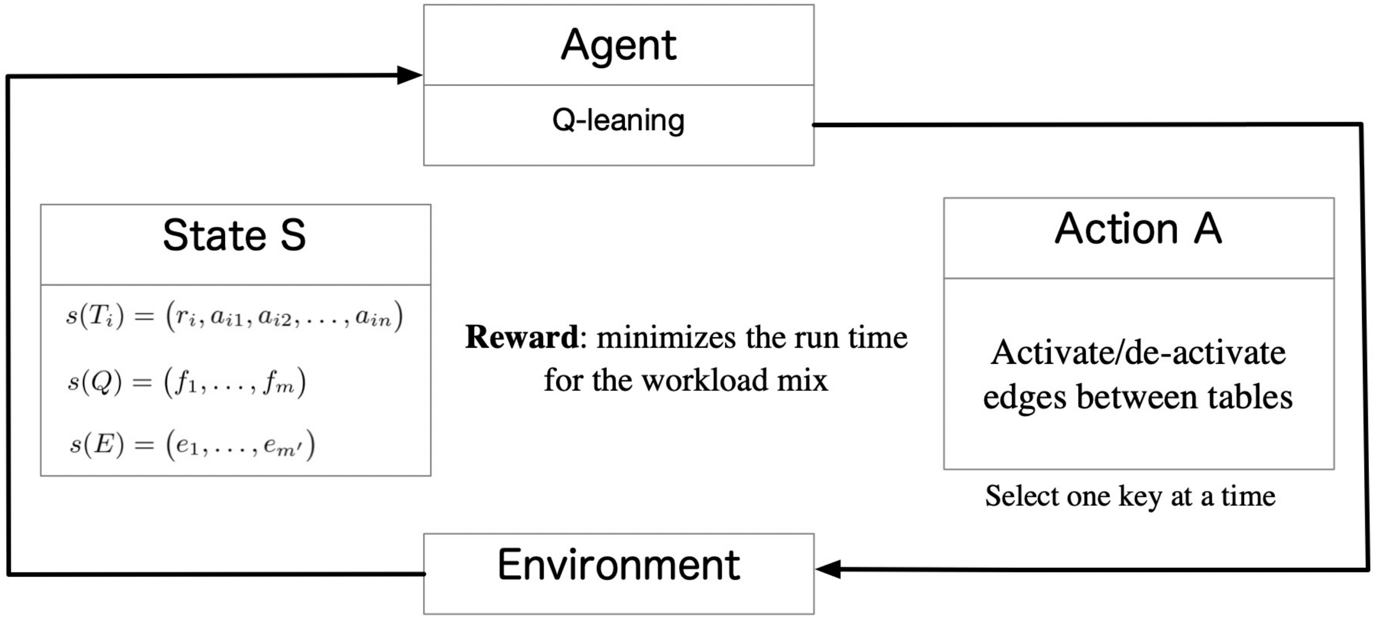
DRL for Offline Partition-Key Selection

Feature Extraction

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

Model Construction

- To select from enumerous partition-key combinations and support new queries →
- (1) Use DQN to partition or replicate tables; (2) Pre-train a cluster of RL models.



Benjamin Hilprecht, Carsten Binnig, Uwe Röhm. Towards learning a partitioning advisor with deep reinforcement learning. SIGMOD 2019.

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

Online Optimizing NP-hard Problems

Online Optimization for Query Rewrite

☐ Motivation:

☐ Many queries are poorly-written

- Terrible 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)
- No evaluation of different rewrite orders

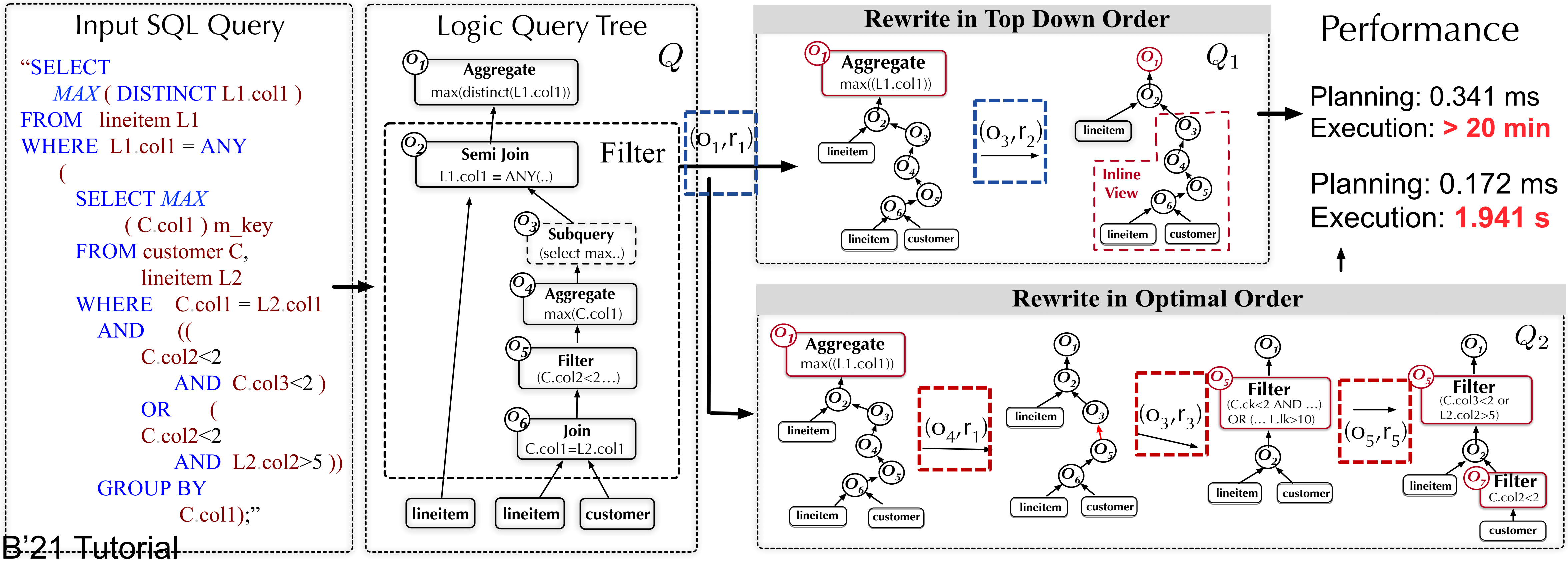
☐ Trade-off in SQL Rewrite

- **Best Performance:** Enumerate for the best rewrite order
- **Minimal Latency:** SQL Rewrite requires low overhead (milliseconds)

Online Optimization for Query Rewrite

☐ Problem Definition

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.



Online Optimization for Query Rewrite

☐ Challenge:

- **The rewrite space is large**
  - Exponential to the number of rewrite rules
- **Search rewrite space within time constraints**
  - Rewrite within milliseconds;
- **Estimate rewrite benefits by multiple factors**
  - Reduced costs after rewriting
  - Future cost reduction if further rewriting the query

MCTS for Query Rewrite

☐ Feature Extraction

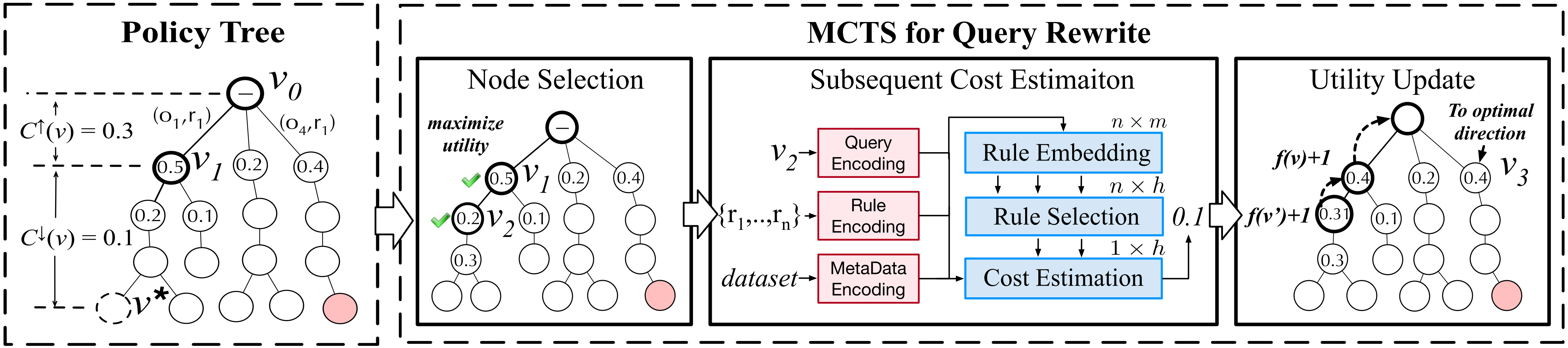
- **A slow query may have various rewrite of different benefits →**
- **Policy Tree Model**
  - **Node v_i**: any rewritten query
  - **$C^{\uparrow}(v_i)$**: previous cost reduction
  - **$C^{\downarrow}(v_i)$**: subsequent cost reduction

☐ Model Construction

- **To select from enumerous rewrite orders →**
- **(1) Policy Tree Search Algorithm**

$$\mathcal{U}(v_i) = (C^{\uparrow}(v_i) + C^{\downarrow}(v_i)) + \gamma\sqrt{\frac{ln(\mathcal{F}(v_0))}{\mathcal{F}(v_i)}}$$

- **(2) Multiple Node Selection**



A Learned Query Rewrite System using Monte Carlo Tree Search.

Take-aways of Query Rewrite

- **Traditional query rewrite method is unaware of cost, 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

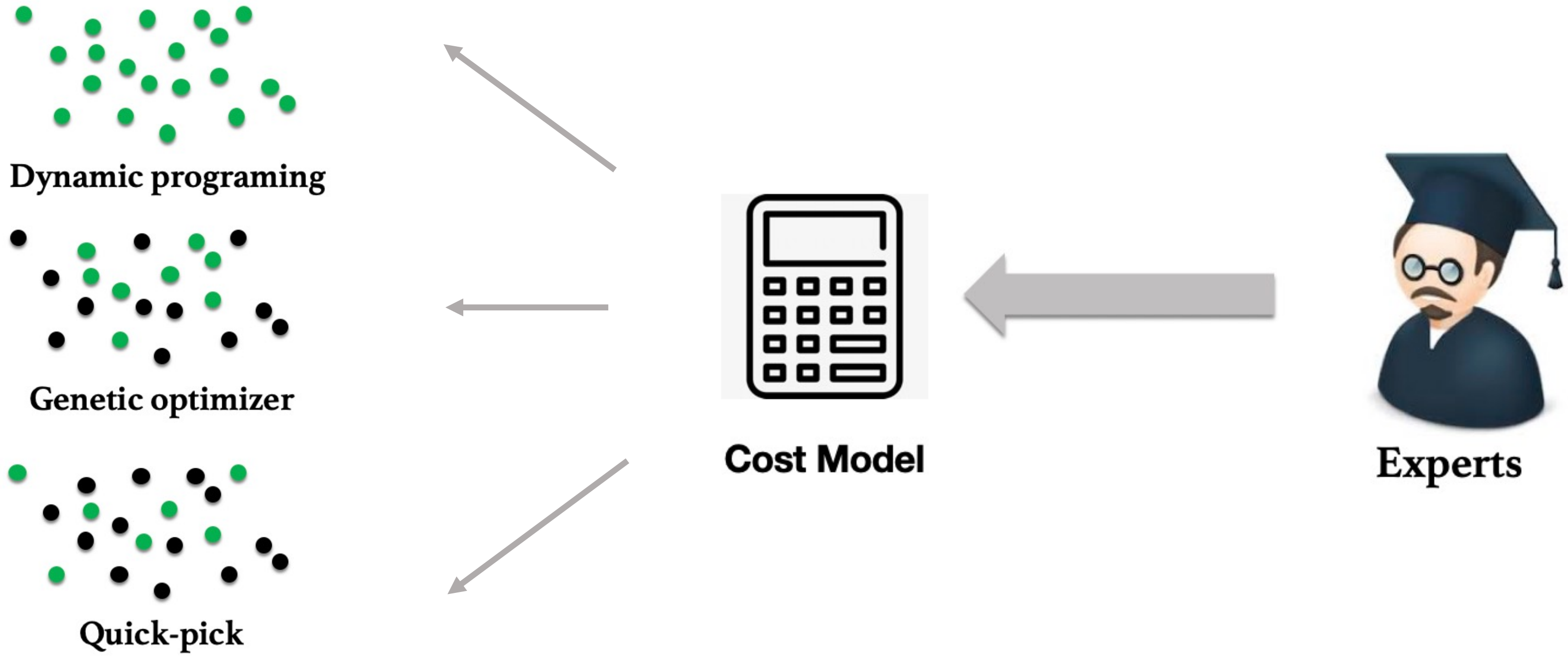
Plan Enumerator

- **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
  - **Steer existing optimiers can gain higher performance**
    - Hint join orders; Hint operator types

Join Order Enumerator

☐ Problem Definition

Given a SQL query, select the "cheapest" join ordering (according to the cost model).



Join Order Enumerator

☐ Method Classification

☐ Offline Optimization Methods.

- Characteristic: given Workload，RL based.
- **Key idea:** Use existing workload to train a learned optimizer, which predicts the plan for future queries.

☐ Online Optimization Methods.

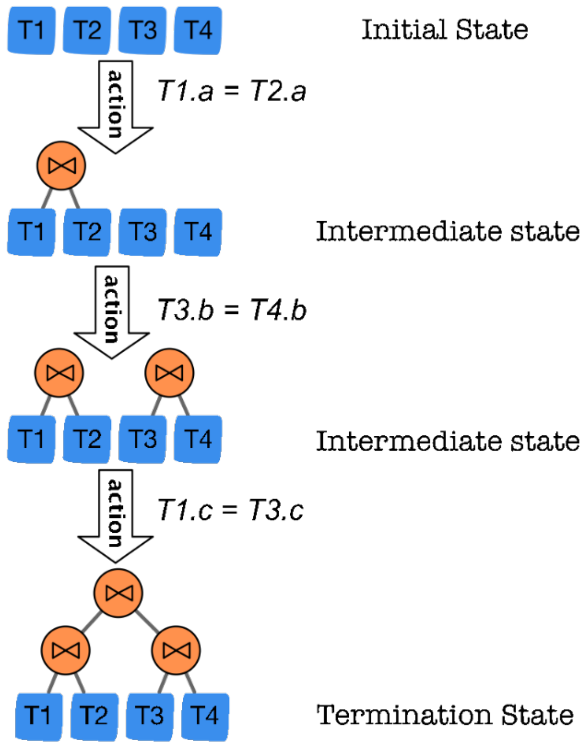
- Characteristic: No workload, but rely on customized Database.
- **Key idea:** The plan of a query can be changed during execution. The query can switch to another better plan. It learns when the database executes the query.

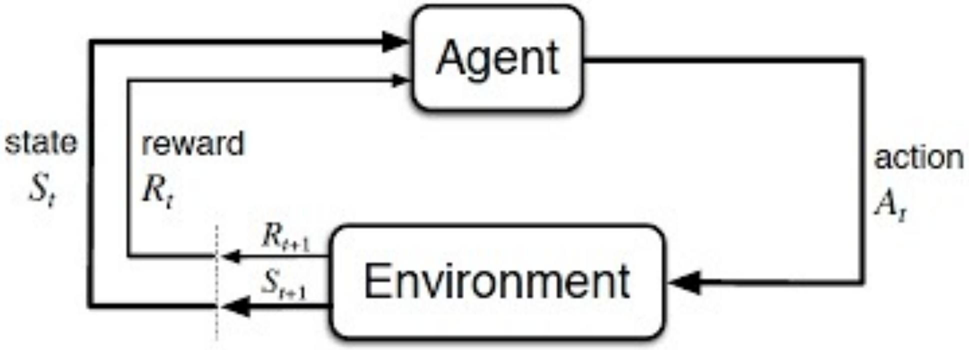
Offline Optimization for Join Order Enumerator

Map into RL Models (DQ, ReJOIN) [1,2]

- Agent : optimizer
- Action: join
- Environment: Cost model, database
- Reward： Cost ,Latency
- State : join order

```
Select *
From T1,T2,T3,T4
Where T1.a = T2.a
   and T3.b = T4.b
   and T1.c = T3.c
```





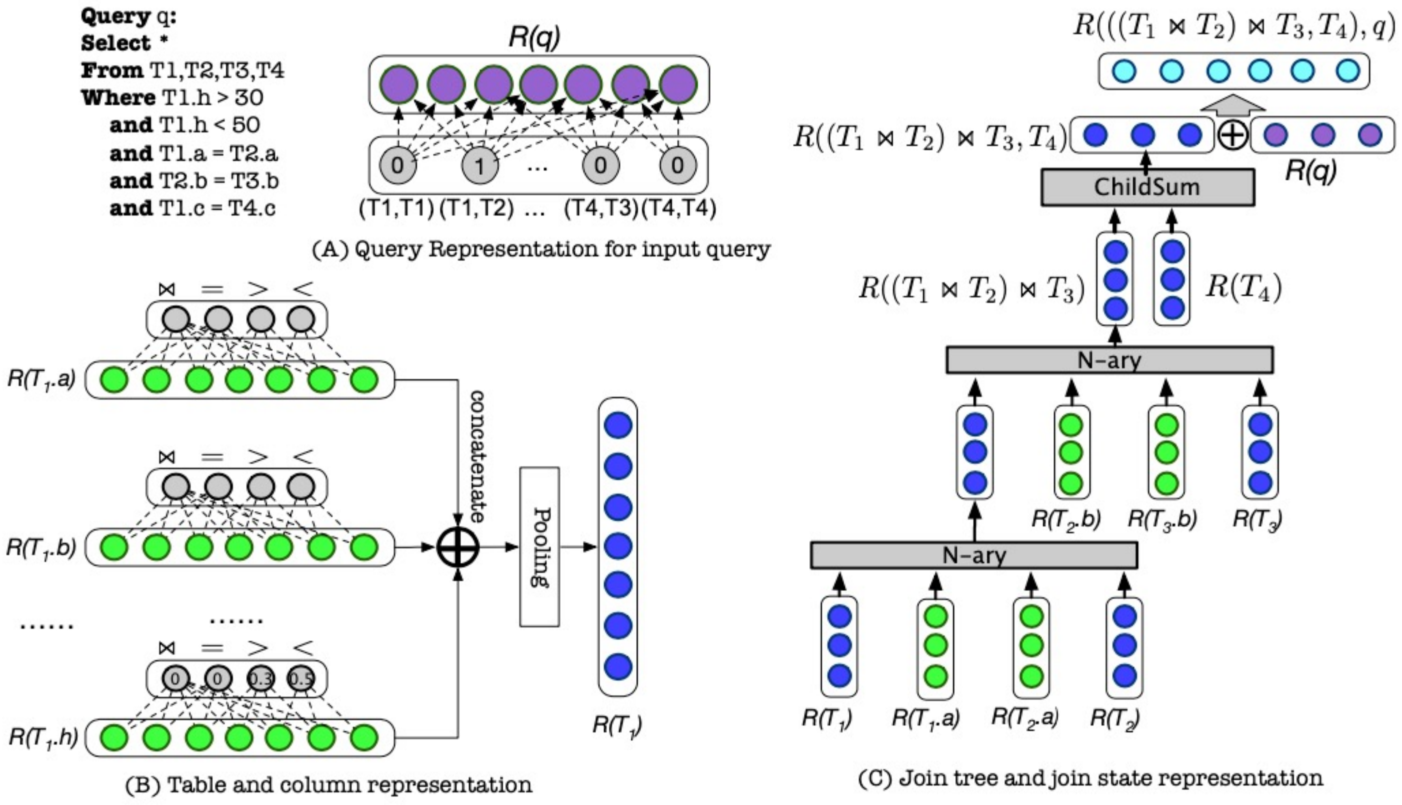
1. Marcus R, Papaemmanouil O. Deep reinforcement learning for join order enumeration
2. Krishnan S, Yang Z, Goldberg K, et al. Learning to optimize join queries with deep reinforcement learning

Feature Encoding for Join Order Enumerator

☐ Feature Extraction

- **The structural information of the execution plan is vital to join order selection →**

- **Encode the operator relations and metadata features of the query**
- **Embed the query features with Tree-LSTM; Decide join orders with RL model**

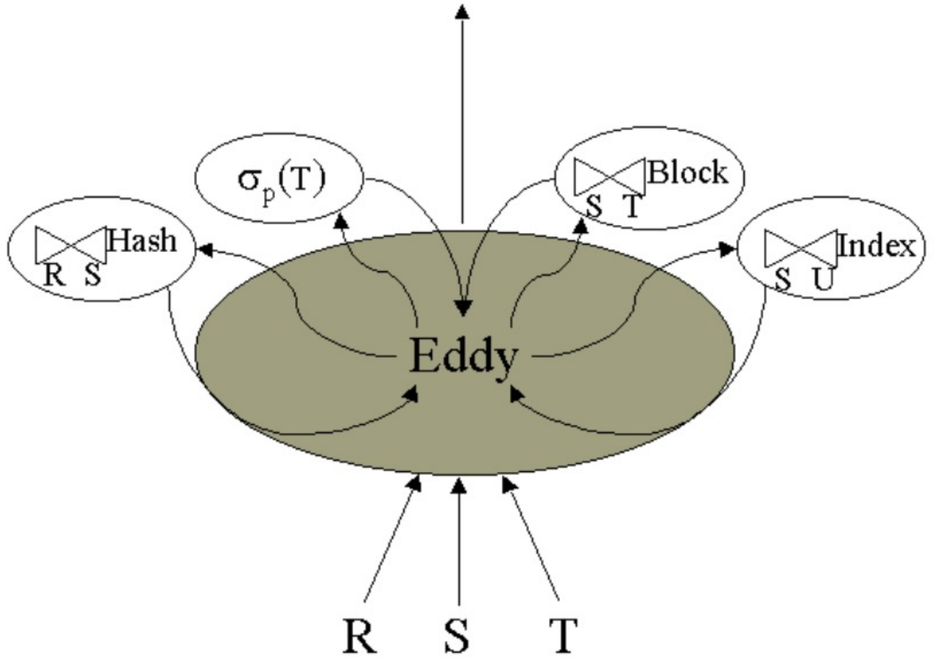


X. Yu, G. Li, and C.C. et al. Reinforcement learning with tree-lstm for join order selection. In ICDE, 2020.

Online Optimization for Join Order Enumerator

☐ Update execution orders of tuples on the fly

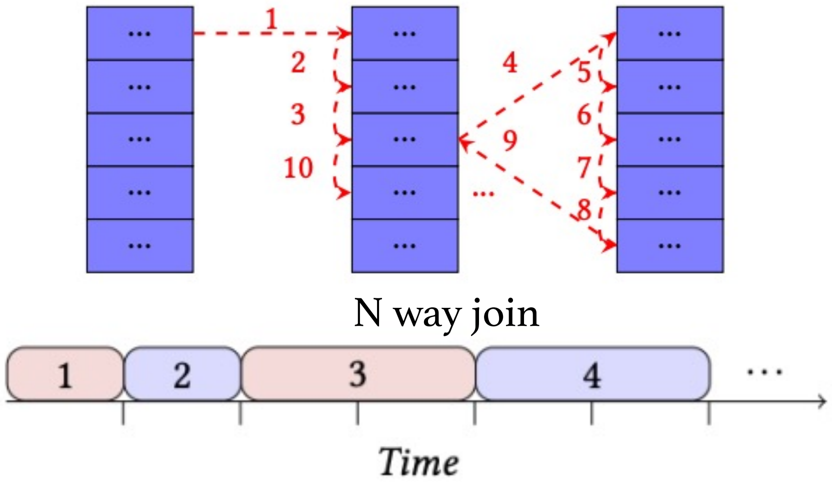
- Update the plan on the fly and <u>preserve the execution state</u> →
- Tuples flows into *the* Eddy from input relations (e.g., R, S, T);
- Eddy routes tuples to corresponding operators (the order is adaptively selected by the operator costs);
- Eddy sends tuples to the output only when <u>the tuples have been handled by all the operators</u>.

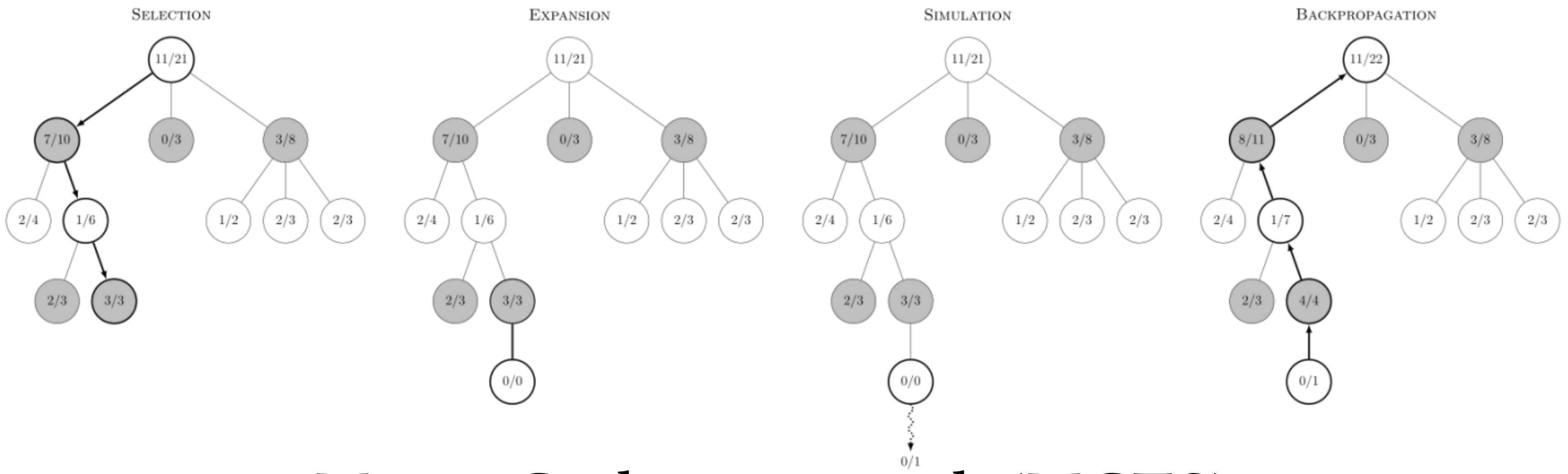


Ron Avnur and Joseph M. Hellerstein. Eddies: Continuously Adaptive Query Processing. SIGMOD, 2000.

MCTS for Join Order Enumerator

- **Support online reorder with MCTS** →
  - **Do not require pre-training**
  - **Time Slides: 0.001s**
    - **Learn during runtime**
  - **Customize Database**
    - **Switch Plan in Low Latency**





Monte Carlo tree search (MCTS).

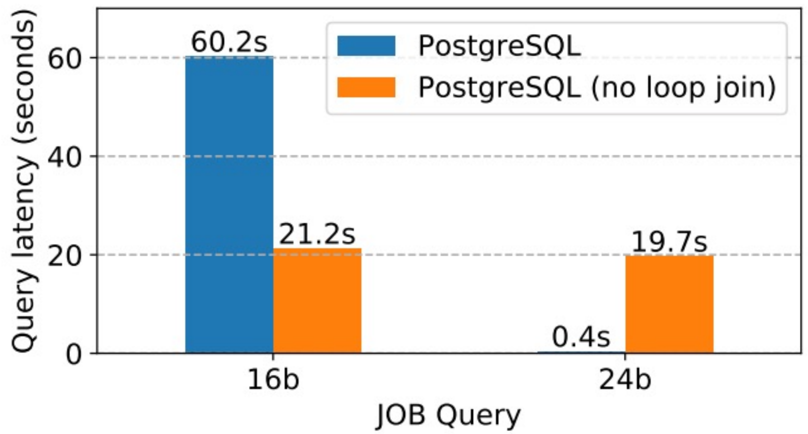
Trummer, et al Skinnerdb: Regret-bounded query evaluation via reinforcement learning. In SIGMOD, 2019.

Join Order Enumerator

	Quality	Training Cost	Adaptive (workload)	Adaptive (DB Instance)
Traditional Methods [Genetic algorithms] [Dynamic Programming]	Low	Low	✓	High
Offline Optimization Methods [NEO VLDB2019] [RTOS ICDE2020]	High	High	×	Medium
Online Optimization Methods [Eddies SIGMOD2000] [SkinnerDB SIGMOD2019]	Medium	Low	✓	Low

Online Optimization for Plan Hinter

- **Enhance query optimization with minor changes**
  - E.g., Activate/Deactivate loop join for different queries

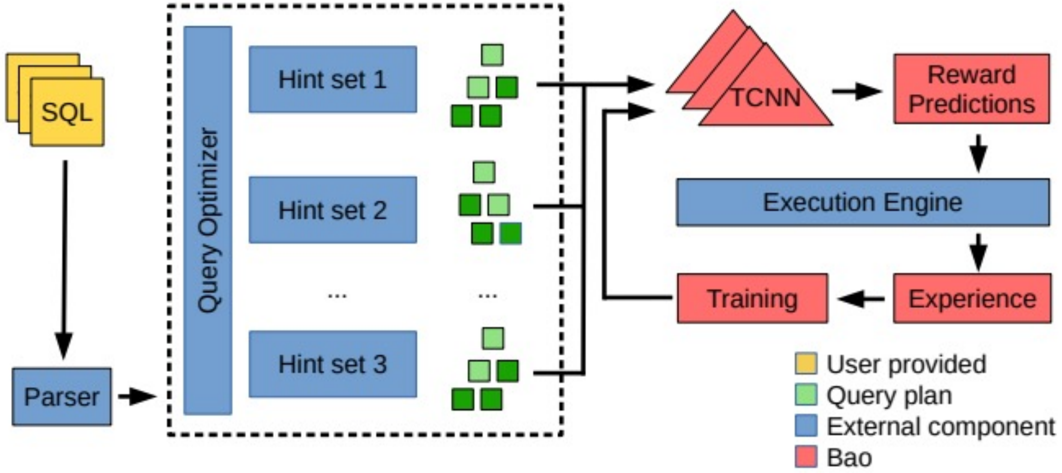


- **Model Plan Hinter as a Multi-armed Bandit Problem**
  - Model each hint set $HSet_i$ as a query optimizer

$$HSet_i : Q \rightarrow T$$

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

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



Ryan Marcus et al. Bao: Making Learned Query Optimization Practical. In SIGMOD, 2021.

Take-aways of Plan Enumerator

- Learning based algorithm usually gives the plan with low time complexity, especially for large queries.
- Offline learning methods use the sampled workload to pretrained the model. It will give good plans for the incoming queries.
- A new database (updates) will lead to model retraining.
- 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.

Regression Problems

Regression Problems

Database estimation problems can be modeled as regression problems, which fit the high-dimension input variables into target features (e.g., cost, utility) and estimate the value of another variable.

- **Cardinality/Cost Estimation** aims to estimate the cardinality of a query and a regression model (e.g., deep learning model) can be used.

- **Index/View Benefit Estimation** aims to estimate the benefit of creating an index (or a view), and a regression model can be used to estimate the benefit.

- **Query Latency Estimation** aims to estimate the execution time of a query and a regression model can be used to estimate the performance based on query and concurrency features.

Automatic Cardinality/Cost Estimation

☐ Motivation:

☐ One of the most challenging problems in databases

- ➢ Achilles Heel of modern query optimizers

☐ Traditional methods for cardinality estimation

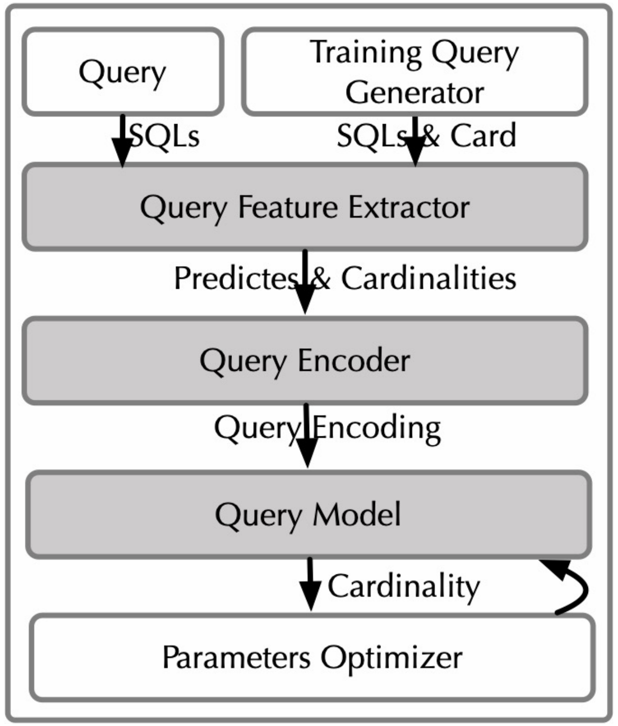
- ➢ Sampling (on base tables or joins)
- ➢ Kernel-based Methods (Gaussian Model on Samples)
- ➢ Histogram (on single column or multiple columns)

☐ Traditional cost models

- ➢ Data sketching/data histogram based methods
- ➢ Sampling based methods

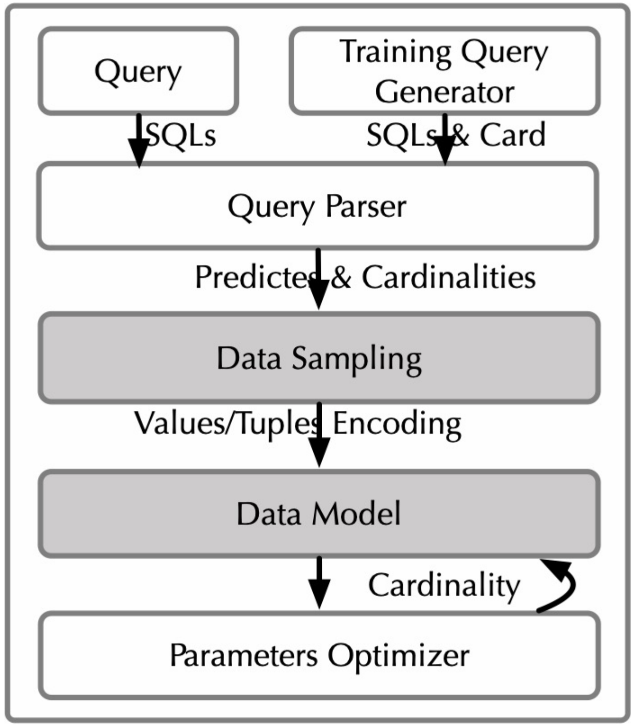
Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. How good are query optimizers, really? In VLDB, 2015.

Categories of Cardinality Estimation



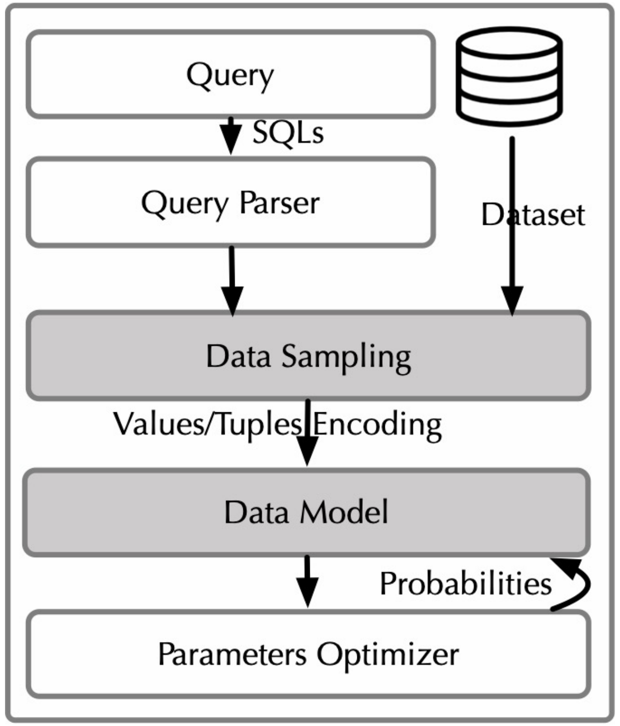
(1) Supervised Query Methods

- Multi-set Convolutional network
- Tree-based ensemble



(2) Supervised Data Methods

- Gaussian kernel
- Uniform mixture model



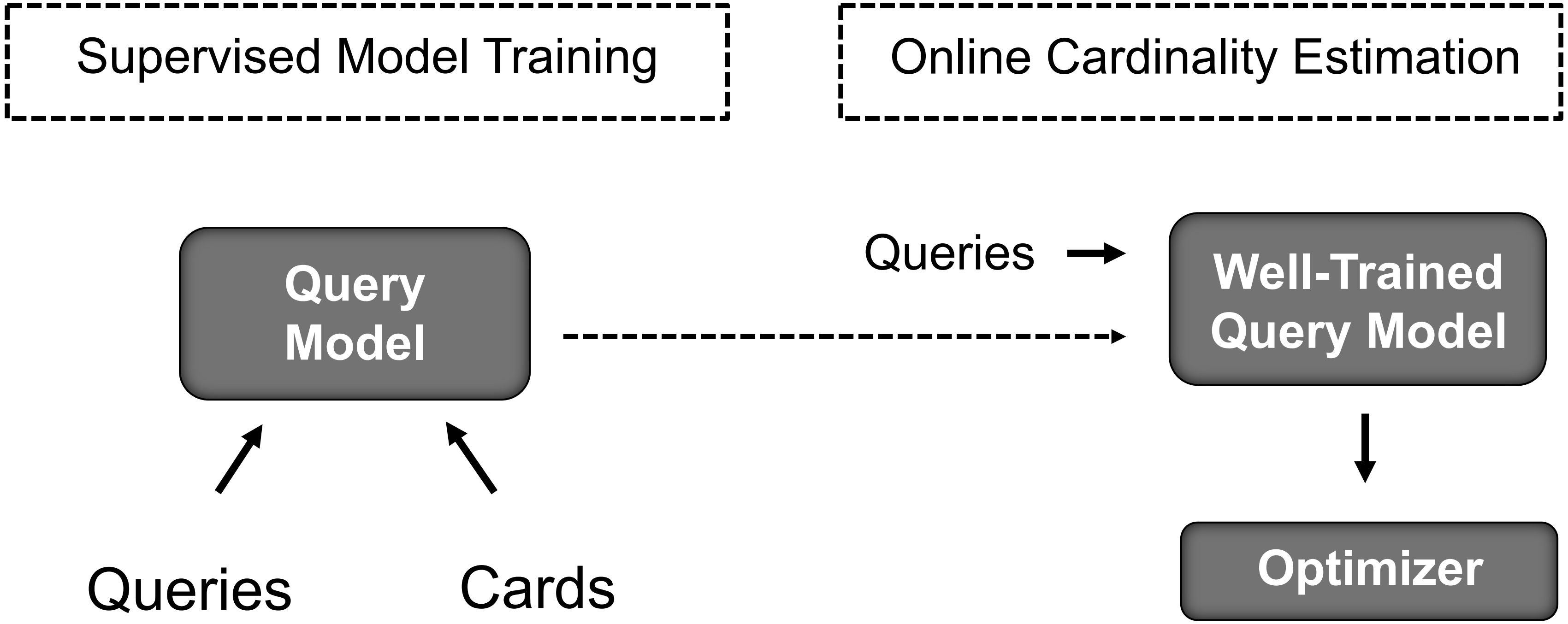
(3) Unsupervised Data Methods

- Autoregressive
- Sum product network

1 Supervised Query Methods for Cardinality Estimation

□ Problem Definition

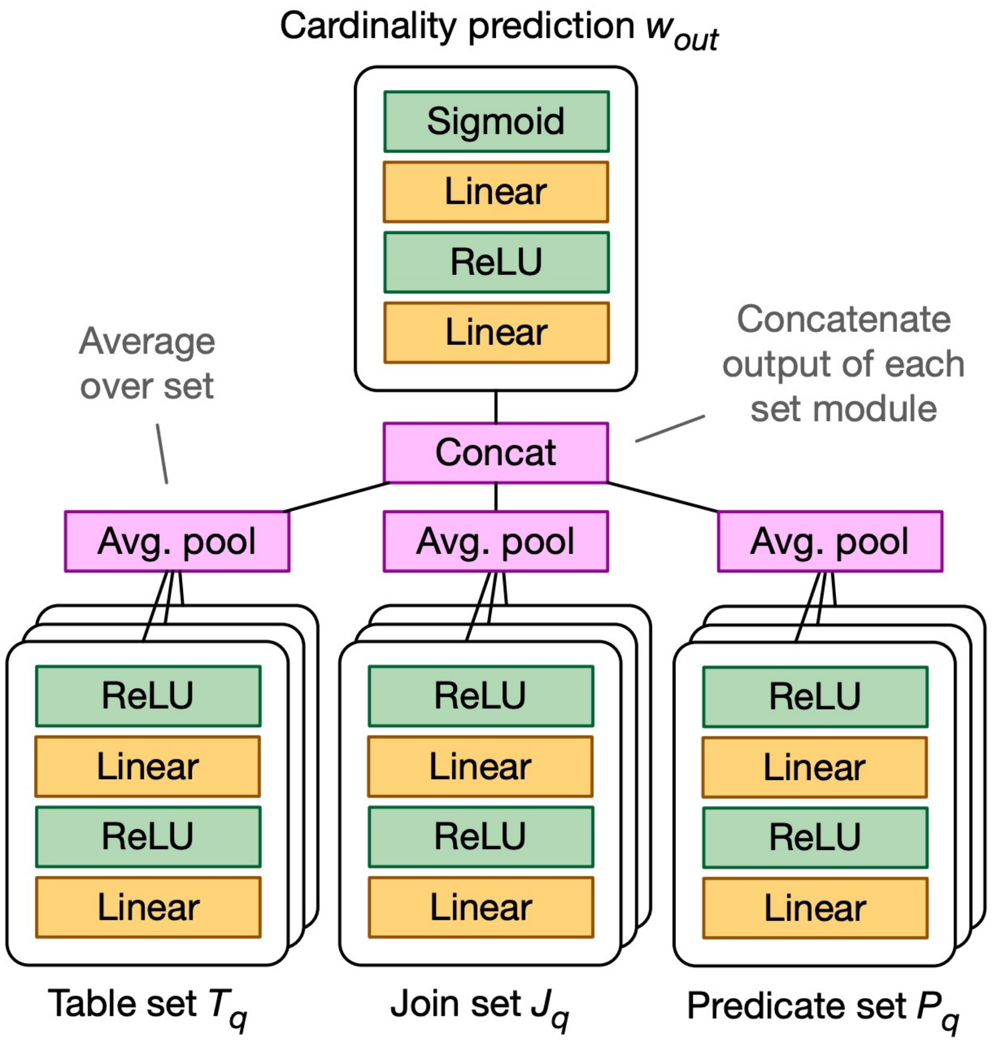
A regression problem: learn the mapping function between query Q and its actual cardinality



1.1 Deep Learning for Cardinality Estimation

☐ Model Construction

- **Multi-set Convolutional Neural Network**
  - ➢ Linear Models for different part of SQL (table, joins, predicates)
  - ➢ Pooling Varying-sized representations (avg pooling)
  - ➢ Concatenate different parts



A. Kipf, T. Kipf, B. Radke, V. Leis, P. A. Boncz, and A. Kemper. Learned cardinalities: Estimating correlated joins with deep learning. In CIDR, 2019.

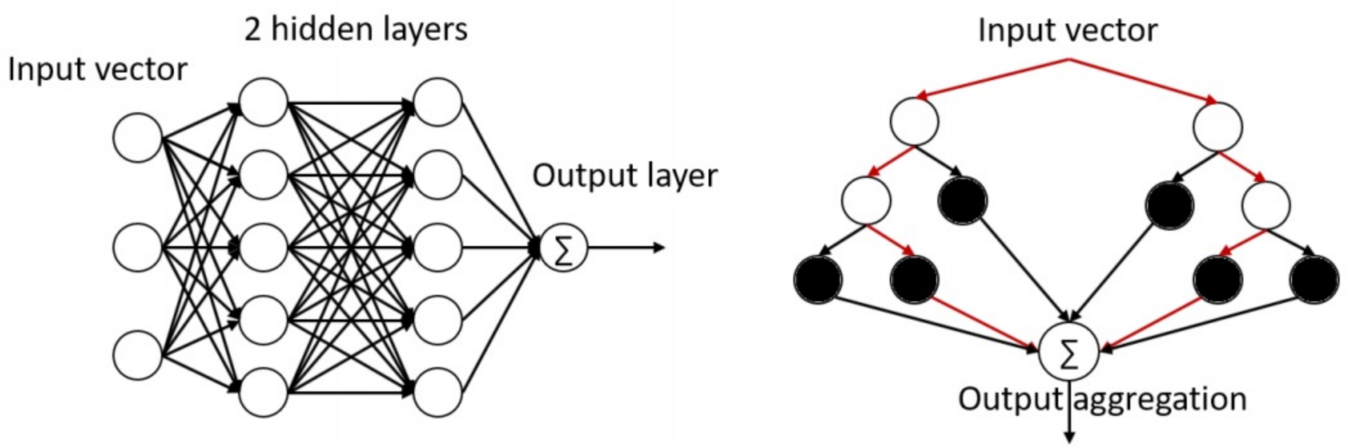
1.2 Tree-Ensembling for Cardinality Estimation

☐ Model Construction

- **Challenge: Traditional cost estimation methods assume column independency**
  - ➢ **Any conjunctive query on columns *C* can be represented as:**

$$(c_1 \leq lb_1 < c_2) \wedge (c_3 < ub_1 \leq c_4) \wedge (c_5 \leq ub_2 \leq c_6)$$

  - ➢ **Tree-based ensembles: pass query encoding vectors through the traversal of multiple binary trees**



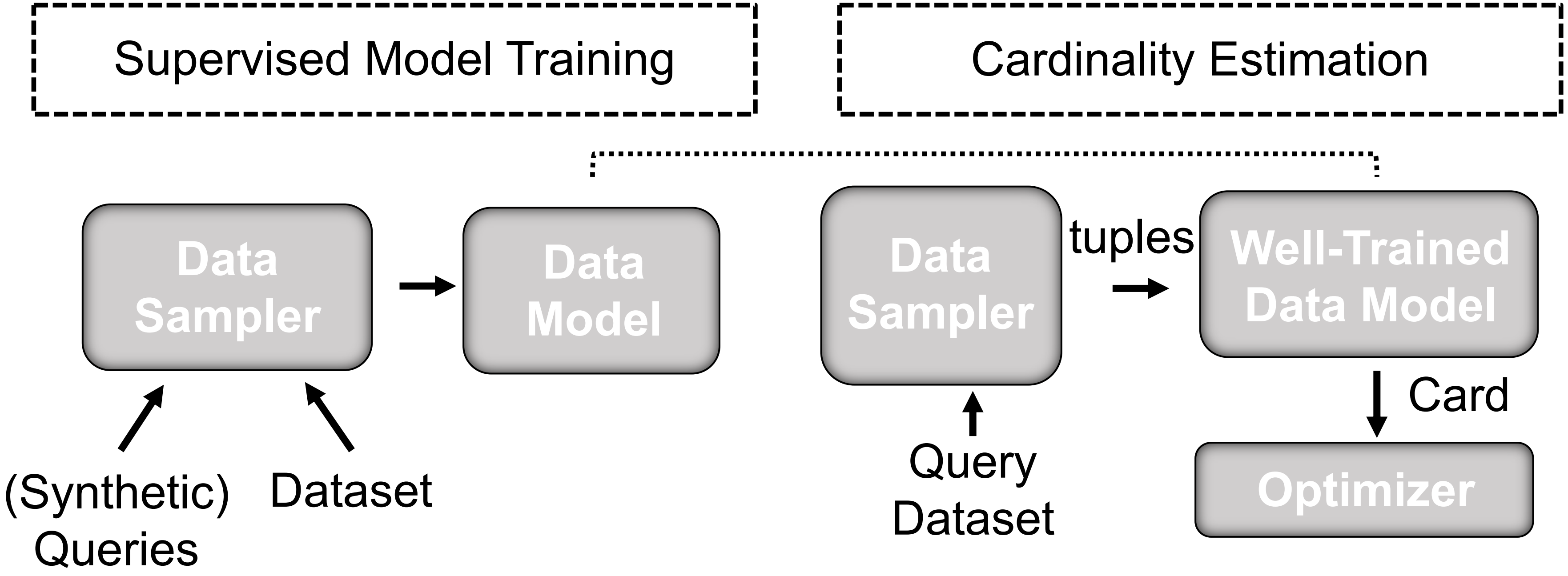
(a) Neural network with 2 hidden layers (b) Tree-based ensembles with 2 trees

A. Dutt, C. Wang, A. Nazi, S. Kandula, V. R. Narasayya, and S. Chaudhuri. Selectivity estimation for range predicates using lightweight models. PVLDB, 2019.

2 Supervised Data Methods for Cardinality Estimation

☐ Problem Definition

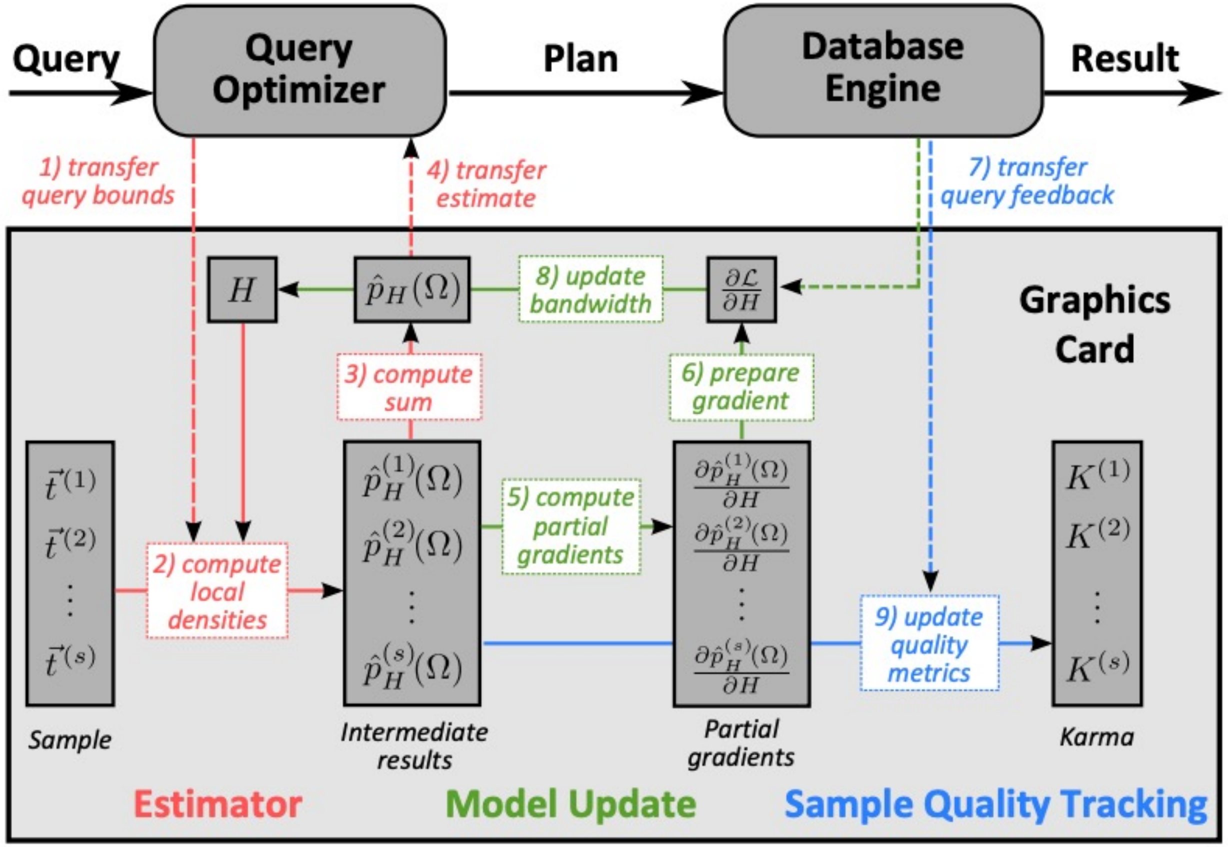
A density estimation problem: learn a joint data distribution of each data point



2.1 Kernel-Density for Cardinality Estimation

☐ Model Construction

- **Support point queries on single tables →**
  - ➢ Sample rows from the table and initialize the **bandwidth** (distance from the true distribution) of the kernel density model.
  - ➢ Pick optimal bandwidth via stochastic gradient descent.
  - ➢ Estimate the cardinality based on the kernel density model.



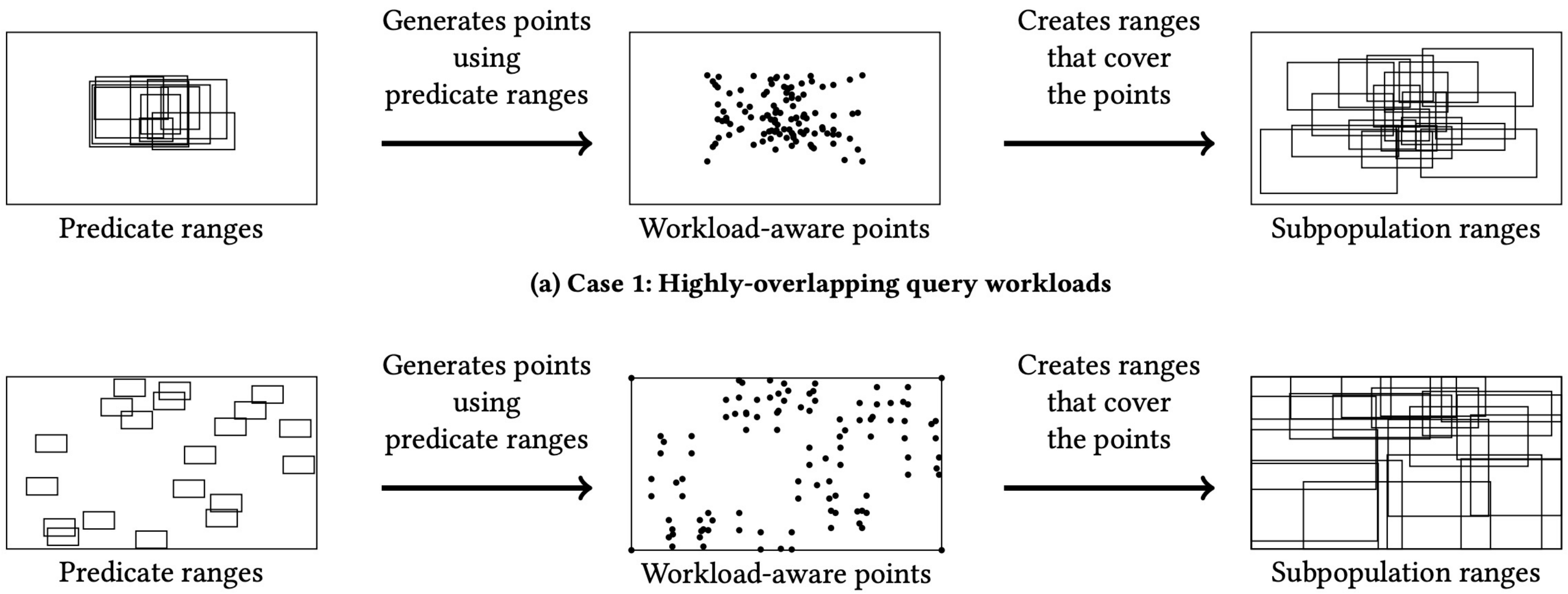
M. Heimel, M. Kiefer, and V. Markl. Self-tuning, gpu-accelerated kernel density models for multidimensional selectivity estimation. SIGMOD, 2015.

2.2 Mixture Model for Cardinality Estimation

☐ Model Construction

- **Support Range Queries →**

➢ Sample points within each history queries.
➢ Generating **subgroups** for the points.
➢ Learn the weights of all the Uniformity Mixture Models for range queries.

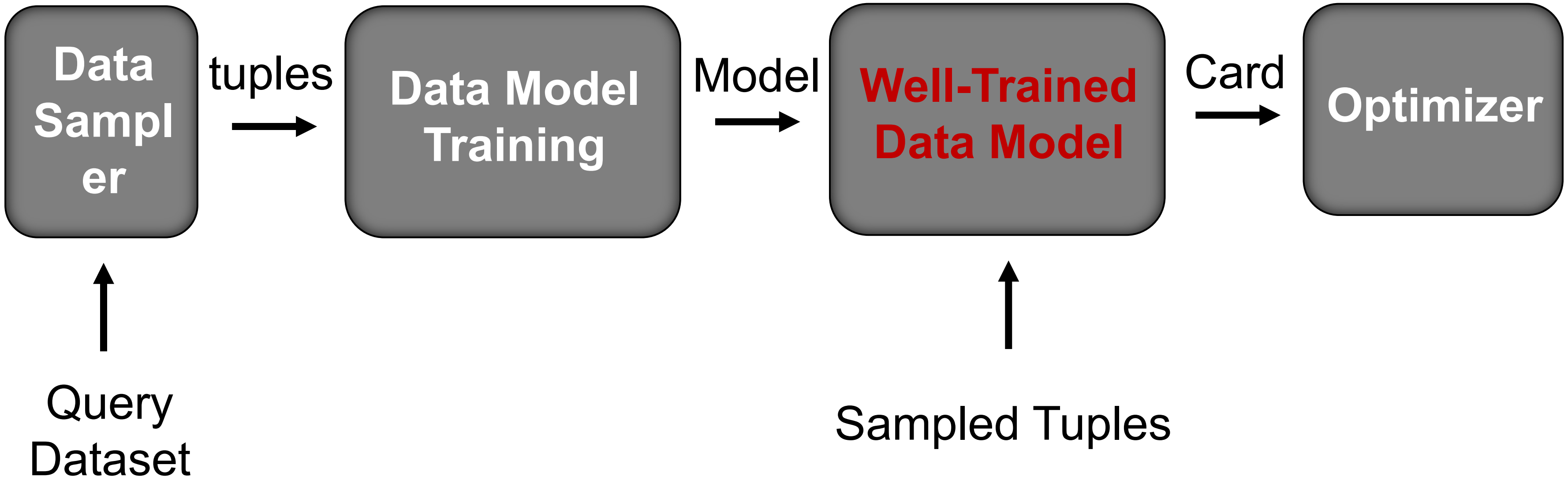


Yongjoo Park, Shucheng Zhong, and Barzan Mozafari. Quicksel: Quick selectivity learning with mixture models. SIGMOD 2020

3 Unsupervised Data Methods for Cardinality Estimation

□ Problem Definition

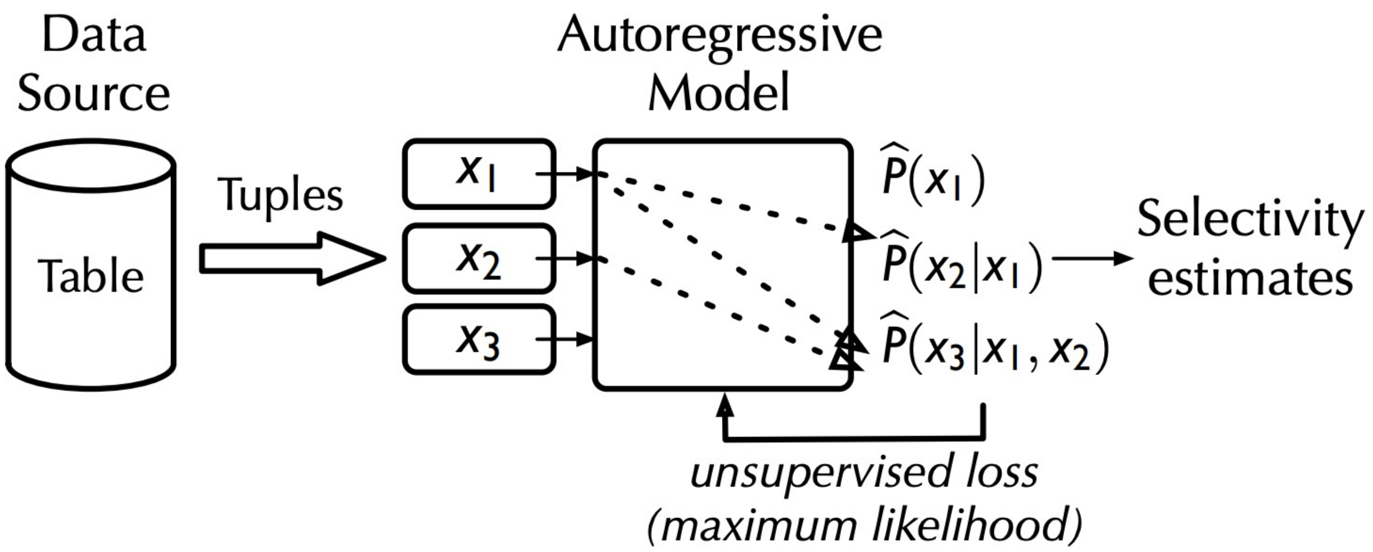
A regression problem: learn a probability function for each data point



3.1 Autoregressive for Cardinality Estimation (single table)

☐ Model Construction

- **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

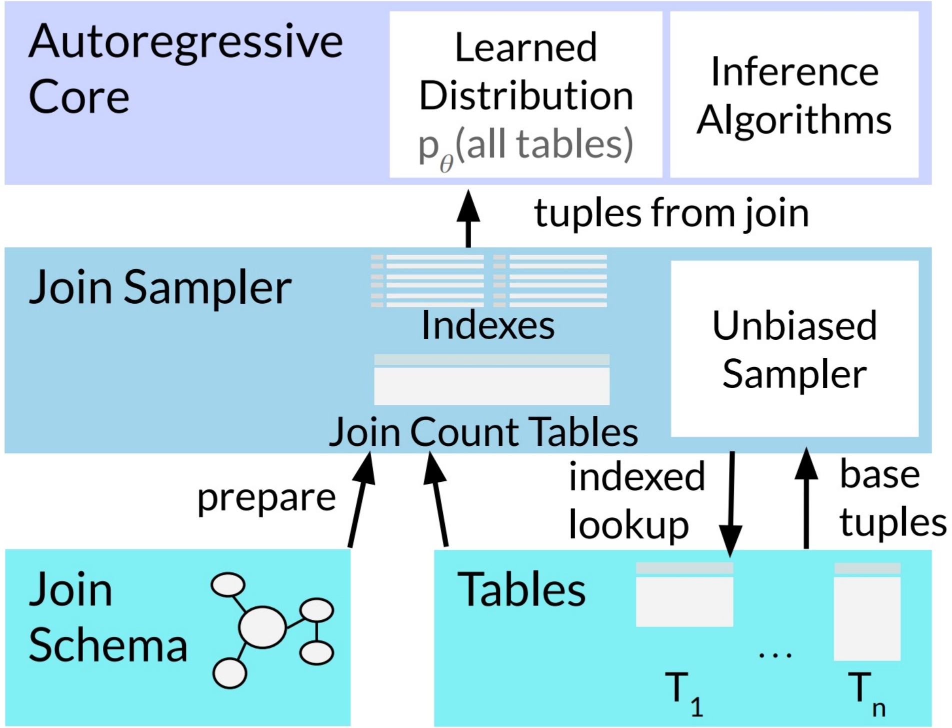


1. S. Hasan, S. Thirumuruganathan, J. Augustine, N. Koudas, and G. Das. Deep Learning Models for Selectivity Estimation of Multi-Attribute Queries. In sigmod, 2020.
2. Z. Yang, E. Liang, A. Kamsetty, C. Wu, Y. Duan, X. Chen, P. Abbeel, J. M. Hellerstein, S. Krishnan, and I. Stoica. Deep Unsupervised Cardinality Estimation. PVLDB, 13(3): 279-292, 2019.

3.2 Autoregressive for Cardinality Estimation (multi-tables)

☐ Model Construction

- **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
  - ➢ Down sampling some tuples when estimating query with only a subset of tables according to the fanout scaling.



Zongheng Yang, Amog Kamsetty, Sifei Luan, Eric Liang, Yan Duan, Xi Chen, and Ion Stoica. NeuroCard: One Cardinality Estimator for All Tables. PVLDB, 14(1): 61-73, 2021

3.3 Sum-Product Network for Cardinality Estimation

☐ Model Construction

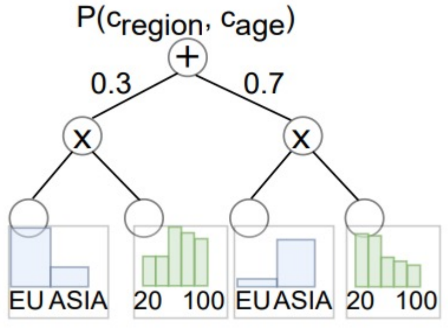
- **Different data distributions over the tables, which are independent from each other →**
  - ➢ Split data table into multiple segments and columns in each segment are near independent.
  - ➢ SPN: Sum for different filters and Product for different joins.
  - ➢ RSPN is for AVG aggregation, NULL values support, non-key attributes modeling and updatability.

c_id	c_age	c_region
1	80	EU
2	70	EU
3	60	ASIA
4	20	EU
...	...	...
998	20	ASIA
998	25	EU
999	30	ASIA
1000	70	ASIA

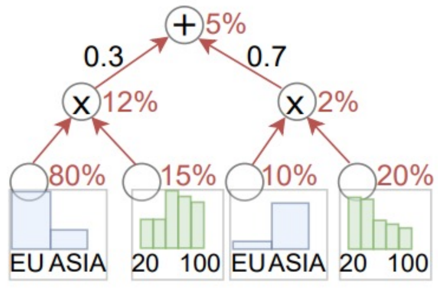
(a) Example Table

c_age	c_region
80	EU
70	EU
60	ASIA
20	EU
...	...
...	...
20	ASIA
25	EU
30	ASIA
70	ASIA

(b) Learning with Row/Column Clustering



(c) Resulting SPN



(d) Probability of European Customers younger than 30

Benjamin Hilprecht, Andreas Schmidt, Moritz Kulessa, Alejandro Molina, Kristian Kersting, and Carsten Binnig. DeepDB: Learn from Data, not from Queries! PVLDB 13, 13(7): 992-1005, 2020

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

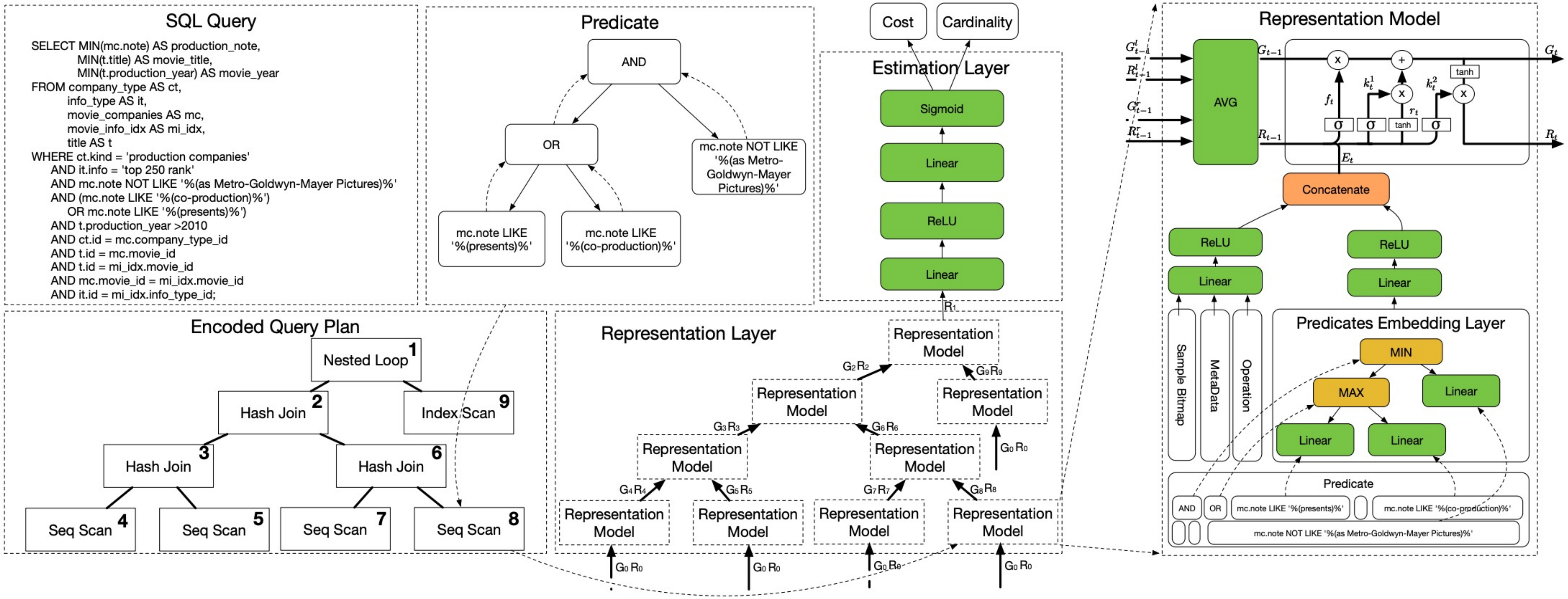
□ Estimation Difficulity

- Cost is harder to estimate than cardinality, which considers multiple factors (e.g., seq scan cost, cpu usage)

Tree-LSTM for Cost Estimation

◻ Model Construction

- **Traditional cost estimation uses estimated card, which is inaccurate without predicate encoding →**

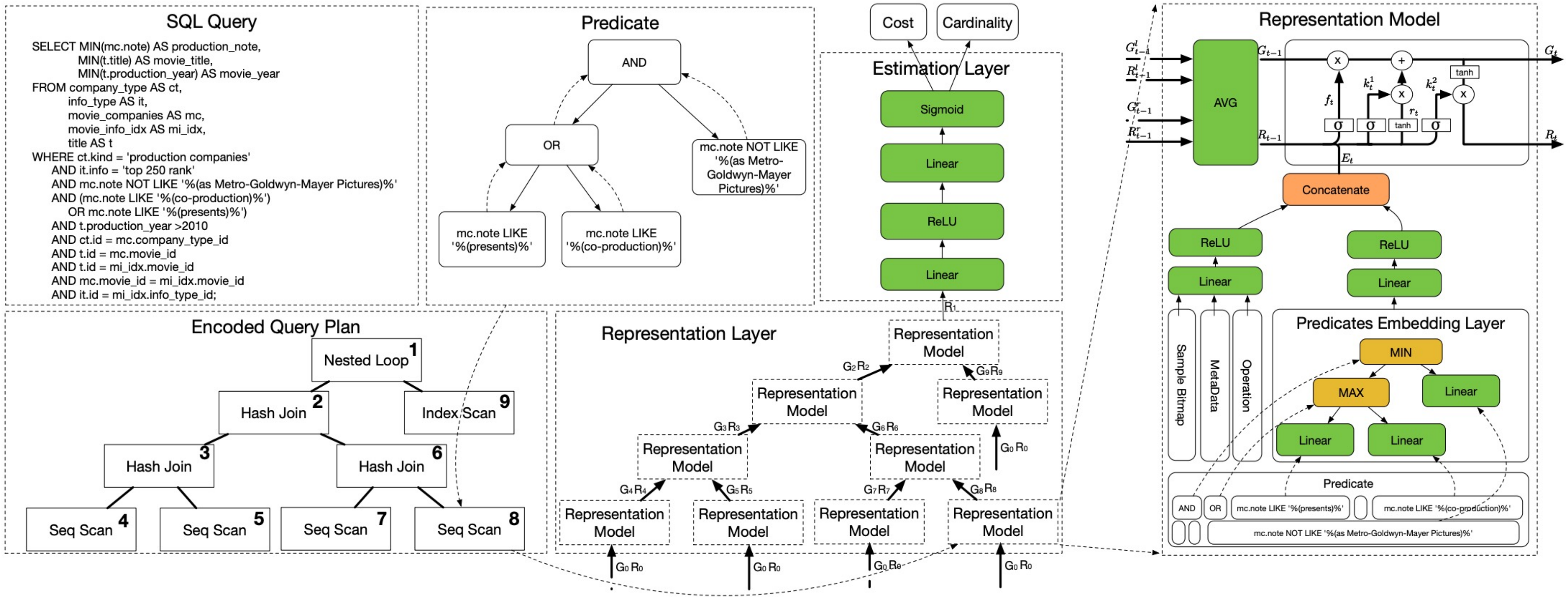


J. Sun and G. Li. An end-to-end learning-based cost estimator. PVLDB, 13(3):307–319, 2019.

Tree-LSTM for Cost Estimation

☐ Model Construction

- ➢ **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



J. Sun and G. Li. An end-to-end learning-based cost estimator. PVLDB, 13(3):307–319, 2019.

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-drive 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.**

Deep Learning for Benefit Estimation

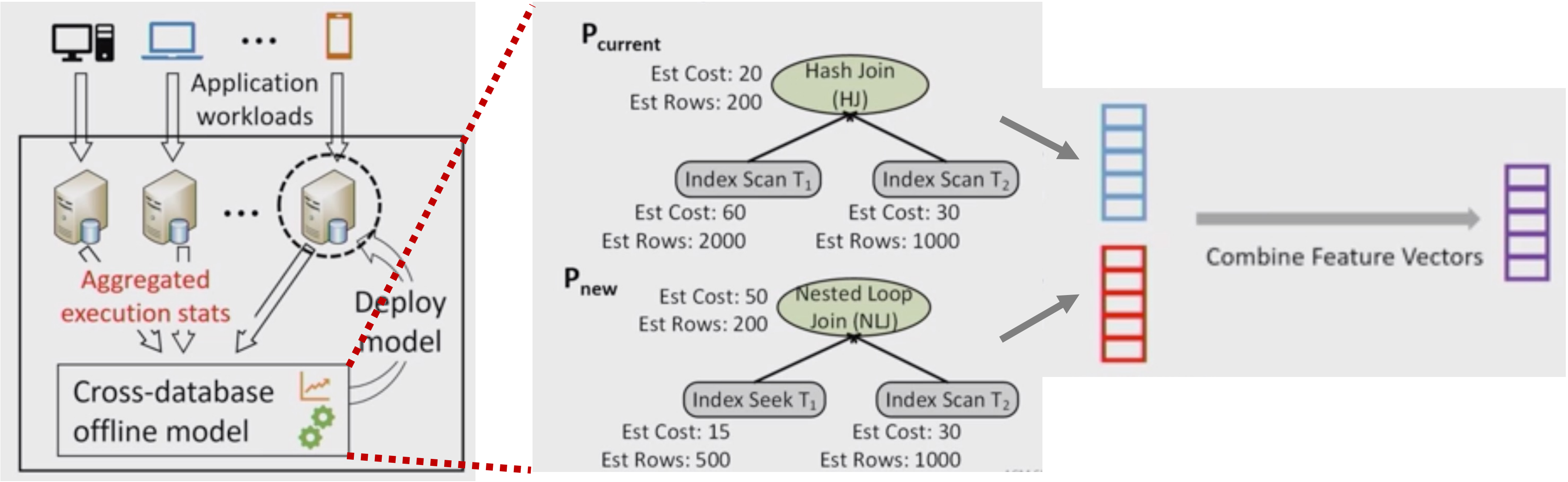
- **Challenge**
  - **The index/view benefit is hard to evaluate**
    - Multiple evaluation metrics (e.g., index benefit, space cost)
    - Cost estimation by the optimizer is inaccurate
  - **Interactions between existing data structures**
    - Multiple column access, Data refresh
    - Conflicts between MVs

1 Deep Learning for Index Benefit Estimation

☐ Model Construction

- ☐ **Motivation: It is critical to compare execution costs of plans and decide index benefits →**

- ➢ **Training Data:** Workloads and execution feedback from customers
- ➢ **Well-trained Evaluation Model:** Predict the index performance
- ➢ Use the evaluation model to create indexes with performance gains



Bailu Ding, Sudipto Das, et al. AI meets ai: leveraging query executions to improve index recommendations. In SIGMOD, 2019.

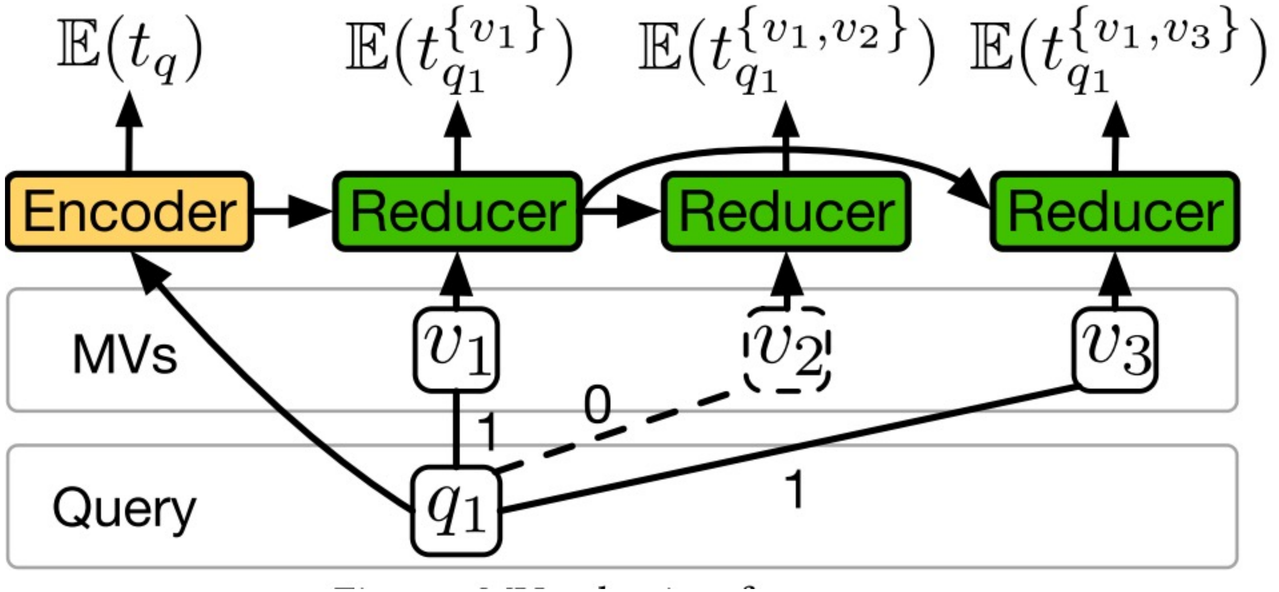
2 Encoder-Decoder for View Benefit Estimation

☐ Feature Extraction

- **Previous work take candidate views as fixed length →**
- **Encode various number and length of queries and views with an *encoder-reducer model*, which captures correlations with *attention***

☐ Model Construction

- **It is hard to jointly consider MVs thatmay have conflicts →**
- **(1) Split the problem into sub-steps that select one MV; (2) Use attention-based model to estimate the MV benefit**



Y. Han, G. Li, H. Yuan, and J. Sun. An autonomous materialized view management system with deep reinforcement learning. In ICDE, 2021.

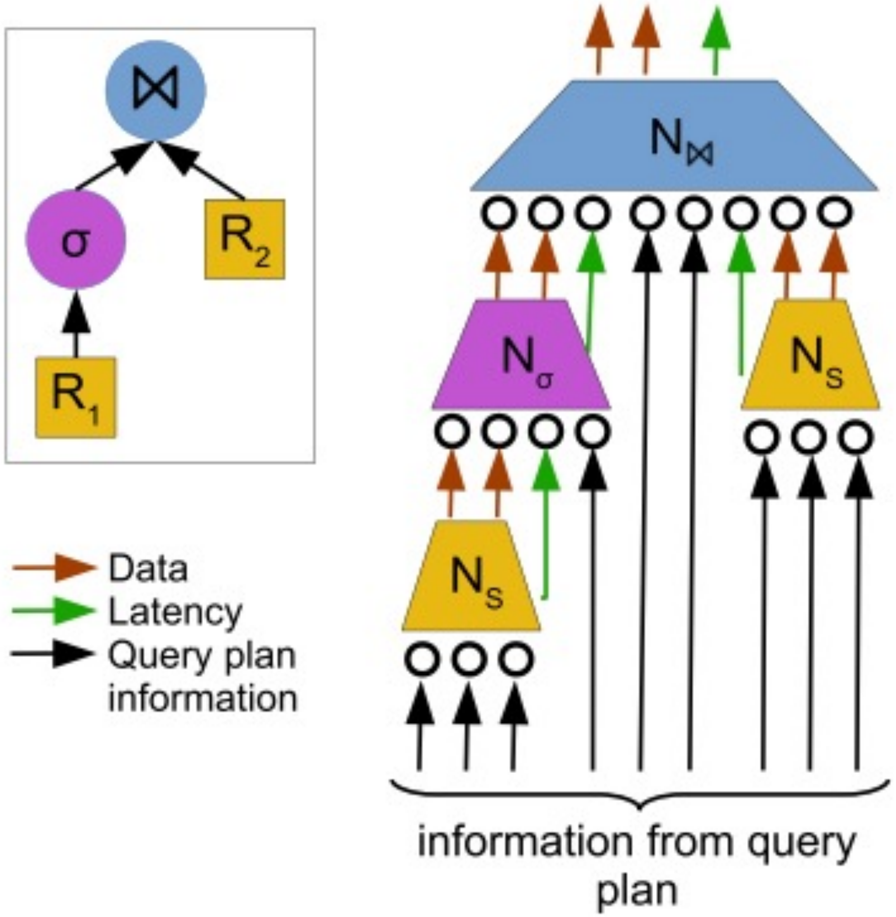
Take-aways of Benefit Estimation

- **Learned utility estimation is more accurate than traditional empirical methods**
- **Learned utility estimation is also accurate for multiple-MV optimization**
- **Query encoding models need to be trained periodically when data update**
- **Open problems:**
  - Benefit prediction for future workload
  - Cost of initialization and future updates

Deep Learning for Query Latency Estimation

☐ Model Construction

- **Performance prediction of single queries →**
  - Represent each operator with a **neural unit**
  - Each neural unit predicts the execution time of its operator
  - Construct a **network** that matches the query structure to predict the query latency
  - Take effects of concurrent queries as parallel operators (e.g., gather, parallel join)

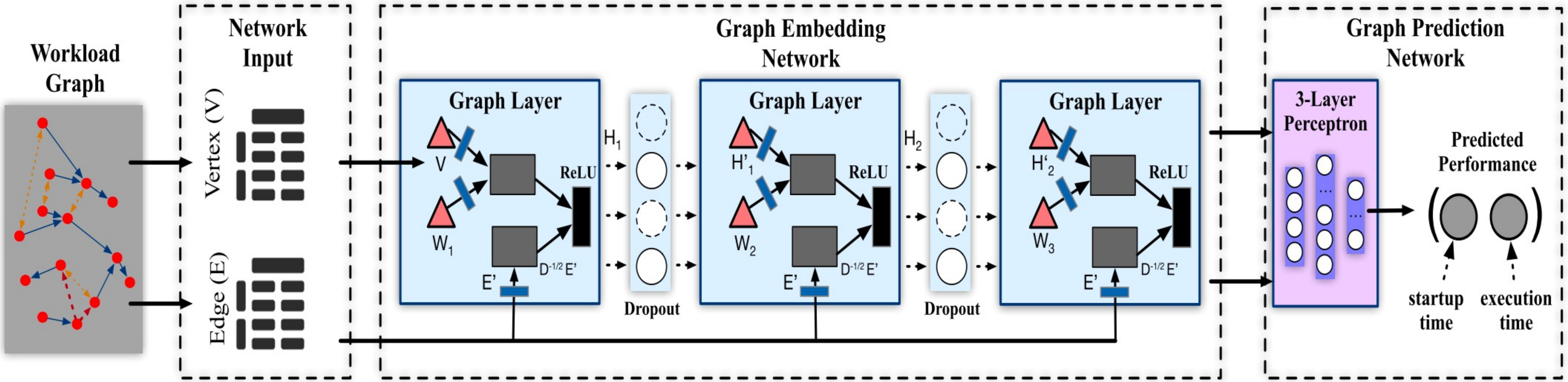


R. Marcus and O. Papaemmanouil. Plan-structured deep neural network models for query performance prediction. Proc. VLDB Endow., 2019.

Graph Embedding for Query Latency Estimation

☐ Model Construction

- **Performance prediction of concurrent queries →**
  - ➢ Represent concurrent queries with a graph model
  - ➢ Embed the graph with graph convolution network and predict the latency of all the operators with a simple dense network



 X. Zhou, J. Sun, G. Li, et al. Query Performance Prediction for Concurrent Queries using Graph Embedding. VLDB, 2020.

Prediction Problems

Prediction Problems

☐ Motivation

☐ Effective Scheduling can Improve the Performance

- ➢ Minimize conflicts between transactions

☐ Concurrency Control is Challenging

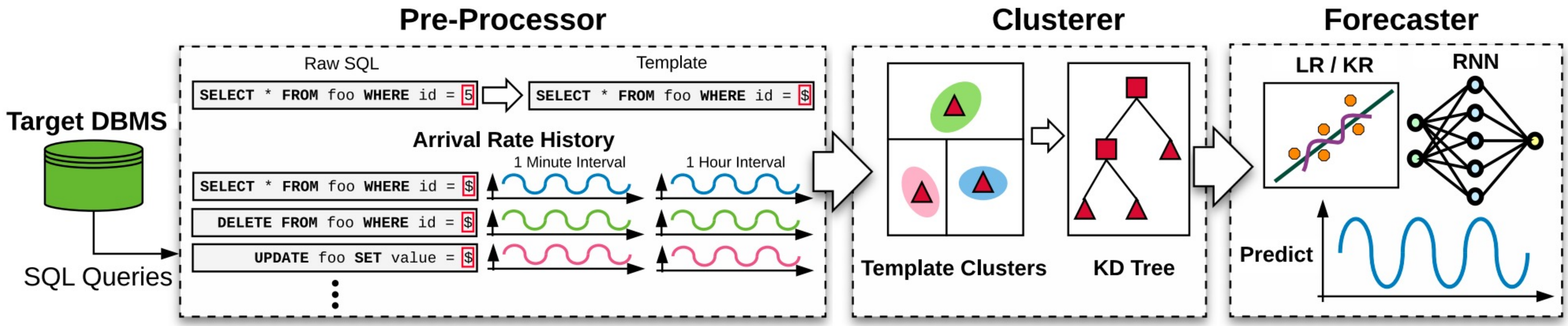
- ➢ #-CPU Cores Increase

☐ Transaction Management Tasks

- ➢ Transaction Prediction
- ➢ Transaction Scheduling

Learned Transaction Prediction

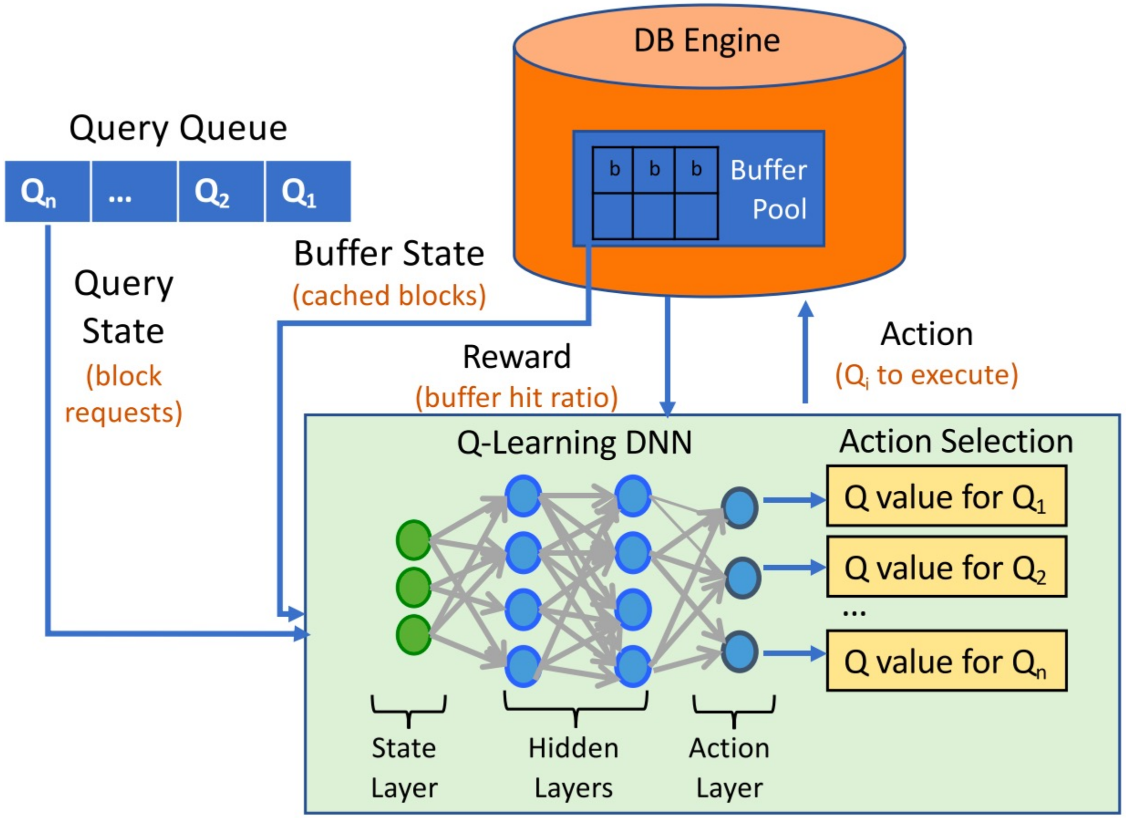
- **Predict the future trend of different workloads →**
  - **Pre-Processor** identifies query templates and the arrival-rate from the workload;
  - **Clusterer** combines templates with similar arrival rate patterns
  - **Forecaster** utilizes ML models to predict arrival rate in each cluster



Lin Ma, Dana Van Aken, and et al. Query-based Workload Forecasting for Self-Driving Database Systems. In SIGMOD, 2018.

Learned Transaction Scheduling

- **Learn to schedule queries to minimize disk access requests →**
  - 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



Chi Zhang, Ryan Marcus, and et al. Buffer Pool Aware Query Scheduling via Deep Reinforcement Learning. In VLDB, 2020.

ML Methods in ML4DB

Summarization of ML4DB Techniques

	Database Problem	Method	Performance	Overhead	Training Data	Adaptivity
Offline NP Problem	knob space exploration	gradient-based [1, 18, 47]	High	High	High	–
		dense network [37]	Medium	High/Medium	High	– / instance
		DDPG [23, 46]	High	High	Low/Medium	query
	index selection	q-learning [19]	–	High	Low	–
	view selection	q-learning [43]	Medium	High	Low	–
		DDQN [9]	High	High	Low	query
	partition-key selection	q-learning [11]	–	High	Low	–
Online NP Problem	join order selection	q-learning [27]	High	High	Low	–
		DQN [26, 42]	High	High	Low	query
		MCTS [38]	Medium	Low	Low	instance
	query rewrite	MCTS [21, 49]	–	Low	Low	query
Regression Problem	cost estimation	tree-LSTM [35]	High	High	High	query
	cardinality estimation	tree-ensemble [7]	Medium	Medium	High	query
		autoregressive [41]	High	High/Medium	Low	data
		dense network [16]	High	High	High	query
		sum-product [12]	Medium	High	Low	data
	index benefit estimation	dense network [5]	–	High	High	query
	view benefit estimation	dense network [9]	–	High	High	query
	latency prediction	dense network [28]	Medium	High	High	query
		graph embedding [50]	High	High	High	instance
	learned index	dense network [3]	–	High	High	query
Prediction Problem	trend prediction	clustering-based [24]	–	Medium	Medium	instance
	transaction scheduling	q-learning [44]	–	High	Low	query

Classical ML Methods

- **Techniques**
  - **Gradient methods (e.g., GP); Regression methods (e.g., tree-ensembling, kernel-density estimation)**
- **Advantages**
  - **Lightweight; Easier to interpret than DL**
- **Disadvantages**
  - **Hard to extend to large data; Complex feature engineering**
- **ML4DB Applications**
  - **Knob Tuning; Cardinality Estimation**

Classical ML Methods

☐ Application Difference

	Feature Engineering	Model Selection
Knob Tuning	• Reduce the knob space with linear regression like Lasso; • Reduce redundant metrics with factor analysis and clustering like k-means;	• Gaussian Process: Search local-optimal settings within the selected knob space • Reuse the historical data by matching workloads by their metric values
Cardinality Estimation	• Assumptions like column independency or linear relations between columns • Determine supported queries like range queries	• Query-based: Define input space as conjunction of the query ranges on data columns (Tree-Ensemble) • Data-based: Partition data into indpendent regions (Sum-Product) or learn column correlations (AR)

Classical ML Methods

- How to apply to a new problem?
  - Problem Modelling: As a regression or gradient-based optimization problems
  - Feature Engineering: Determine the input with feature engineering techniques
  - Model Construction: Select proper classic ML models, collect sample data, and learn the mapping relations
  - Additional Requirements: Reuse classic ML models in limited scenarios (e.g., similar workloads)

Reinforcement Learning Methods

- **Techniques**
  - **Model-based (e.g.,, MCTS+DL);**
  - **Model-free (e.g., value-based like Q-learning, policy-based like DDPG)**
- **Advantages**
  - **High performance on large search space; No prepared data**
- **Disadvantages**
  - **Long exploration time; Hard to migration to new scenarios**
- **ML4DB Applications**
  - **Knob Tuning, View/Index/Partition-key Selection, Optimizer, Workload Scheduling**

Reinforcement Learning Methods

	Input Features	RL Method	Reward Design	Estimation Model
Knob Tuning	• Knobs Values • Innter Metrics • Workloads	• DDPG for both continuous state and continuous actions	• Performance improvements over last tuning action • Performance improvements over first tuning action	• Design a dense network as the estimation (critic) model

Reinforcement Learning Methods

	Input Features	RL Method	Reward Design	Estimation Model
View Selection	• Candidate Views • Built Views • Workload	• DQN for continuous state and discrete actions	• Utility increase on creating the views	• Encoder-decoder for inputs; Nonlinear layers for utility estimation
Index Selection	• Candidate Indexes • Built indexes • Workload		• Utility increase on creating the indexes	• Design a dense network as the estimation model
Partiton-key Selection	• Columns • Tables • Query templates		• Estimated costs beofore/after partitioning	• Design a dense network as the estimation model

Reinforcement Learning Methods

	Input Features	RL Method	Reward Design	Estimation Model
Query Rewrite	• Logical Query • Rewrite Rules • Table Schema	• MCTS for tree search	• Utility increase for future optimal queries	• Multi-head attention for rules, query, data
Join Order Selection	• Physical Plan • Candidate Joins • Table Schema	• DQN for continuous state and discrete actions	• Saved costs	• Design a dense network as the estimation model
Plan Hinter	• Physical Plan • Hint Sets	• Contextual Multi-armed for limited actions	• Saved costs	• Traditional Optimizer

Reinforcement Learning Methods

- How to apply to a new problem?
  - Problem Modelling: Map to the 6 factors in a RL model (state, action, reward, policy, agent, environment)
  - Feature Characterization: Select target-related features as the state of the RL problem
  - Model Construction: Select proper RL models (e.g., MCTS, DQN, DDPG), design the networks and the reward function
  - Additional Requirements: E.g., encode the query costs with Deep Learning; encode the join relations with GNN

Deep Learning Methods

- **Techniques**
  - **Dense Layer ((non)-linear); Convolutional Layer; Graph Embedding Layer; Recurrent Layer**
- **Advantages**
  - **Approximate the high-dimension relations**
- **Disadvantages**
  - **Data-consuming**
- **ML4DB Applications**
  - **Cost Estimation; Benefit Estimation; Latency Estimation**

Deep Learning Methods

	Input Features	Feature Encoding	Model Design
Cost Estimation	• Physical Plan	• Encode operators with LSTM	• Plan-structured Neural Network
Benefit Estimation	• Physical Plan • Optimization Actions (e.g., views. indexes)	• Encode actions like Encoder-Decoder for Views and linear layer for Indexes	• Design a dense network as the estimation model
Latency Estimation	• Physical Plan • Query Relations • DB State	• Encoder query correlations with graph covolutions	• Design a K-layer graph embedding network for K-hop neighbors

Deep Learning Methods

- How to apply to a new problem?
  - Input Features: Select features that affect the estimation targets (e.g., latency, utility)
  - Encoding Strategy: Encode based on the feature structures (e.g., Graph embedding for query relations)
  - Model Design: Design the network structures (e.g., layers, activation functions, loss functions) based on the input embedding (e.g., fixed-length or varied-length)

Open Problems of ML4DB

Open Problem #1: Reduce Model Training Overhead

- **Lightweight Model Training**
  - **Featurization:** Some features are *not available* in real-word scenarios, e.g., by privacy constraints;
  - **Data Collection:** Costly to *collect data* on different datasets/ databases, e.g., high collection latency, overhead;
  - **Model Migration&Application:** ML models trained on small datasets are hard to generalize to large datasets

- **Possible solutions: few-show learning; from data-driven to knowledge-driven; super-large pre-trained model**

Open Problem #2: Validate Learning-based Models

☐ Model Validation

- ➢ **Whether a model is effective?**
- ➢ **Whether a model outperforms existing ones?**
- ➢ **Whether a model can adapt to new scenarios?**

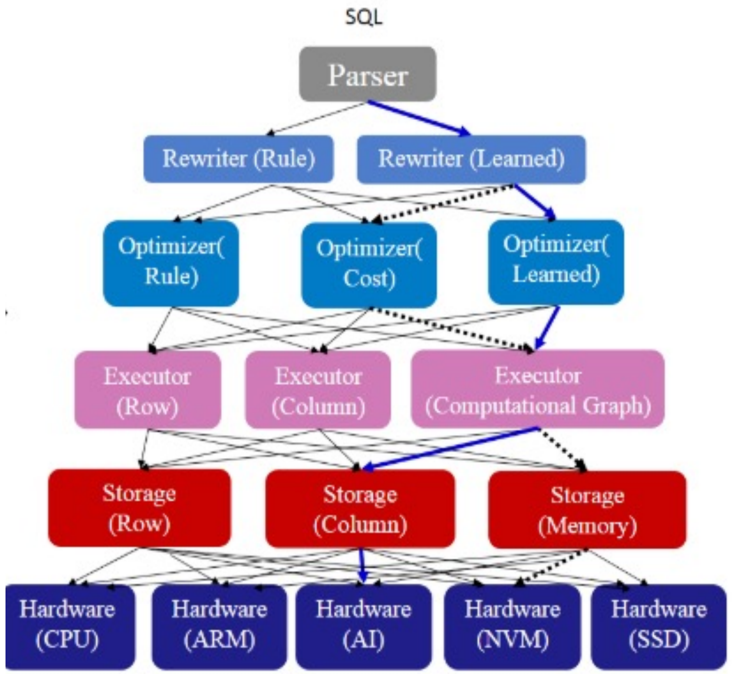
Open Problem #3: One Model Fits Various Scenarios

- **High Adaptability**
  - **Workloads:** query operators; plan structures; underlying data access
  - **Datasets:** tables; columns; data distribution; indexes / views; data updates
  - **DB Instances:** state metrics (DB, resource utilization): hardware configurations
  - **DBMSs:** MySQL; PostgreSQL; MongoDB; Spark
- **Possible Solutions: common knowledge extraction; meta learning**

Open Problem #4: Automatic Learned Model Selection

◻ Automatic Database Assembling

- ➢ Automatically select ML models/algorithms for different tasks
- ➢ Evaluate the overall performance



Database Assembling

Category	Method
Supervised Learning	Linear Regression Logistic Regression Decision Tree Deep Learning
Unsupervised Learning	K-Means Clustering Association Rules Reinforcement Learning
Descriptive Statistics	Count-Min Sketch Data Profiling

The Stack of ML Algorithms

Open Problem #5: Unified Database Optimization

□ Arrange Multiple Database Optimization Tasks

- □ **Multiple Requirements:** (1) Optimizer can produce good plans with not very accurate estimator; (2) Creating indexes may incur the change of optimal knobs
- □ **Hybrid Scheduling:** Arrange different optimization tasks based on the database configuration and workload characters
- □ **Optimization Overhead:** Achieve maximum optimization without competing resources with user processes

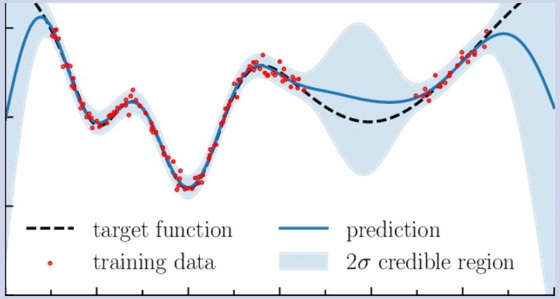
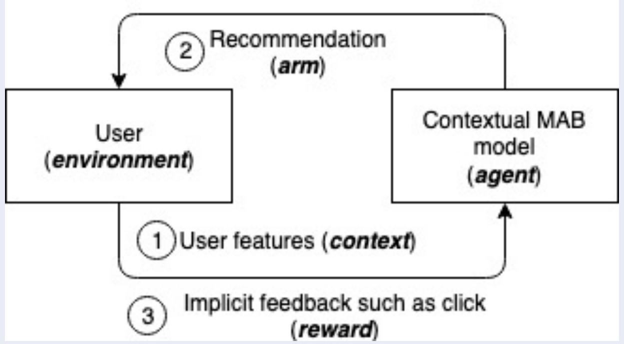
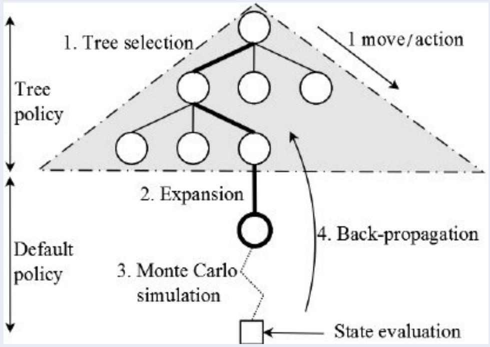
✓ **Challenges:** various task features; correlations between tasks; trend changes

Thanks

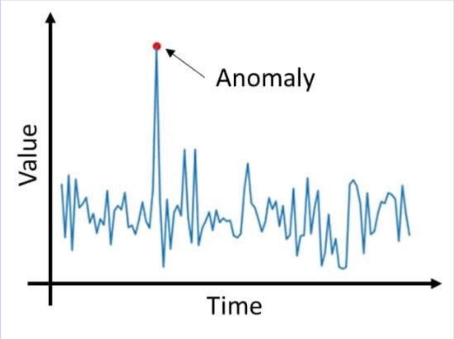
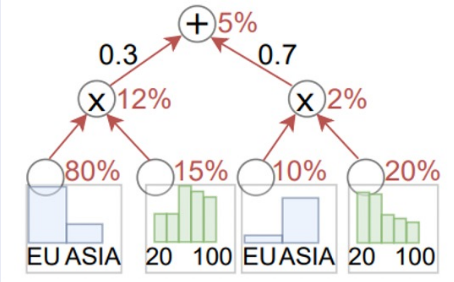
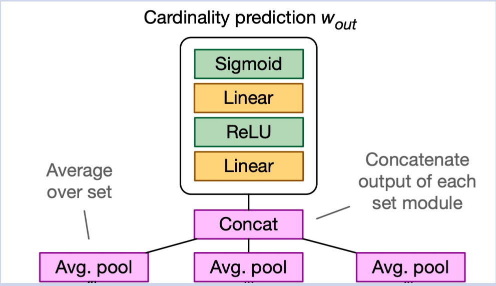
Traditional Methods and Problems

- ☐ **Manual-based Methods (e.g., knob tuning)**
  - It is costly and time-comsuming for DBAs to optimize components
- ☐ **Heuristic Search/Equations/Rules (e.g., cost/view/index estimation)**
  - Produce sub-optimal solutions; cannot learn from historical data; fail to handle complex scenarios
- ☐ **Optimal algorithms (e.g., join order selection, view selection)**
  - Assumptions may not be satisfied in most scenarios

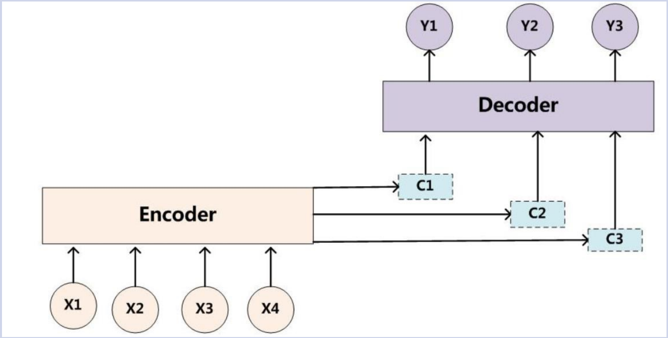
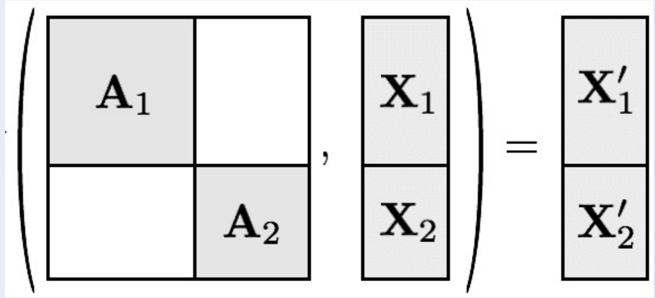
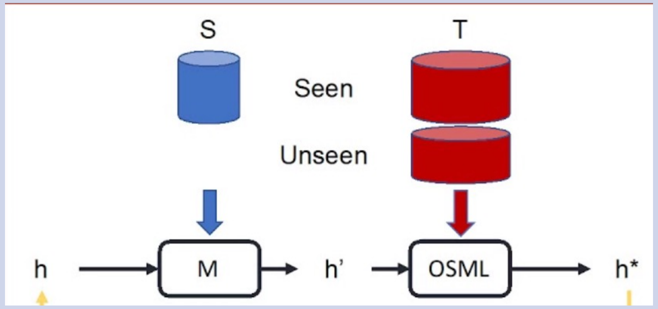
ML Models for Optimization Problems

ML Method	Description	Example	DB Tasks
Gradient-based Methods	Approximate the data distribution with gaussian functions, and select the optimal point by the guidance of gradients		Knob Tuning; Cardinality Estimation
Contextual Multi-armed Bandit	Maximize the reward by repeatedly selecting from a fixed number of arms		Plan Hint; Knob Tuning; MV Selection; Index Selection; Database Partition; Join Order Selection; Workload Schedule
Deep Reinforcement Learning	Learn the selection (<u>actor</u>) or estimation (<u>critic</u>) policy with neural networks		
Monte Carlo Tree Search	Repeated iterations of four steps (<u>selection</u>, <u>expansion</u>, <u>simulation</u>, <u>back-propagation</u>) until termination		Query Rewrite; Online Join Order Selection

ML Models for Regression Problems

ML Method	Description	Example	DB Tasks
Statistical ML	Build a regression model to approximate real distribution based on sampled data		Cardinality Estimation; Trend Prediction
Sum-Product Network	Learn distributions with Sum for different filters and Product for different joins		Cardinality Estimation
Deep Learning (e.g., DNN, CNN, RNN)	Learn the mapping relations from the input features to the targets by graident descent		Knob Tuning; Cardinality Estimation; Cost Estimation

ML Models for Others

<table>
<tr><th>ML Method</th><th>Description</th><th>Example</th><th>DB Tasks</th></tr>
<tr><td>Generative Model (e.g., Encoder-Decoder)</td><td><u>Encode varied-length input features</u> into fixed-length vector with mechanisms like multi-head attention</td><td>
</td><td>MV Selection</td></tr>
<tr><td>Graph Convolutional Network</td><td><u>Encode graph-structured input features</u> with convolutions on the vertex features and their K-hop neighbor vertices</td><td>
</td><td>Query Latency Prediction</td></tr>
<tr><td>Meta Learning</td><td><u>Use the base models to form the target model</u> based on the task similarity and the prediction accuracy during usage</td><td>
</td><td>Knob Tuning</td></tr>
</table>