Artificial Intelligence Meets Database: An Overview

Guoliang Li

Tsinghua University

Artificial Intelligence Meets Database

AI4DB

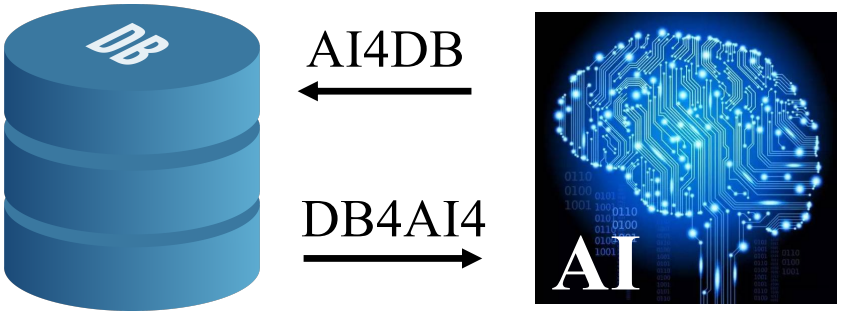
Manual → Automatic

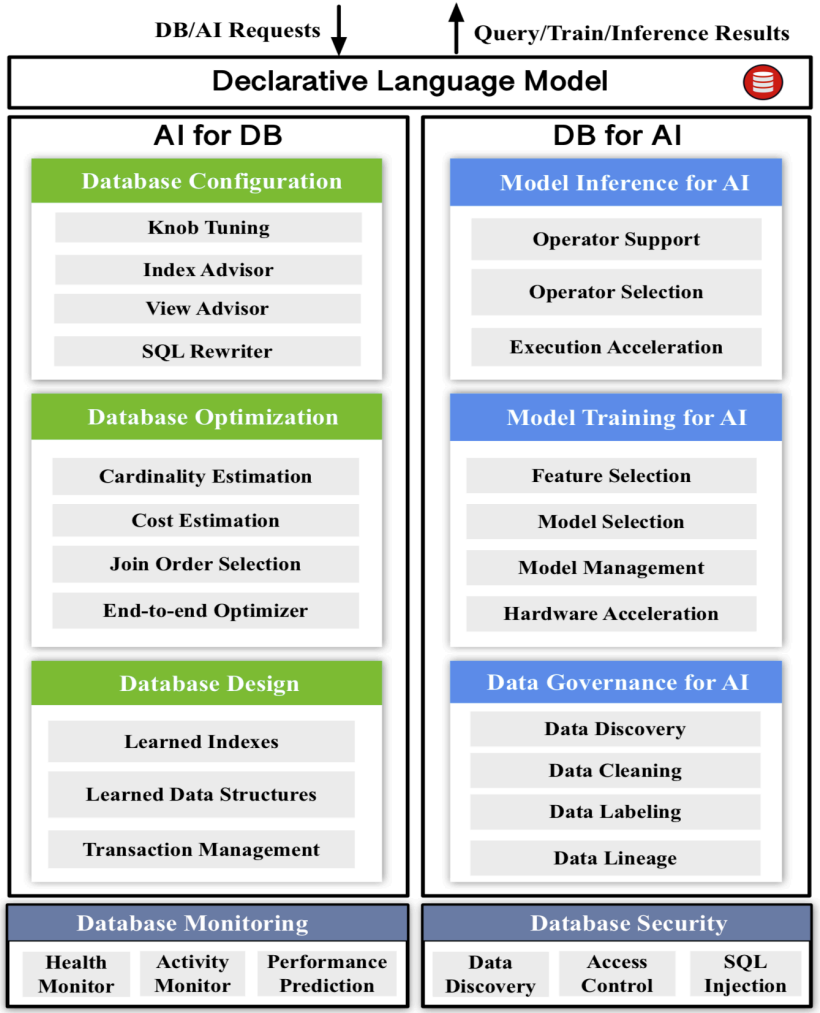
- ☐ Self-optimization
- ☐ Self-configuration
- ☐ Self-monitoring
- ☐ Self-healing
- ☐ Self-security
- ☐ Self-design

DB4AI

AI → as easy as DB

- ☐ Declarative AI
- ☐ AI optimization
- ☐ Data governance
- ☐ Model management
- ☐ AI+DB hybrid model
- ☐ AI+DB hybrid inference





Revisit DB Systems: Driving Factors

database development

File System

Relations → **E**asy to manipulate data

Relational Model, SQL

transactions → **A**tomicity, **C**onsistency, **I**solation, **D**urability

OLTP: rollbacks, triggers, locking, logging, …

data analysis → **H**igh Quality

OLAP: JIT compilation, vectorized execution, …

Scale out → **U**nstructured Data; **H**igh Scalability

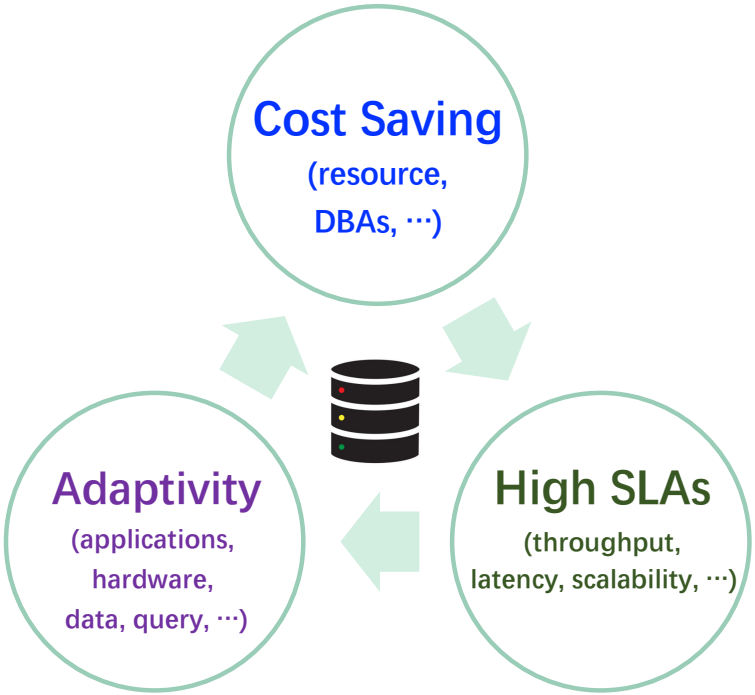
NoSQL: data models, indexing, partitioning, …

transactions +big data → **H**igh Scalability; **ACID**

Distributed OLTP: 2PC, Paxos, Distributed SQL…

DBaaS → **F**lexibility, **C**ost-Saving

Cloud-Native: compute-storage disaggregation, …



Learned Database

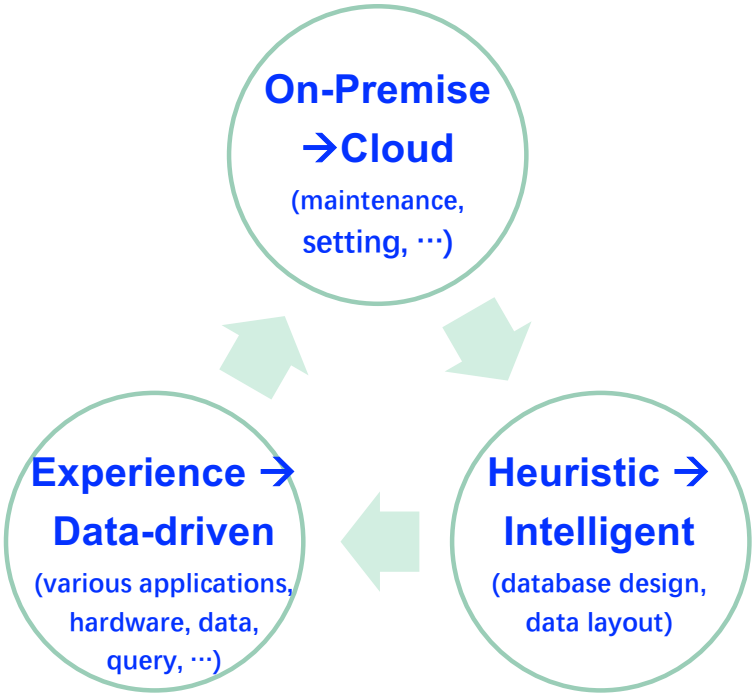
New Opportunities: What Can AI Bring for DB?

- **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

Learned Database

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



Learned Database

Double-Edged Sword: 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;
- **Adaptivity**
- **Training Data**

AI4DB Techniques: Motivation

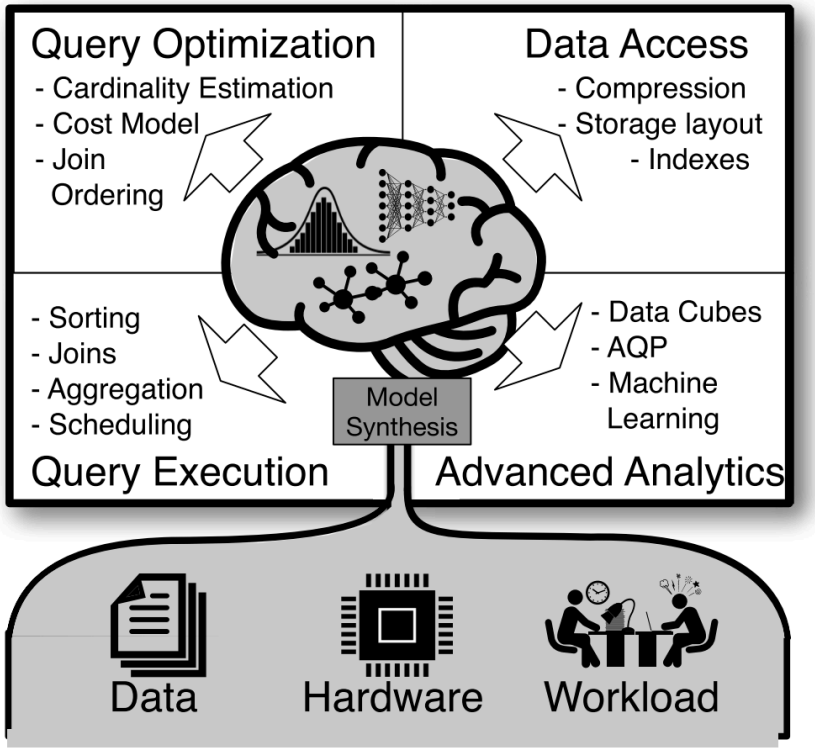
☐ Learned Database Kernel

- **Cardinality/Cost Estimation, Query Rewrite, Plan Generation**

●**Manual → Autonomous**

●**Heuristic → Intelligent**

●**Empirical → Data-Driven**



AI4DB Techniques: Motivation

☐ Learned Database Configuration

- Automate database configurations, e.g., DRL for knob tuning, binary classifier for index selection.

✗ **Labor-intensive tuning**

✗ **Time-consuming tuning**

✓ **Rich tuning experience**

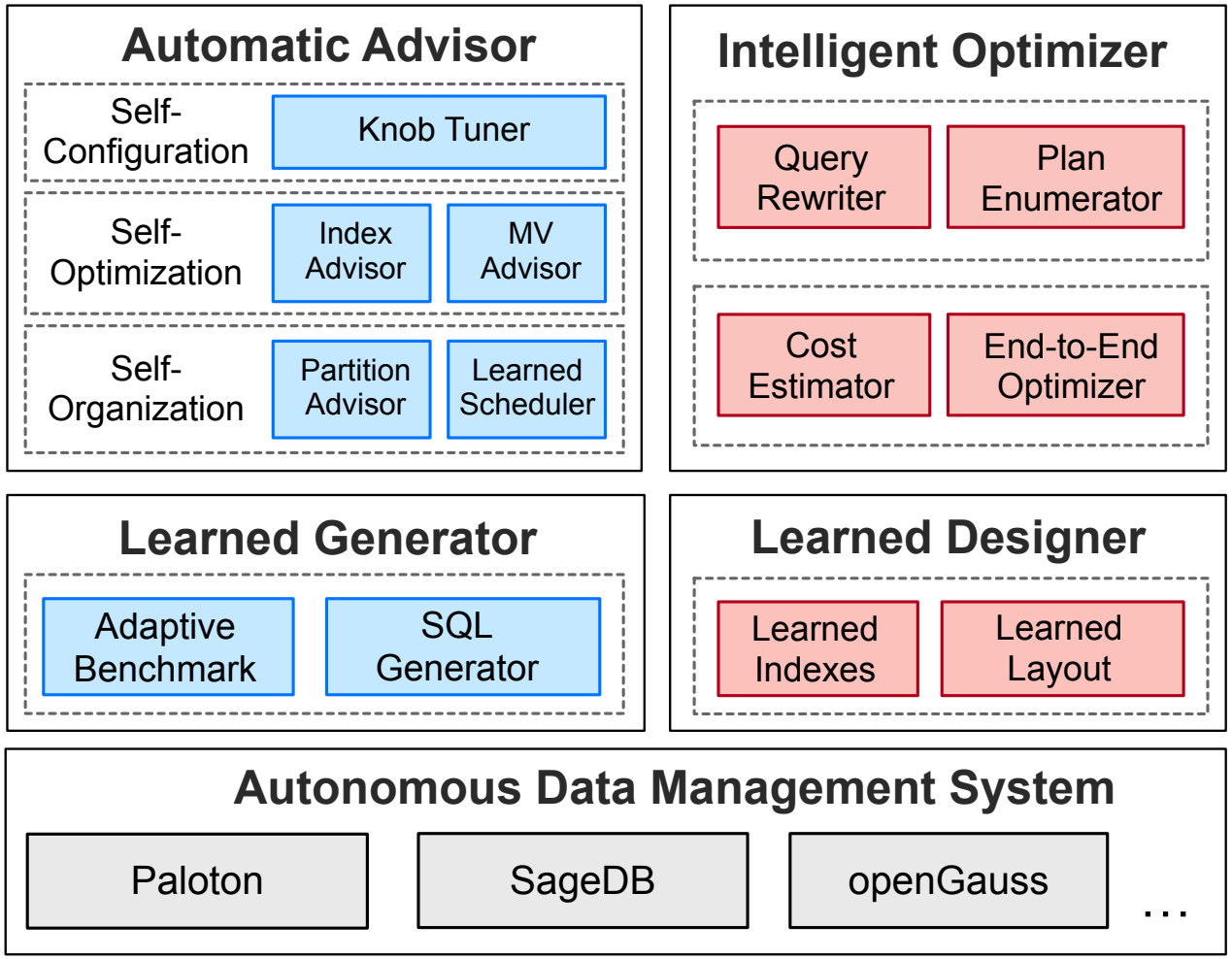
✓ Automatic tuning

✓ Low tuning latency

✗ Adaptivity for different DBs

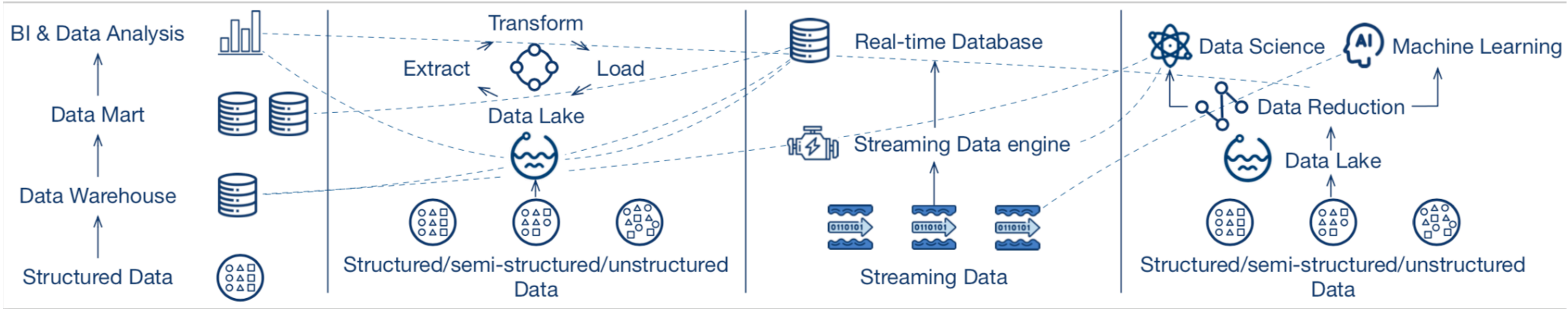
AI4DB Problems

- **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**



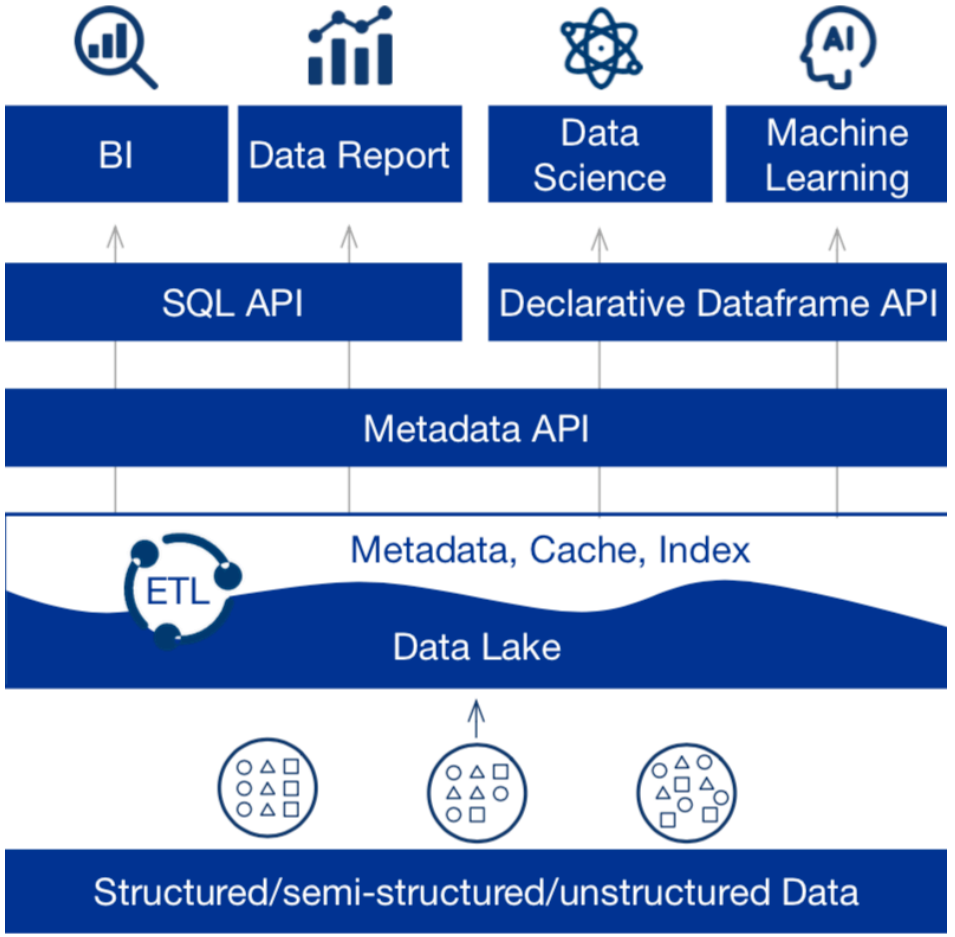
DB4AI Motivation

- Online Inference: T+1 → T+0
- Data Security: ETL → In-DB
- Resource Utilization: data duplicates → one data
- Optimization: DB optimization on learning models
- DB usability: easy to use



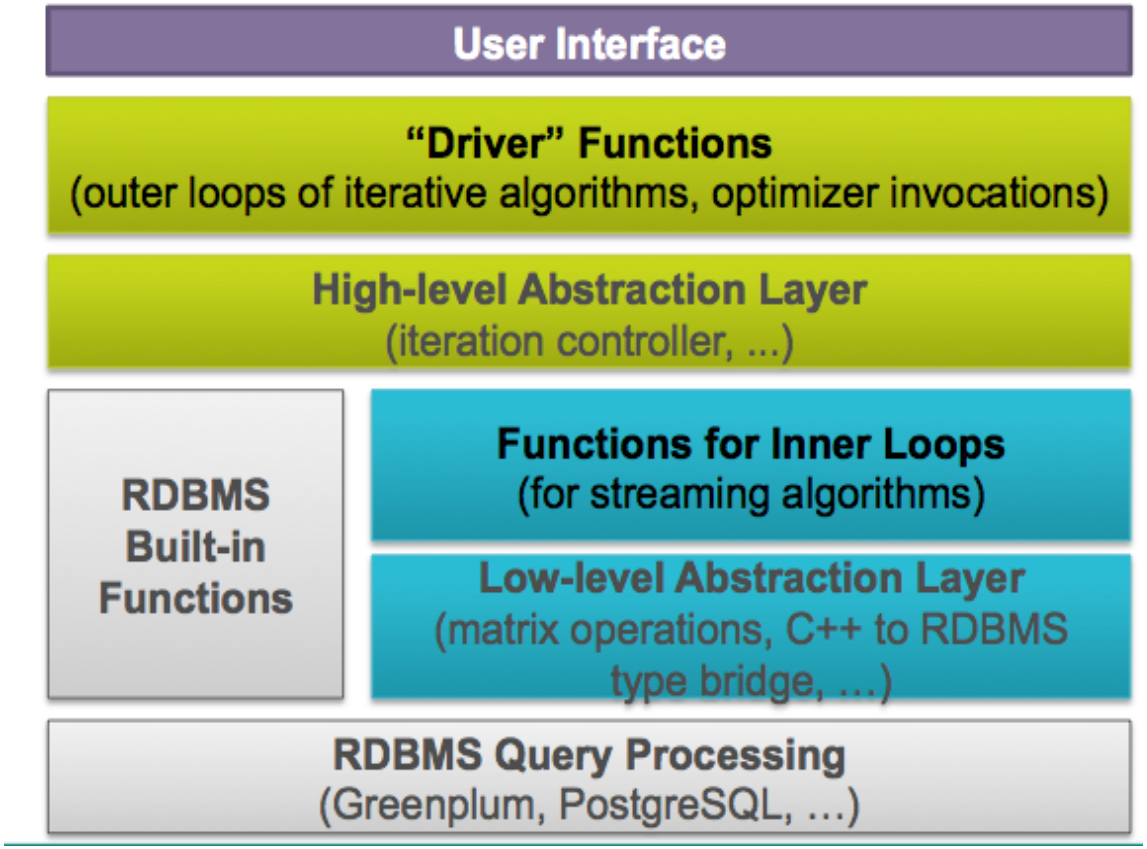
DB4AI Motivation

- One Data
  - Unstructured
  - Structured
  - Semi-structured
- One Analytics
  - SQL
  - Machine Learning
  - Data Science
  - Business Intelligence (BI)



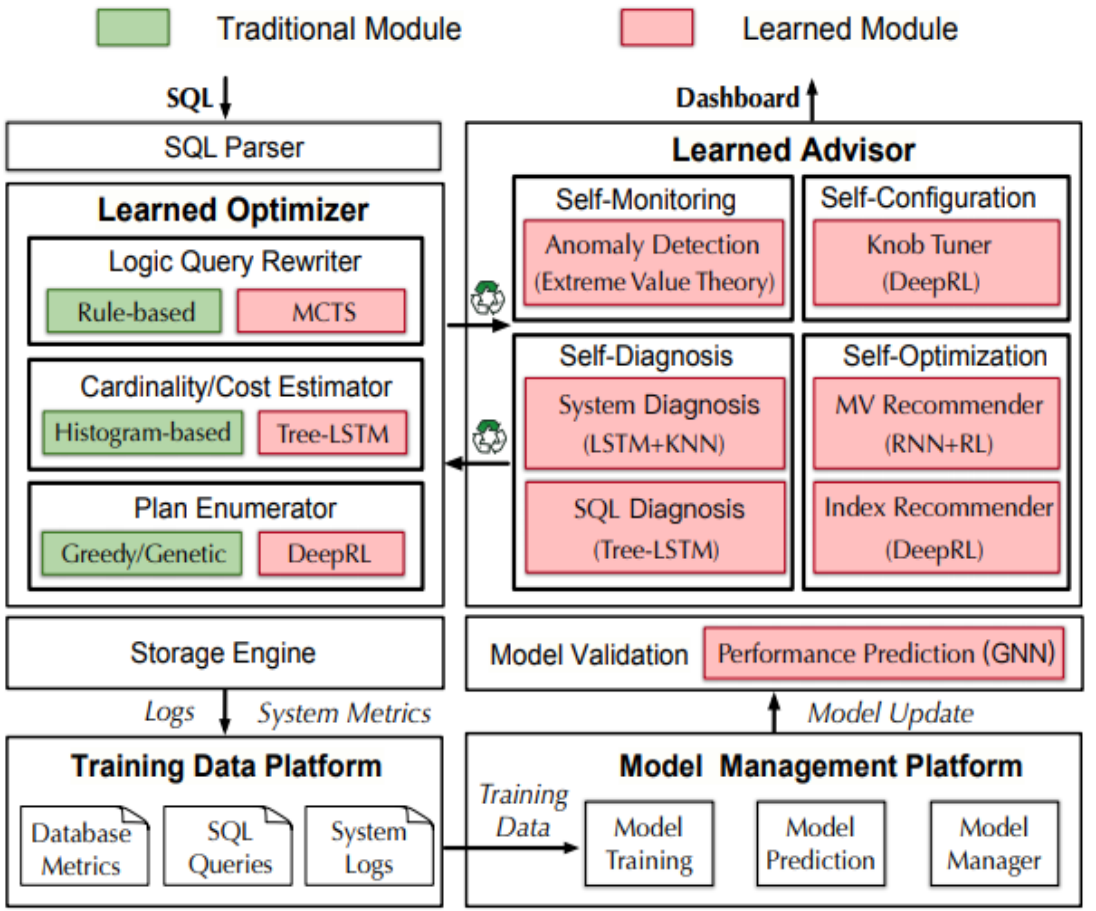
DB4AI

- Declarative AI
  - AI to SQL
  - SQL completeness
  - SQL advisor
- AI optimizations
  - Cost estimation
  - Auto parameter
  - Auto model
  - Parallel computing
- Lightweight In-DB Model
  - Training
  - Inference



Autonomous DB System Architecture

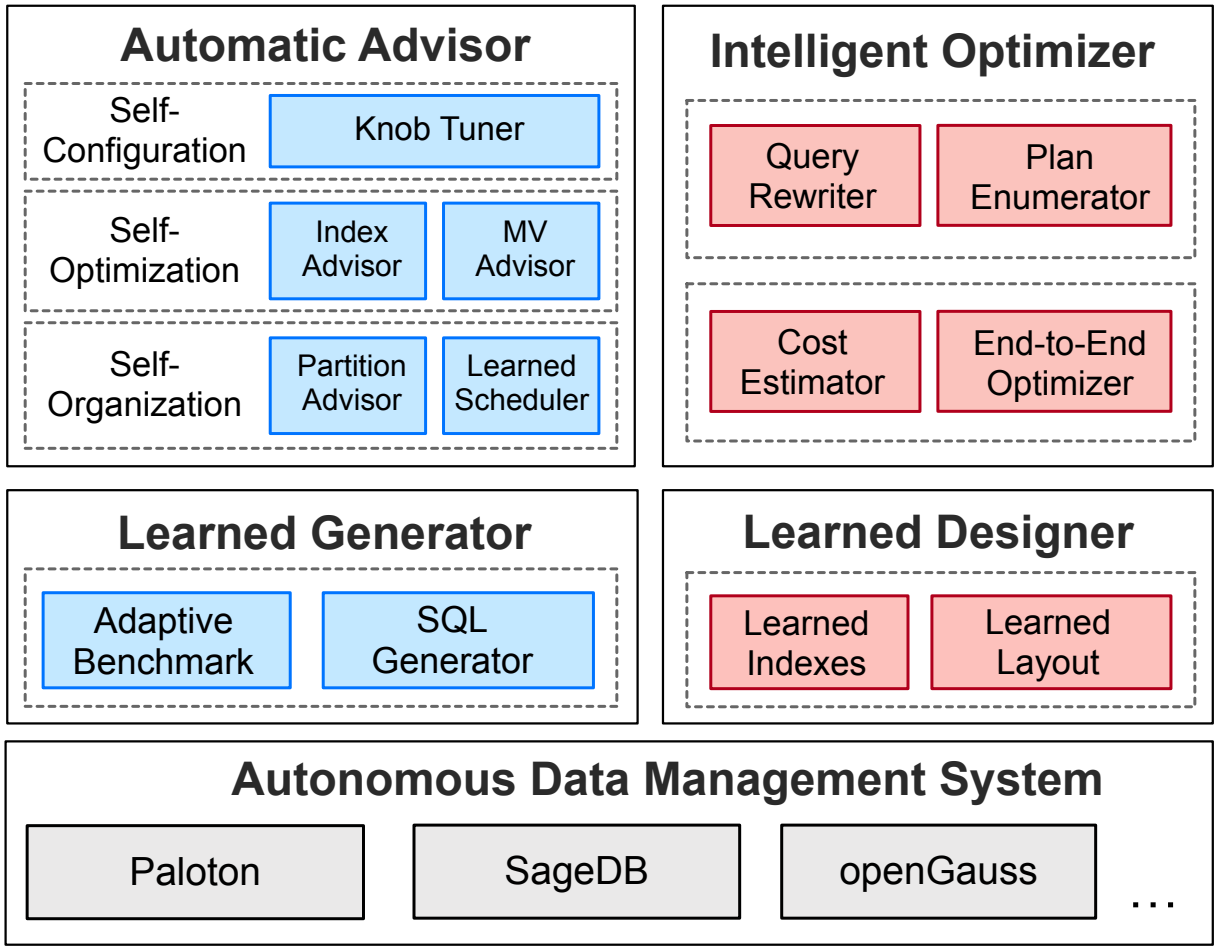
- **Learned Optimizer**
  - Cost Estimation (Tree-LSTM)
  - Logical Optimization (Tree Search)
  - Physical Optimization (RL)
- **Learned Advisor**
  - Monitoring/Diagnosis (LSTM)
  - Configuration (DRL)
  - Optimization (DRL)
- **Model Validator (GNN)**
- **Training-Data/Model Platform**



Guoliang Li. openGauss: An Autonomous Database System. VLDB 2021

AI4DB: 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**



Overview of AI4DB

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

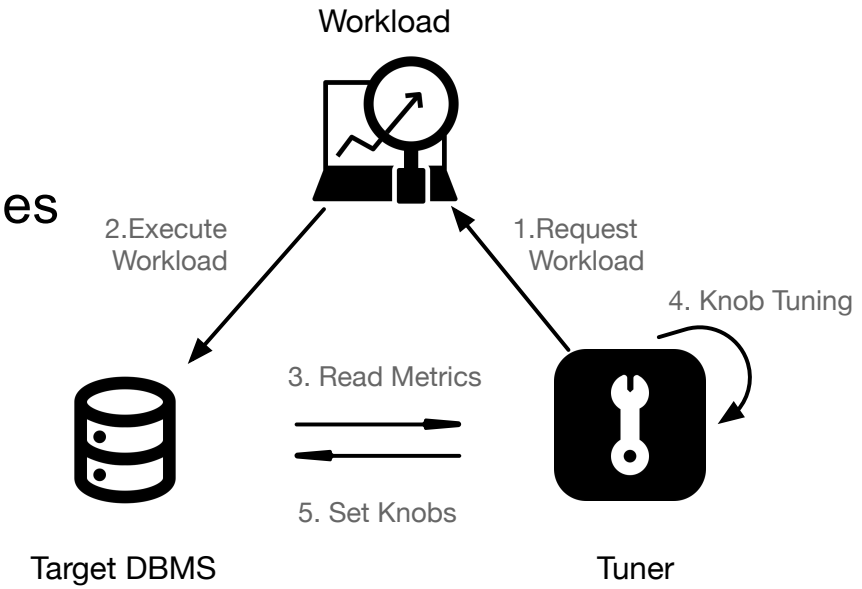
Method	Task	Target	Training Data
Clustering Algorithm	Trend Forecast	High accuracy	Huge
Reinforcement Learning	Workload Scheduling	High performance	--

Learned Advisors

- ☐ Learned Knob Tuning
- ☐ Learned Index Advisor
- ☐ Learned View Advisor
- ☐ Learned Partition Advisor
- ☐ Learned Data Generation

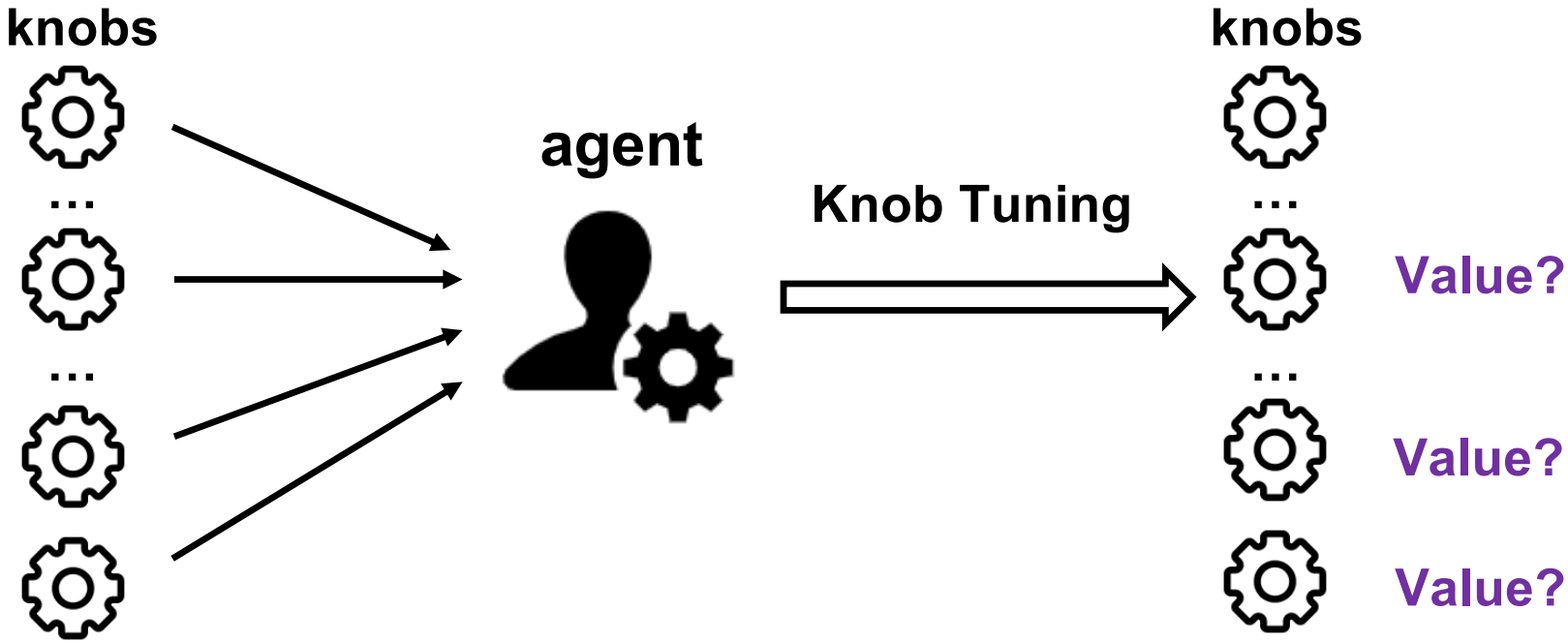
Knob Tuning

- **A Constrained Optimization Problem**
  - Given a suite of knobs B and a target T, knob tuning aims to find the optimal values of B, so as to meet T for the incoming workload.
- **Knobs**
  - concurrency control, optimizer settings
  - memory management, background processes
- **Targets**
  - Performance (throughput, latency)
  - Resource Usage (e.g., CPU utilization)



Offline Optimization for Knob Tuning

Problem Definition: Consider a database with different workloads, the target is to find the optimal knob settings to meet required SLA (service-level agreement).

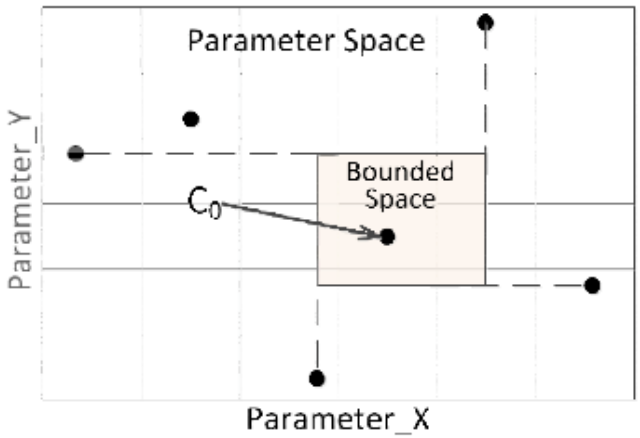


Offline Optimization for Knob Tuning

- **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 the metrics.
  - DBMSs have numerous system knobs with continuous values, which makes it harder to find optimal knobs.

Traditional Knob Tuning Methods

- **Motivation: Most users only utilize default knob settings and cause performance regression**
- **Basic Idea: Greedily select local-optimal knob settings with bound-and-search algorithm**
- **Challenge: Optimal settings change with tuning goals and workloads**
- **Solutions:**
  - **Sample Phase:** Divide each knob range into k intervals and sample k settings that cover all the value ranges
  - **Search Phase:** Select the best sampled setting and build search space around the best setting



Random Sampling: Some important settings may not be sampled

Yuqing Zhu et al. BestConfig: Tapping the Performance Potential of Systems via Automatic Configuration Tuning. In SoCC, 2017.

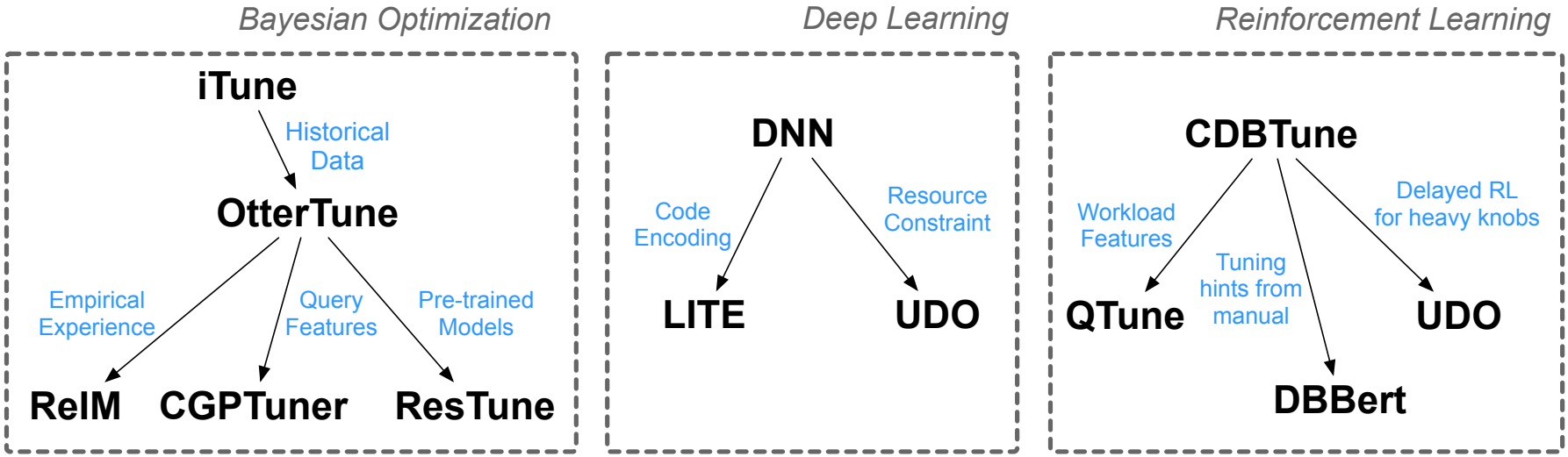
Learning-based Knob Tuning

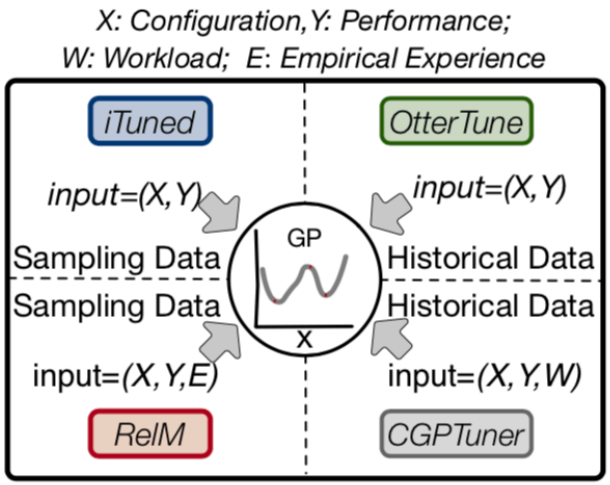
- **Why heuristics → Machine Learning ?**
  - **A large number of configuration knobs**
    - Total > 400
    - *Heuristic Method:* waste much time in search from huge knob space
  - **Knobs control nearly every aspect and have complex correlations**
    - One-knob-at-a-time is inefficient
    - *Heuristic:* The relations are non-monotonic
  - **Learn from the historical tuning**
    - *Heuristic:* Restart tuning from scratch each time

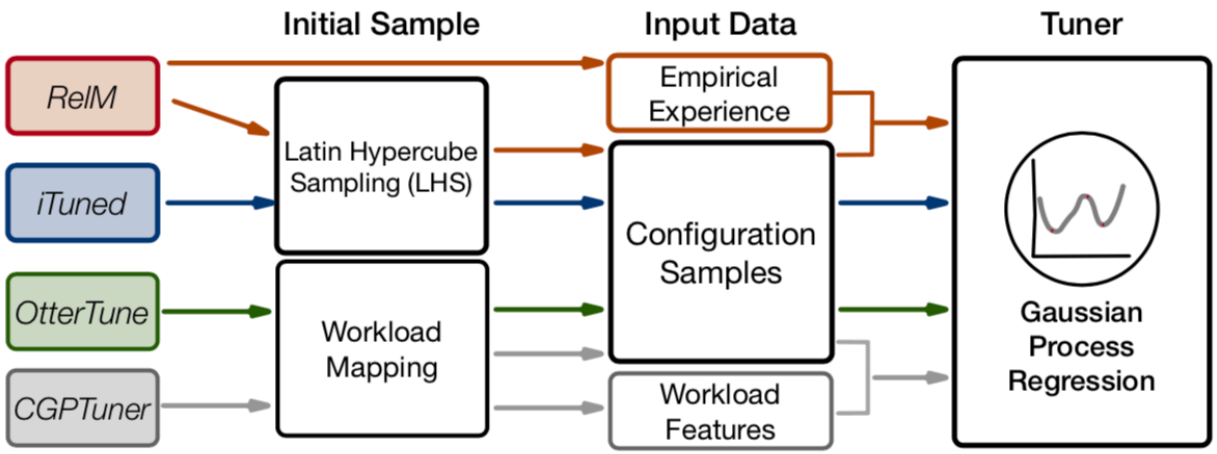
Hi, list. I've just upgraded pgsql from 8.3 to 8.4. I've used pgtune before and everything worked fine for me. And now i have **~93%** **cpu** load. Here's changed values of config:

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

Learning-based Knob Tuning

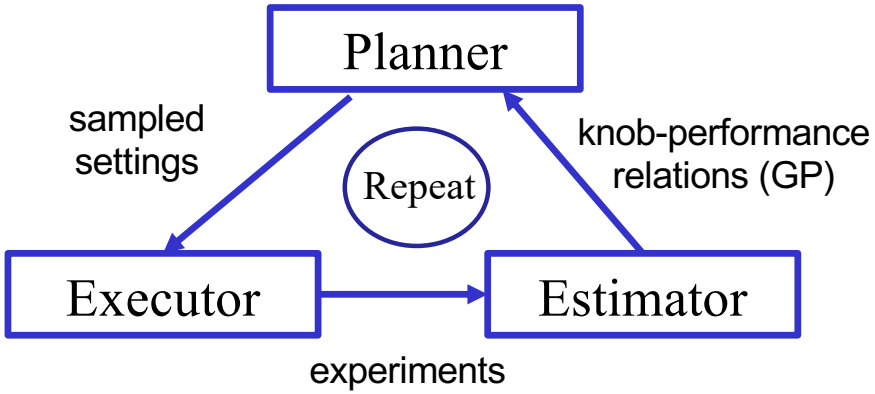


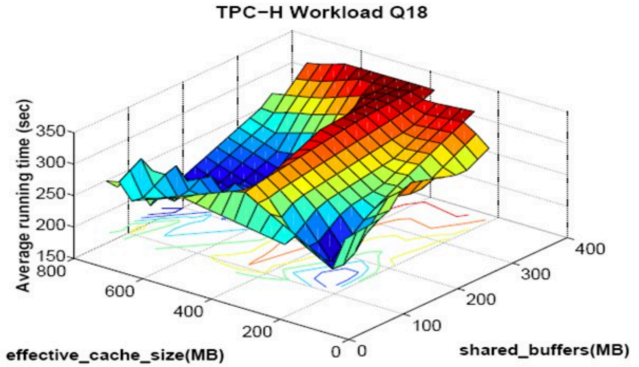




(1.1) Bayesian Optimization for Knob Tuning

- **Motivation: Only a few knobs have significant effects to the performance**
- **Basic Idea: Explore the knob-performance relations by experiments**
- **Challenge: Identify important knobs and their values efficiently**
- **Solution:**
  - **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



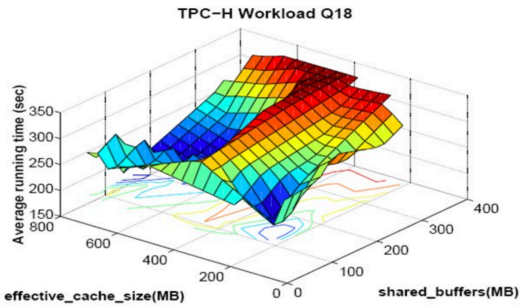


knob-performance relations

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

(1.1) Bayesian Optimization for Knob Tuning

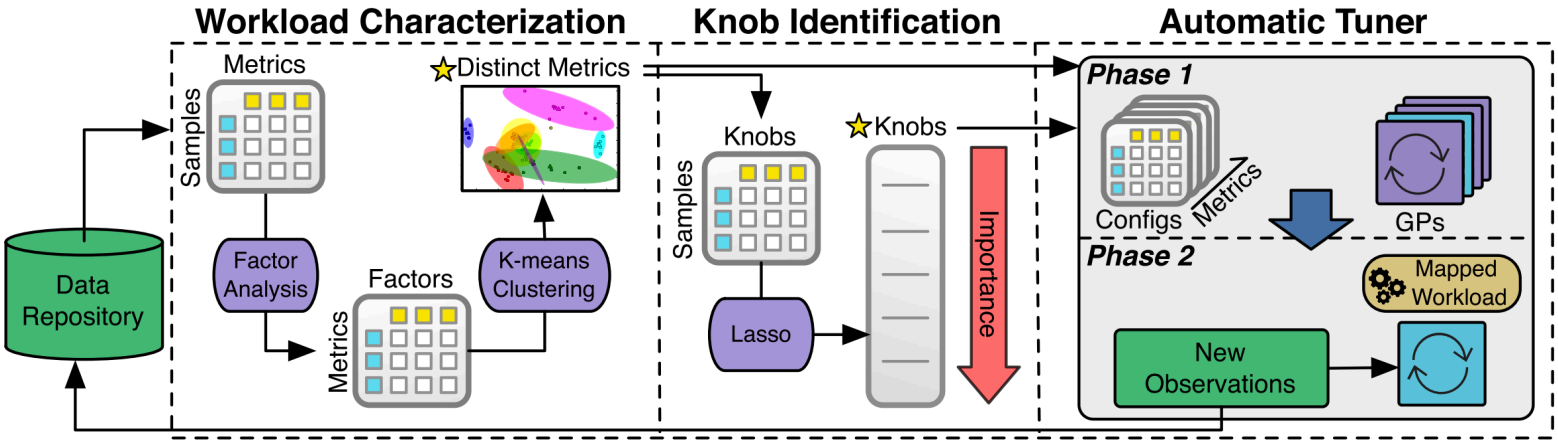
- **Motivation: Only a few knobs have significant effects to the performance**
- **Basic Idea: Explore the knob-performance relations by experiments**
- **Challenge: Identify important knobs and their values within hours**
- **Solution:**
  - **Planner:** Adaptively sample some knob settings
  - **Executor:** 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
- **Limitations**
  - **Sampling configurations from scratch is inefficient**
  - **Knob-performance relations are extremely complex**
  - **Important workload features are not utilized**



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

(1.2) Bayesian Optimization + Historical Data

- **Data-driven: Optimize tuning performance 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



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

(1.3) Bayesian Optimization + Empirical Experience

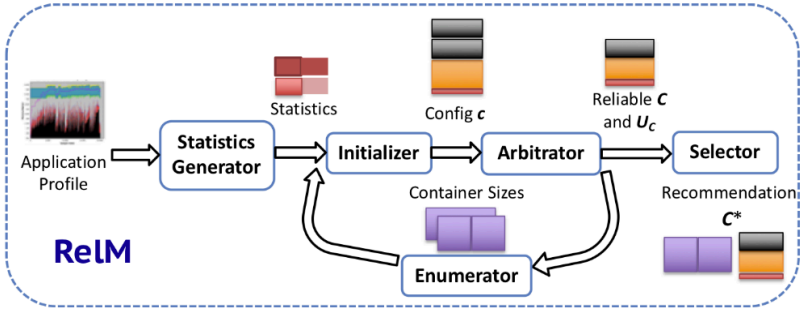
- **Motivation: Expert experience can make learned tuning more robust**
  - e.g., limit the minimal shard buffer size
- **Basic Idea: Utilize expert experience to optimize tuning**
- **Solution**
  - **Empirically compute input features at resource/APP/VM levels**

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

$\boldsymbol{x}$: Tested knob setting
$\boldsymbol{M_i}$: Code overhead value
$\boldsymbol{m_c}$: Required cache storage
$\boldsymbol{m_o}$: GC settings

  - **Rely on empirical features to estimate tuning performance**

(1) Input: Empirical features, Initialized knob values;
(2) Model: Gaussian Process;
(3) Target: Tuning Performance.



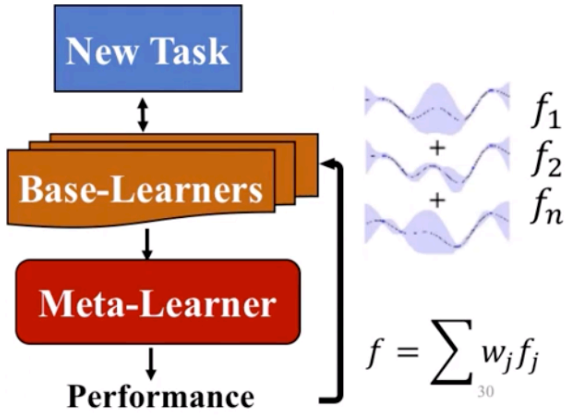
Mayuresh Kunjir, Shivnath Babu. Black or White? How to Develop an AutoTuner for Memory-based Analytics. SIGMOD 2020.

(1.4) Bayesian Optimization + Pretrained Models

- **Motivation: Learning-based tuning is hard to migrate to new scenarios**
- **Basic Idea: Improve migration capability with pre-trained tuning models**
- **Solution:**
  - Characterize the common workload features
    - Reserved SQL words (e.g., SELECT, DISTINCT)
  - Cluster tuning models on historical workloads to generate Base Leaners;
  - For a New Task, generate *Meta Learner* based on the Base Leaners (similarity weight: g_i);
    - The Meta Learner *M* is a gaussian process model:

$$\textit{mean value}\ \ \mu_M(\theta) = \frac{\sum_{i=1}^{T+1} g_i \mu_i(\theta)}{\sum_{i=1}^{T+1} g_i} \qquad \textit{variance}\ \ \sigma_M^2(\theta) = \sum_{i=1}^{T+1} v_i \sigma_i^2(\theta),$$

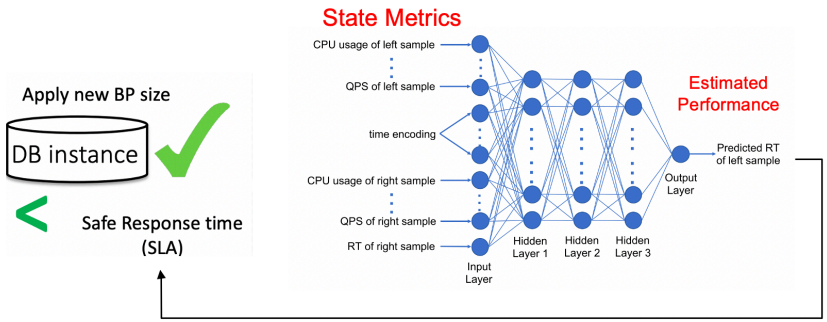
  - Fine-tune the *Meta Learner* by running the new workload;
  - Recommend promising knobs with Meta Learner.



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

(2.1) Deep Learning for Knob Tuning

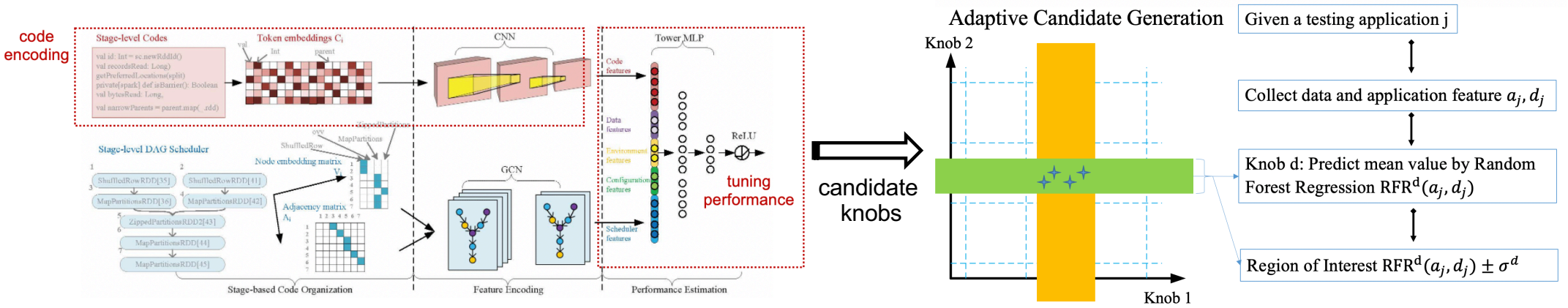
- **Motivation: Expensive to run workloads for evaluating tuning effects**
- **Basic Idea: Estimate tuning effects without running workloads**
- **Challenge: Many metrics affect the performance**
- **Solution:**
  - Collect DB metrics: [logical-read, QPS, CPU usage, response time];
  - Initialize a buffer 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.



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

(2.2) Deep Learning + Code Encoding

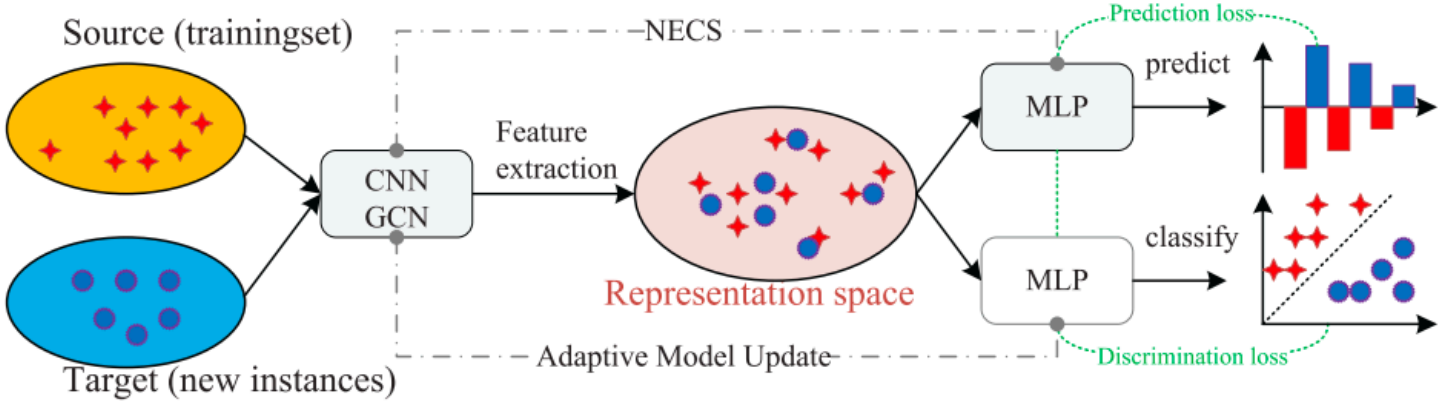
- **Motivation: Spark code involves complex semantics, and it is costly to migrate tuning models from small datasets to large datasets**
- **Basic Idea: Restrict the tuning region by predicting the performance**
  - **Knob Sampling:** Sample candidate knob settings based on the data and code features;
  - **Code Instrumentation:** Enrich semantic features by adding the Spark API;
  - **Performance Prediction:** Predict the performance with encoded code, data, knob, DAG.



Chen Lin, Junqing Zhuang, Jiadong Feng, Hui Li, Xuanhe Zhou, Guoliang Li. Adaptive code learning for Spark configuration tuning. ICDE, 2022.

(2.2) Deep Learning + Code Encoding

- **Motivation: Spark code involves complex semantics, and it is costly to migrate tuning models from small datasets to large datasets**
- **Basic Idea: Restrict the tuning region by predicting the performance**
  - **Knob Sampling:** Sample candidate knob settings based on the data and code features;
  - **Code Instrumentation:** Enrich the code features by adding the Spark API tokens;
  - **Performance Prediction:** Predict the performance with *encoded code*, *data*, *knob*, DAG features;
  - **Generalization to Big Datasets:** When dataset changes, utilize adversarial learning to capture the domain-invariant features and update the performance model with newly collected samples.

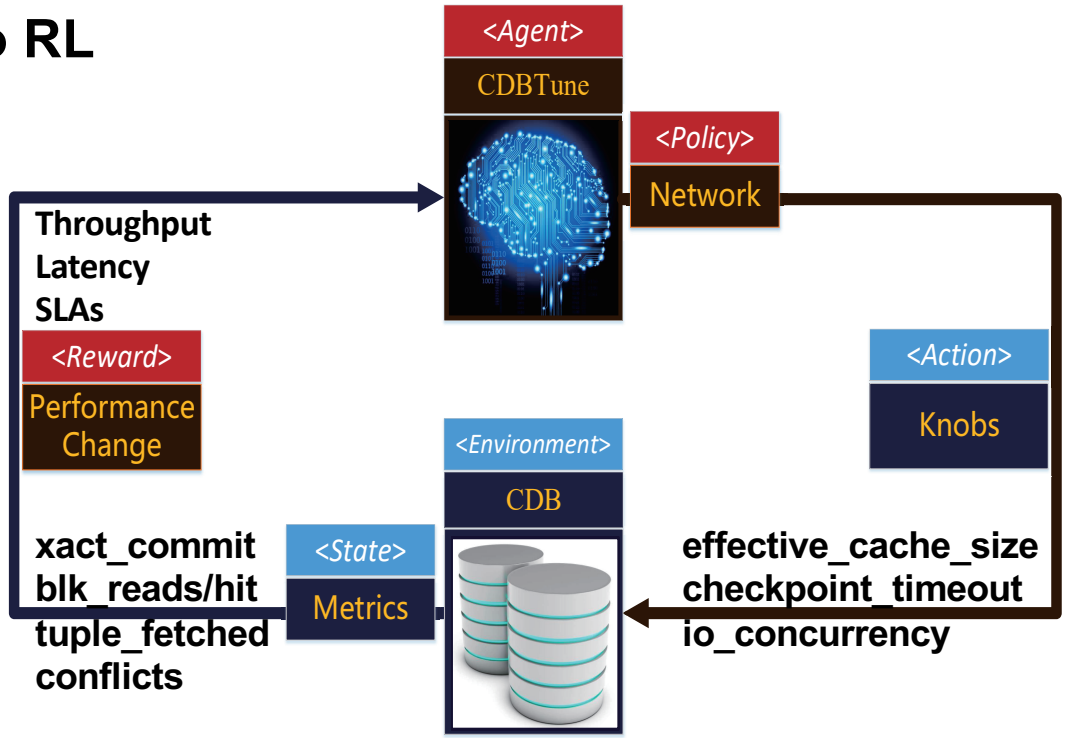


Chen Lin, Junqing Zhuang, Jiadong Feng, Hui Li, Xuanhe Zhou, Guoliang Li. Adaptive code learning for Spark configuration tuning. ICDE, 2022.

(3.1) Reinforcement Learning for Knob Tuning

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

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.

(3.1) Reinforcement Learning for Knob Tuning

- **Issue1: How to choose an appropriate RL approach**
  - **Challenge: Many continuous runtime 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.

(3.1) Reinforcement Learning for Knob Tuning

Issue2: How to train an RL-based Model (e.g., DDPG)

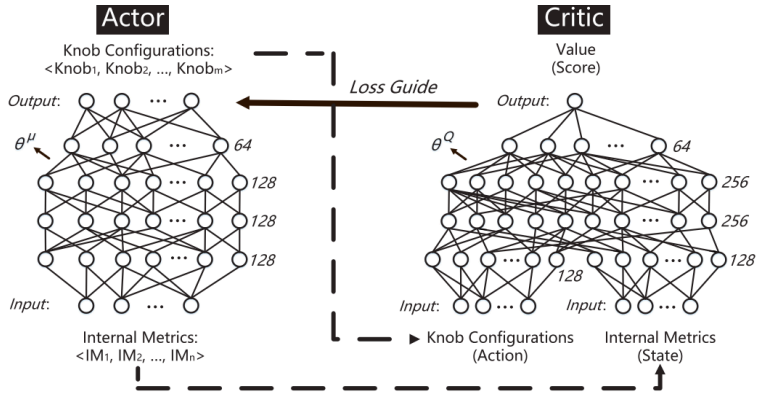
Challenge: Optimize the tuning strategy with execution rewards

- **Design effective reward function *r* *(current benefit)*:**

$$r = \begin{cases} ((1+\Delta_{t\to 0})^2 - 1)|1+\Delta_{t\to t-1}|, \Delta_{t\to 0} > 0 \\ -((1-\Delta_{t\to 0})^2 - 1)|1-\Delta_{t\to t-1}|, \Delta_{t\to 0} \leqslant 0 \end{cases}$$

Improvement over default setting

Improvement over (t-1) setting



- **Actor Network Training:** Update with the score estimated by the Critic

$$\nabla_{\theta^{\pi_A}} \pi_A = \nabla_{A_i} Q(S_i', A_i|\pi_C) \cdot \nabla_{\theta^{\pi_A}} \pi_A(S_i'|\theta^{\pi_A})$$

$Q(S_i', A_i|\pi_C)$ → The output of Critic

- **Critic Network Training:** Update with accumulated *long-term benefit*:

$$L = (Q(S_i', A_i|\pi_C) - Y_i)^2$$

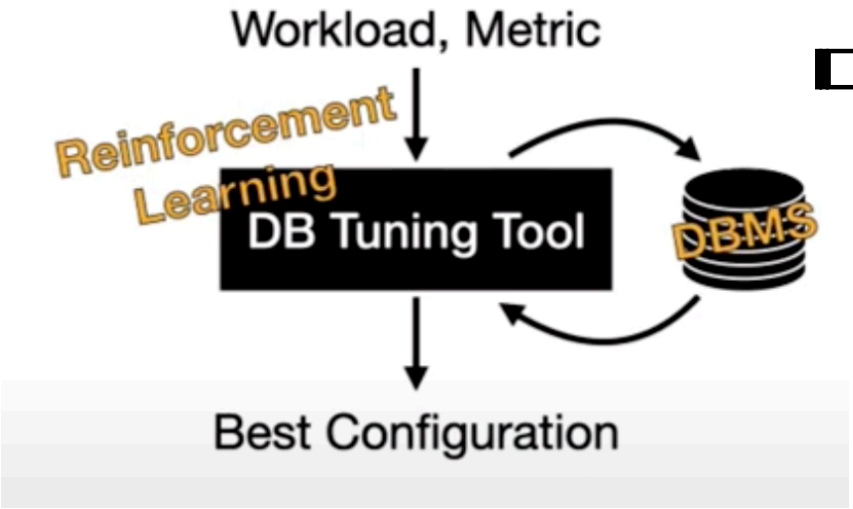
$$Y_i = R_i + \tau \cdot Q(S_{i+1}', \pi_A(S_{i+1}'|\theta^{\pi_A})|\pi_C)$$

Y_i → Long-term benefit based on the reward

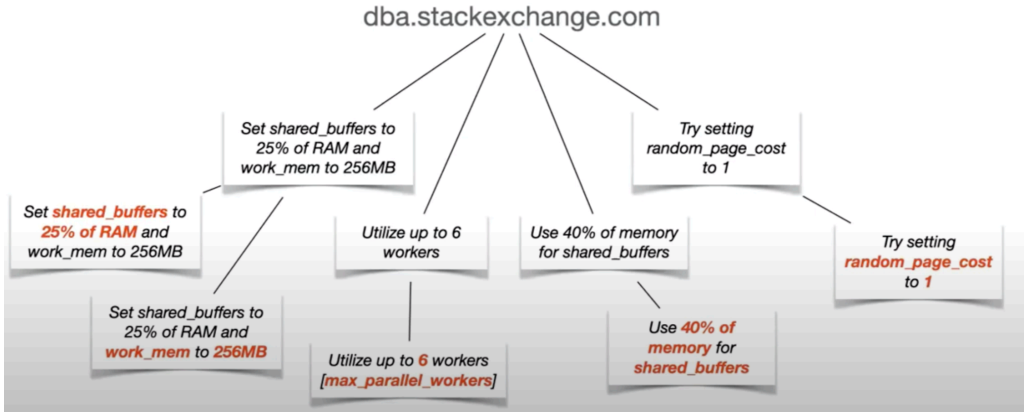
Guoliang Li, Xuanhe Zhou, Shifu Li, Bo Gao. QTune: A Query-Aware Database Tuning System with Deep Reinforcement Learning. VLDB 2019.

(3.2) Reinforcement Learning + Tuning Hints

- Limitations in RL-based tuning
  - High tuning overhead
  - Require DBAs (e.g., decide the knob ranges)



- Basic Idea: Tuning hints from manual
  - (1) Collect tuning hints from website



Extract hints from manual

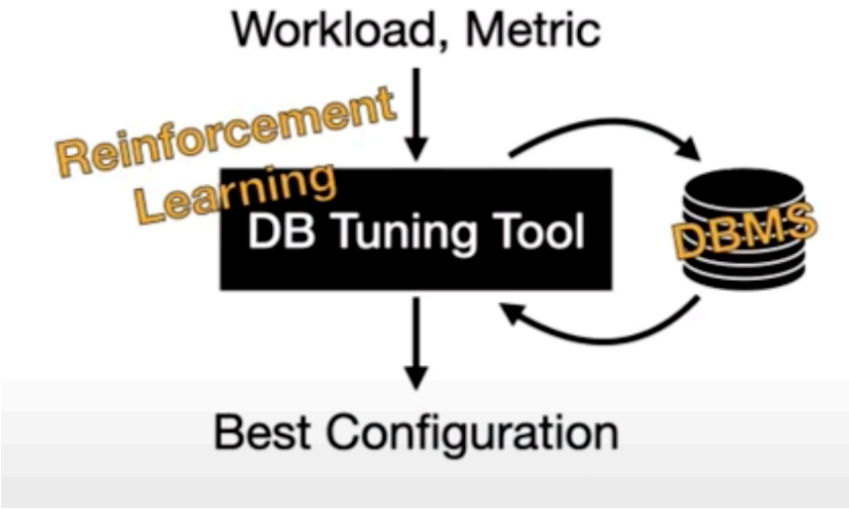
$$Parameter = Value\ [* System\ Property][* Constant]$$

Given in Text RAM/Disk/Cores

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

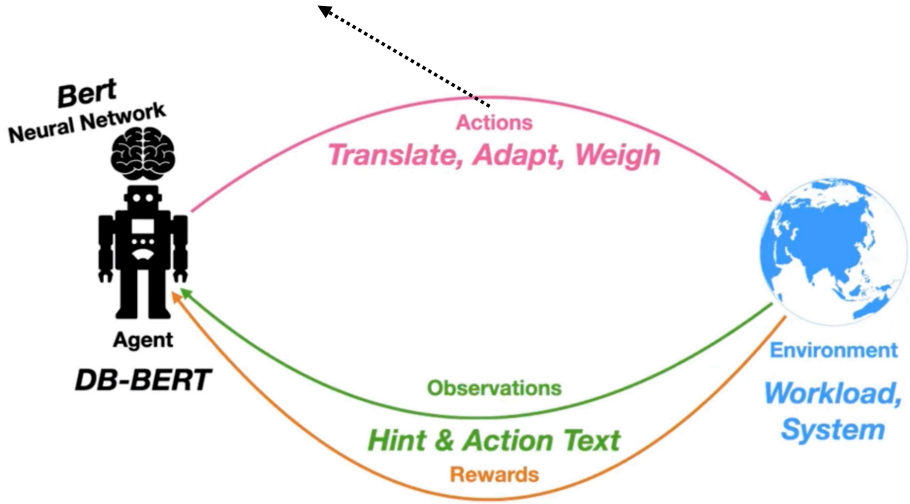
(3.2) Reinforcement Learning + Tuning Hints

- Limitations in RL-based tuning
  - High tuning overhead
  - Require DBAs (e.g., decide the knob ranges)



- Basic Idea: Tuning hints from manual
  (2) Apply the tuning hints with the reinforcement learning model

$$Parameter = Value\ [* System\ Property][* Constant]$$



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

Summarization of Learned Knob Tuning

	Quality	Training Efficiency	Training Data	Adaptivity
Gaussian Process (historical data)	✓	--	✓✓	✓
Gaussian Process (+ empirical features)	✓	✓	✓	✓✓
Gaussian Process (pre-trained models)	✓	✓	✓✓	✓✓
Deep Learning (resource issues)	✓	✓	✓✓	✓
Deep Learning (+ code encoding)	✓✓	✓	✓✓	✓✓
Reinforcement Learning (from scratch)	✓✓	--	No Prepared Data	✓
Reinforcement Learning (+ tuning hints)	✓	--	✓✓✓	✓✓

Take-aways of Knob Tuning

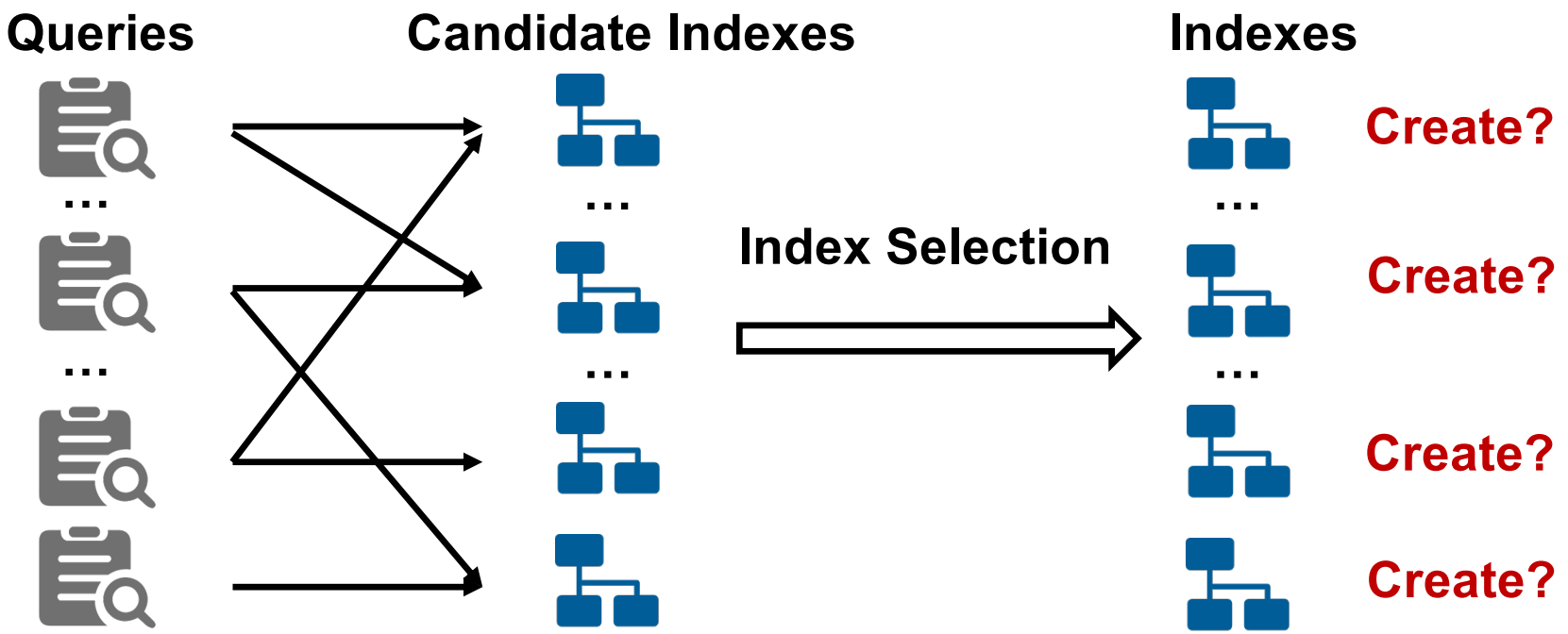
- **Gradient-based GP methods reduce the tuning complexity by filtering out unimporant features.** However, it heavily relies on training data, and requires other migration techniques to adapt to new scenarios
- **Deep learning method considers both query performance and resource utilization.** And they can significantly reduce the tuning overhead.
- **Reinforcement learning methods take longest training time, e.g., hours, from scratch.** It 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

Learned Advisors

- ☐ Learned Knob Tuning
- ☐ Learned Index Advisor
- ☐ Learned View Advisor
- ☐ Learned Partition Advisor
- ☐ Learned Data Generation

Index Management

Problem Definition: Given a set of queries W and resource constraint D (e.g., disk limit), create a collection of indexes so as to optimize the execution of these queries under the constraint D. → NP-hard

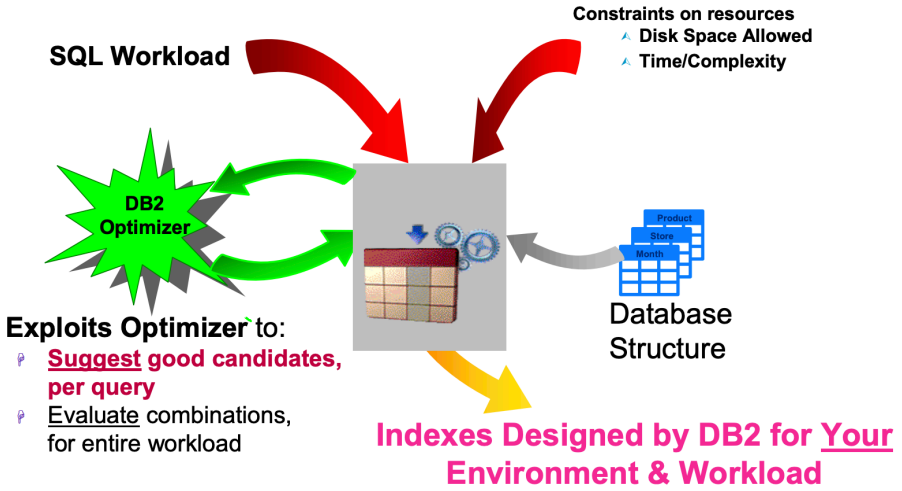


Index Management

- Index Benefit Estimation
  - The benefit of building an index on a column
- Index Selection
  - Column selection
  - Index-type selection, e.g., B-tree, Hash, bitmap
- Index Update
  - Adding or removing an index

Heuristic Index Selection

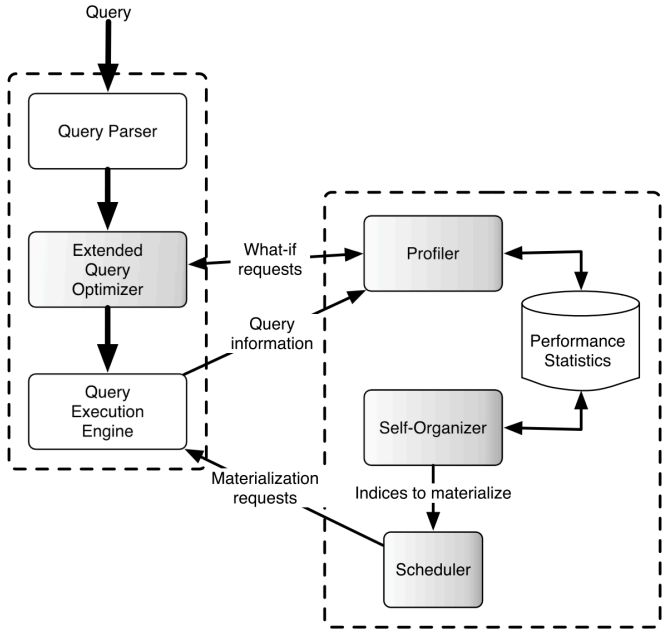
- Motivation: Proper indexes can significantly improve the performance
- Basic Idea: Model index selection as a knapsack problem and heuristically find the best indexes under disk limit
- Challenge: There are correlations between indexes (e.g., index sizes)
- Solution:
  - Model index selection as a knapsack problem
    - Item: Candidate index
    - Item weight: Index size
    - Value: Cost reduced by the index
  - Heuristically select the highest-benefit indexes
    - Benefit: Cost Reduction / Index Size by optimizer



G. Valentin, M. Zuliani, D. C. Zilio, et al. *DB2 advisor: An optimizer smart enough to recommend its own indexes*. In ICDE 2000.

Heuristic Index Selection for Dynamic Workloads

- Motivation: Performance gets unstable for dynamic workloads
- Basic Idea: Split workloads into epochs and finetune indexes for each epoch
- Challenge: Online index update for new queries
- Solution:
  - Divide a workload into epochs of queries
  - Generate candidate indexes for each new query
    - Index Benefit: average latency reduction for the queries within the same epoch
    - Benefit Estimation: Estimate the index benefit through what-if call (*similar queries have similar index benefits*)
    - Update the index set and statistics
  - Create indexes with highest index benefit at each epoch



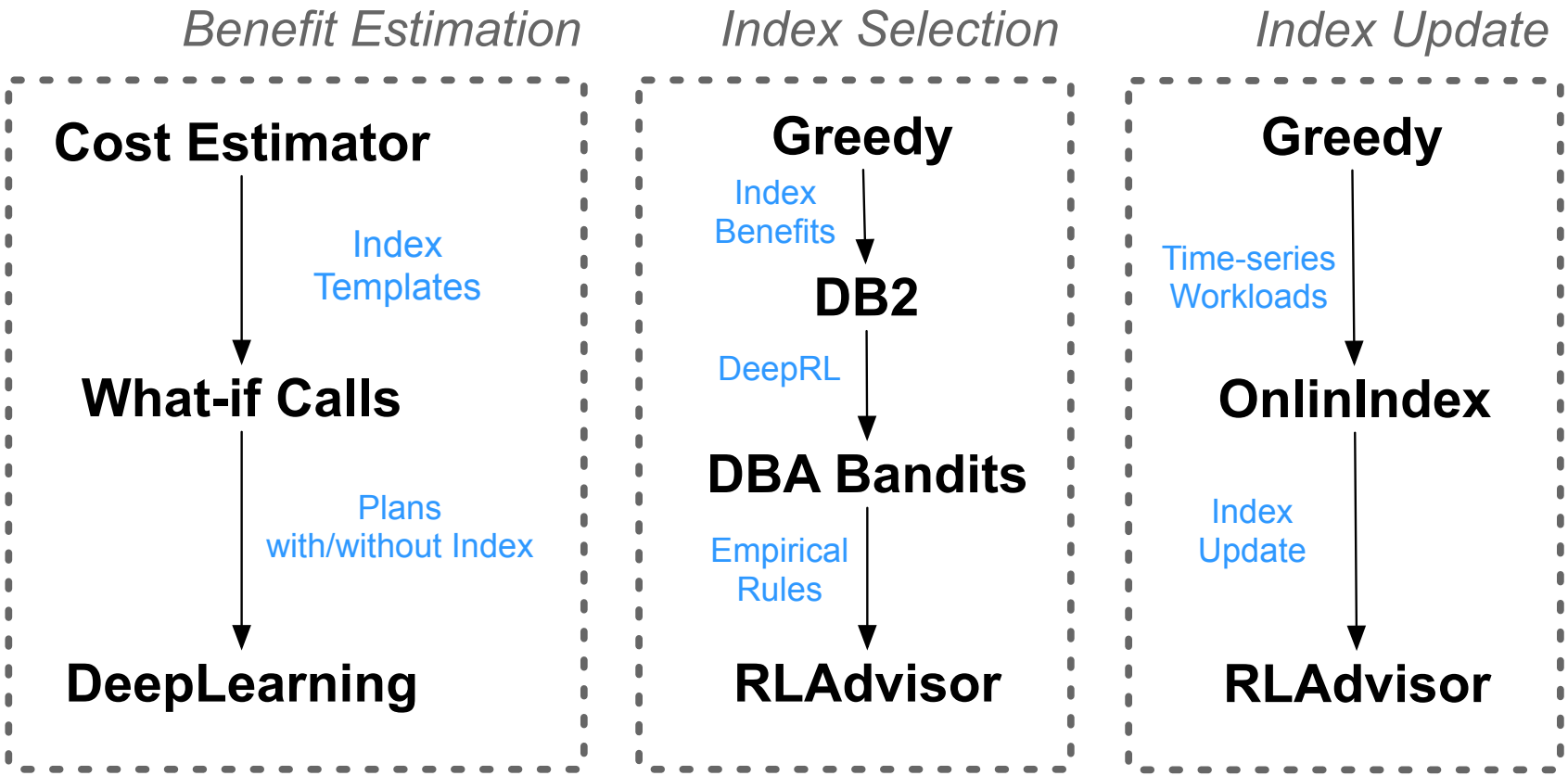
K. Schnaitter, S. Abiteboul, T. Milo and N. Polyzotis. *On-Line Index Selection for Shifting Workloads*. In ICDE 2007.

Learning-based Index Selection

☐ Why heuristics → learned index selection?

- **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);
- **Find better solutions from numerous candidate indexes**
  - Columns have different access frequencies, data distribution
- **Redundant indexes may cause negative effects**
  - Increase maintenance costs for update/delete operations

Learning-based Index Recommendation



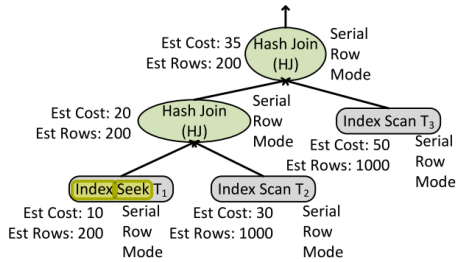
Index Benefit Estimation

☐ Challenge

- **The index benefit is hard to evaluate**
  - ➢ Multiple evaluation metrics (e.g., index benefit, space cost)
  - ➢ Cost estimation by the optimizer is inaccurate
- **Correlations with other components**
  - ➢ Multiple column access, data refresh
  - ➢ Conflicts between Indexes

Deep learning for Index Benefit Estimation

- Motivation: Critical to estimate index benefits by comparing execution costs of plans with/without created indexes
- Core Idea: Model benefit estimation as an ML classification task
- Challenge: Hard to accurately estimate the index benefits
- Solution:
  - Prepare training data
    - Query Plans + Costs under different indexes
  - Train the classification model
    - Input: Two query plans with/without indexes
    - Output: 1 denotes performance gains; 0 denotes no gains
  - Solve the index selection problem
    - Use the model to create indexes with performance gains



(a) Example query plan.

EstNodeCost

Seek_Row_Serial	10
Scan_Row_Serial	80
HJ_Row_Serial	55
NLJ_Row_Serial	0
MJ_Row_Serial	0
...	...

LeafWeightEstRows WeightedSum

Seek_Row_Serial	200
Scan_Row_Serial	2000
HJ_Row_Serial	4600
NLJ_Row_Serial	0
MJ_Row_Serial	0
...	...

(b) Feature channels for the plan.

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

Learning-based Index Selection

- **Challenges**
  - **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 is expensive**
    - Hard to estimate the number of involved pages

Reinforcement Learning for Index Selection

- **Motivation: Index selection using reinforcement learning**
- **Challenge 1: How to extract candidate indexes?**
  - Extract candidate indexes from query predicates 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.

- **Challenge 2: 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

large state space →

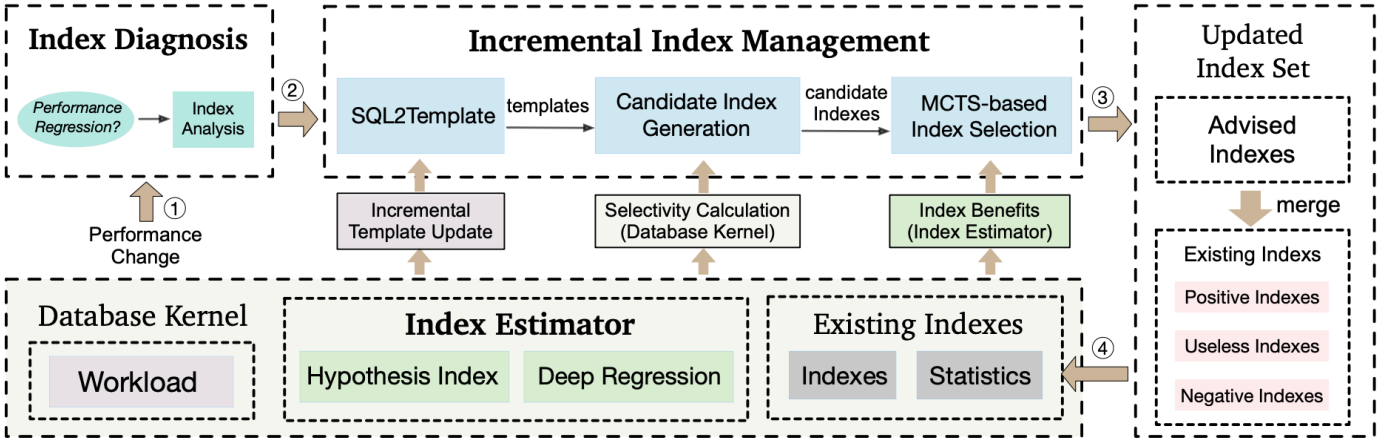
discrete action space →

DQN Model

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

MCTS for Index Update

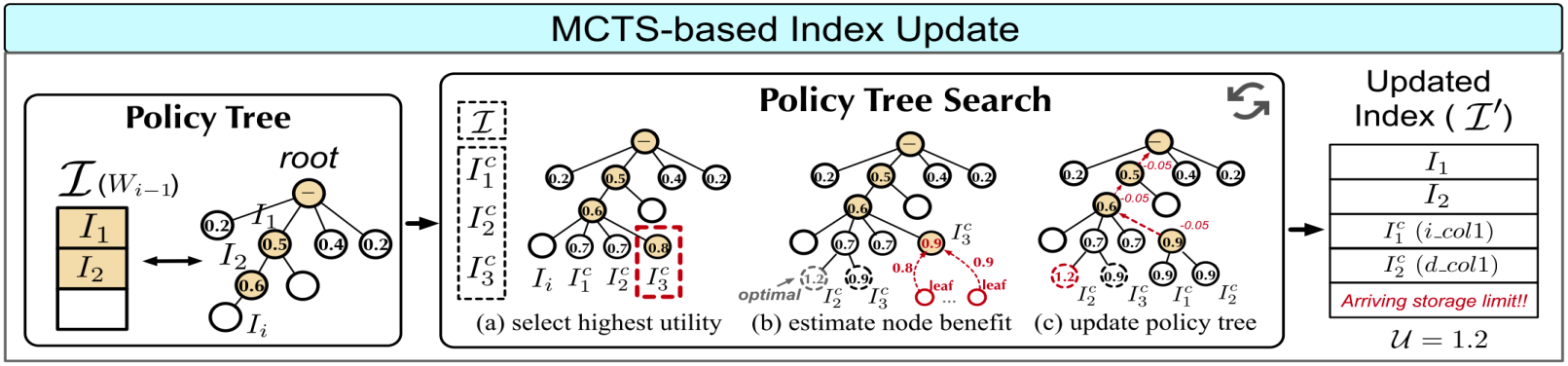
- Motivation: Existing methods cannot incrementally update indexes
- Basic Idea: Incrementally add/remove indexes with MCTS
- Challenge: Consider both the read and write queries
- Solution:
  - Index Diagnosis (anomaly detection)
  - Incremental Index Update (policy tree search)
  - Index Benefit Estimation (deep regression)



Xuanhe Zhou, Luyang Liu, et al. AutoIndex: *An Incremental Index Management System for Dynamic Workloads*. ICDE, 2022.

MCTS for Index Update

- **Motivation: Existing methods cannot incrementally update indexes**
- **Basic Idea: Incrementally add/remove indexes with MCTS**
- **Challenge: Consider both the read and write queries**
- **Solution:**
  - **Index Problem Diagnosis: Detect whether the performance regression is caused by index issues;**
  - **Candidate index extraction: Cluster queries → Map to query templates → Extract candidate indexes;**
  - **Incremental Index Update: Initialize a policy tree with existing indexes → Add new candidate indexes;**
  - **Index Benefit Estimation: Index Update Costs = seek_tuples * cpu_cost + insert_tuples * cpu_index_tuple_cost**



Xuanhe Zhou, Luyang Liu, et al. AutoIndex: *An Incremental Index Management System for Dynamic Workloads*. ICDE, 2022.

Summarization of Index Management

	Optimization Targets	Training Efficiency	Training Data	Adaptive
Deep Learning	Accurate Estimation	high	numerous data	query changes
Reinforcement Learning	High Performance	high computation costs	no prepared Data	query changes
MCTS	High Performance for index update	trade-off (costs, performance)	a few prepared data	query changes

Take-aways of Index Advisor

- **Learned index estimation is more robust than cost models**
- **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

Learned Advisors

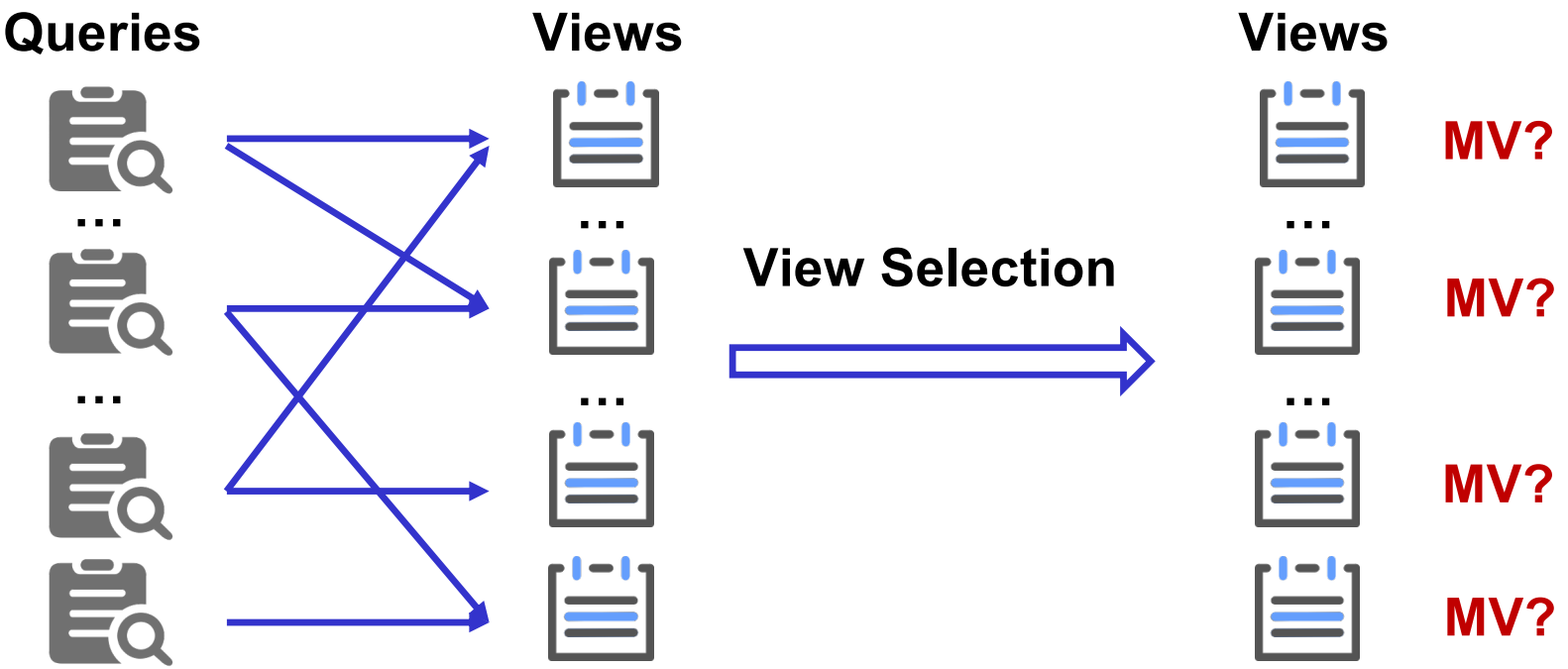
- ☐ Learned Knob Tuning
- ☐ Learned Index Advisor
- ☐ Learned View Advisor
- ☐ Learned Partition Advisor
- ☐ Learned Data Generation

View Management

- **View Benefit Estimation**
  - The benefit of building a materialized view (MV) for a subquery
- **View Selection**
  - Which subquery to create an MV
- **View Update/Refresh**
  - Adding or removing an MV

View Selection

Problem Definition: Given a workload *Q* and a space budget, select optimal subqueries to materialize (MVs), including (i) MV benefit estimation; (ii) MV Selection; (iii) MV update; (iv) MV rewrite.



View Selection

- **Materialized Views (MVs) can 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

Traditional View Selection Methods

- **Given a workload, select and maintain materialized views that minimize the total latency within a limited materialized view storage space (NP-hard).**
- **Traditional Methods**
  - **Greedy:** WATCHMAN, DynaMat, CloudViews
  - **Genetic:** EA, Hybrid-GHCA
  - **Coral Reefs Optimization Algorithm:** CROMVS
  - **Backtracking Search Optimization Algorithm:** BSAMVS-penalty
  - **Integer Linear Programming:** BIGSUBS, HAWC

Learned View Management

- **Limitations of Traditional Methods**
  - **View's benefit estimation. Not accurate.**
    - **Traditional models is not accurate for view benefit/multiple view benefit estimation.**
    - **Hard to estimate materialized view update cost.**
  - **View selection. Not generalizable.**
    - **Designed and work well for specific scenarios or workloads.**
    - **Rely on assumptions that are not always right**
  - **View update. Long Delay.**
    - **Based on accumulated benefits and creation cost of views.**
    - **Hard to estimate the a view's future benefit and recreation cost.**

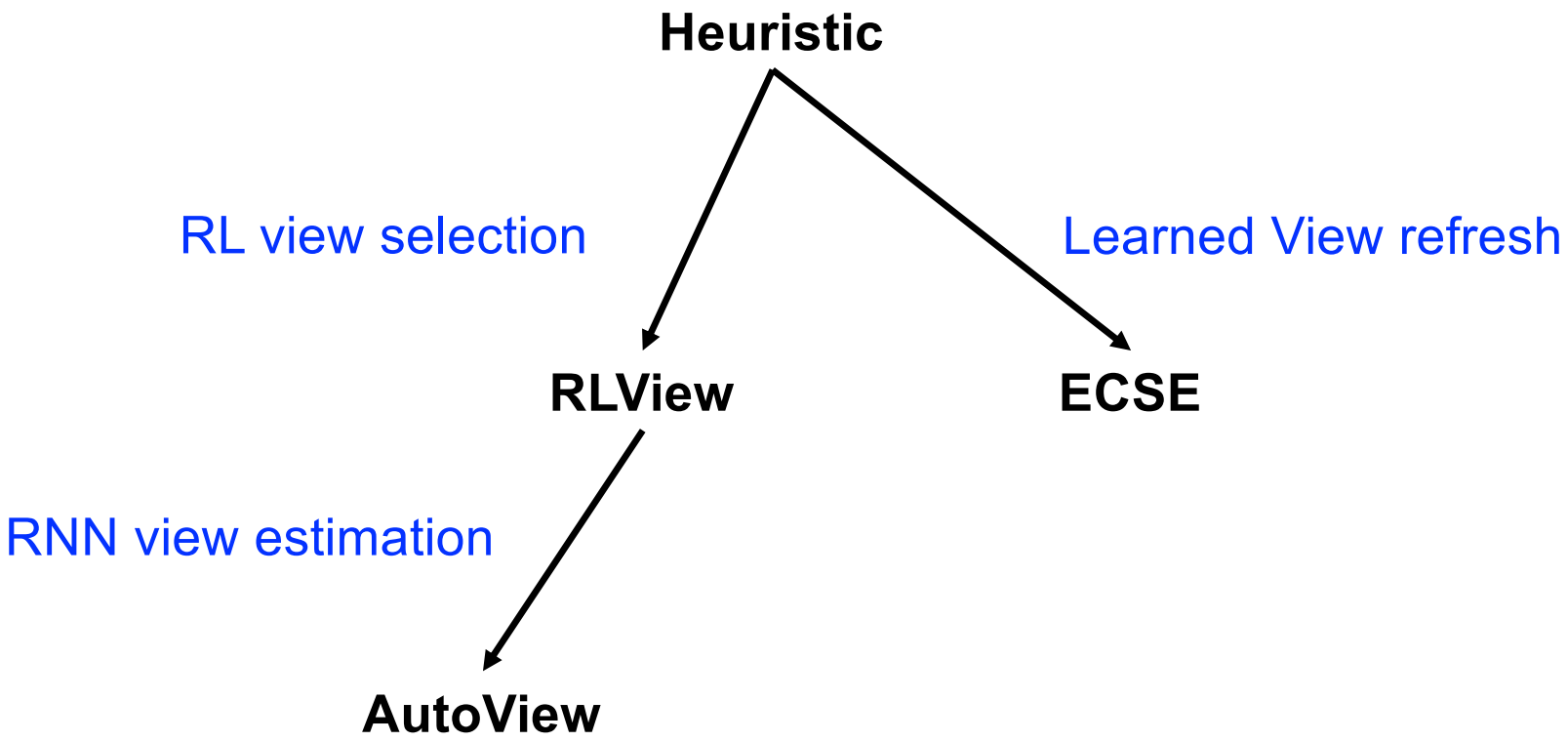
Learned View Management

- **Motivation**
  - **Estimate view benefit accurately.**
    - **Learned based methods from real runtime statistics.**
      (Also verified in learned cardinality and learned join order selection)
  - **Generalizable on different workloads.**
    - **Learns from historical workloads and learns directly from the view selection performance without human experience.**
  - **Predict views' future benefit.**
    - **Learns from historical MV utilization and predict future benefit and update cost.**

Learned View Management

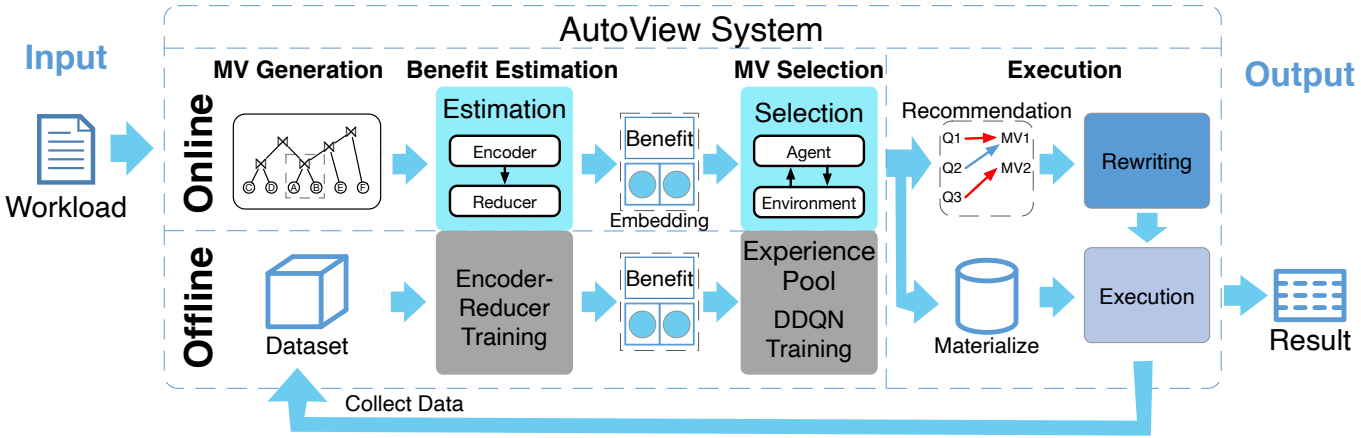
- Challenges
  - View and query need to be encoded for neural networks.
  - New models need to be designed for view benefit estimation.
  - View selection models should be efficient and flexible.
- Optimization Goals
  - View Quality
  - Model Adaptivity
  - Support view update

Learned View Management



Learned View Estimation: AutoView

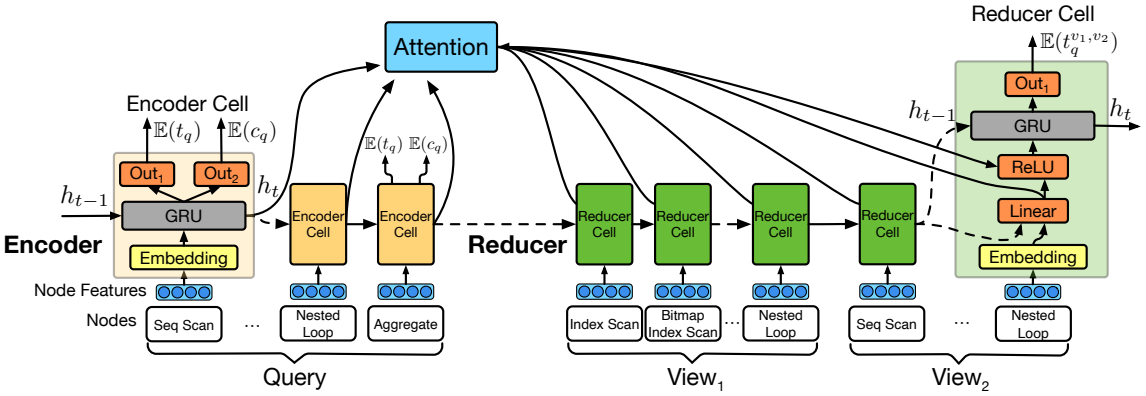
- **Motivation**
  - Estimate views' benefit more accurately.
  - Support variable number of views in RL for view selection.
- **Challenges**
  - Views have different benefits on queries in workload.
  - Hard to extend state representation after model training.

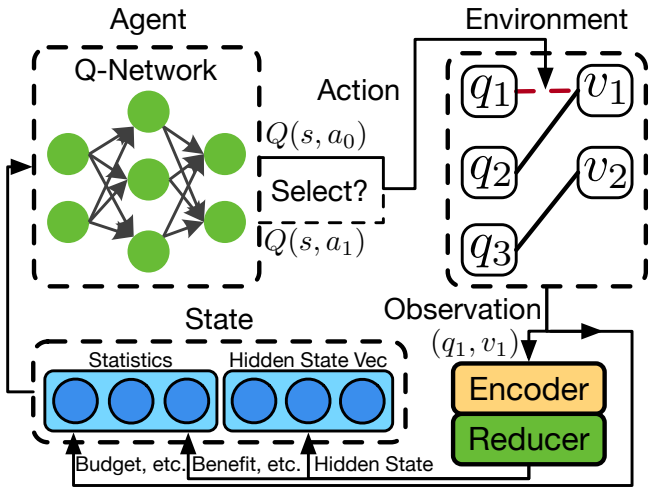


Y. Han, G. Li, H. Yuan, and J. Sun. An autonomous materialized view management system with deep reinforcement learning. In ICDE, 2021.

Learned View Estimation: AutoView

- **Estimate the query-view benefits with encoder-reducer model:**
  - Two LSTM network for query and views, which captures query-MV correlations with attention.
- **Select optimal query-view combinations with reinforcement learning iteratively.**





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

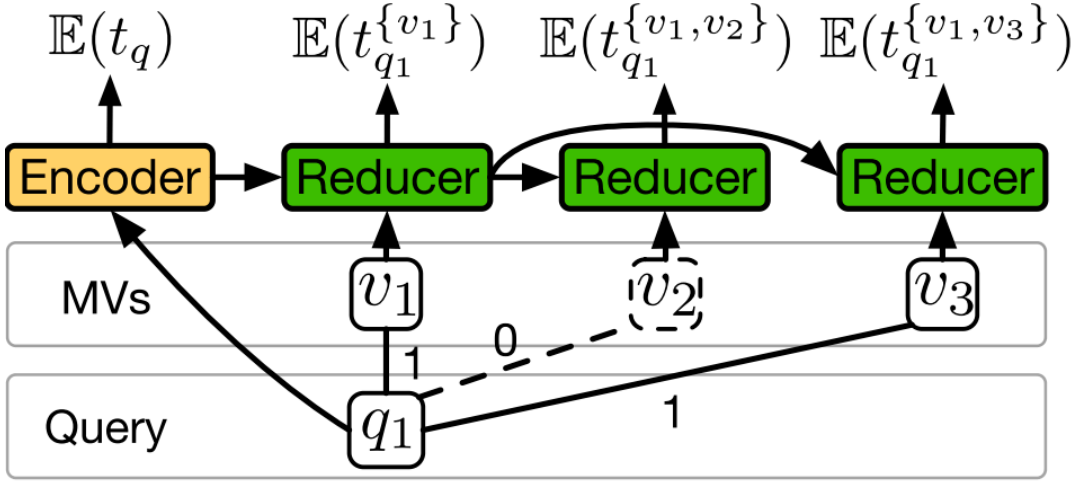
Learned View Estimation: AutoView

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

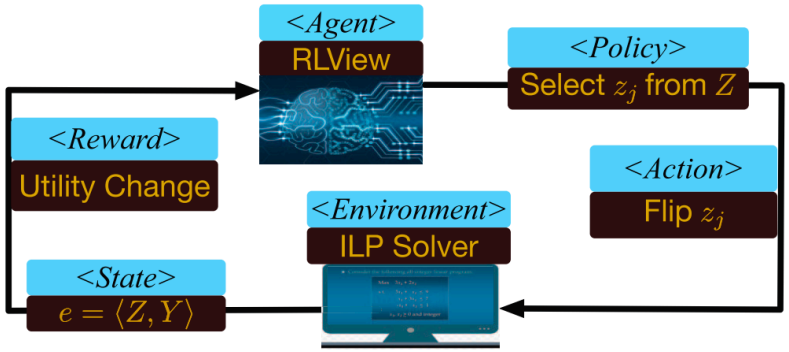
- It is hard to jointly consider MVs with conflicts →
- (1) Split the problem into sub-steps that select one MV;
- (2) Use attention-based model to estimate the MV benefit



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

Learned View Selection: RLView

- **Motivation**
  - RL performs well on combinatorial optimization problem.
- **Challenges**
  - How to solve view selection problem in RL framework.
- **Solutions**
  - Cluster equivalent queries and select the least overhead ones as the candidate;
  - Represent MVs as a fixed-length state vector and solve with DQN model;
  - Estimate the MV benefits with DNN.

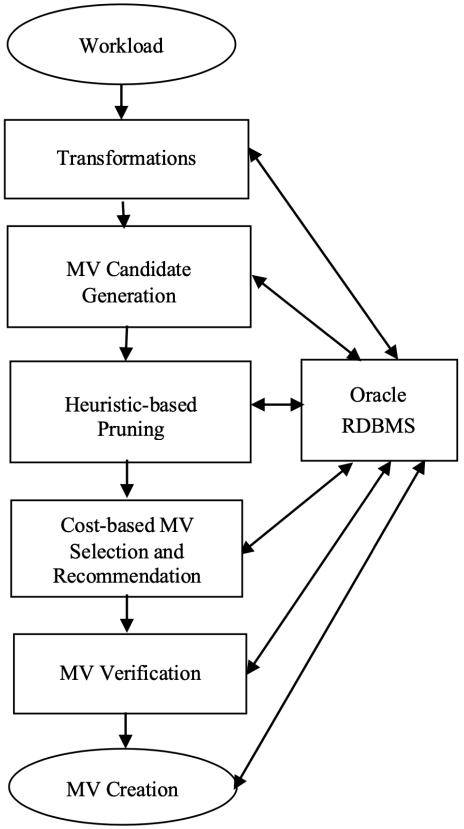


$Z = \{z_j\}$:z_j is a 0/1 variable indicating whether to materialize the subquery s_j
$Y = \{y_{ij}\}$:y_{ij} is a 0/1 variable indicating whether to use the view v_{s_j} for the query q_i

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

Learned View Update: ECSE

- **Motivation**
  - Support MV refresh.
- **Challenges**
  - Hard to estimate refresh benefit/cost from historical workload.
- **Solutions**
  - Traditional view generation, estimation, and selection;
  - Use a neural network model to predict future DML operations and MV usage for scheduling the refresh.
  - Use linear regression to estimate MV refresh time with
    - MV size, refresh method, affected number of rows,
    - previous refreshes time.



Ahmed, R., Bello, R., Witkowski, A. Kumar. Automated generation of materialized views in Oracle. VLDB 2020.

Learned View Management: Comparison

Method	View Quality	Adaptability	View Update	View Estimation	View Selection	View Update
RLView	Medium	Low	No	Learned	Learned	-
AutoView	High	High	No	Learned	Learned	-
ECSE	Medium	Medium	Yes	Heuristic	Heuristic	Learned

Learned View Advisor: Take-away

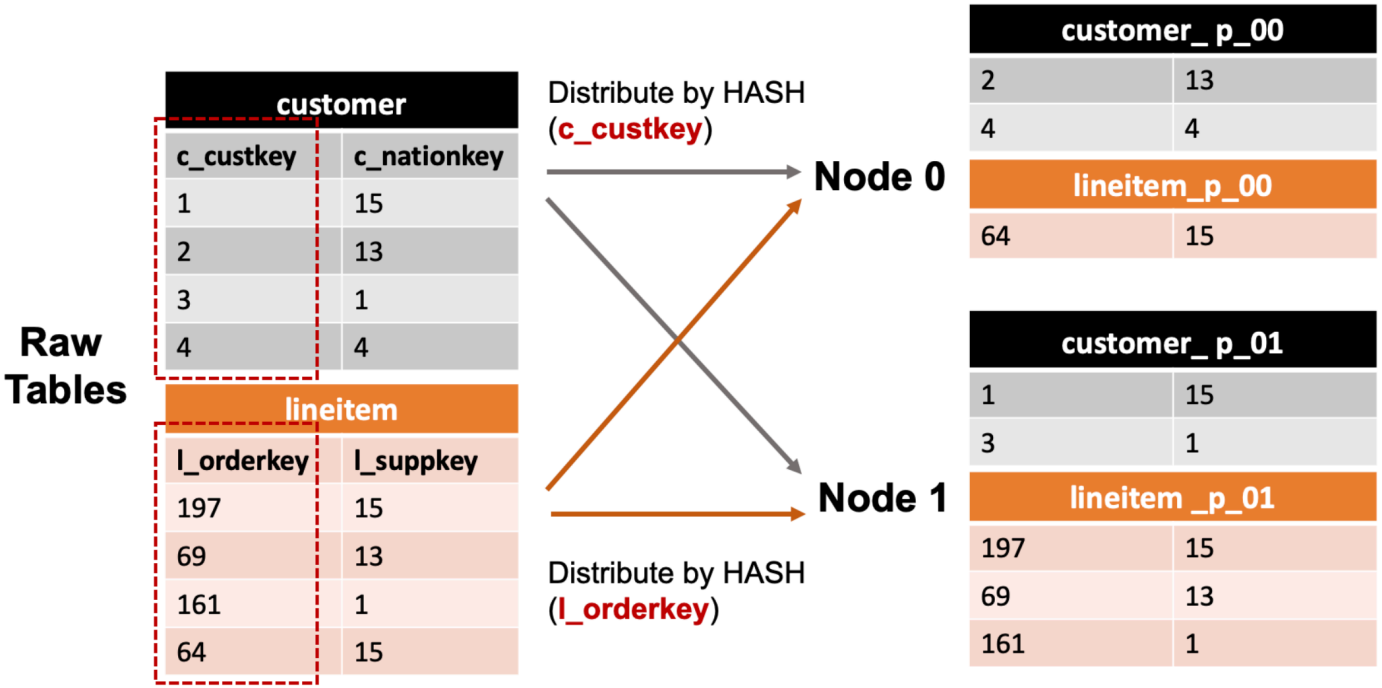
- **Learned view selection gains higher performance than heuristics**
- **Learned view selection works well for read workloads**
- **Learned view benefit estimation is more accurate than traditional empirical methods**
- **Learned view benefit estimation is accurate for multiple-view optimization**
- **Open Problems:**
  - Learned MV update/refresh
  - Learned MV rewrite

Learned Advisors

- ☐ Learned Knob Tuning
- ☐ Learned Index Advisor
- ☐ Learned View Advisor
- ☐ Learned Partition Advisor
- ☐ Learned Data Generation

Database Partition

Problem Definition: Given tables $\{T_1, T_2, \ldots, T_m\}$ and a partition function F, database partition selects columns for each table T_i as the partition key, and allocate the tuples in T_i into partitions using F, such that the workload performance is optimal.



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

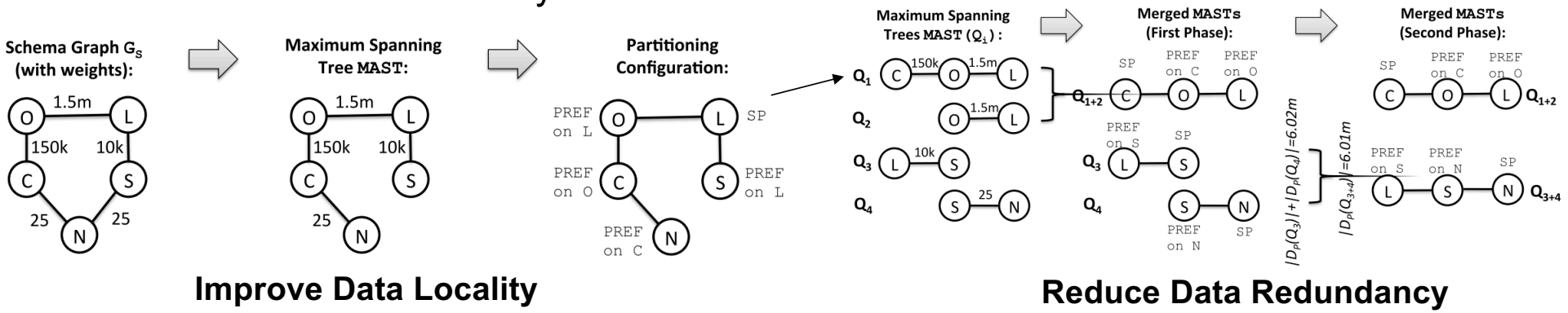
Heuristic Database Partition for OLAP Workloads

- **Motivation**
  - Reduce the network costs by judiciously partitioning tables
- **Core Idea**
  - Heuristically co-partition (the tuples of the referenced table are on the same node of referencing table) tables by foreign-key relations
- **Challenge**
  - It is hard to find a suitable partitioning scheme (for many tables with join correlations) that maximizes data locality.
  - There can be different partition schemes. How to merge them so as to reduce the data redundancy caused by co-partitioning.

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

Heuristic Database Partition for OLAP Workloads

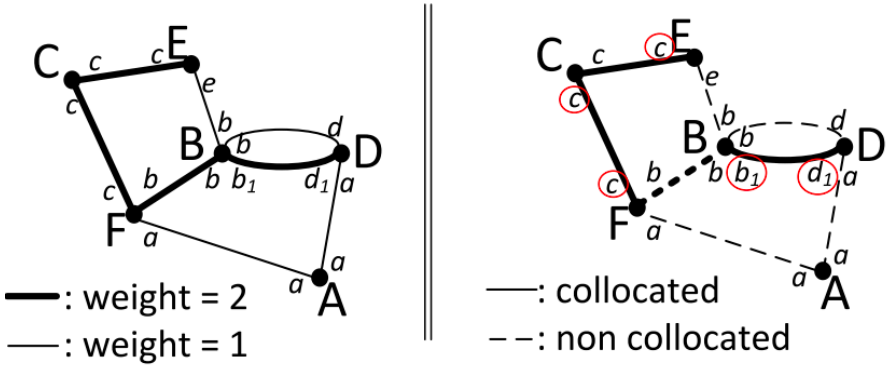
- **Represent the specific dataset schema → Build a graph mode**
  - Initialize a graph model G,
    - *Nodes*: tables, *Edges:* foreign keys, *Edge weight*: the size of smaller table connected to the edge
- **Improve data locality (reduce network costs) → Partition by join predicates**
  - REF partitioning: a table is co-partitioned by the join predicate that refers to another table;
  - Utilize maximum spanning tree to extract subsets of edges (a partition strategy) that (1) partition all the tables and (2) maximize the data locality.
- **Full data locality may introduce duplicate tuples → Merge duplicated partitions**
  - Utilize dynamic programming to merge candidate partition strategies so as to find the one with minimal data redundancy.



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

Traditional Database Partition

- **Motivation: Partition on join columns can significantly reduce the network communication and reduce execution costs**
- **Core Idea: Combine exact and heuristic algorithms to find good partition strategies for different workloads**
- **Challenge: Picking join columns as partition keys is NP-complete**
- **Solution**
  - **Build a Join Multi-Graph**
    - *Vertices* are tables, *Edges* denote join relations
  - **Partition with hybrid partitioning algorithms**
    - *Exact algorithm:* Assume each table only uses a column; and turn into an integer programming problem;
    - *Heuristic algorithm:* Select the table columns with largest edge weights



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

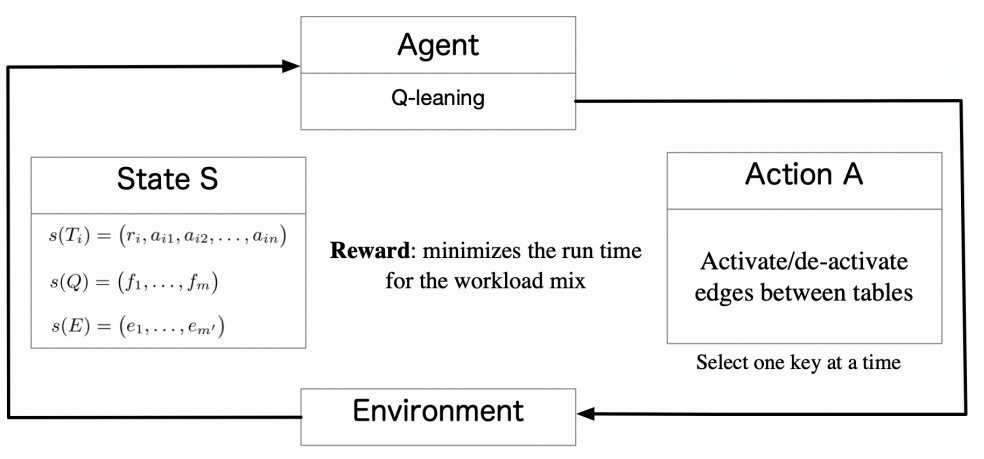
Learning-based Database Partition

☐ Motivation:

- ➢ **Consider both the data balance & access efficiency**
  - Place partitions on different nodes to speedup queries
  - Trade-off based on workload and data features
- ➢ **Combine ML to optimize the NP optimization problem**
  - **Combinatorial problem:** 61 TPC-H columns, 145 query relations, 2.3 x 10^{18} candidate combinations

Reinforcement Learning for Database Partition

- **Motivation: OLAP Workloads contain complex and recursive queries**
- **Core Idea: Explore column combinations as partition keys with RL**
- **Challenge: Characterize partition features; Migrate to new workloads**
- **Solution**
  - **Extract partition features as a vector**
    - **[tables, query frequencies, foreign keys]**
  - **Use DQN to partition the tables for a workload**
    - **Iteratively partition tables by long-term reward**
  - **Support new workloads with trained models**
    - **Train a cluster of DQN models on typical workloads;**
    - **Pick models whose workloads are similar to the new workload to partition tables.**



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 under complex workloads (e.g., with multiple joins)**
- **Learned key-selection partition has much higher partitioning latency (e.g., data collection, model training)**
- **Open Problems:**
  - Adaptive partition for relational databases
  - Partition quality prediction
  - Improve partition availability with replicates

Learned Advisors

- ☐ Learned Knob Tuning
- ☐ Learned Index Advisor
- ☐ Learned View Advisor
- ☐ Learned Partition Advisor
- ☐ Learned Data Generation

Automatic Query Generation

☐ Motivation

- Companies generally will not release their data and 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?**

Definiation：Given a scheme and constraints (e.g., cost/ cardinality ranges), we generate k SQL queries which can (i) legally execute in the databse 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；	RL →	☐ Construct a LSTM-based critic to predict the long-term benefits of any intermediate queries; utilize actor to explore new tokens；
☐ It is hard to generate diverse SQLs；	→	☐ Construct a probablistics model to ensure the diversity of generated queries；
☐ Grammar and syntax constraints need to be considered to generate legal queries；	→	☐ Construct a FSM to prune illegal tokens for current intermediate queries；

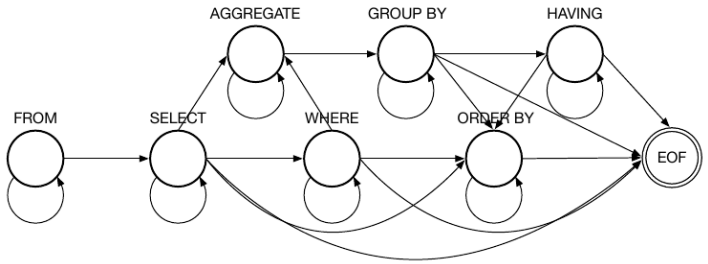
Lixi Zhang, Chengliang Chai, Xuanhe Zhou, Guoliang Li. LearnedSQLGen: Constraint-aware SQL Generation using Reinforcement Learning. SIGMOD 2022.

Automatic Query Generation

Query Legality

➢ SQL Grammar:

- FSM



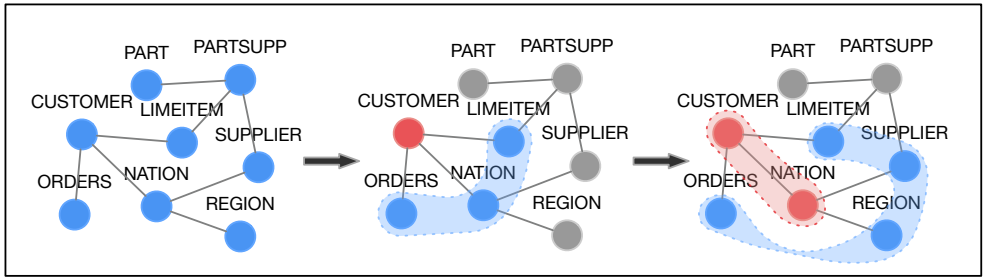
Advantage:

✓ 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

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

Automatic Training Data Generation

Motivation

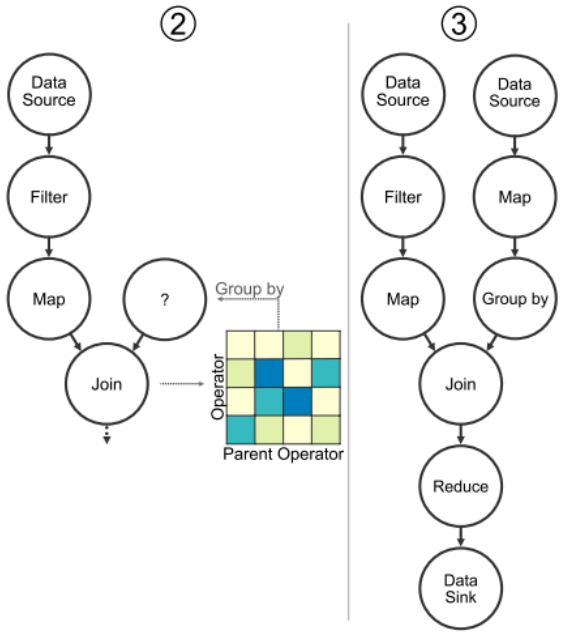
- **Machine learning is widely adopted in database components**

- **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

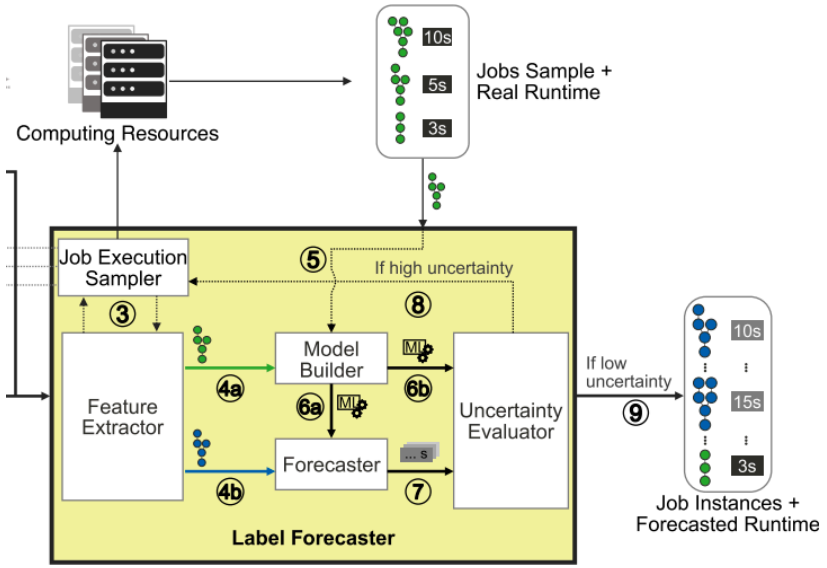
- **Challenges in existing workload generators (TPC-H, sqlsmith)**
  - Limited SQL templates; while real queries have various structures;
  - Fail to label the SQL queries (e.g, cost, execution time)
- **Core Idea: Reduce the labeling time by generating many query jobs and estimating the job latency**
  - SQL Sampling
    - A few real SQL queries + sample data;
  - Plan Synthesis
    - Generate abstract plans from the real SQLs;
    - Collect statistics, e.g., distribution of the longest plan paths;
    - Generate job by imitating the structures/patterns of the plans,
      - E.g., for join operator, they select the operator (Group by) as the child node with the max possibility (the transition matrix)



Francesco Ventura. Expand your training limits! generating training data for ML-based data management. SIGMOD, 2021.

Automatic Training Data Generation

- **Challenges in existing workload generators (TPC-H, sqlsmith)**
  - Limited SQL templates; while real queries have various structures;
  - Fail to label the SQL queries (e.g, cost, execution time)
- **Core Idea: Reduce the labeling time by generating many query jobs and estimating the job latency**
  - SQL Sampling
    - A few real SQL queries + sample data;
  - Plan Synthesis
  - Label Forecasting
    - **Sample and execute jobs** → Get the real latency (labels)
    - **Build an estimator** → Evaluate the latency and uncertainty of the unexecuted jobs
    - **Incrementally sample jobs** → Reduce the uncertainty



Francesco Ventura. Expand your training limits! generating training data for ML-based data management. SIGMOD, 2021.

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

Prediction Problems

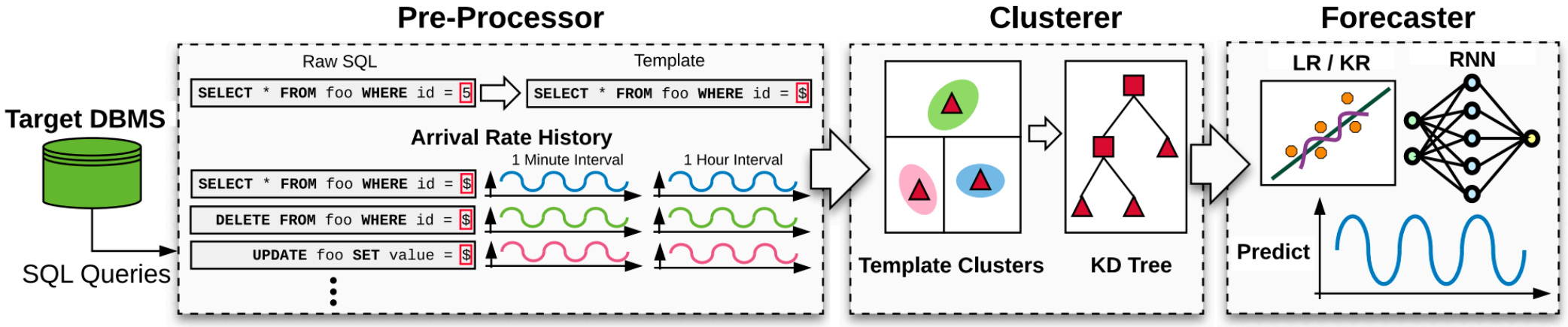
Prediction Problems

☐ 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 Workload 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

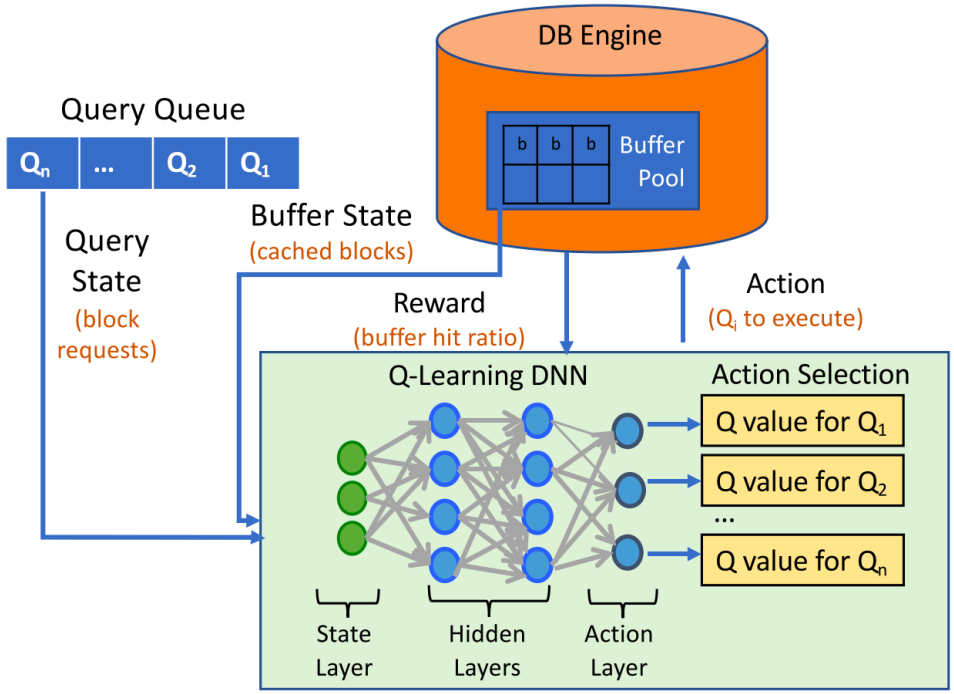


Lin Ma, Dana Van Aken, and et al. Query-based Workload Forecasting for Self-Driving Database Systems. In SIGMOD, 2018.

Learned Workload 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



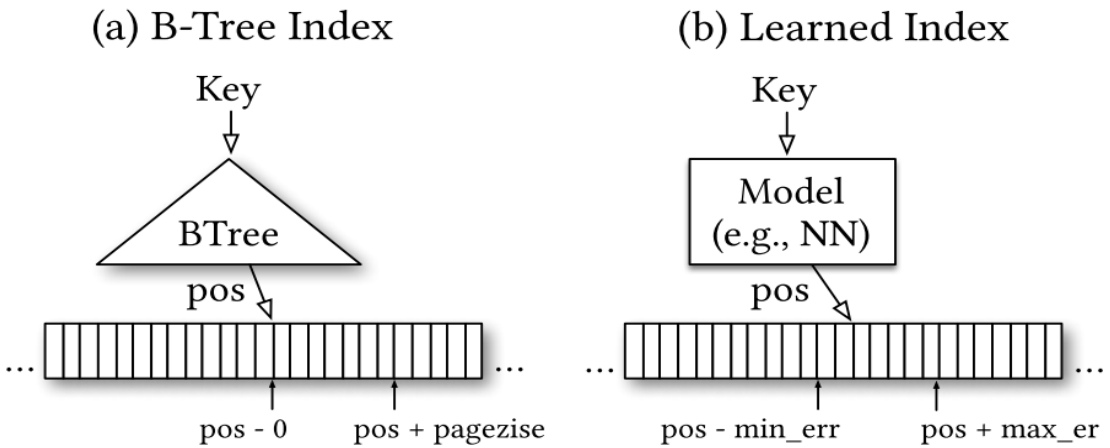
Chi Zhang, Ryan Marcus, and et al. Buffer Pool Aware Query Scheduling via Deep Reinforcement Learning. In VLDB, 2020.

Learned Indexes

Basic Idea of Learned Index

- Model the cumulative distribution function(CDF) of the data to predict the location as:

$$p = F(Key) * N$$



- Data sampling → Training CDF → Predict approximate location → Search precise location

Kraska, T., Beutel, A., Chi, et al. The case for learned index structures. SIGMOD, 2018.

Why 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
- ➢ Highly parallel, adapt to modern hardware like GPU and TPU

Learned Index: Formulation

☐Problem Formulation

– Given a set of key-value pairs, index is a data structure that improves the speed of data retrieval operations such as: lookup the value of the key, range query, nearest neighbor query, etc.

☐Traditional Methods

– B-Tree, ART, R-Tree,…

☐Limitations

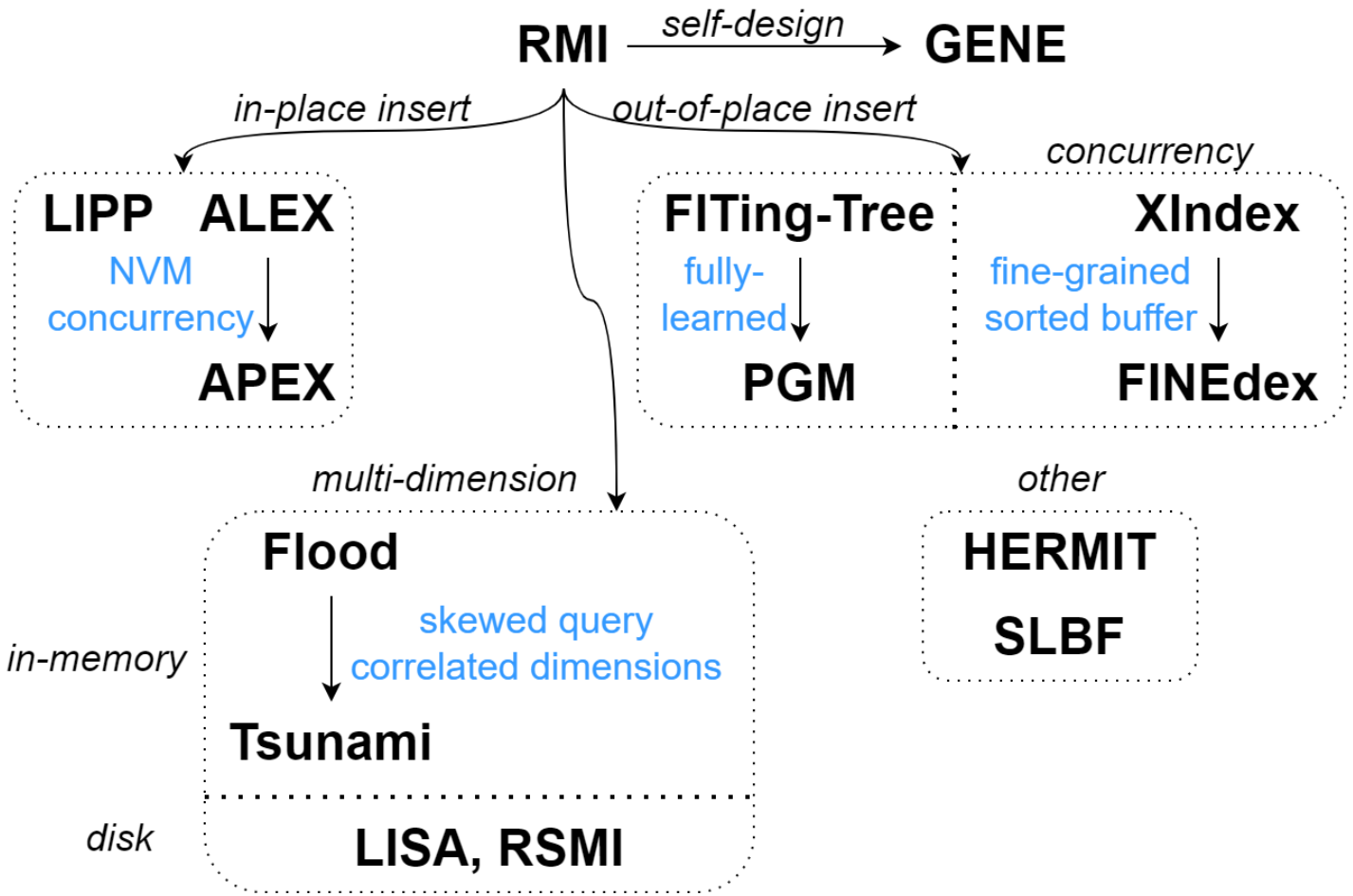
– Unaware of data and workload distribution

– Trade-off between space and access efficiency

Learned Index: Challanges

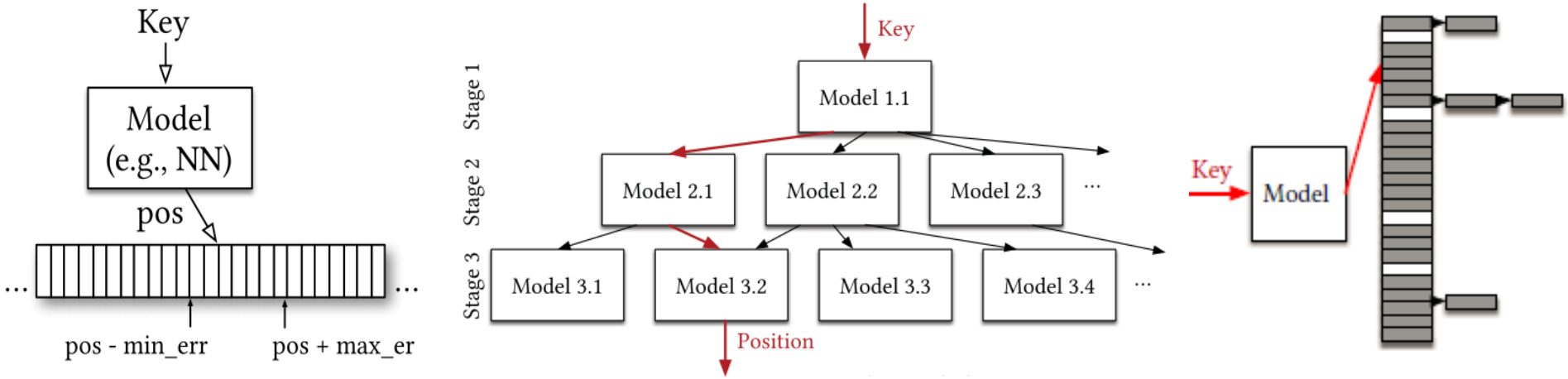
- **Advantages**
  - Space efficient, only store several parameters
  - Faster access if the model fit well, predict the position
- **Challenges**
  - Support update, concurrency, and persistency
- **Optimization Goals**
  - Higher throughput
  - Less space
  - Robustness

Learned Index: Lineage



Learned Index: RMI

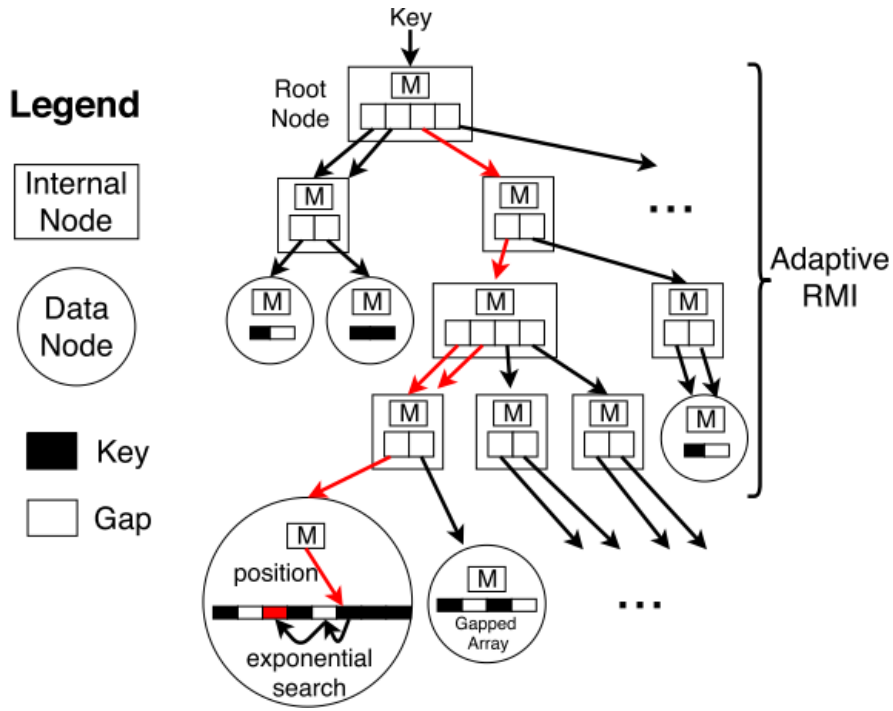
- **Motivation: indexes are models**
- **Challenge: difficult for the "last mile" to reduce error**
  - **Range index**: approximate location as $p = CDF(Key) * N$, model by hierarchy of simple neural networks, search precise location within error-bounded range
  - **Hash index**: CDF as hash function to reduce conflict



Kraska, T., Beutel, A., Chi, et al. The case for learned index structures. SIGMOD 2018

Updatable Learned Index: ALEX

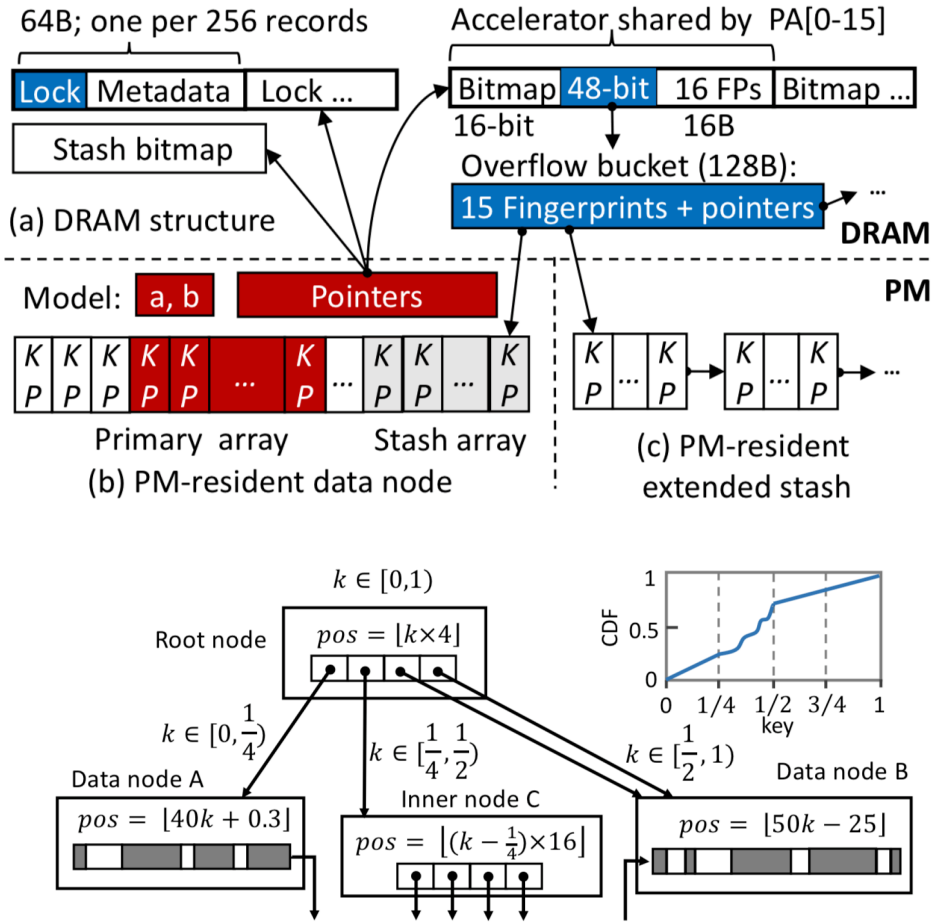
- **Motivation: support update**
- **Challenge: adaptive to dynamic data distribution**
  - Linear model, only exponential search in data nodes
  - Use **gapped array layout** in data nodes to accelerate insert
  - **Cost model**: predict latency of lookup and insert, expand/split data node if slower than a threshold (e.g. 1.5× that at creation)



Ding, J., Minhas, U. F., Yu, J., et al. ALEX: An Updatable Adaptive Learned Index. SIGMOD 2020

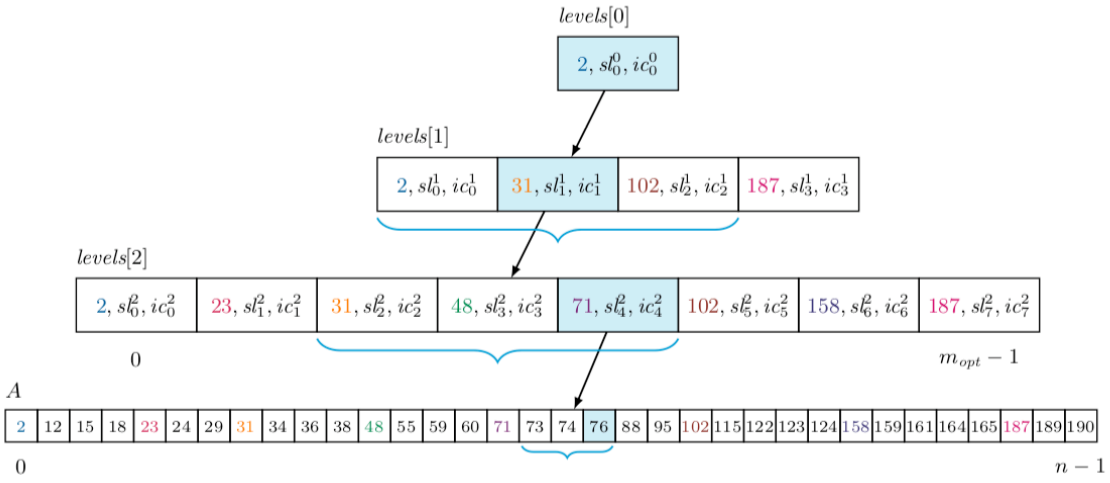
Persistent Learned Index: APEX

- **Motivation: NVM-optimized ALEX**
- **Challenge: lower write bandwidth, crash consistency**
  - **Reduce write**: linear model in data node as hash function, collision solved by sequential scan and chaining
  - **Concurrency**: reader-writer lock for inner node, fine-grained optimistic lock for data nodes' non-structural update
  - **Crash recovery**: nodes out-of-place expand/split, undo-log before new node prepared, redo-log after



Updatable Learned Index: PGM

- Motivation: support update, fully-dynamic
- Challenge: adaptive to dynamic data distribution
  - Piecewise Geometric Model index (PGM-index)
  - I/O-optimally the predecessor search problem while taking succinct space
  - adaptive not only to the key distribution but also to the query distribution



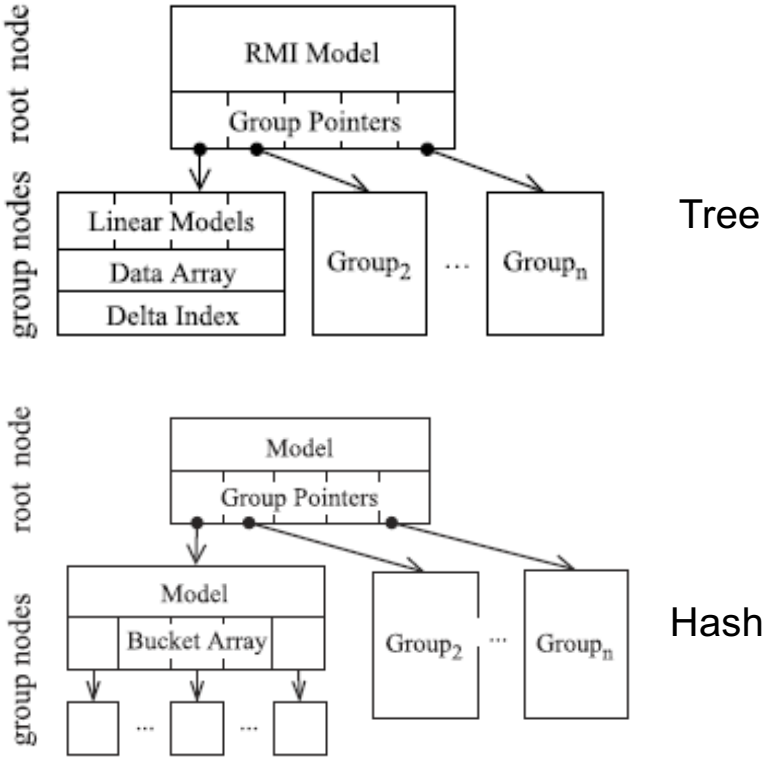

```
BUILD-PGM-INDEX(A, n, ε)
1  levels = an empty dynamic array
2  i = 0; keys = A
3  repeat
4    M = BUILD-PLA-MODEL(keys, ε)
5    levels[i] = M; i = i + 1
6    m = SIZE(M)
7    keys = [M[0].key, ..., M[m − 1].key]
8  until m = 1
9  return levels in reverse order
```

```
QUERY(A, n, ε, levels, k)
1  pos = f_r(k), where r = levels[0][0]
2  for i = 1 to SIZE(levels) − 1
3    lo = max{pos − ε, 0}
4    hi = min{pos + ε, SIZE(levels[i]) − 1}
5    s = the rightmost segment s′ in
       levels[i][lo, hi] such that s′.key ≤ k
6    t = the segment at the right of s
7    pos = ⌊min{f_s(k), f_t(t.key)}⌋
8  lo = max{pos − ε, 0}
9  hi = min{pos + ε, n − 1}
10 return search for k in A[lo, hi]
```

FERRAGINA P, VINCIGUERRA G. The PGM-index: A fully-dynamic compressed learned index with provable worst-case bounds. VLDB 2020

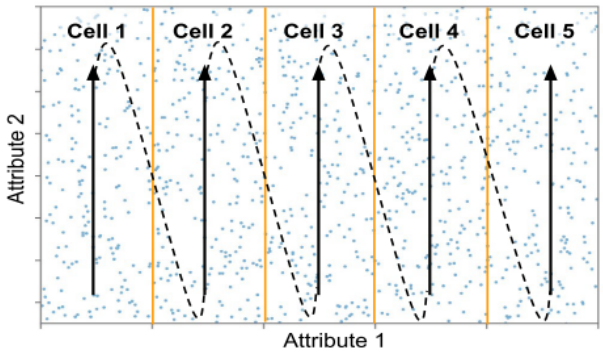
Concurrent Learned Index: XIndex

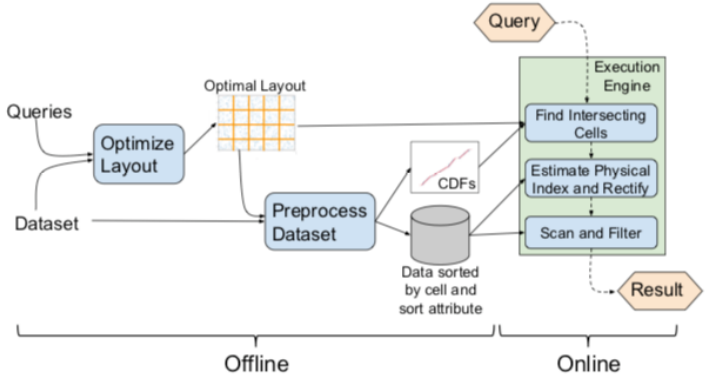
- **Motivation: handle concurrent write**
- **Challenge: update in-place with a non-blocking scheme**
  - Two write types: in-place update, insert into buffer
  - **Two-phase compaction** to preserve effect of update:
    - first merge pointers to group's data and buffer
    - then copy the value
  - Similar design for the hash index, similar two-phase resize

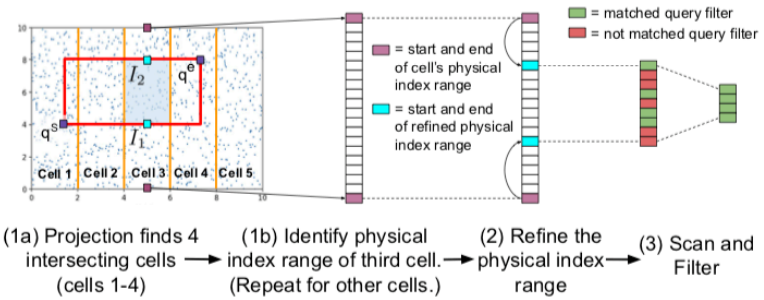


Multi-D Learned Index: Flood

- **Motivation: multi-dimensional in-memory read-optimized**
- **Challenge: optimize for data and query distribution**
  - Variant of grid index, cells sorted by 1st, 2nd, … column; within cell, points sorted by the last column
  - Gradient descent to find the optimal number of segments for each column using sample of dataset and workload
  - Use RMI to learn CDF of each column to even out segments and predict position



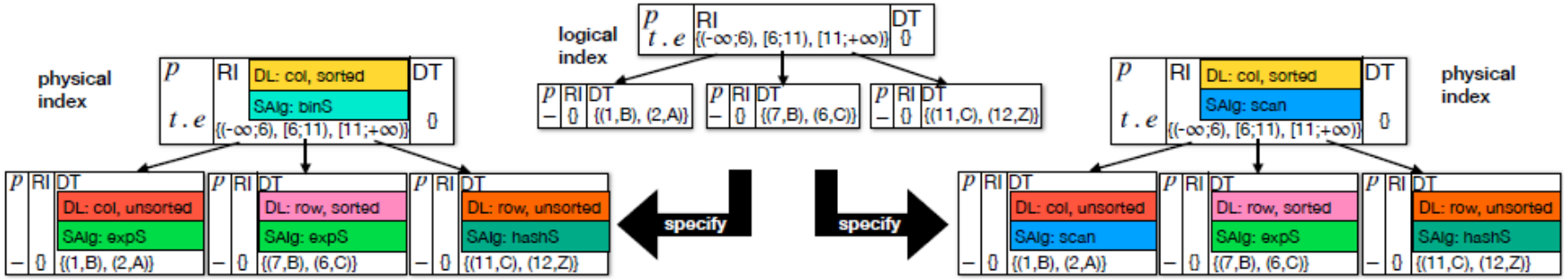




Nathan, V., Ding, J., Alizadeh, M., et al. Learning multi-dimensional indexes. SIGMOD 2020

Learned Index Generation: GENE

- **Motivation: self-design indexes**
- **Challenge: generalize to a genetic index framework**
- **Genetic Algorithm**
  - **Node framework**: child mapping, data, data layout and search method
  - **Population**: a set of indexes (e.g. initially a single node)
  - **Mutations**: change particular node's implementation, or merge/split nodes horizontally and vertically
  - **Fitness function**: optimize indexes for the runtime given workload



Learned Index: Comparison

Learned index	Model	Update	Concurrency	Persistency
RMI	simple NN	no	no	no
ALEX	linear	yes	no	no
Flood	simple NN	no	no	no
XIndex	simple NN, linear	yes	yes	no
APEX	linear	yes	yes	yes
GENE	any function	no	no	no

Learned Index: Take-away

- Though some research has already verified the benefit of learned index, performance in **industrial workloads** still needs to be studied, especially in **update distribution-drift** and **multi-dimension** situation.
- Open problems
  - Types of ML models to use
  - More efficiently support update, concurrency, persistency
  - Robustness: more adaptive to update distribution drift
  - Self-design: learn faster, or amortize learning cost
  - Make learned index applicable to industrial database systems

Learned Data Layout

Motivation

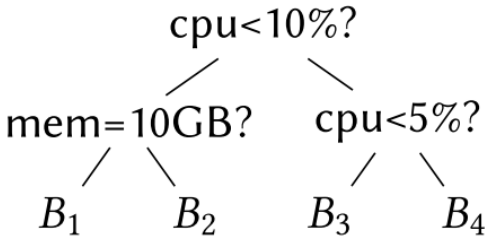
- **To reduce the #-data read from disk**
  - Split data into data blocks (main-memory, secondary storage)
  - in-memory min-max index for each block

- **It is challenging to partition data into data blocks**
  - Numerous ways to assign records into blocks

    Traditional: assign by arrival time; hash/range parititon

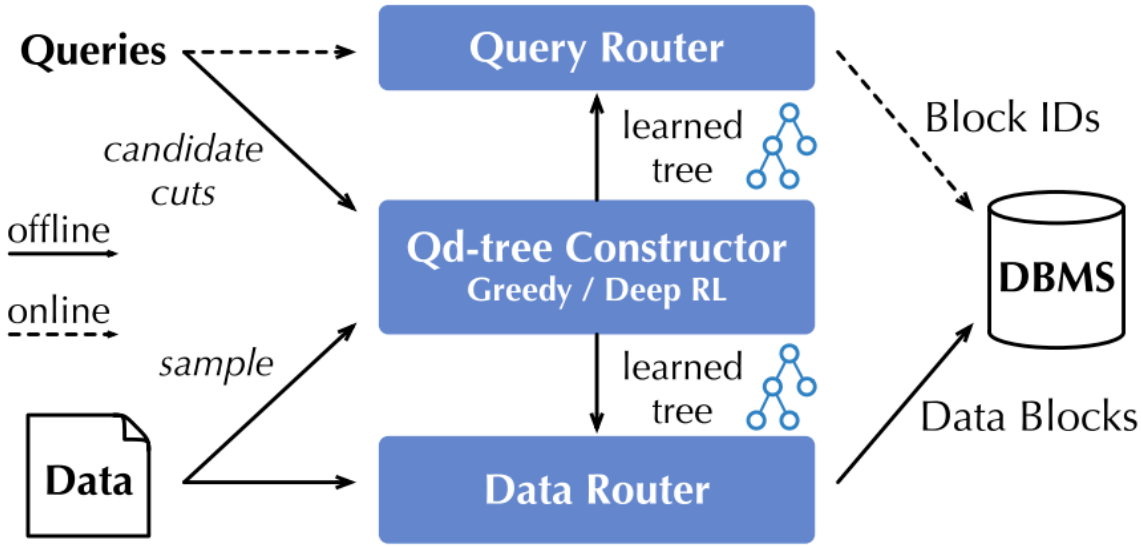
Learned Data Layout (Qd-tree)

- **Qd-tree: Learning Branch Predicates**
  - Root Node: The whole data space
  - Other Nodes: A part of the whole space



Example Qd-tree

- **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



Zongheng Yang, et al. Qd-tree: Learning Data Layouts for Big Data Analytics. SIGMOD, 2020.

Learned Data Layout: Join Predicates

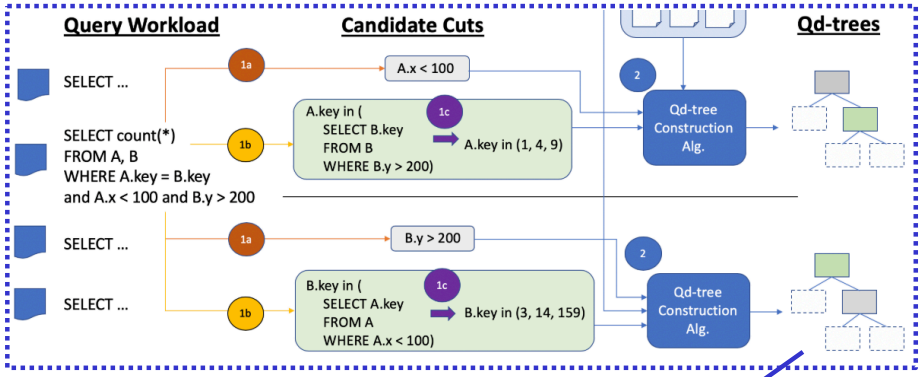
Motivaiton

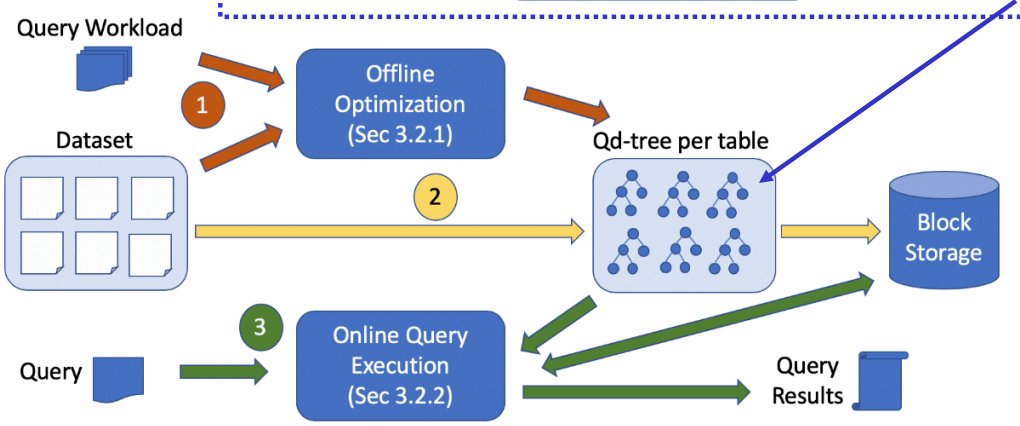
- **Traditonal:** either provide rare data skipping (zone maps), or require careful manual designs (Z-order)
- **Qd-tree:** only optimize singe-table layouts

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





Jialin Ding, et al. Iinstance-Optimized Data Layouts for Cloud Analytics Workloads. SIGMOD, 2021.

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
  - **Persistent**, **Update, Concurrency Control, Recovery**

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

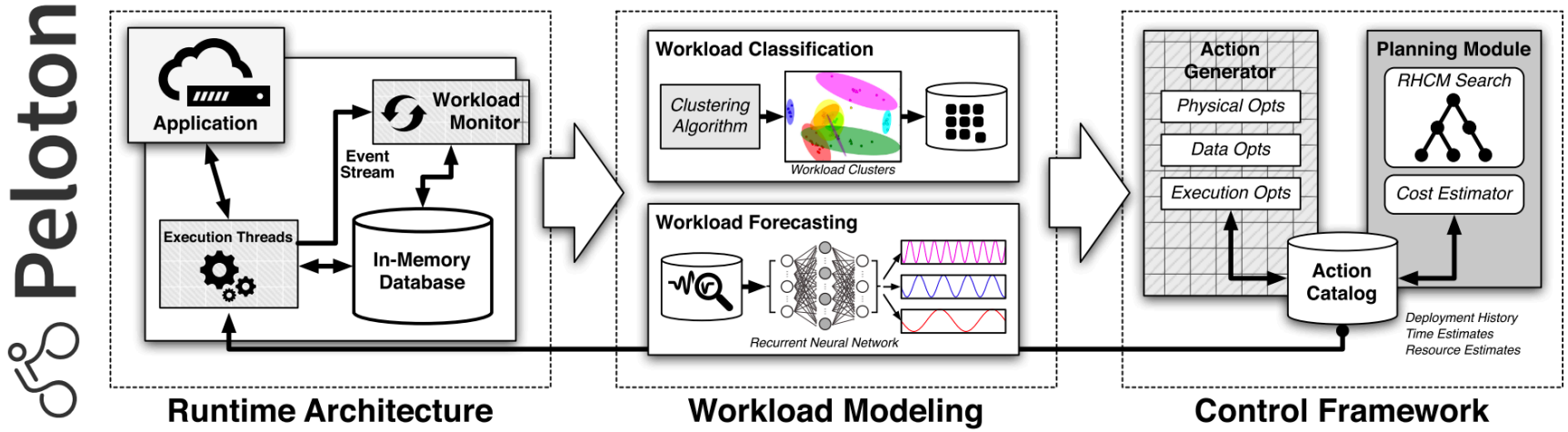
- ➢ Local optimum with high training overhead

☐Commercial Practices of AI4DB Works

- ➢ Heavy ML models are hard to implement inside kernel
- ➢ A uniform training platform is required

Peloton

- **Schedule optimization actions via workload forecasting**
  - **Embedded Monitor:** Detect the event stream
  - **Workload Forecast Model:** Future workload type
  - **Optimization Actions:** Tuning, Planning



Andy Pavlo, et al. Self-Driving Database Management Systems. In CIDR, 2017.

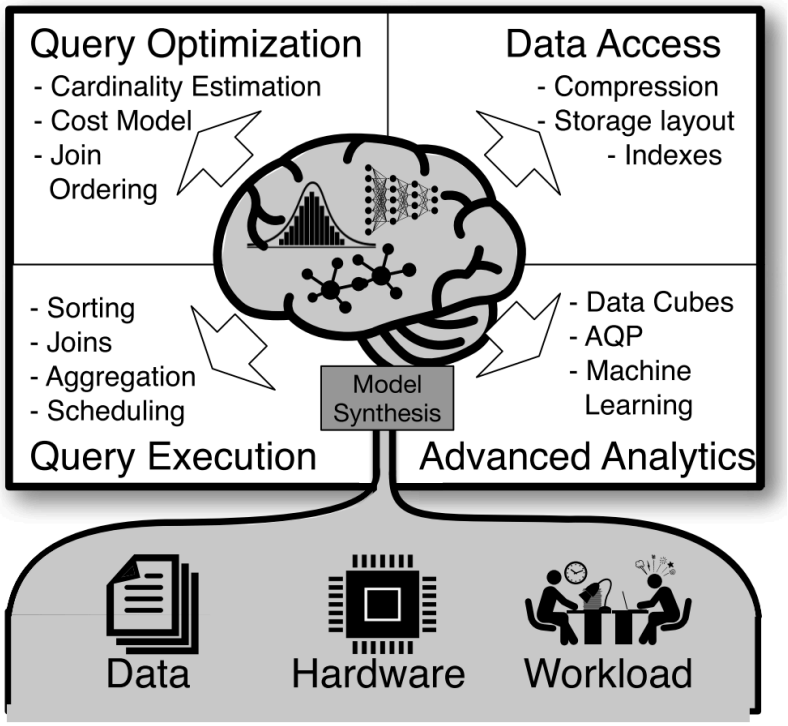
SageDB

☐ Customize DB design via learning the Data Distribution

- Learn Data Distribution by Learned CDF

$$M_{CDF} = F_{X_1,...X_m}(x_1,...x_m) = P(X_1 \leq x_1,...,X_m \leq x_m)$$

- Design Components based on the learned CDFs
  - Query optimization and execution
  - Data layout design
  - Advanced analytics

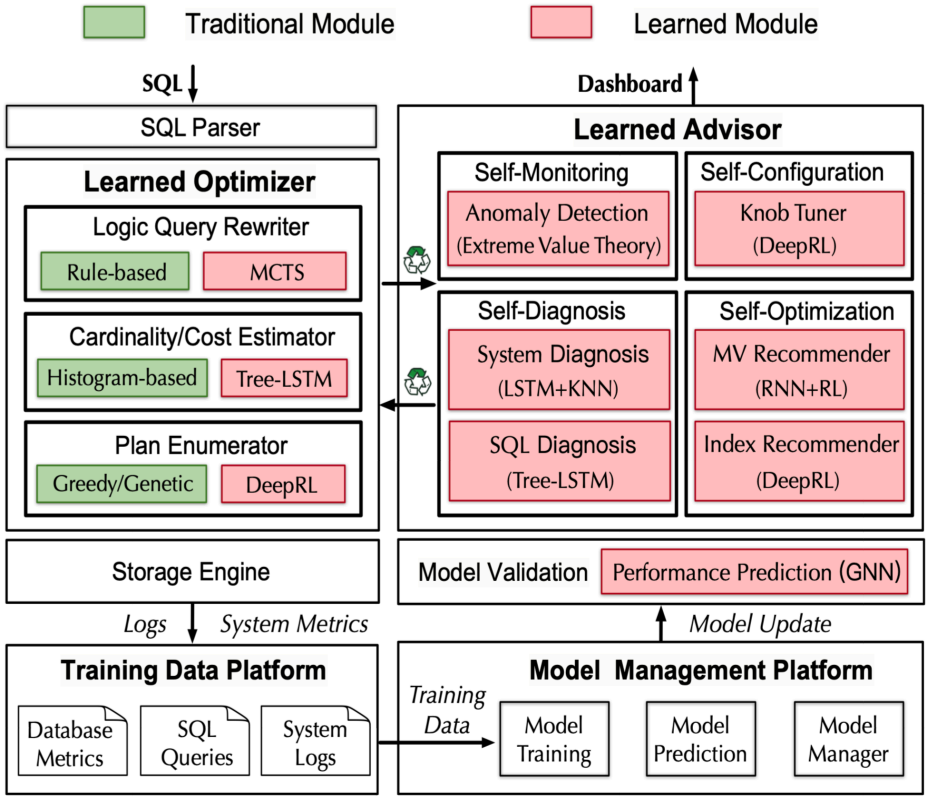


Tim Kraska, et al. SageDB: A Learned Database System. In CIDR, 2019.

openGauss

☐ Implement, validate, and manage learning-based modules

- ➢ **Learned Optimizer**
  - Query Rewriter
  - Cost/Card Estimator
  - Plan Enumerator
- ➢ **Learned Advisor**
  - Self-Monitoring
  - Self-Diagnosis
  - Self-Configuration
  - Self-Optimization
- ➢ **Model Validation**
- ➢ **Data/Model Management**



Guoliang Li, et al. openGauss: An Autonomous Database System. In VLDB, 2021.

Thanks

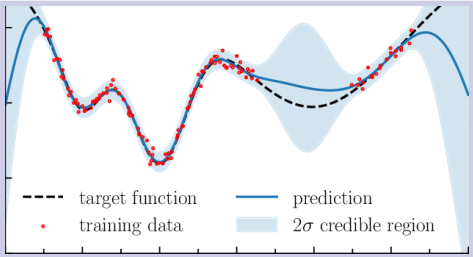
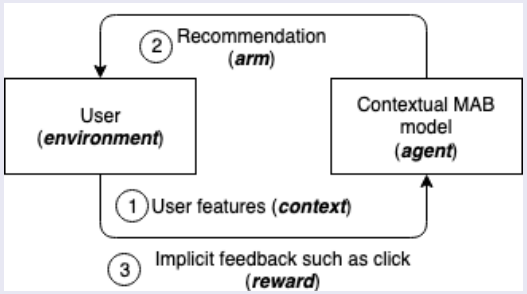
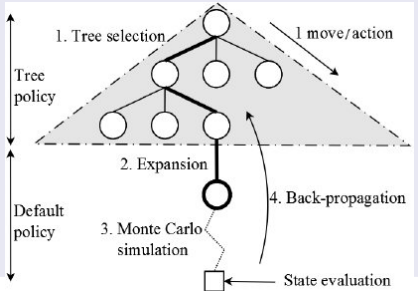
Machine Learing for Databases

Empirical Methods	Heuristic Search/Rules	e.g., knob tuning
	Dynamic Programming	e.g., Index Selection
	Maximum Spanning Tree	e.g., Database Partition
Supervised ML	Gaussian Process	e.g., knob tuning
	Bayesian Optimization	e.g., knob tuning
	CNN	e.g., card Estimation
	Tree-based Ensemble	e.g., card Estimation
	Kernel Density Estimation	e.g., card Estimation
	Uniform Mixture Model	e.g., card Estimation
	Causal Model	e.g., System Diagnosis
	Clustering Algorithns	e.g., System Diagnosis
	Annotated Plan Graph	e.g., System Diagnosis
	Dense Neural Network	e.g., knob tuning, view selection, index selection, lesarned Index, transactions, query latency prediction
	Encoder-Decoder	e.g., view selection
	Tree-LSTM	e.g., Cost estimation, plan enumerator
	Graph Neural Network	e.g., workload performance prediction
Unsupervised ML	AutoRegressive	e.g., card Estimation
	Sum-Product Network	e.g., card estimation
Semi-supervised ML	Meta Learning	e.g., knob tuning
	Pre-Training Network	e.g., query encoding
(Deep) Reinforcement Learning	DDPG	e.g., knob tuning
	DQN	e.g., view selection, index selection, plan enumerator
	Q-learning	e.g., view Selection , database partition, transactions
	MCTS	e.g., plan enumerator

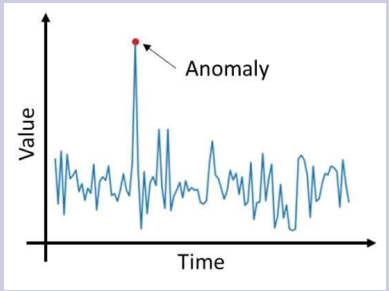
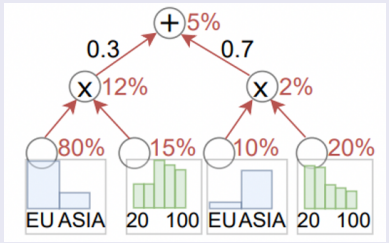
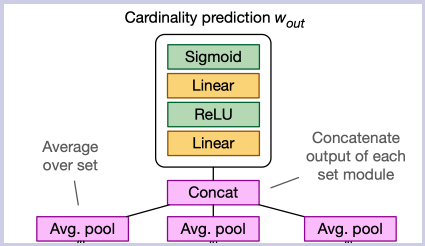
Summarization of AI4DB Techniques

	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

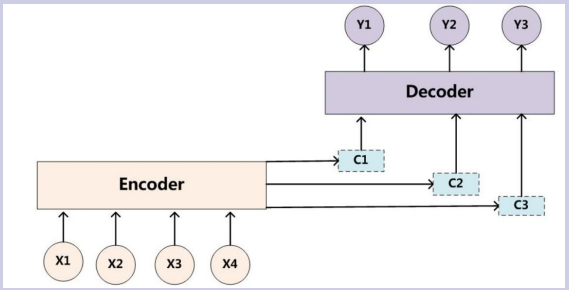
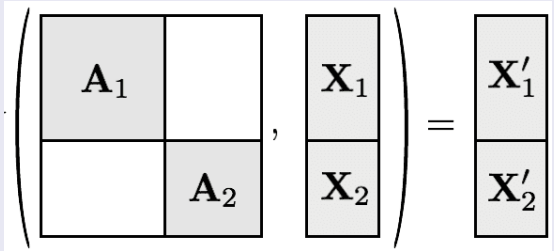
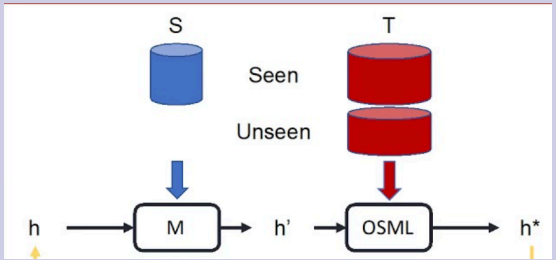
ML Models for Optimization Problems

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 (actor) or estimation (critic) policy with neural networks		
Monte Carlo Tree Search	Repeated iterations of four steps (selection, expansion, simulation, back-propagation) 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

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

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: Challenges

- 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)

Classical ML Methods

	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)

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: Challenges

- 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

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

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: Challenges

- 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)

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