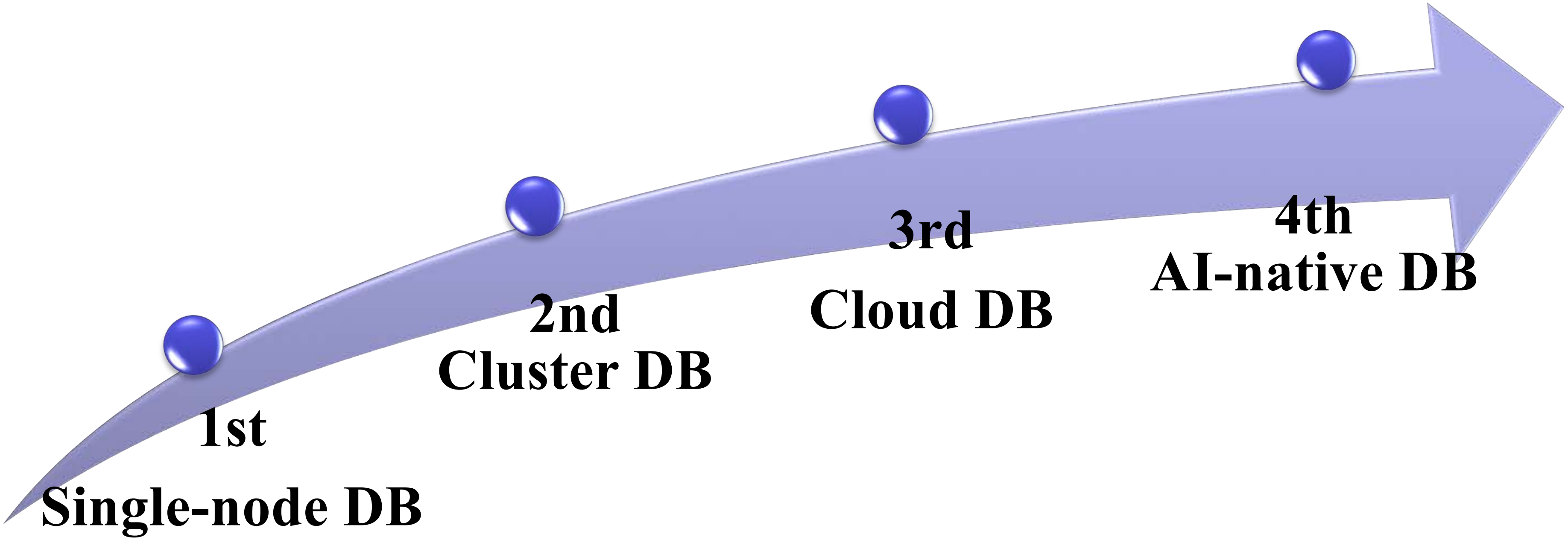
AI-Native Database

AI4DB & DB4AI

Guoliang Li
Tsinghua University

Redefine database architecture



1970s	1990s	2010s	2020s
Relation & SQL	Large scale	Internet scale	Heterogenous
High performance	High availability	Scale-out	Autonomous

An intelligent era calls for a more intelligent database

AI-Native Database

AI4DB

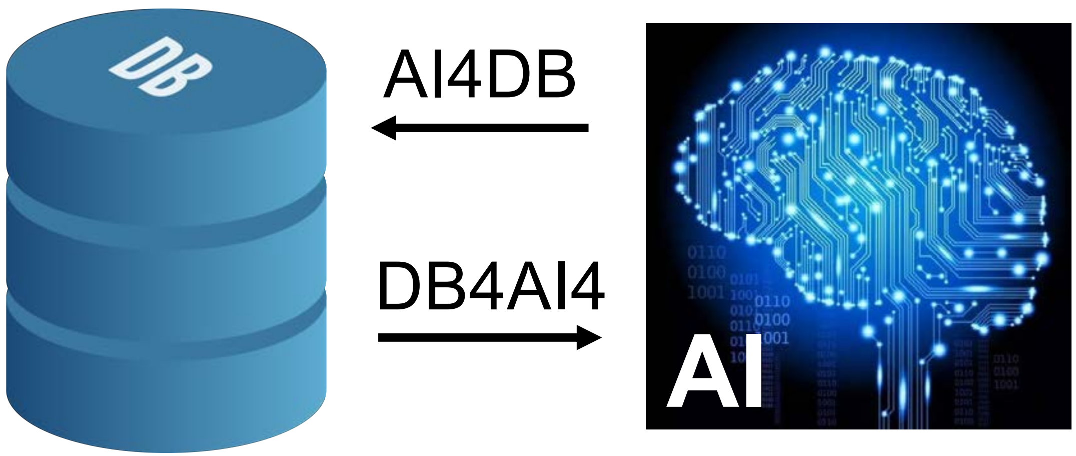
Manual → Automatic

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

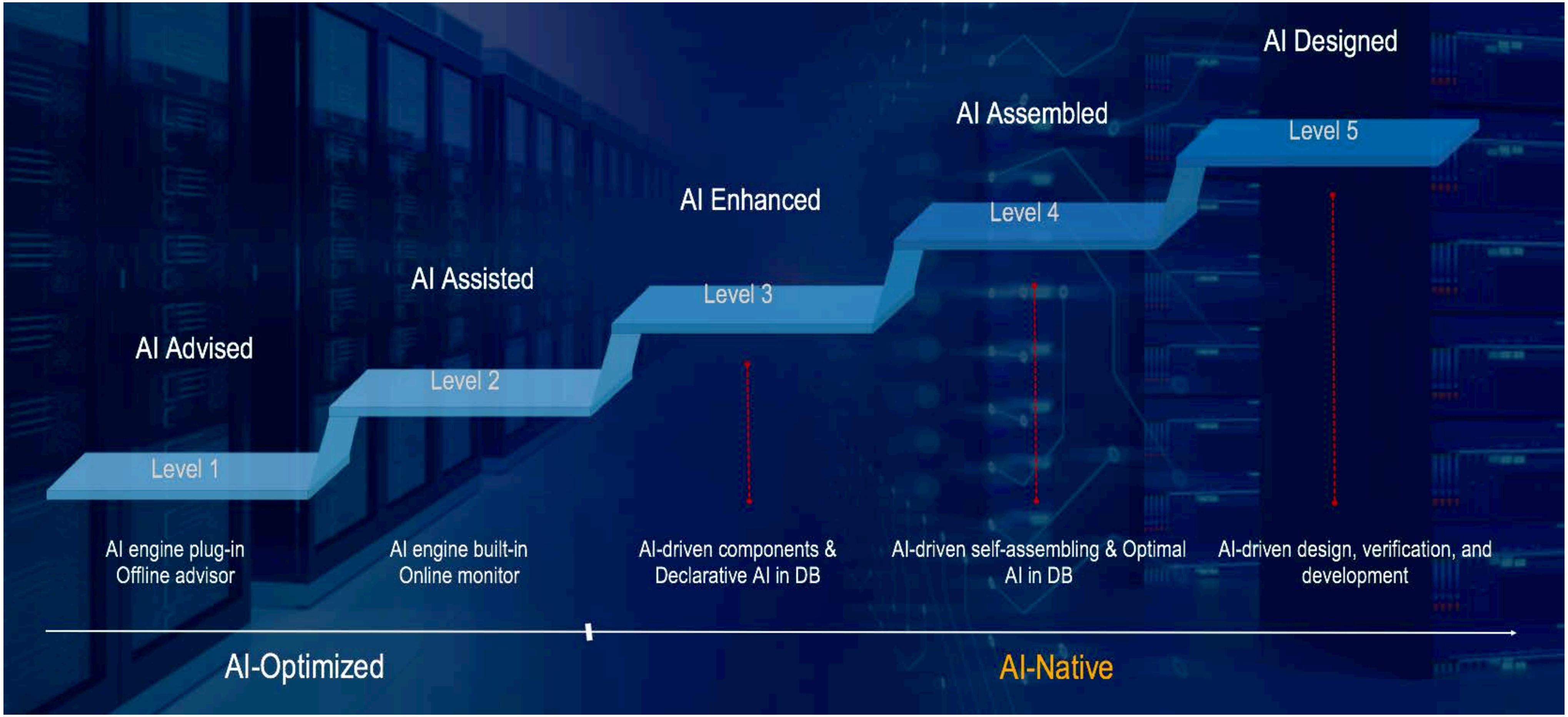
DB4AI

AI → as easy as DB

- ☐ Declarative AI
- ☐ AI optimization
- ☐ Data governance
- ☐ Data provenance
- ☐ Model management



AI-Native Database



Knob tuning
Index advisor
View advisor

Workload
modeling
scheduling

Learned index
optimizer
Declarative AI

Self assembling
AI optimization

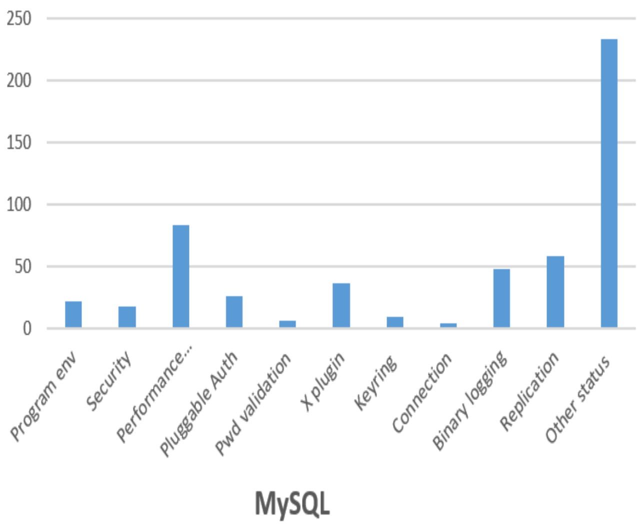
Self design
AI&DB Fusion

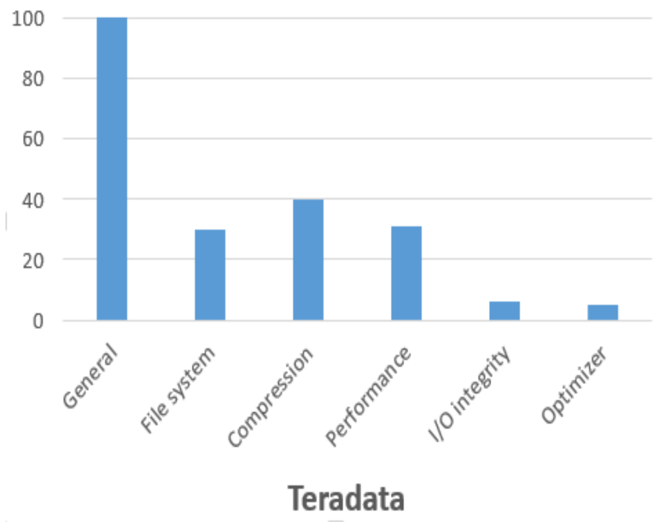
Level 1: AI-advised DB

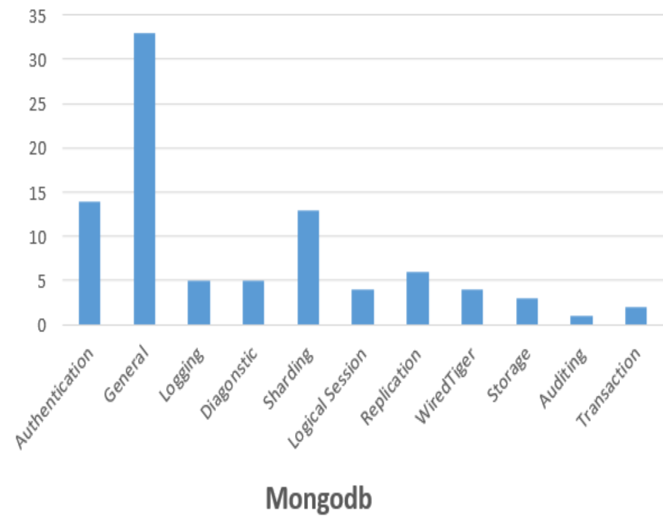
- Database advisor for making database more intelligent
  - Database Configuration
    - Knob tuning
    - Workload management
    - Automatic Upgrade
  - Database Optimization
    - Index advisor
    - View advisor
    - Partition advisor

AI for Knob Tuning

- **Automatic Tuning is important and challenging**
  - **Tunable options control nearly all aspects of runtime operations.**
  - **The number of knobs in a DBMS is huge and the relationships are complex.**

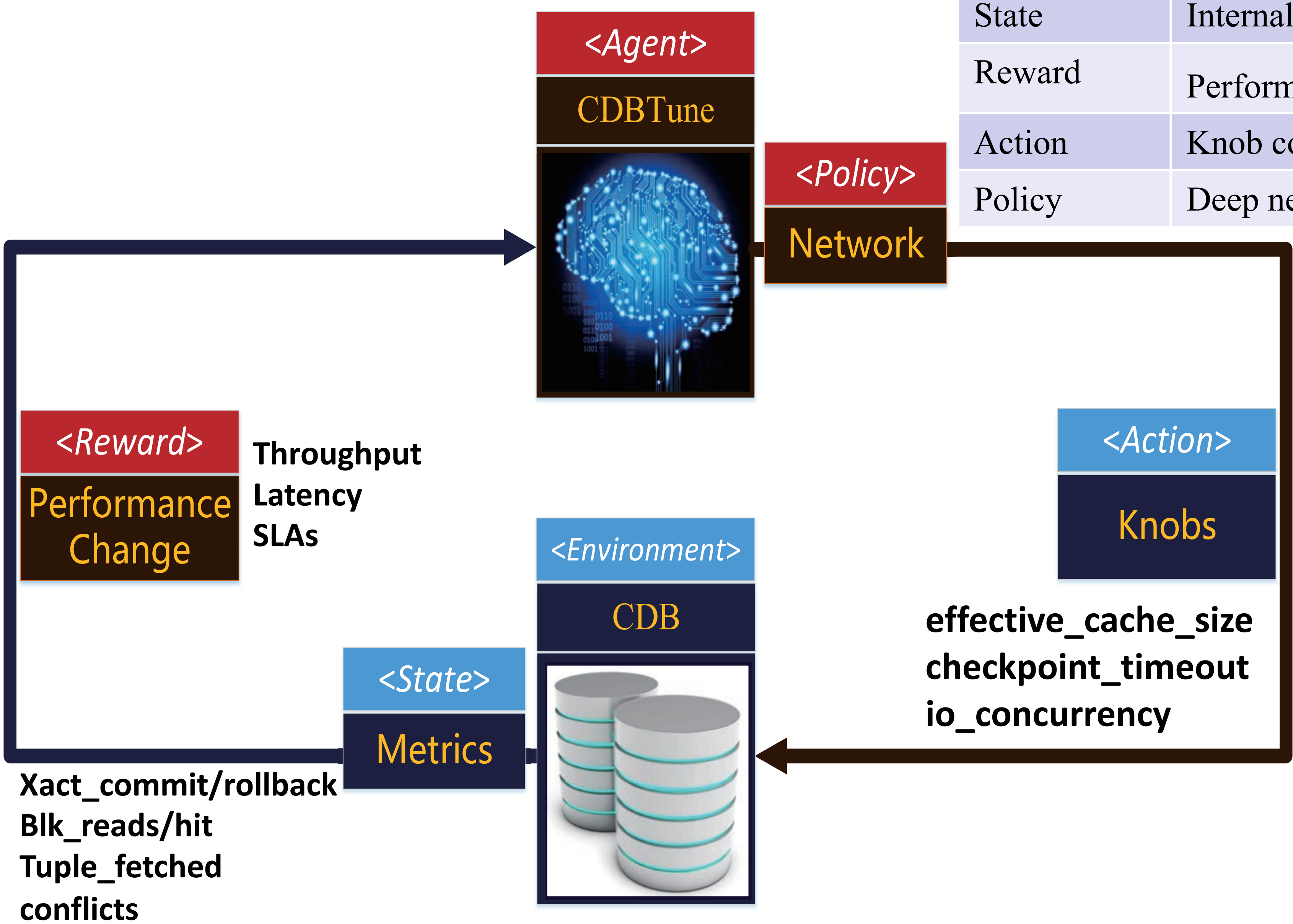






CDBTune

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



CDBTune

□CDBTune

- using deep reinforcement learning (DRL), an end-to-end automatic CDB (**C**loud **D**ata**B**ase) tuning system
  - deep deterministic policy gradient method (DDPG)
  - try-and-error strategy
- Characteristics:
  - end-to-end learning
  - using a limited number of samples
  - high-dimensional continuous knobs recommendation
  - reducing the possibility of Local Optimum
  - good adaptability
  - accelerates the convergence speed

CDBTune: Working Mechanism

☐Offline Training

- Step 1: builds a training model
- Step 2: trains the training model
  - Training Data
  - Training Model
  - Training Data Generation

☐Online Tuning

- Step 3: utilizes the model to recommend knob settings for an online tuning request
- Step 4: updates the training model by taking the tuning request as training data

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

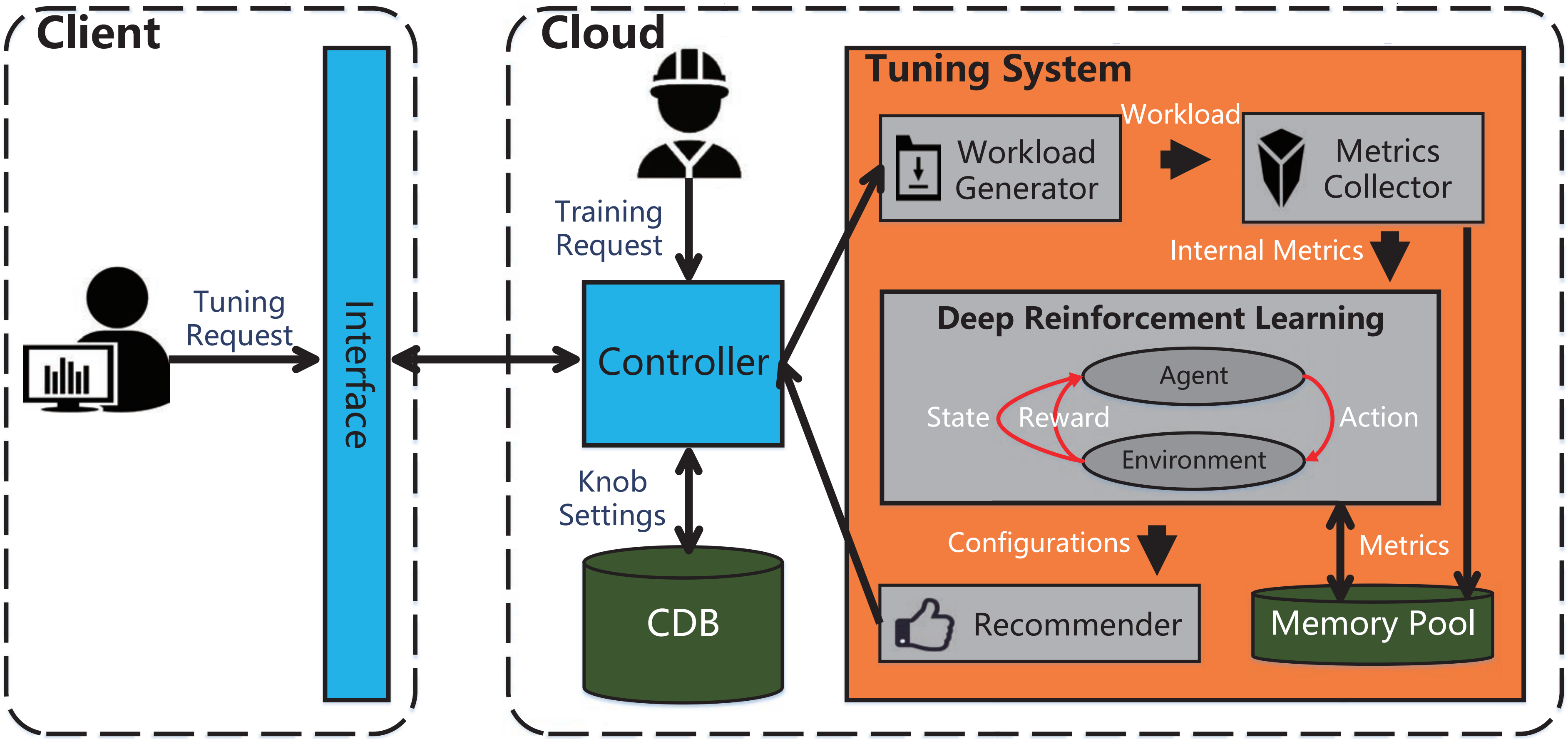
CDBTune

send request to the server through the local interface

interacts information among the client, CDB and CDBTune

conducts stress testing

collects and processes related metrics



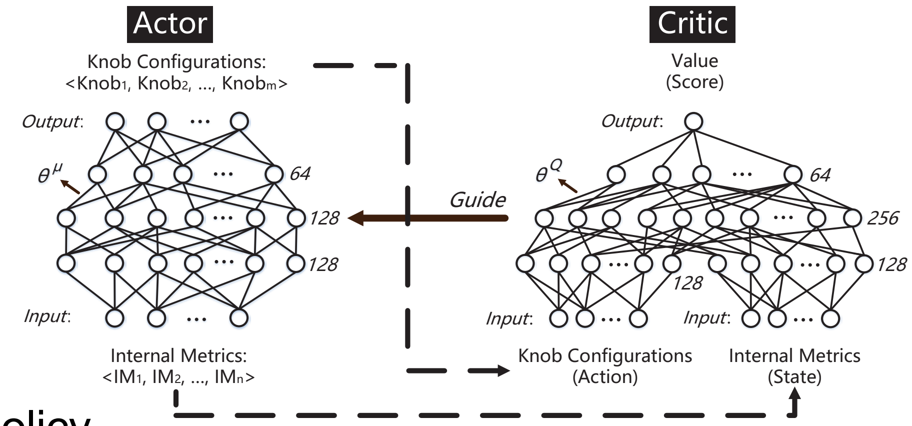
outputs the knob configurations

store processed data

Reinforcement Learning

◻ Reinforcement Learning

- Method: DDPG
- Goal: learn the best policy



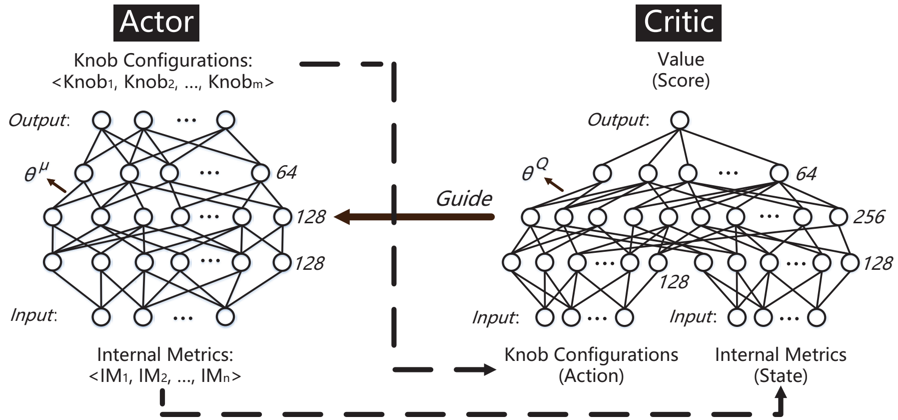
◻ Six key elements in RL

- **Agent**
  - receives reward and state, updates the policy
- **Environment**
  - Environment is the tuning target, specifically an instance of CDB
- **State**
  - the current state of the agent, i.e., the 63 metrics
  - describe the state at time t as s_t
- **Reward**
  - a scalar described as r_t
- **Action**
  - described as a_t corresponds to a knob tuning operation
- **Policy**
  - described as $\mu(s_t)$
  - a mapping from state to action

DDPG

☐ About DDPG

- a policy-based method, combination of DQN and actor-critic
- learn the policy with high dimensional states and actions



☐ DDPG design

- **Policy function: $a_t = \mu(s_t | \theta^{\mu})$**
  - θ^{μ}: mapping the state s_t to the value of action a_t
- **Critic function: $Q(s_t, a_t | \theta^Q)$**
  - represent the value (score) with specific action a_t and state s_t
  - θ^Q is learnable parameters
- **Inheriting from Bellman Equation and DQN:** $Q^{\mu}(s,a) = \mathbb{E}_{r_t, s_{t+1} \sim E}[r(s_t, a_t) + \gamma Q^{\mu}(s_{t+1}, \mu(s_{t+1}))]$
  - policy $\mu(s)$ is deterministic, s_{t+1} is the next state, $r_t = r(s_t, a_t)$ is the reward function, and γ is a discount factor
- **Minimize the training objective:** $\min L(\theta^Q) = \mathbb{E}[(Q(s, a|\theta^Q) - y)^2]$
  - where $y = r(s_t, a_t) + \gamma Q^{\mu}(s_{t+1}, \mu(s_{t+1}) | \theta^Q)$

Reward Function

- **About Reward Function**
  - feedback information between the agent and environment
  - guides the agent to learn by telling what behavior is right or wrong
- **The design of the reward function**
  - r, T and L denote reward, throughput and latency
  - 1. At time t, calculate the rate of performance change Δ from time $t-1$ and the initial time to time t respectively.
  - 2. Reward function: use r to denote the sum of rewards of throughput and latency:

$$\Delta T = \begin{cases} \Delta T_{t\to 0} = \dfrac{T_t - T_0}{T_0} \\ \Delta T_{t\to t-1} = \dfrac{T_t - T_{t-1}}{T_{t-1}} \end{cases}$$

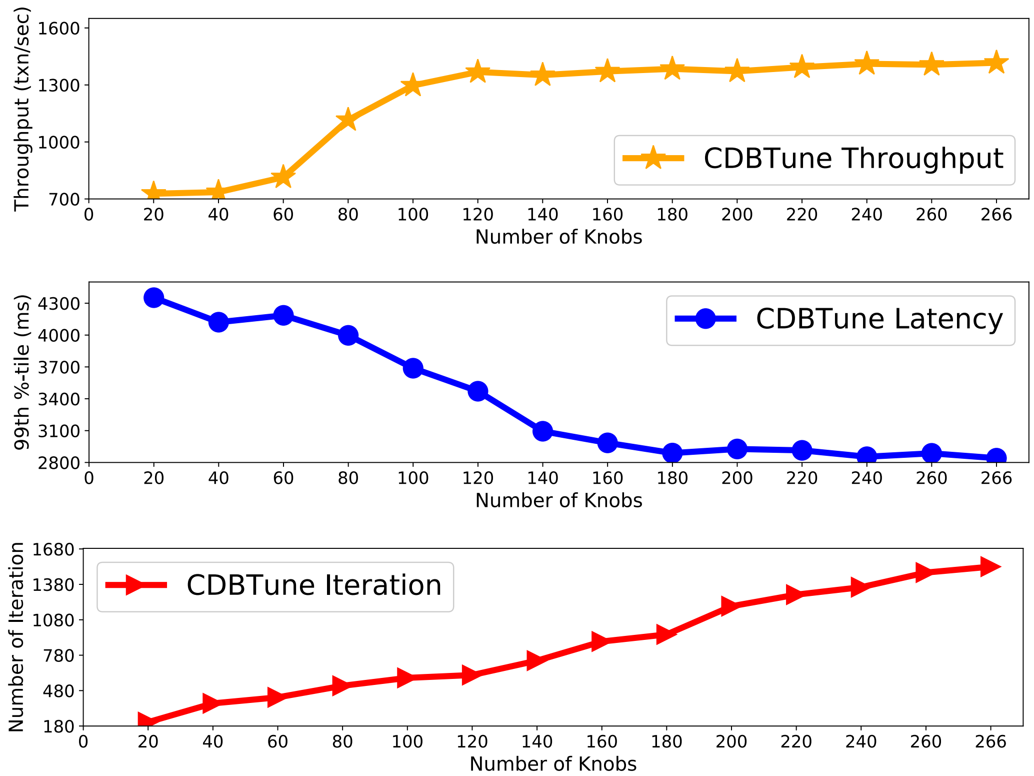
$$r = c_T * r_T + c_L * r_L$$

$$\Delta L = \begin{cases} \Delta L_{t\to 0} = \dfrac{-L_t + L_0}{L_0} \\ \Delta L_{t\to t-1} = \dfrac{-L_t + L_{t-1}}{L_{t-1}} \end{cases}$$

  - r_T : the reward of throughput
  - r_L : the reward of latency
  - r : the sum of rewards of throughput and latency
  - c_T and c_L are different coefficients

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

Results



Workload	BestConfig		DBA		OtterTune	
	T	L	T	L	T	L
RW	↑ 68.28%	↓ 51.65%	↑ 4.48%	↓ 8.91%	↑ 29.80%	↓ 35.51%
RO	↑ 42.15%	↓ 43.95%	↑ 4.73%	↓ 11.66%	↑44.46%	↓ 23.63%
WO	↑ 128.66%	↓ 61.35%	↑ 46.57%	↓ 43.33%	↑ 91.25%	↓ 59.27%

Results on Postgres

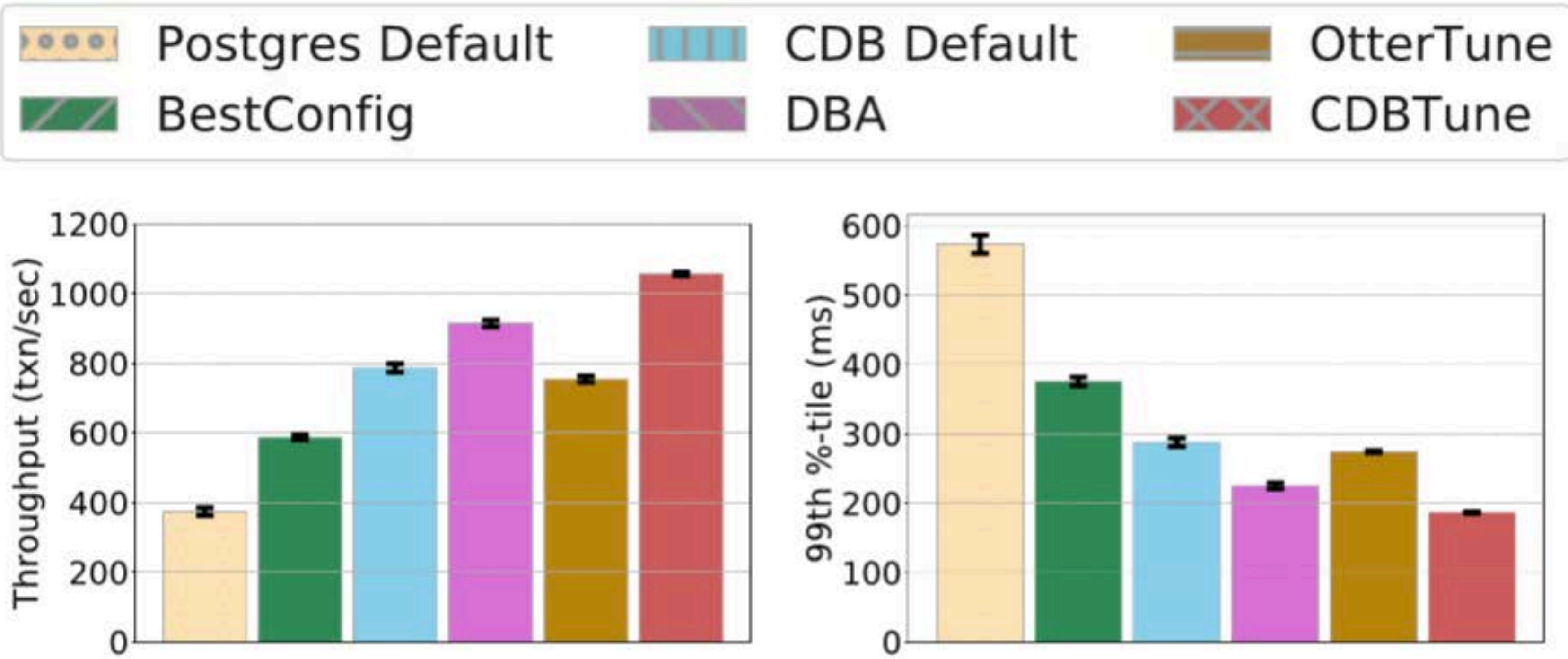
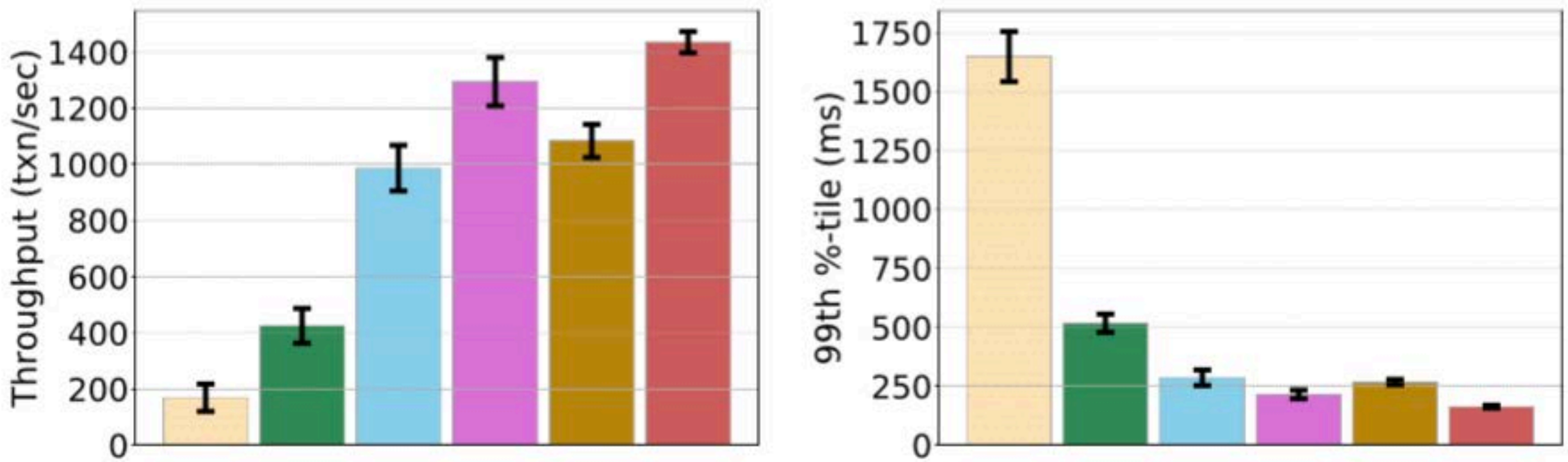


Figure 14: Performance comparison for TPC-C workload using instance CDB-D among CDBTune, Postgres default, CDB default, BestConfig, DBA and OtterTune (on Postgres).

Results on MySQL



(a) TCP-C (Throughput) (b) TCP-C (99%-tile Latency)

Figure 18: Performance on TPC-C for local MySQL.

Results on MongoDB

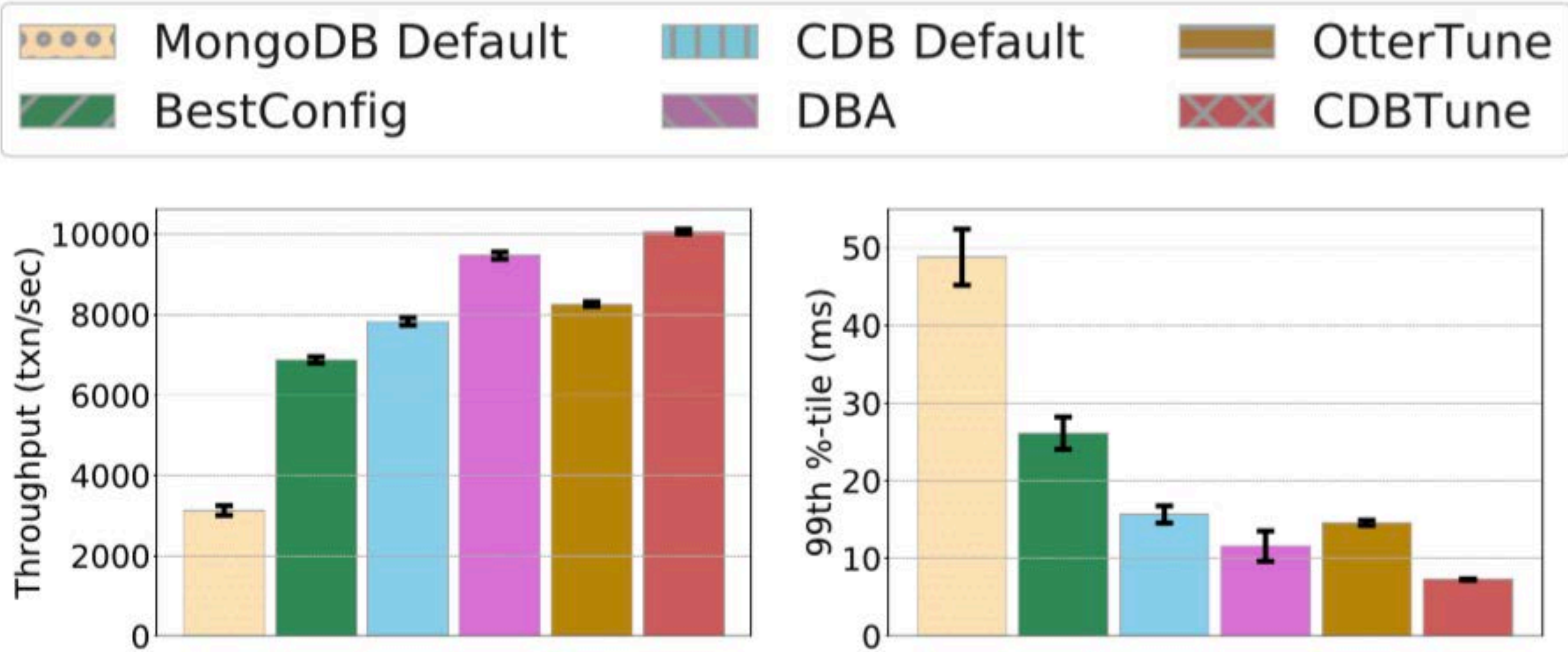
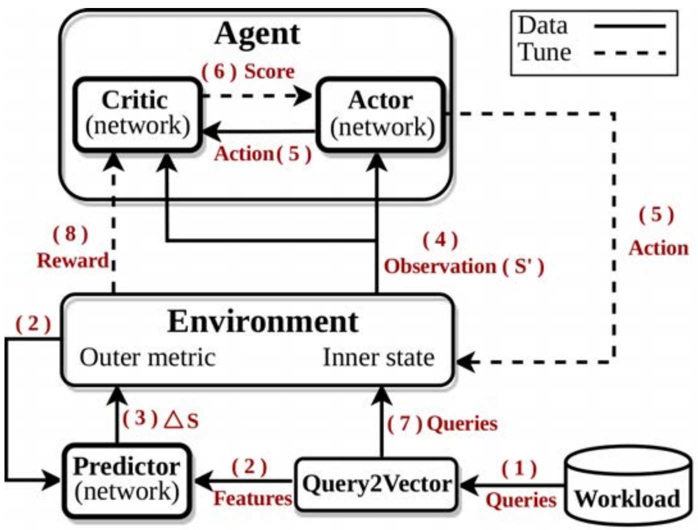
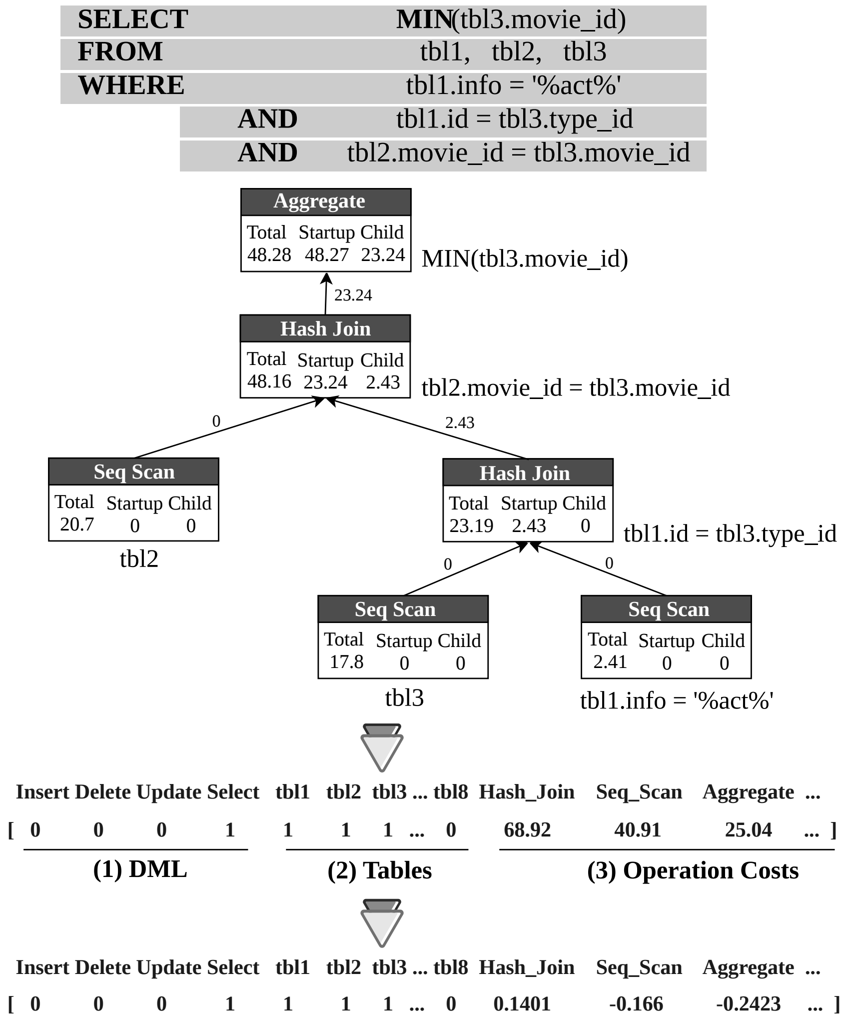


Figure 13: Performance comparison for YCSB workload using instance CDB-E among CDBTune, MongoDB default, CDB default, BestConfig, DBA and OtterTune (on MongoDB).

QTune: Query-Aware Tuning

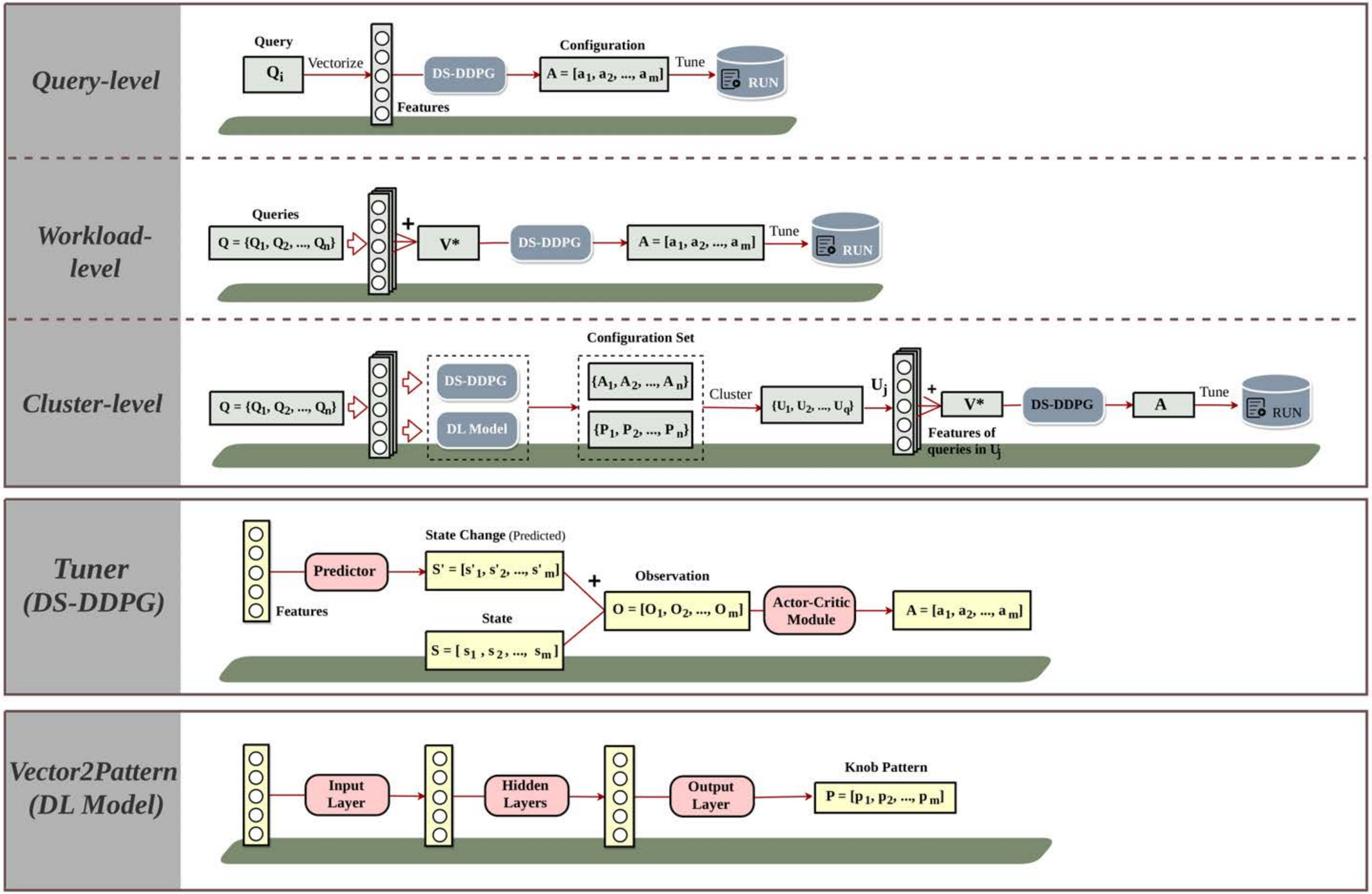
- Query-aware tuning
- Encoding queries
- Encoding cost
- Double state tuning





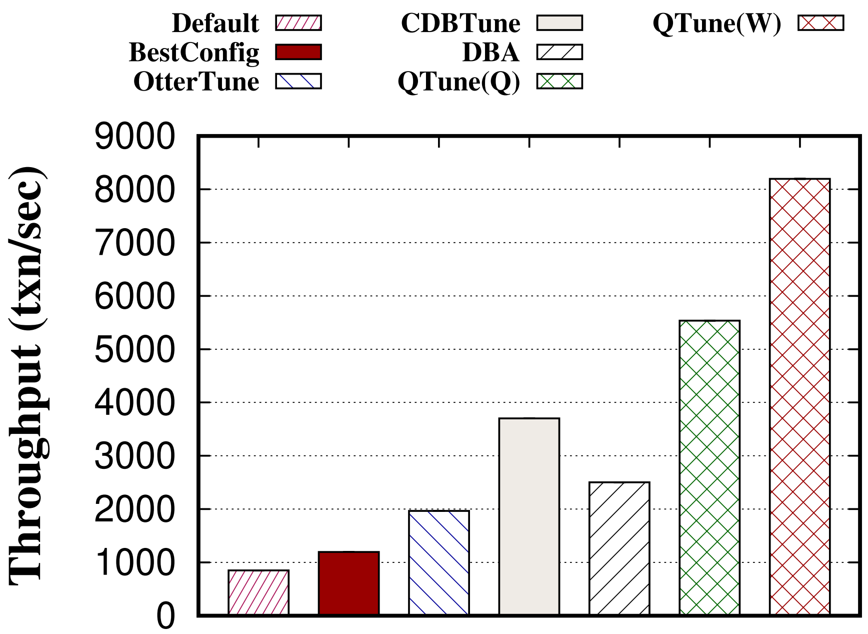
Guoliang Li, Xuanhe Zhou. QTune: A Query-Aware Database Tuning System with Deep Reinforcement Learning. VLDB 2019

QTune

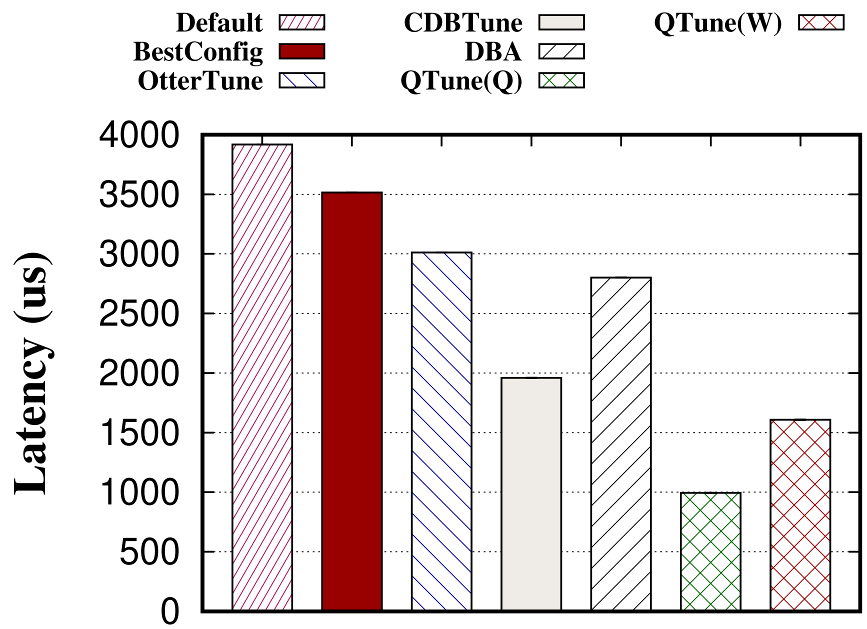


Query-level
Query
Q_i
Vectorize
Features
DS-DDPG
Configuration
$A = [a_1, a_2, ..., a_m]$
Tune
RUN
Workload-level
Queries
$Q = \{Q_1, Q_2, ..., Q_n\}$
+
V*
DS-DDPG
$A = [a_1, a_2, ..., a_m]$
Tune
RUN
Cluster-level
$Q = \{Q_1, Q_2, ..., Q_n\}$
DS-DDPG
DL Model
Configuration Set
$\{A_1, A_2, ..., A_n\}$
$\{P_1, P_2, ..., P_n\}$
Cluster
$\{U_1, U_2, ..., U_q\}$
U_j
+
V*
Features of queries in U_j
DS-DDPG
A
Tune
RUN
Tuner (DS-DDPG)
Features
Predictor
State Change (Predicted)
$S' = [s'_1, s'_2, ..., s'_m]$
+
State
$S = [s_1, s_2, ..., s_m]$
Observation
$O = [O_1, O_2, ..., O_m]$
Actor-Critic Module
$A = [a_1, a_2, ..., a_m]$
Vector2Pattern (DL Model)
Input Layer
Hidden Layers
Output Layer
Knob Pattern
$P = [p_1, p_2, ..., p_m]$

QTune



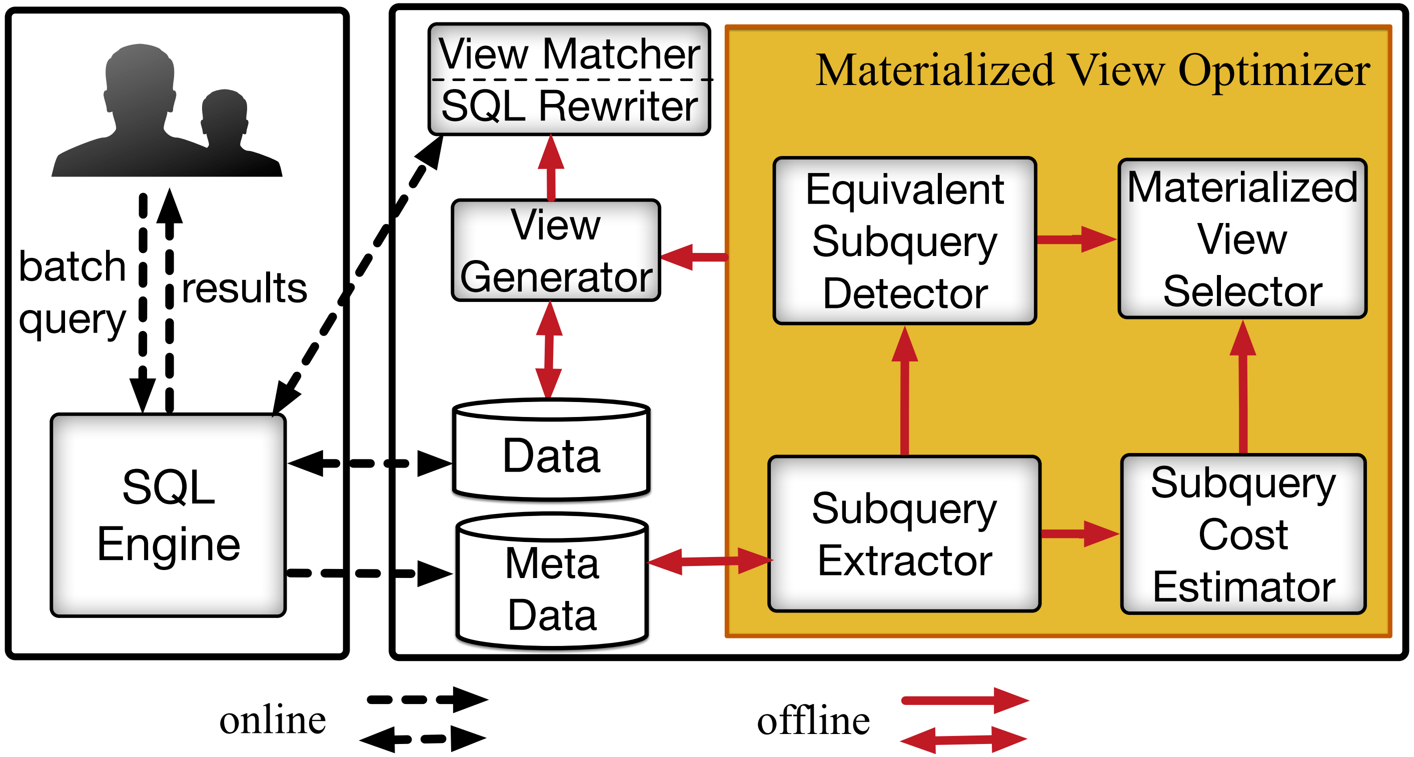
(a) Sysbench (RW)



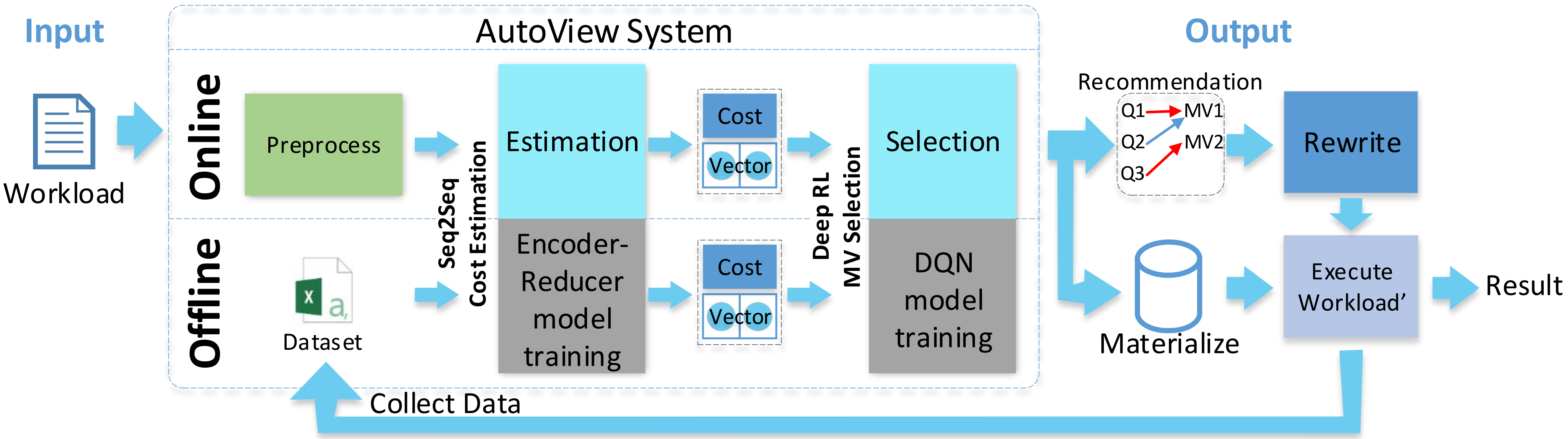
(d) Sysbench (RW)

View Advisor

- Equivalent subquery detector
- Subquery cost/benefit estimator
- View selector



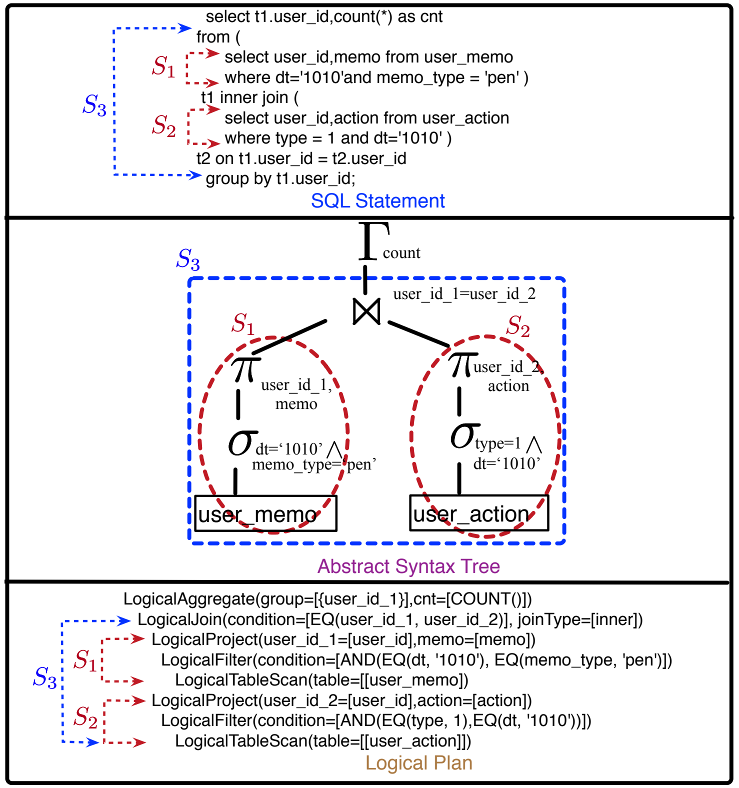
View Advisor: Framework

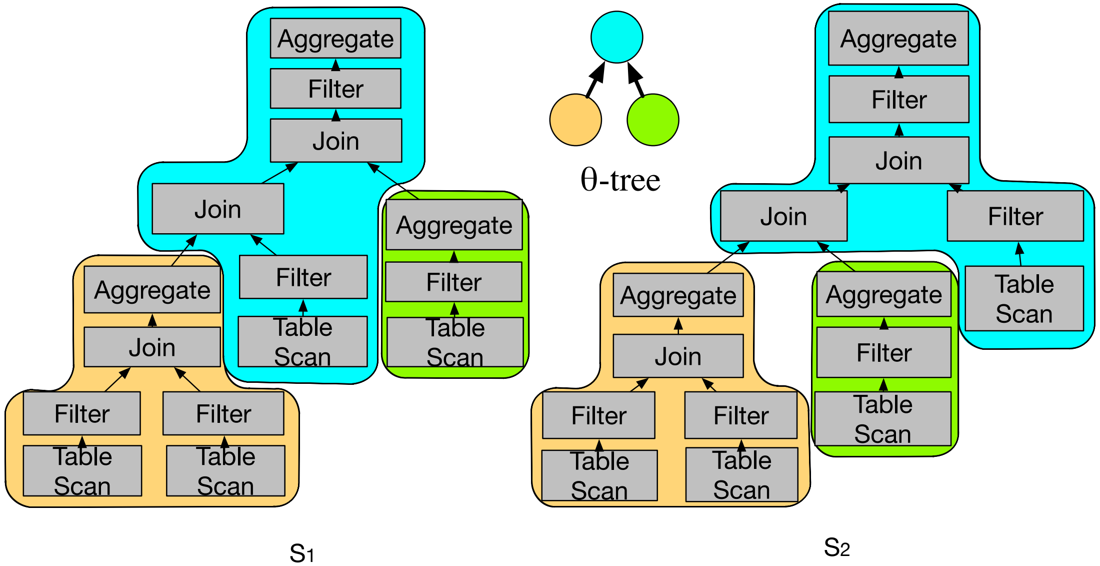


1. MV Candidate Generation
2. MV Estimation Model - encoder-reducer
3. MV Selection - DRL
4. MV-aware Query Rewriting

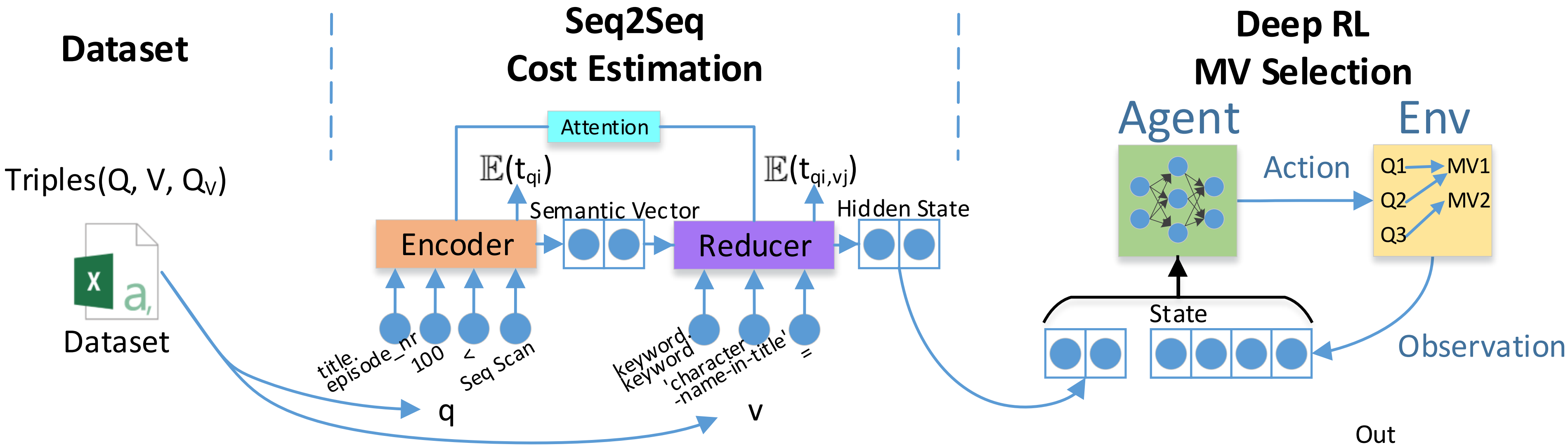
View Advisor: Equivalent Subquery

- Expensive to verify equivalence
- Extract SPJG segments
- Evaluate SPJG segments

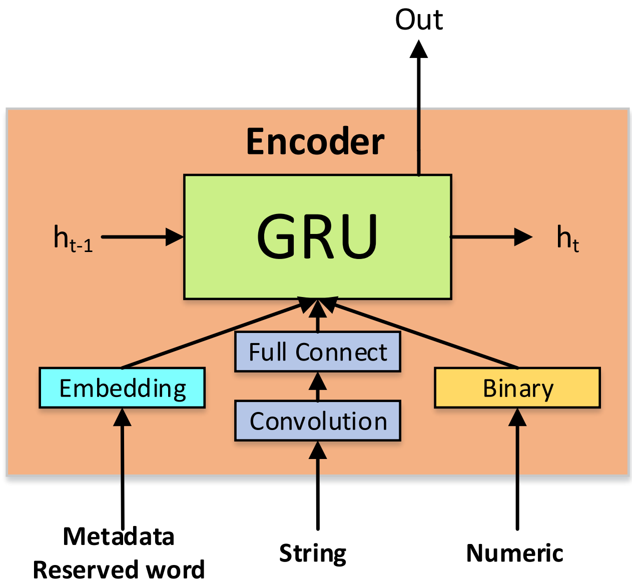




View Advisor: Cost Estimator

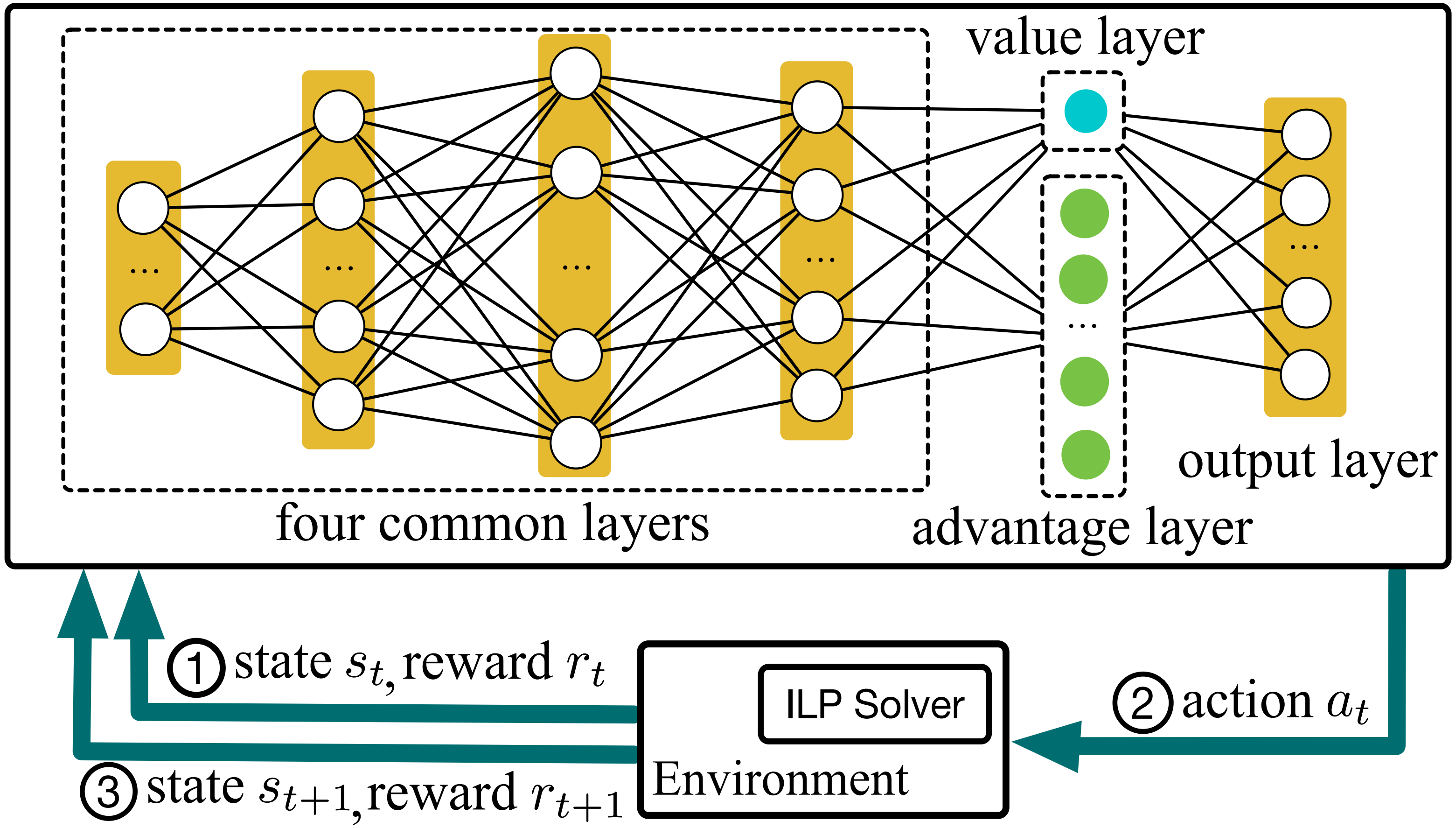


1. Serializing and Encoding
2. Encoder-Reducer Model
   - encoder – cost without view
   - reducer – cost with view
3. Attention Mechanism
4. Fine-tuning

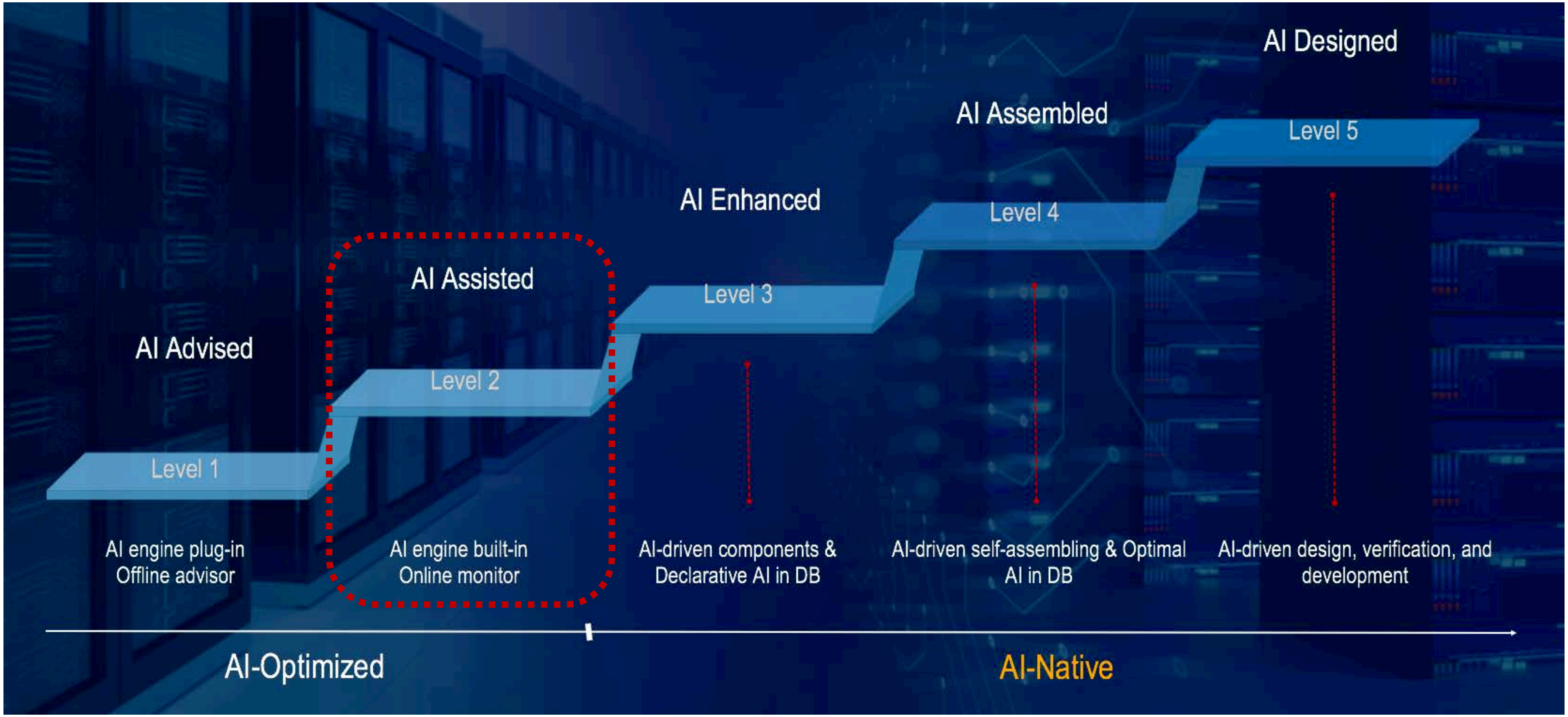


View Advisor: Subquery Selector

- Learn the benefit of materializing a view of a subquery



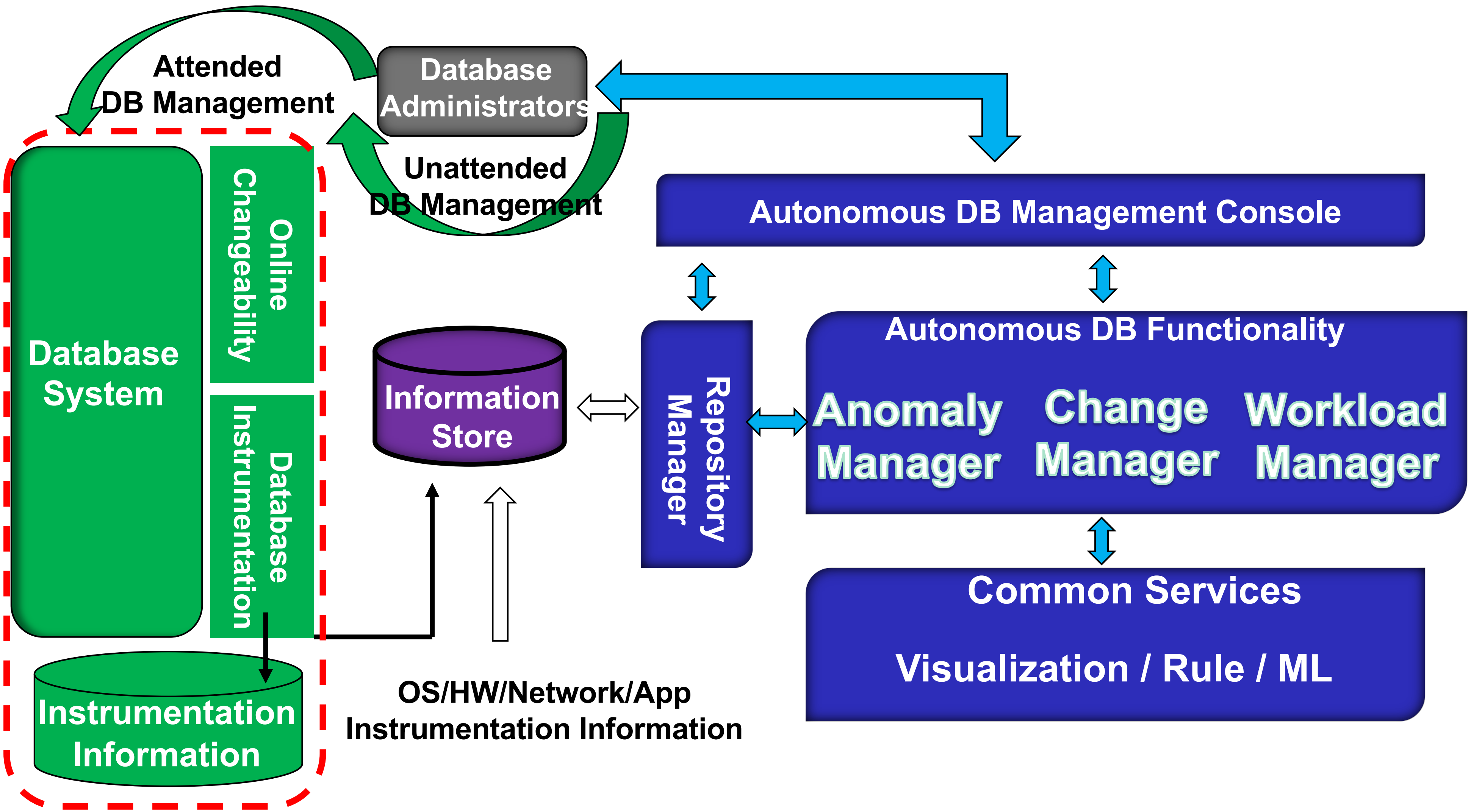
AI-Native Database



Level 2: AI-Assisted DB

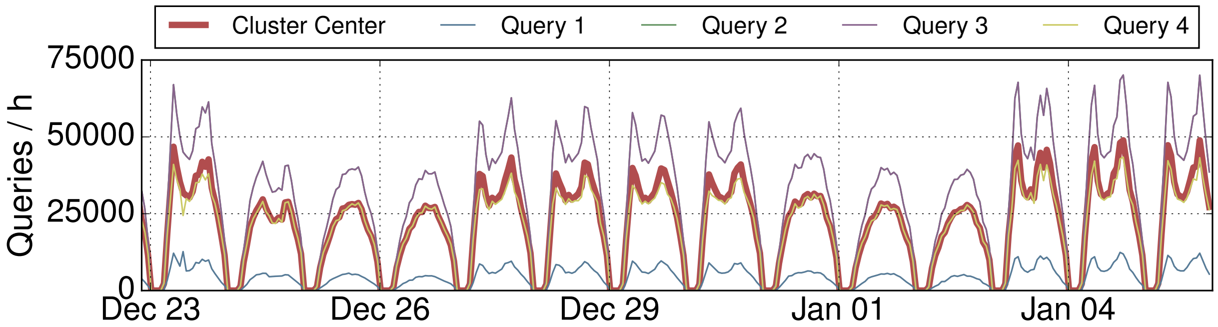
- Monitor and manage DB
  - Self monitoring
    - Statistics, workload, system
  - Self diagnosis
    - Error detection
  - Self healing
    - Failure recovery
  - Self configuring
    - Workload, upgrade
  - Self optimizing
    - SQL rewriter, online statistics

Self Monitoring

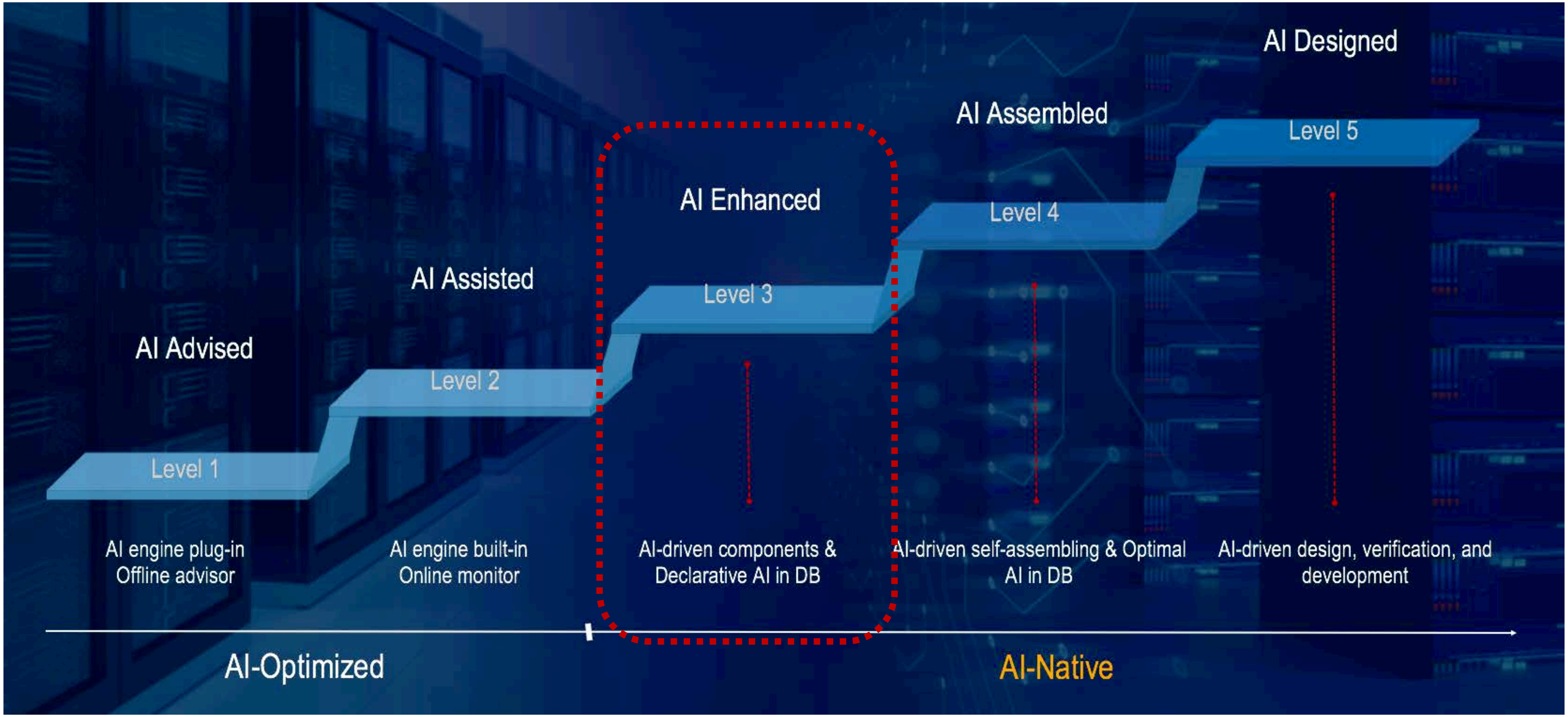


Workload Configuration

- **Workload modelling**
  - Feature, cost, latency, resources
- **Workload scheduling**
  - Prioritize workload
  - OLAP & OLTP
- **Workload prediction**
  - Predicting workloads



AI-Native Database

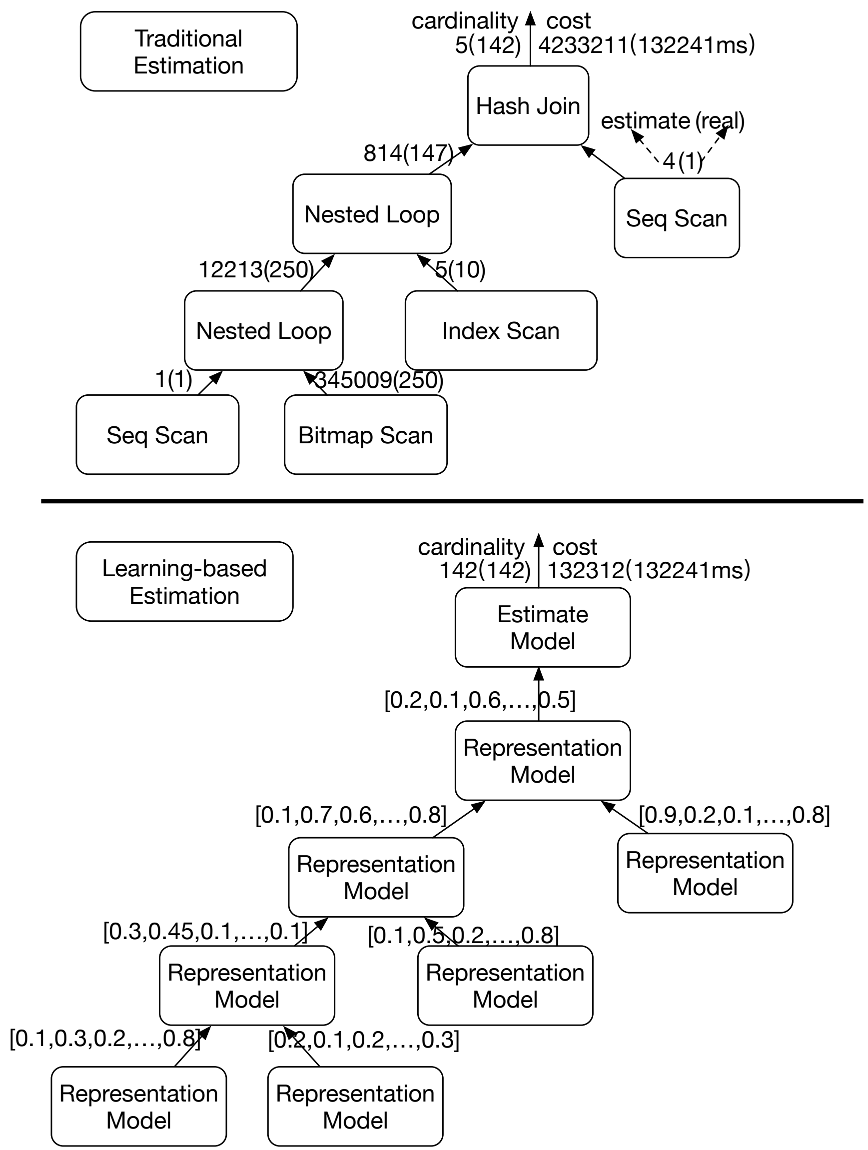


Level 3: AI-Enhanced DB

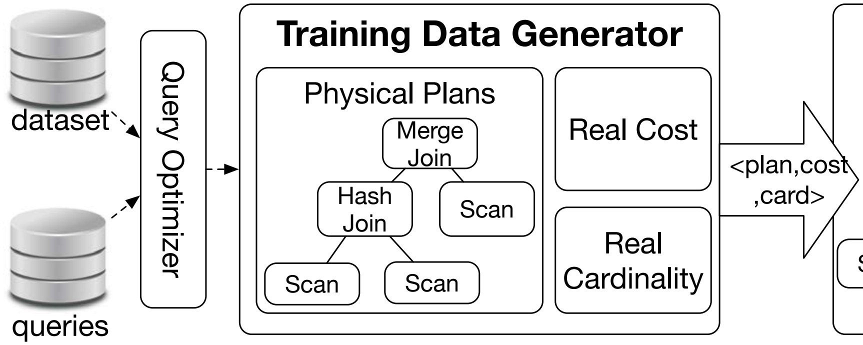
- **AI4DB: Learned DB components**
  - Learned Index
  - Learned Cost estimator
  - Learned Optimizer
  - Learned Statistics
- **DB4AI: Declarative AI**
  - Use SQL for using AI algorithms
  - Lower the burden of using AI

Learned Cost Estimator

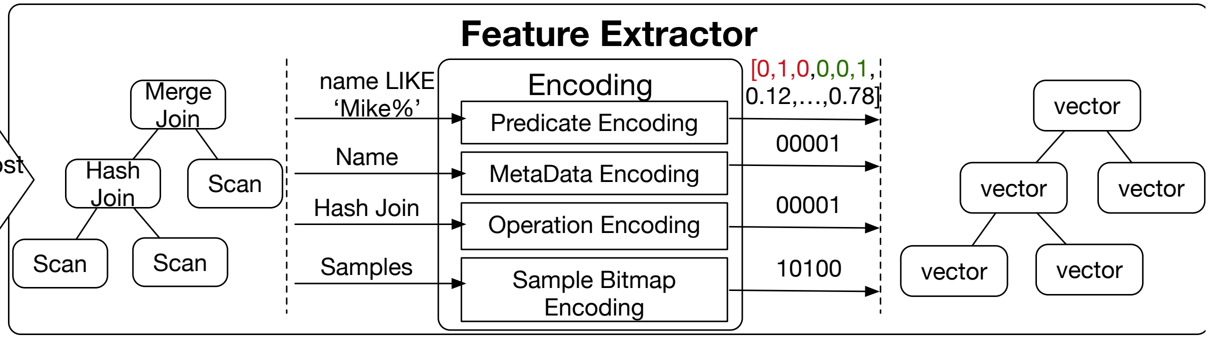
- **Traditional Cost Estimator**
  - Histogram
  - Sketch
  - Empirical functions
  - Failed for correlations between multiple tables
- **Learned Cost Estimator**
  - Estimation model
  - Tree-structure model
  - Predicate embedding



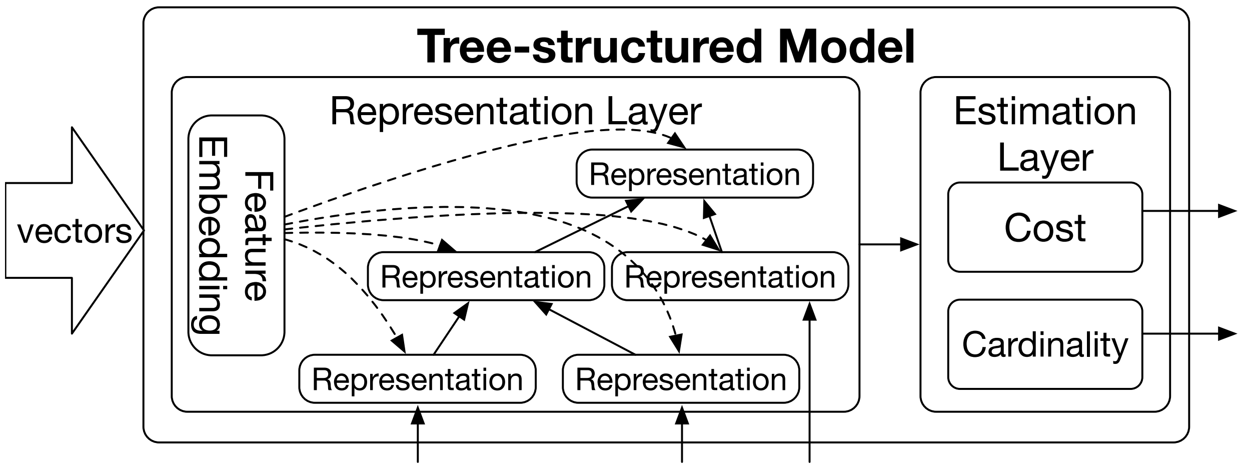
Learned Cost Estimator



1 Training Data Generator



2 Feature Extractor

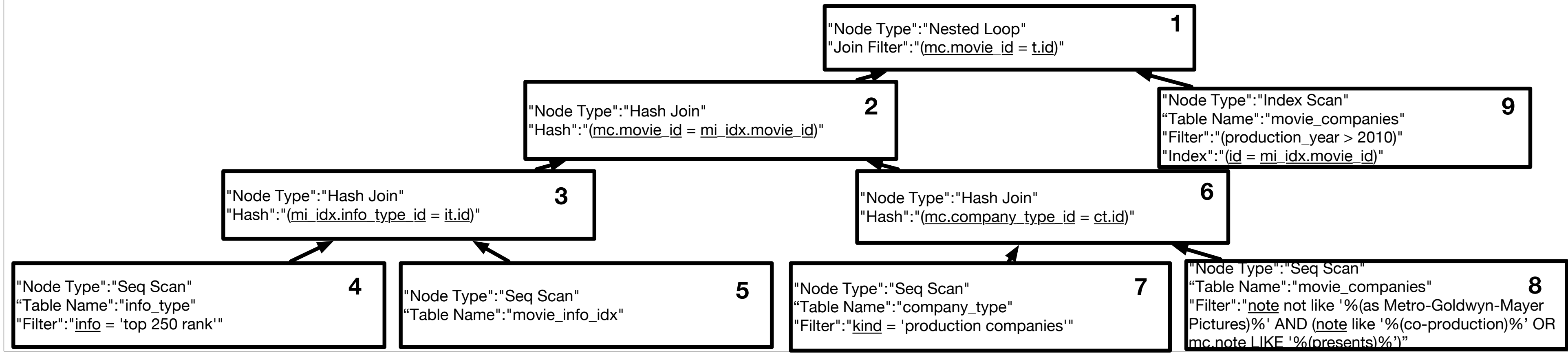


3 Tree-structured Estimation

Learned Cost Estimator

Feature Extraction

Execution Plan



One-hot Encoding

Operator	one-hot
=	0001
>	0010
not like	0100
like	1000

Operation	one-hot
Nested Loop	0001
Hash Join	0010
Seq Scan	0100
Index Scan	1000

Table Name	one-hot
info_type	0001
movie_info_idx	0010
company_type	0100
movie_companies	1000

Column	one-hot
mc.movie_id	000000000001
t.id	000000000010
mi_idx.movie_id	000000000100
mi_idx.info_type_id	000000001000
it.id	000000010000
it.info	000000100000
mc.company_type_id	000001000000
ct.id	000010000000
ct.kind	000100000000
mc.note	001000000000
mc.production_year	010000000000
mc.id	100000000000

Sample Datasets

info_type.info
top 260 rank
top 270 rank
top 250 rank
top 250 rank

movie_companies.note
(as Metro-Goldwyn-Mayer Pictures) (co-production)
(2006)(USA)(TV)
(2006)(worldwide)(TV)
(2011)(UK)(TV)

movie_companies.production_year
1987
2013
1995
1966

company_type.kind
production companies
distributors
special effects companies
miscellaneous companies

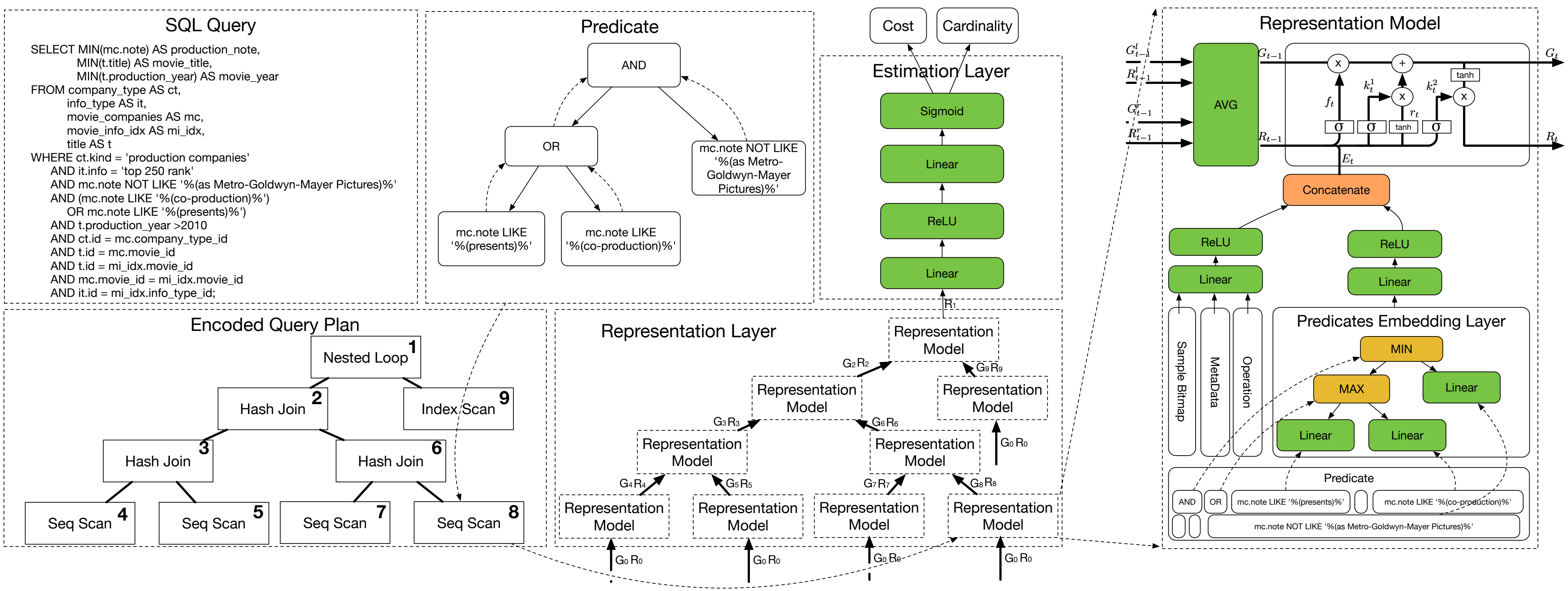
Dictionary

Token	Representation
'top 250 rank'	0.14,0.43, …,0.92
'production companies'	0.51,0.22, …,0.11
'(as Metro-Goldwyn-Mayer Pictures)'	0.91,0.35, …,0.25
'(co-production)'	0.37,0.11, …,0.02
‘(presents)’	0.13,0.41, …,0.76

id	Operation	MetaData	Predicate	Sample Bitmap
1	0001	padding	000000000001 0001 000000000010	padding
2	0010	padding	000000000001 0001 000000000100	padding
3	0010	padding	000000001000 0001 000000010000	padding
4	0100	0001	000000100000 0001 0.14,0.43,…,0.92	0011
5	0100	0010	padding	1111
6	0010	padding	000001000000 0001 000010000000	padding
7	0100	0100	000100000000 0001 0.51,0.22,…,0.11	1000
8	0100	1000	001000000000 0100 0.91,0.35,…,0.25 001000000000 1000 0.37,0.11,…,0.02 001000000000 1000 0.13,0.41,…,0.76	1000
9	1000	1000	010000000000 0010 0.81 100000000000 0001 00000000100	0100

Learned Cost Estimator

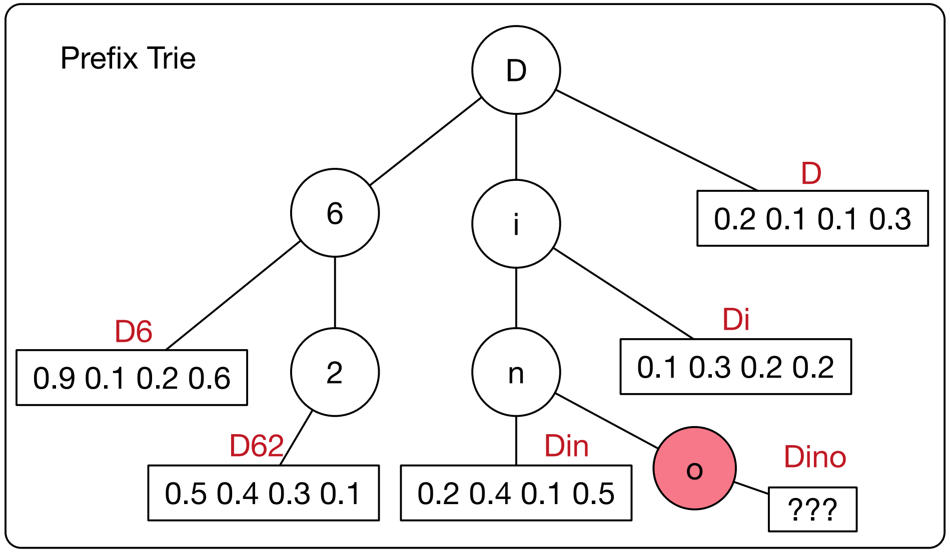
Tree-structured Model



Learned Cost Estimator

String Embedding

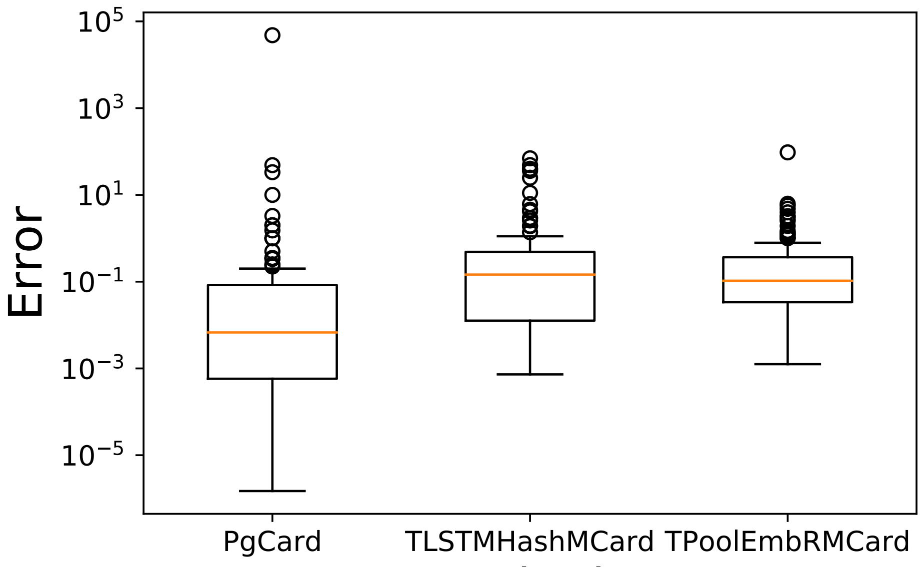
	Rules
"Din" → "Din"	$\langle Prefix, P_t(\text{"}Din\text{"}), 3\rangle$ $\langle Prefix, P_C P_t(\text{"}in\text{"}), 3\rangle$
"Dinos" → "Din"	$\langle Prefix, P_t(\text{"}D\text{"})P_l, 3\rangle$ $\langle Prefix, P_C P_l, 3\rangle$ $\langle Prefix, P_C P_t(\text{"}i\text{"})P_l, 3\rangle$ $\langle Prefix, P_C P_t(\text{"}in\text{"})P_l, 3\rangle$ $\langle Prefix, P_t(\text{"}Din\text{"})P_l, 3\rangle$
"Dinos " → "Din"	$\langle Prefix, P_t(\text{"}D\text{"})P_l P_s, 3\rangle$ $\langle Prefix, P_C P_l P_s, 3\rangle$ $\langle Prefix, P_C P_t(\text{"}i\text{"})P_l P_s, 3\rangle$ $\langle Prefix, P_C P_t(\text{"}in\text{"})P_l P_s, 3\rangle$ $\langle Prefix, P_t(\text{"}Din\text{"})P_l P_s, 3\rangle$
"Dinos in" → "Din"	$\langle Prefix, P_t(\text{"}D\text{"})P_l P_s P_l, 3\rangle$ $\langle Prefix, P_C P_l P_s P_l, 3\rangle$ $\langle Prefix, P_C P_t(\text{"}i\text{"})P_l P_s P_l, 3\rangle$ $\langle Prefix, P_C P_t(\text{"}in\text{"})P_l P_s P_l, 3\rangle$ $\langle Prefix, P_t(\text{"}Din\text{"})P_l P_s P_l, 3\rangle$
"Dinos in " → "Din"	$\langle Prefix, P_t(\text{"}D\text{"})P_l P_s P_l P_s, 3\rangle$ $\langle Prefix, P_C P_l P_s P_l P_s, 3\rangle$ $\langle Prefix, P_C P_t(\text{"}i\text{"})P_l P_s P_l P_s, 3\rangle$ $\langle Prefix, P_C P_t(\text{"}in\text{"})P_l P_s P_l P_s, 3\rangle$ $\langle Prefix, P_t(\text{"}Din\text{"})P_l P_s P_l P_s, 3\rangle$
"Dinos in Kas" → "Din"	$\langle Prefix, P_t(\text{"}D\text{"})P_l P_s P_l P_s P_C P_l, 3\rangle$ $\langle Prefix, P_C P_l P_s P_l P_s P_C P_l, 3\rangle$ $\langle Prefix, P_C P_t(\text{"}i\text{"})P_l P_s P_l P_s P_C P_l, 3\rangle$ $\langle Prefix, P_C P_t(\text{"}in\text{"})P_l P_s P_l P_s P_C P_l, 3\rangle$ $\langle Prefix, P_t(\text{"}Din\text{"})P_l P_s P_l P_s P_C P_l, 3\rangle$

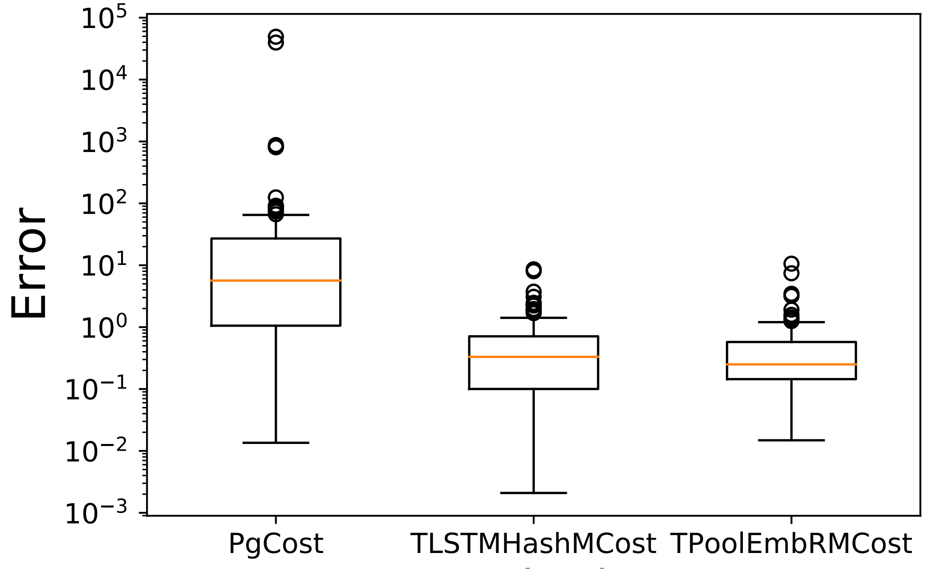


Like ‘Din%’

Learned Cost Estimator

Cost	median	90th	95th	99th	max	mean
PGCost	4.90	80.8	104	3577	4920	105
TLSTMHashMCost	4.47	53.6	149	239	478	24.1
TLSTMEmbNRMCost	4.12	18.1	44.1	105	166	10.3
TLSTMEmbRMCost	4.28	13.3	22.5	104	126	8.6
TPoolEmbRMCost	**4.07**	**11.6**	**17.5**	**63.1**	**67.3**	**7.06**

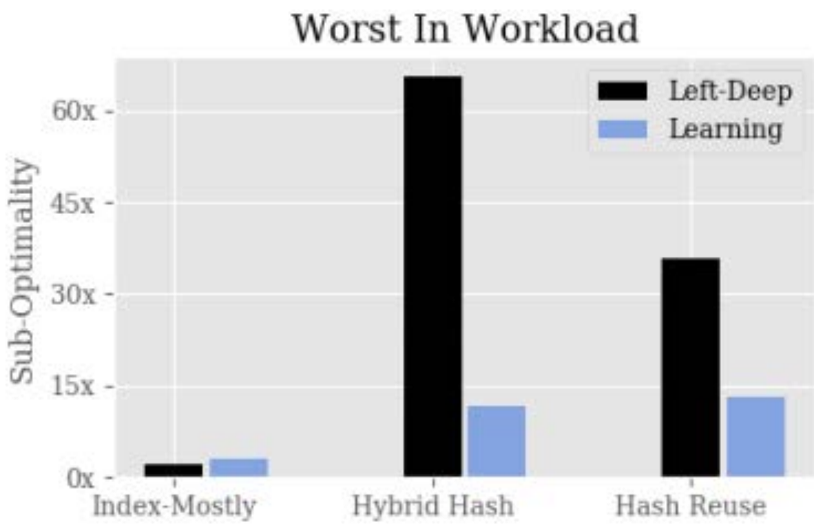




Learned Optimizer

- It is expensive to get the optimal plan
- Estimation is not accurate
  - Cost-based method
  - Rule-based method

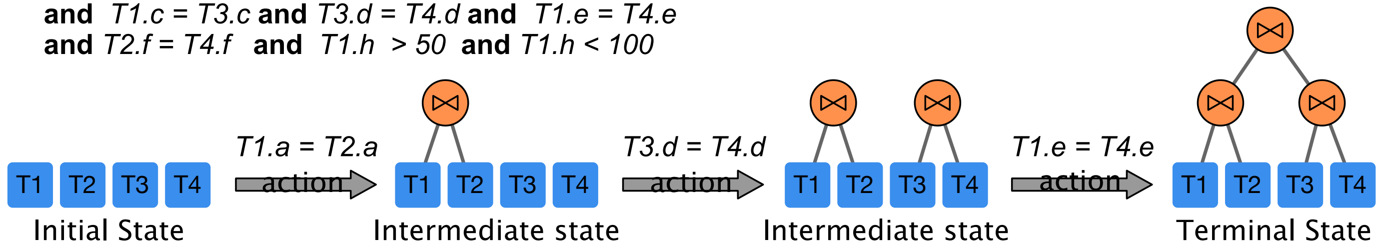
#Relations (n)	#Processing Trees	#Solutions (#Trees $\cdot$ $n!$)
1	1	1
2	1	2
3	2	12
4	5	120
5	14	1,680
6	42	30,240
7	132	665,280
8	429	17,297,280
9	1,430	518,918,400
10	4,862	17,643,225,600
11	16,796	670,442,572,800
12	58,786	28,158,588,057,600
⋮	⋮	⋮

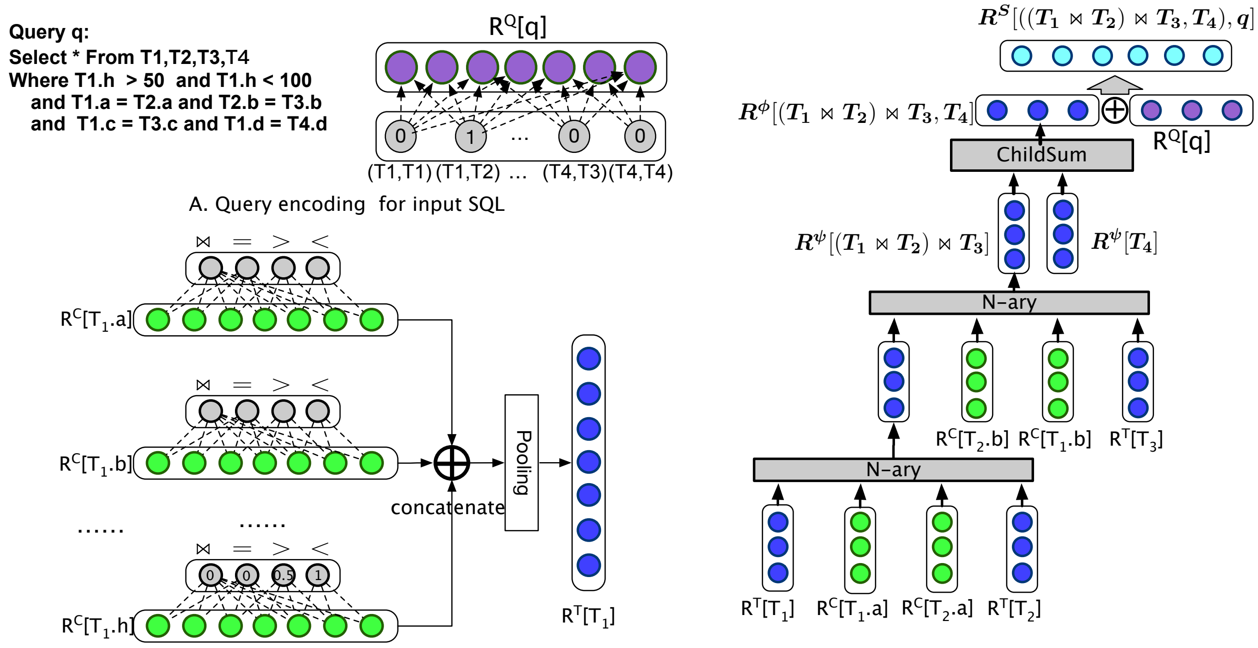


Learned Optimizer

- **Query feature encoding**
- **Reward of a join**

Q1:Select * From *T1,T2,T3,T4* **Where** *T1.a = T2.a* **and** *T2.b = T3.b*
and *T1.c = T3.c* **and** *T3.d = T4.d* **and** *T1.e = T4.e*
and *T2.f = T4.f* **and** *T1.h > 50* **and** *T1.h < 100*



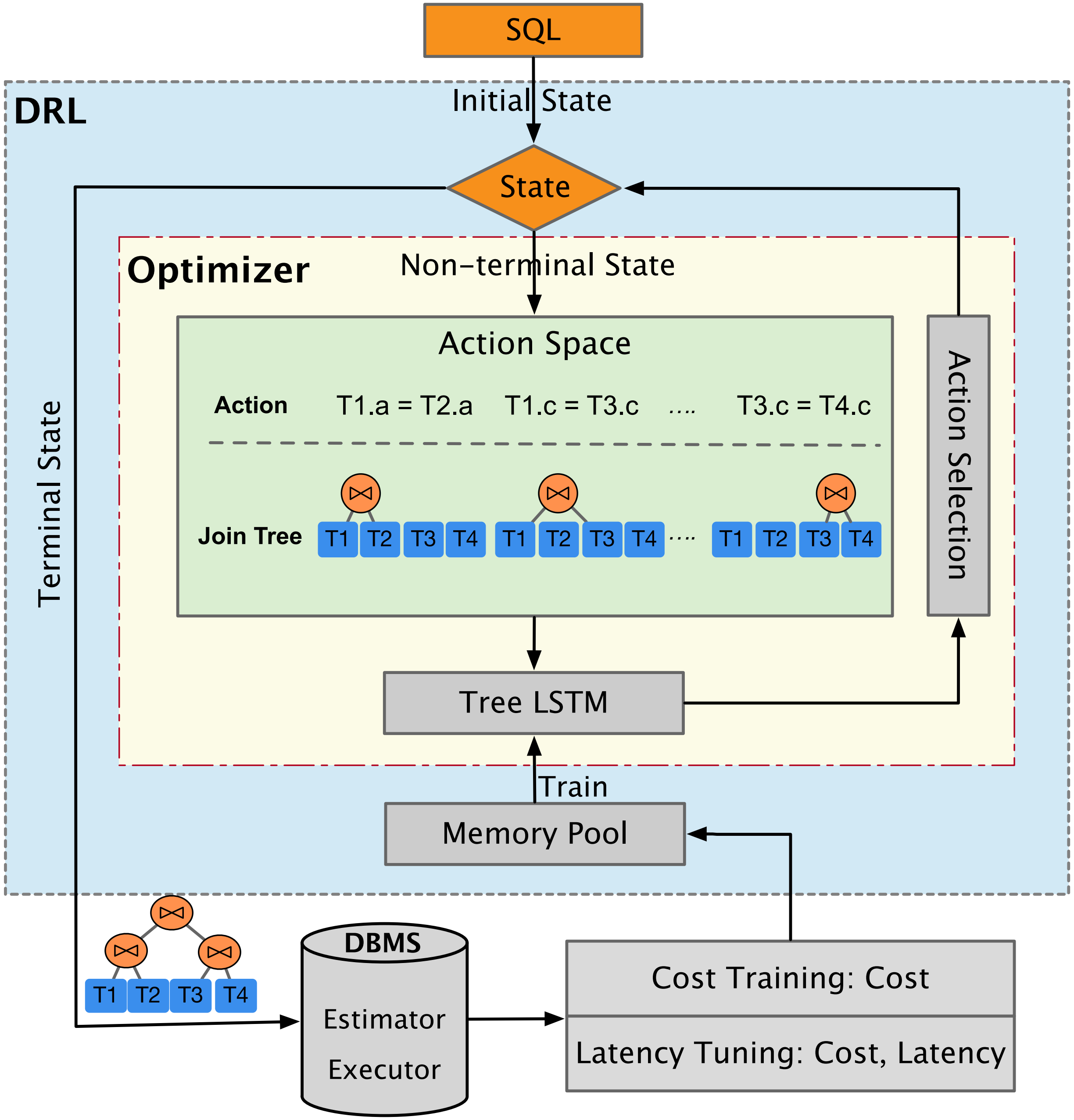


A. Query encoding for input SQL

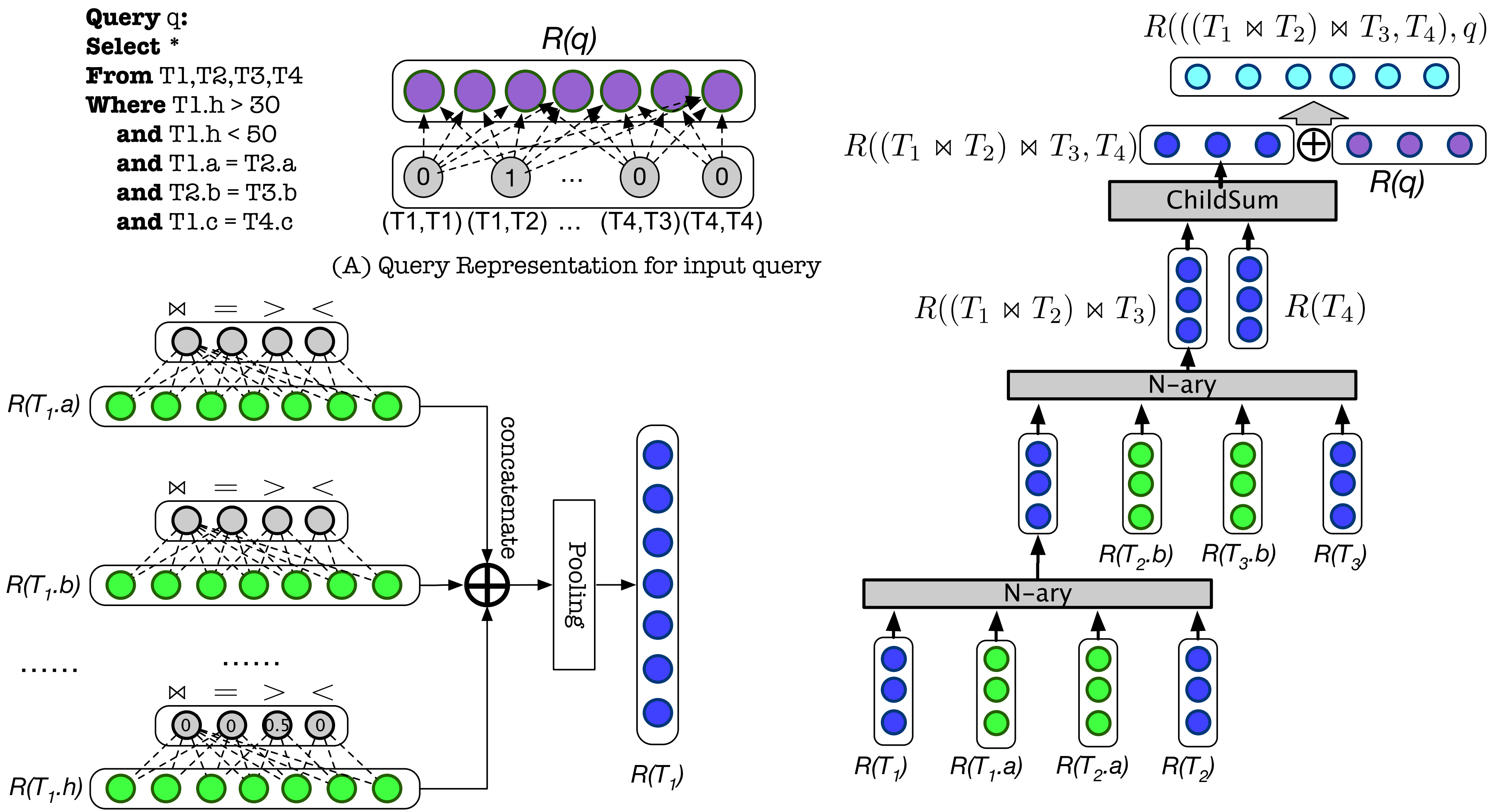
B. Table and column representation

C. Join tree and join state representation

Learned Optimizer



Learned Optimizer



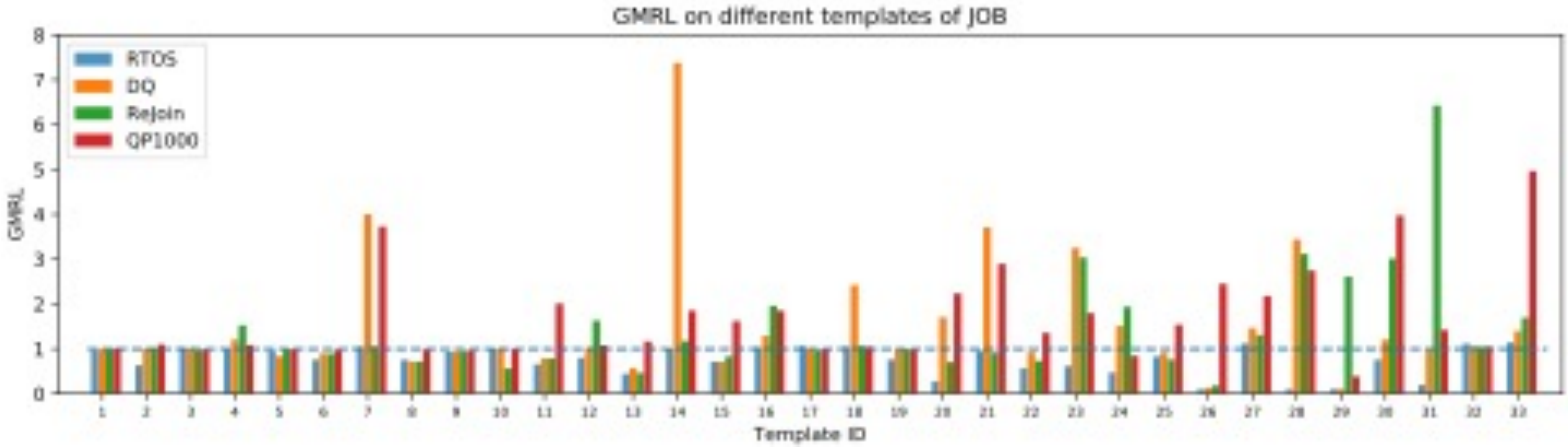
Learned Optimizer

TABLE II
MEAN RELEVANT COST TO DYNAMIC PROGRAMMING

MRC benchmark / algorithm	JOB	TPC-H
RTOS	**1.01**	**1.00**
ReJoin	1.75	**1.00**
QP100	7.81	1.06
QP1000	1.62	**1.00**
DQ	2.34 (1.31)	1.01

TABLE III
EXPONENTIAL MEAN LOG RELEVANT LATENCY (GMRL) TO D

GMRL benchmark / algorithm	JOB	TPC-H
RTOS	**0.67**	**0.92**
ReJoin	1.14	0.96
QP100	NA	1.03
QP1000	1.90	1.00
DQ	1.23	0.99

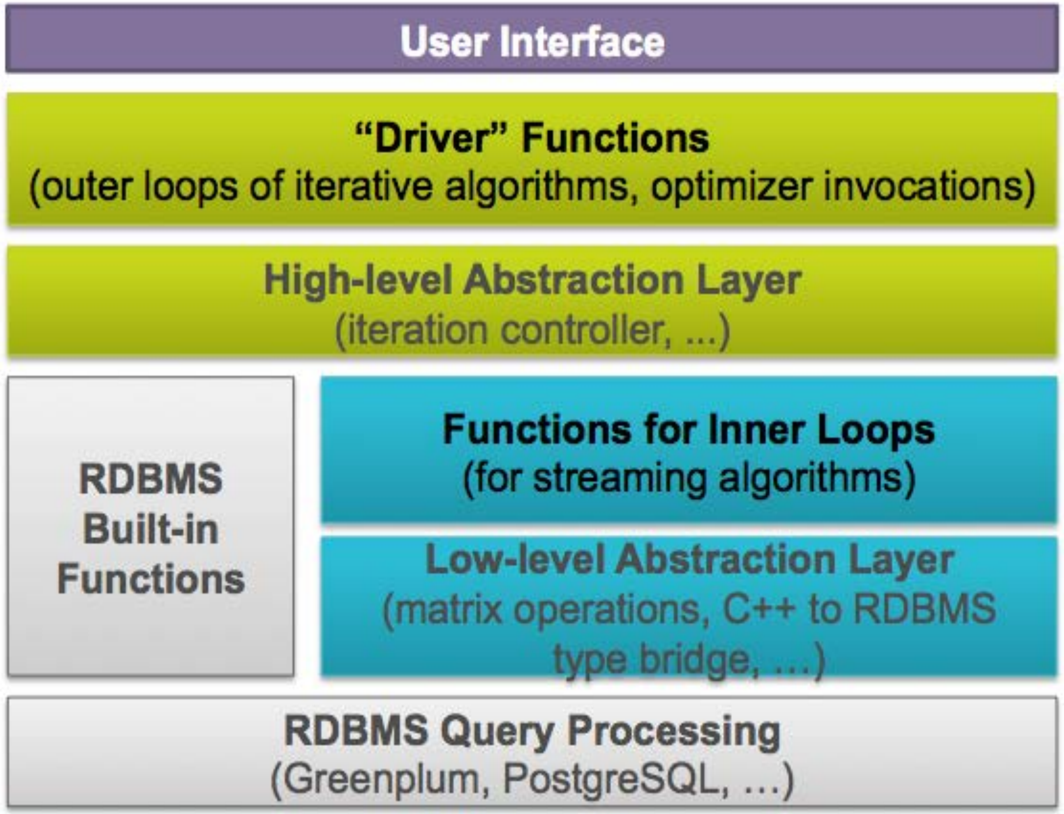


Level 3: AI-Enhanced DB

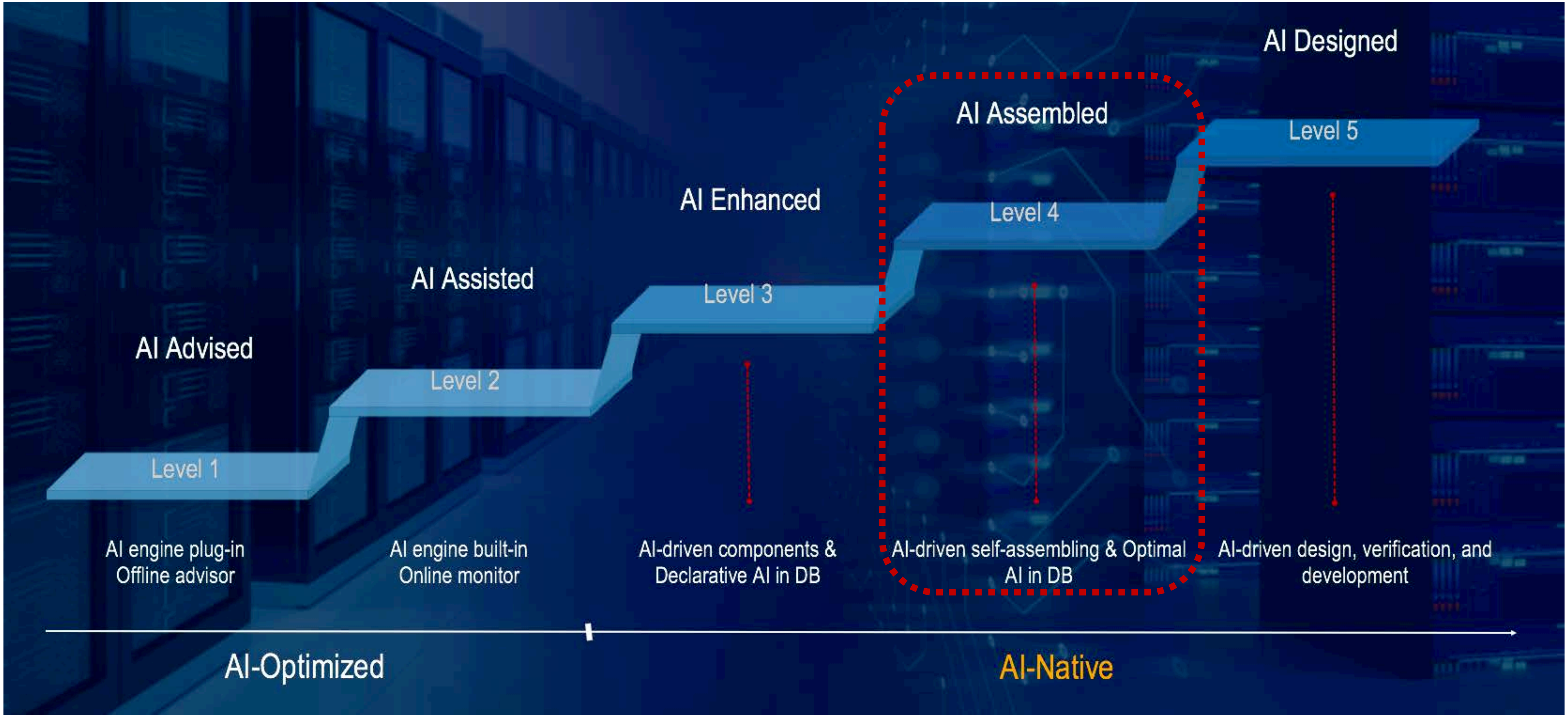
- AI4DB: Learned DB components
  - Learned Index
  - Learned Cost estimator
  - Learned Optimizer
  - Learned Statistics
- **DB4AI: Declarative AI**
  - **Use SQL for using AI algorithms**
  - **Lower the burden of using AI**

DB4AI

- **Declarative AI**
  - **AI to SQL**
  - **SQL completeness**
  - **SQL advisor**
- **AI optimizations**
  - **Cost estimation**
  - **Auto parameter**
  - **Auto model**
  - **Parallel computing**
- **Data Governance**
  - **Data discovery**
  - **Data cleaning and fusion**



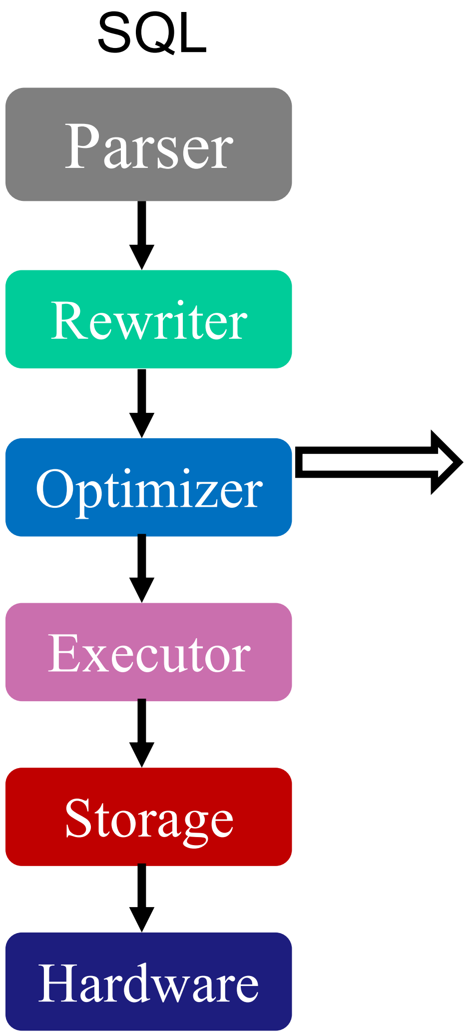
AI-Native Database

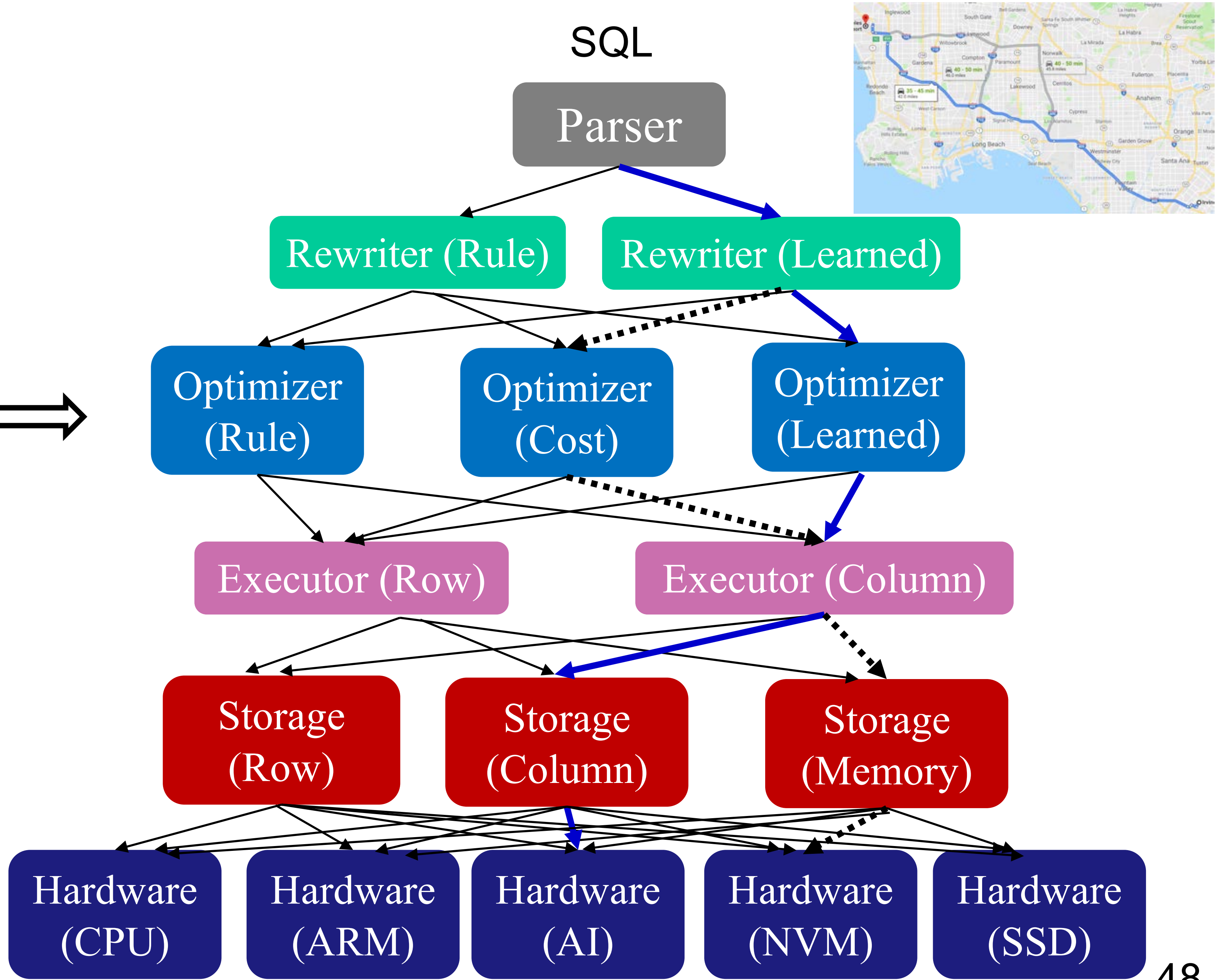


Level 4: AI-Assembled DB

- Self-Assembling
  - Each component has multiple options
    - Optimizer
      - CBO, RBO, Learned
  - Assemble the components as a database
    - Reinforcement learning (RL)
  - From single path to multiple paths
    - Like map navigator
  - Scheduling on diversified hardware
    - Learned tensor model on AI hardware
    - Traditional (cost) model on general hardware

Level 4: AI-Assembled DB

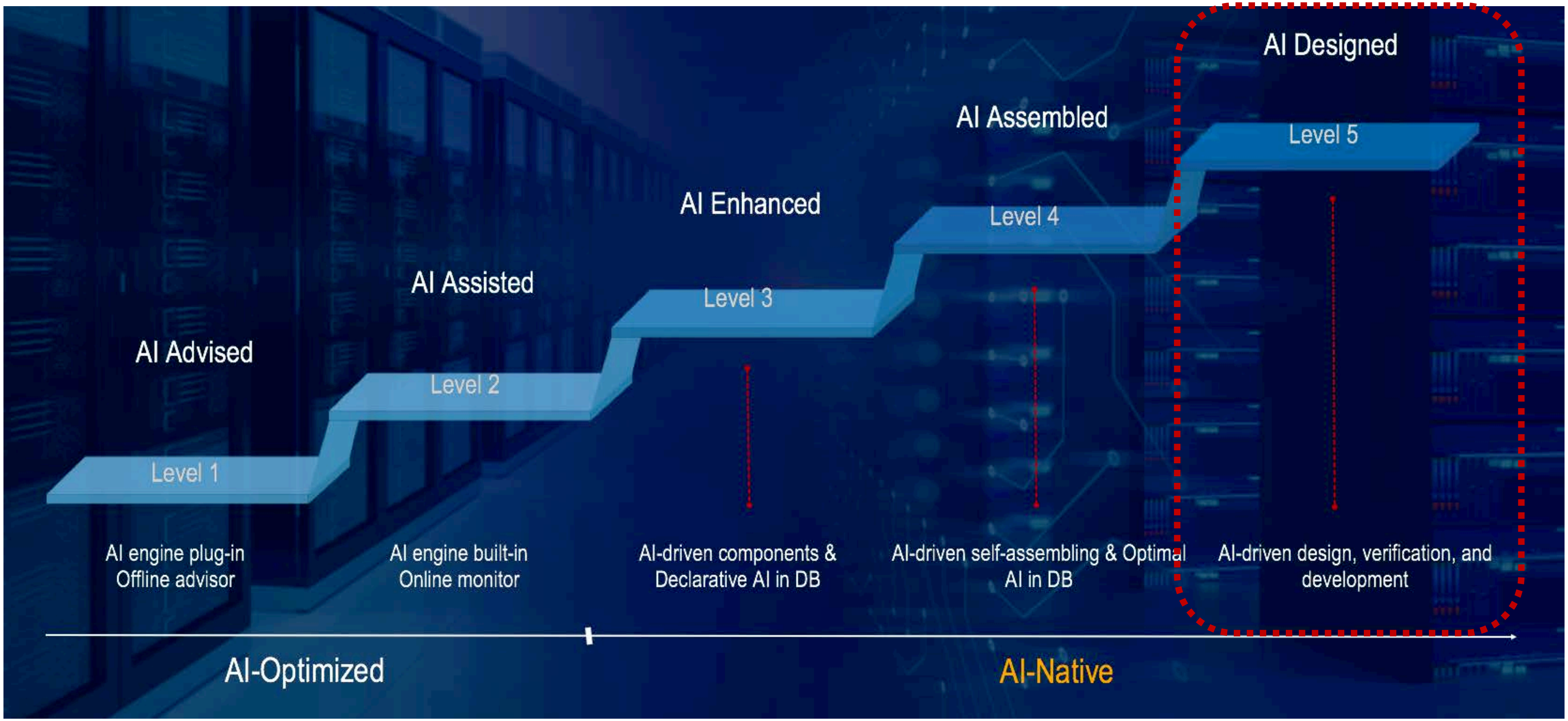




Level 4: AI-Assembled DB

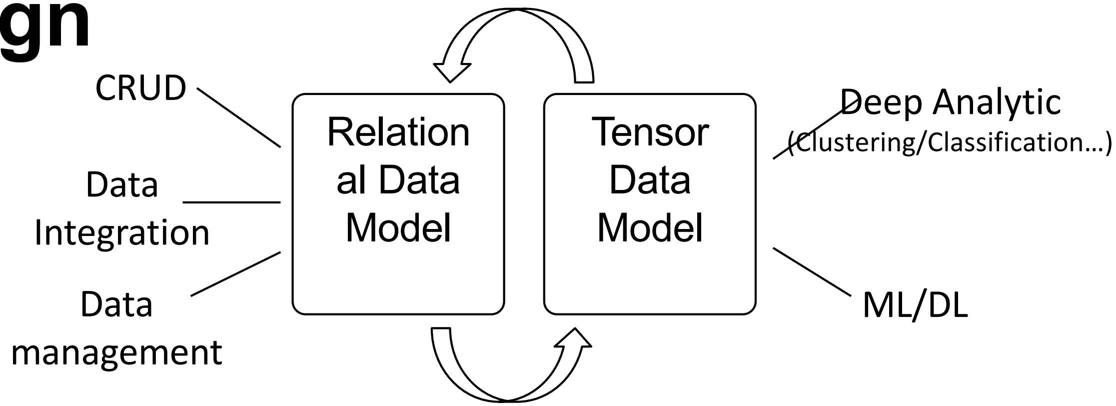
- **Self-Assembling**
  - Each component has multiple options
    - Optimizer
      - CBO, RBO, Learned
  - Assemble the components as a database
    - Reinforcement learning (RL)
  - From single path to multiple paths
    - Like map navigator
  - Scheduling on diversified hardware
    - Learned tensor model on AI hardware
    - Traditional (cost) model on general hardware

AI-Native Database

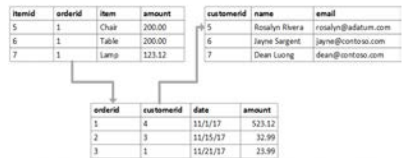


Level 5: AI-Designed DB

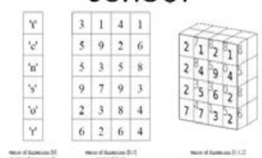
- **AI-based design**
  - Data structure design
  - Transaction design
  - Storage design
  - Index design
  - Optimizer design
- **AI & DB Co-design**
  - Unified model
  - Unified optimization



relational

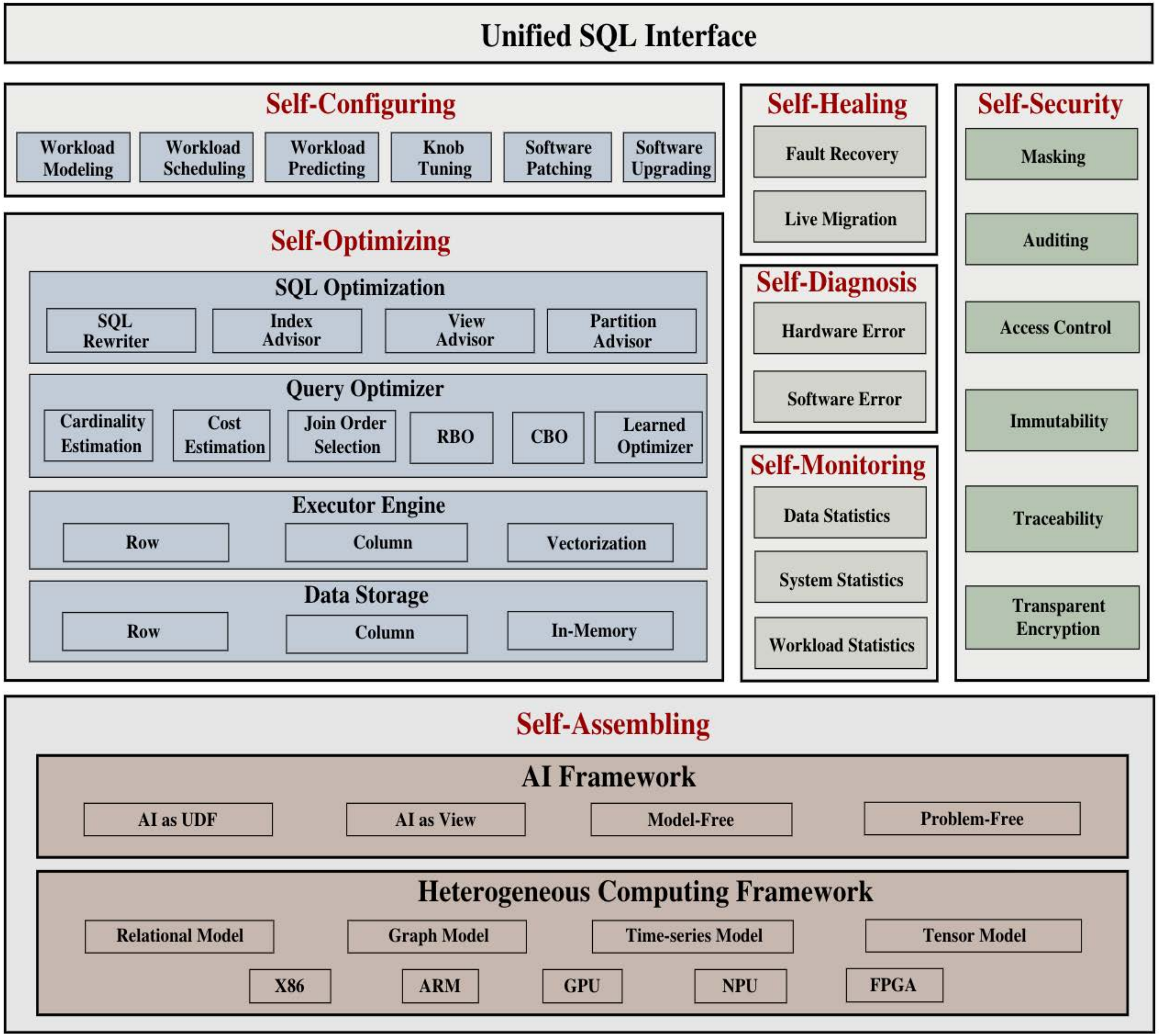


tensor



TPU/NPU

AI-Native Database



Five Levels of DB4AI

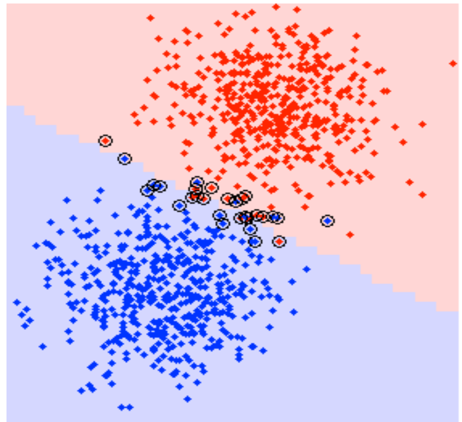
1	AI Advised: • Offline AI-based knob tuning/statistics recommendation, offline data placement; • Offline workload management, offline optimization;
2	AI Assisted: • Self monitoring/tuning: online knob tuning, monitoring; • Self optimization: query tuning, online index/view advisor; • Self diagnoses, healing, protection;
3	AI Enhanced: • Using AI-based algorithms to enhance the core components of database; • Learned index, learned optimizer, learned storage, query engine customization, etc.; • AI in database, declarative AI, DB-optimized AI;
4	AI Assembled: • Functions decoupled as services. Functions deployed on heterogeneous environments; • AI-based algorithms to choose the best execution paths of different services;
5	AI Designed: • DB designed by AI, hardware and software codesign, automatic evaluation. • AI-assisted semi-formal or formal verification for trustworthy and security;

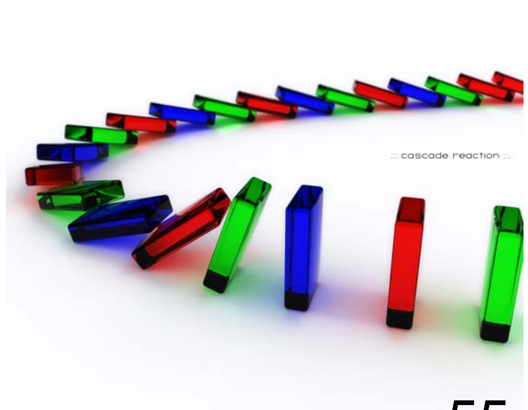
Five Levels of DB4AI

Level	Consumability	Description	AI Skill Level
1	AI Models as UDFs	Algorithms available in the underlying DB system as UDFs or Stored Procedures	High
2	AI Models as Views	Materialized the trained models as ‘views’ which can be utilized by other users. The views will be automatically updated which is triggered by data update or model update	Medium
3	Model-free	No need to specify models. Given a problem, automatically identify the models.	Low
4	Problem-free	No need to specify problems. Automatically identify the problems and models.	Low
5	Full-automatic	Automatically discover AI opportunities, including model selection, algorithm selection and data discovery	Very Low

Lessons Learned

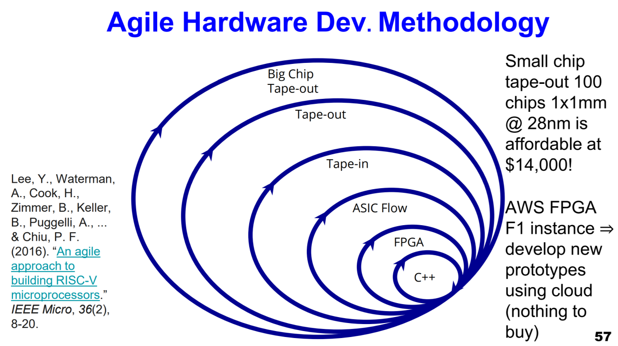
- AI4DB
  - Database can learn from both internal and external “environments” to achieve high performance
  - AI enhances database, especially
    - Fast, flexible and strong adaptability
    - Make DB more intelligent
- DB4AI
  - In-DB AI consumability
  - Make AI easily used in different fields

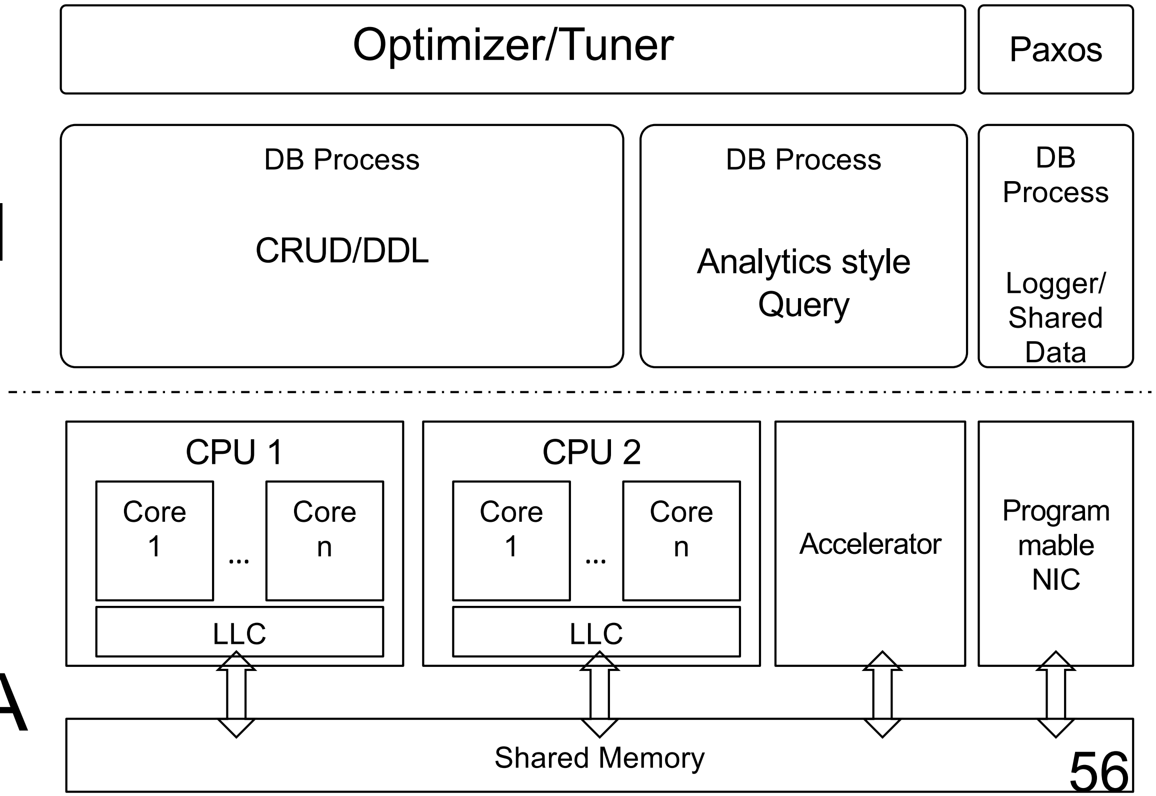




Challenges and Opportunities

- Hardware/Software Co-Design
  - Database chips
  - Tensor model
  - DB evaluation tool
    - Like EDA
- OLAP 2.0
  - Multi-model, DB&AI
- OLTP 2.0
  - New hardware
  - NVM, RDMA, etc.
  - Programable RDMA





Thanks!

Q&A