EDBT ICDT EDINBURGH 2022

Tutorial

Learned Query Optimizer: At the Forefront of AI-Driven Databases

Rong Zhu[1], Ziniu Wu[2], Chengliang Chai[3], Andreas Pfadler[1],

Boling Ding[1], Guoliang Li[3], Jingren Zhou[1]

1 Alibaba Group 阿里巴巴集团

2 Massachusetts Institute of Technology

3 Tsinghua University

Speakers

Rong Zhu
Alibaba Group

Ziniu Wu
MIT

Chengliang Chai
Tsinghua University

Andreas Pfadler
Alibaba Group

Bolin Ding
Alibaba Group

Guoliang Li
Tsinghua University

Jingren Zhou
Alibaba Group

Outline

- ❑ Part 1: Preliminaries
- ❑ Part 2: Learned Individual Component
- ❑ Part 3: Learned Whole QO Module

Tutorial Part A

- ❑ Part 4: Applications and Deployment
- ❑ Part 5: Summary and Future Work
- ❑ Q & A

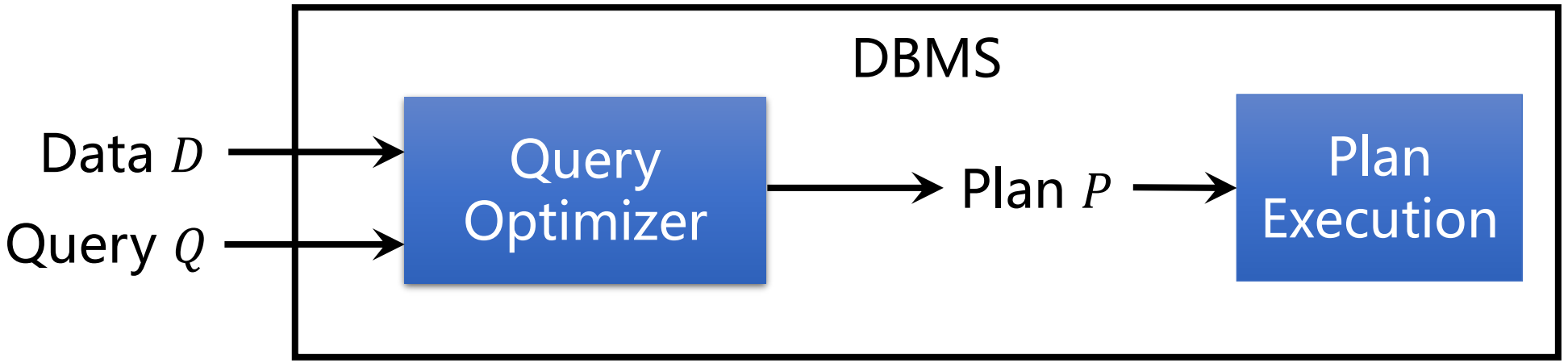
Tutorial Part B

Part 1: Preliminaries

- ❑ Query Optimizer (QO)
- ❑ Learned QO

QO: The Central Role in DBMS

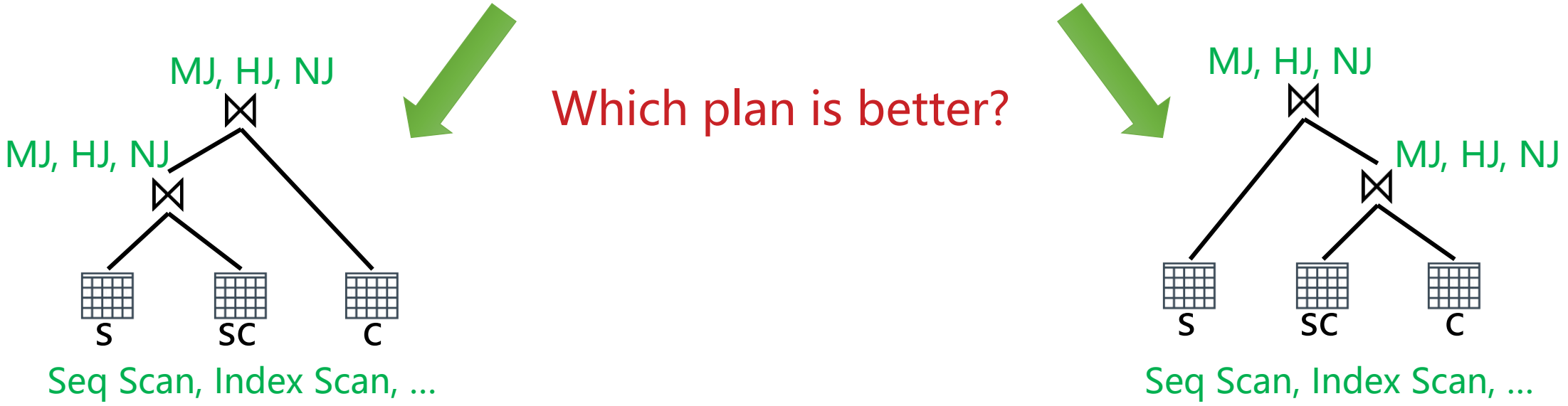
- Generating best physical plan for input query
- Directly relates to the system performance



QO: A Difficult Task

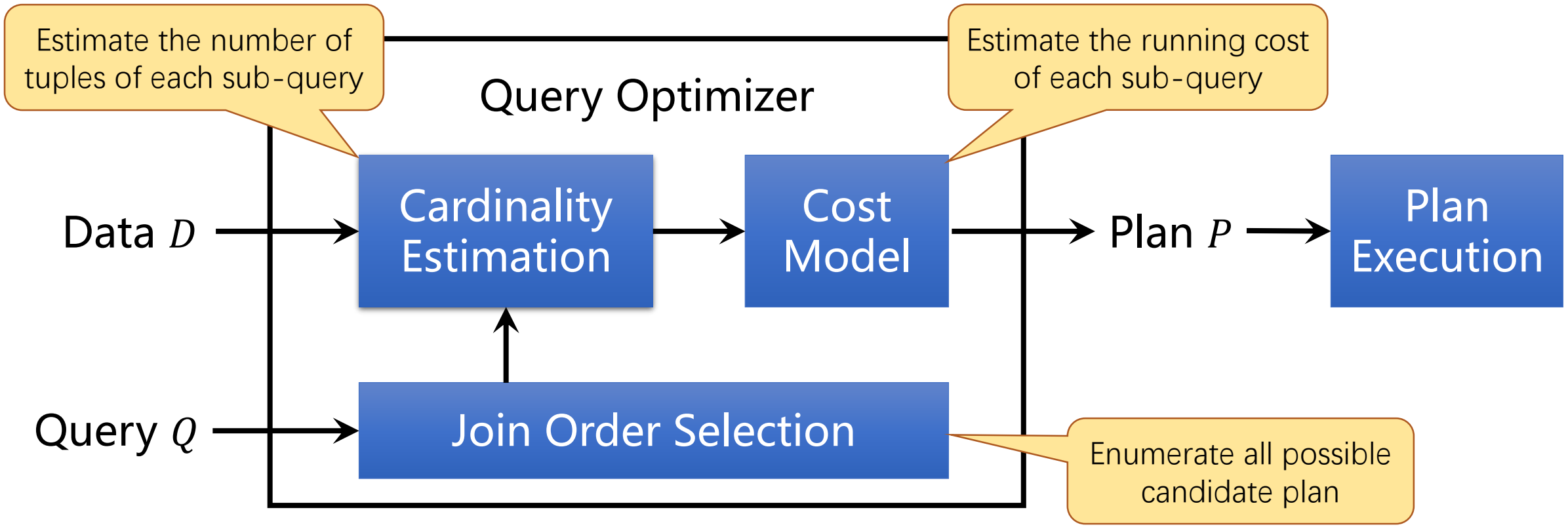
- Different join orders, join methods, scan methods, …

```
SELECT COUNT(*)
FROM s, sc, c
WHERE s.sid = sc.sid AND sc.cid = c.cid
AND s.year < 2009 AND c.credit >= 3
```



QO Architecture

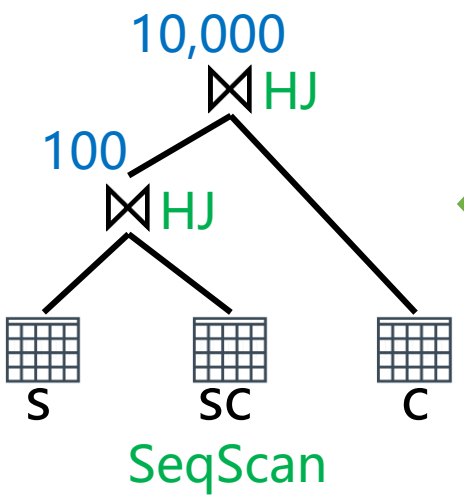
- The general volcano framework



QO Architecture

- The general volcano framework

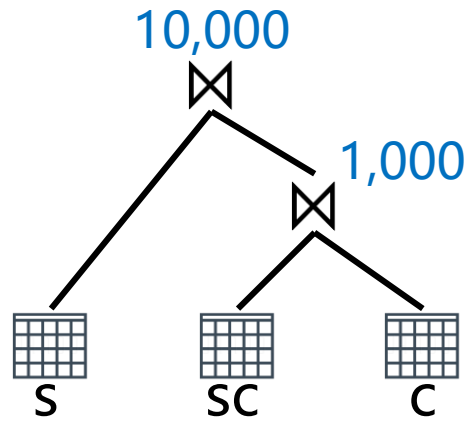
```
SELECT COUNT(*)
FROM s, sc, c
WHERE s.sid = sc.sid AND sc.cid = c.cid
AND s.year < 2009 AND c.credit >= 3
```



SeqScan

Join Cost: HJ < MJ < NJ
Scan Cost: Seq Scan < Index Scan

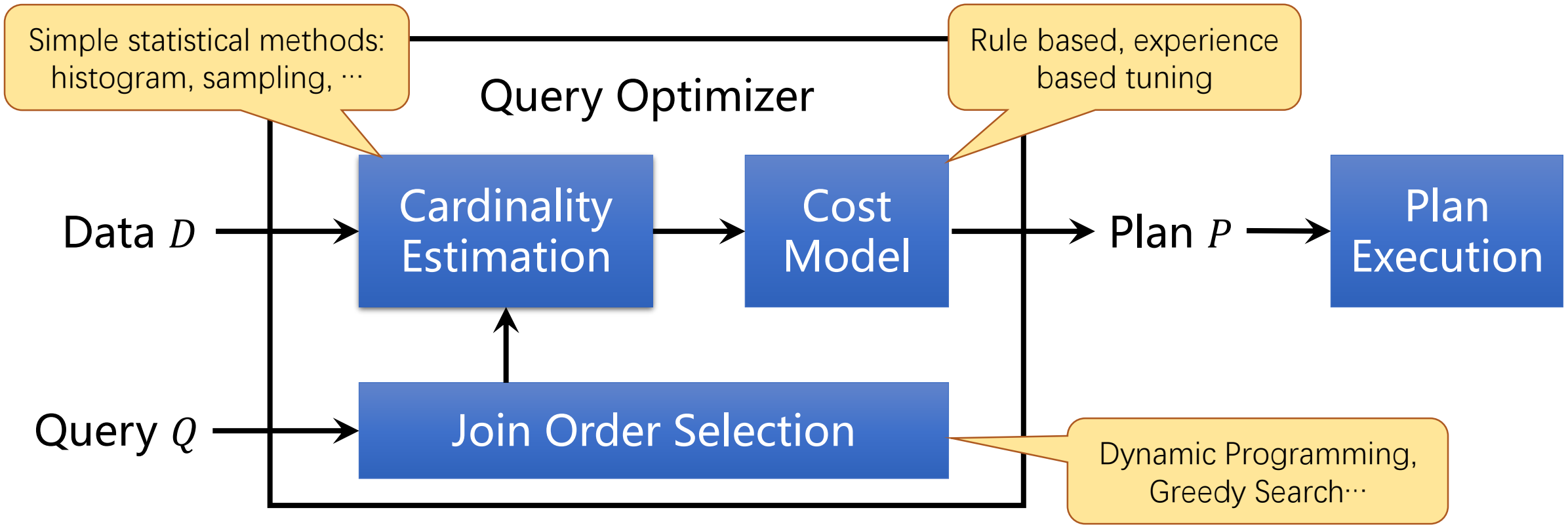
This plan is better!



Seq Scan, Index Scan, ...

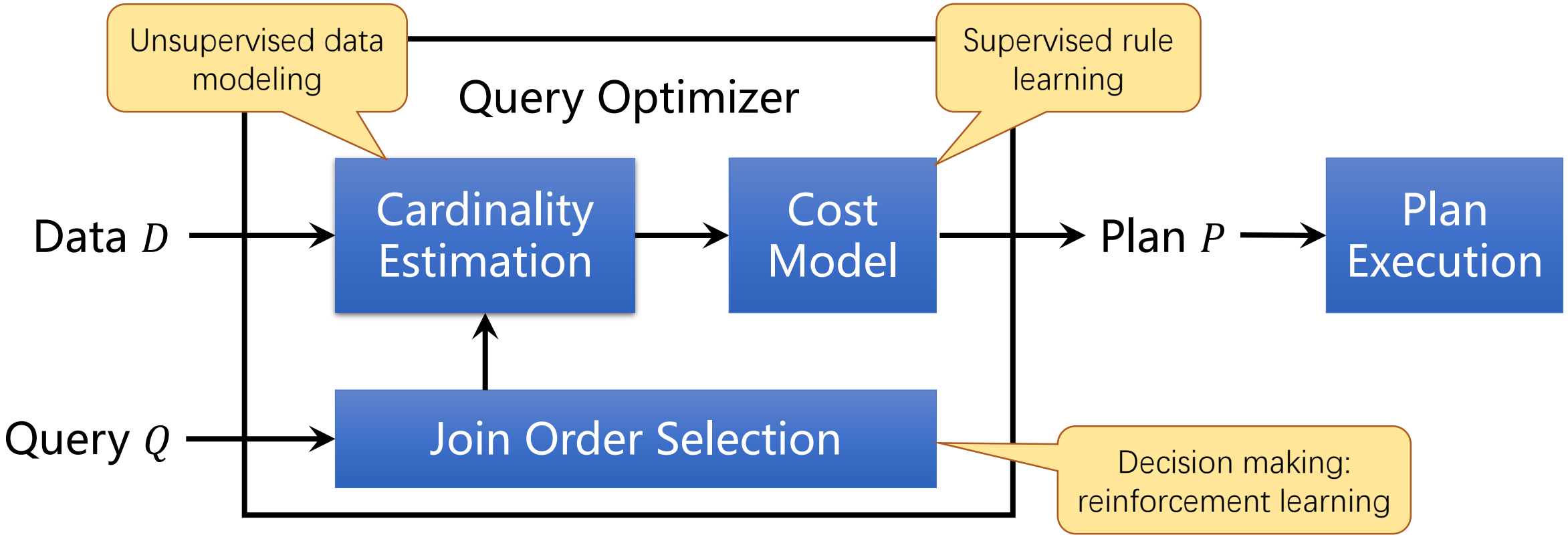
Traditional QO Methods

- Experience driven: not favorable enough



Learned QO: A Pioneer in AI4DB

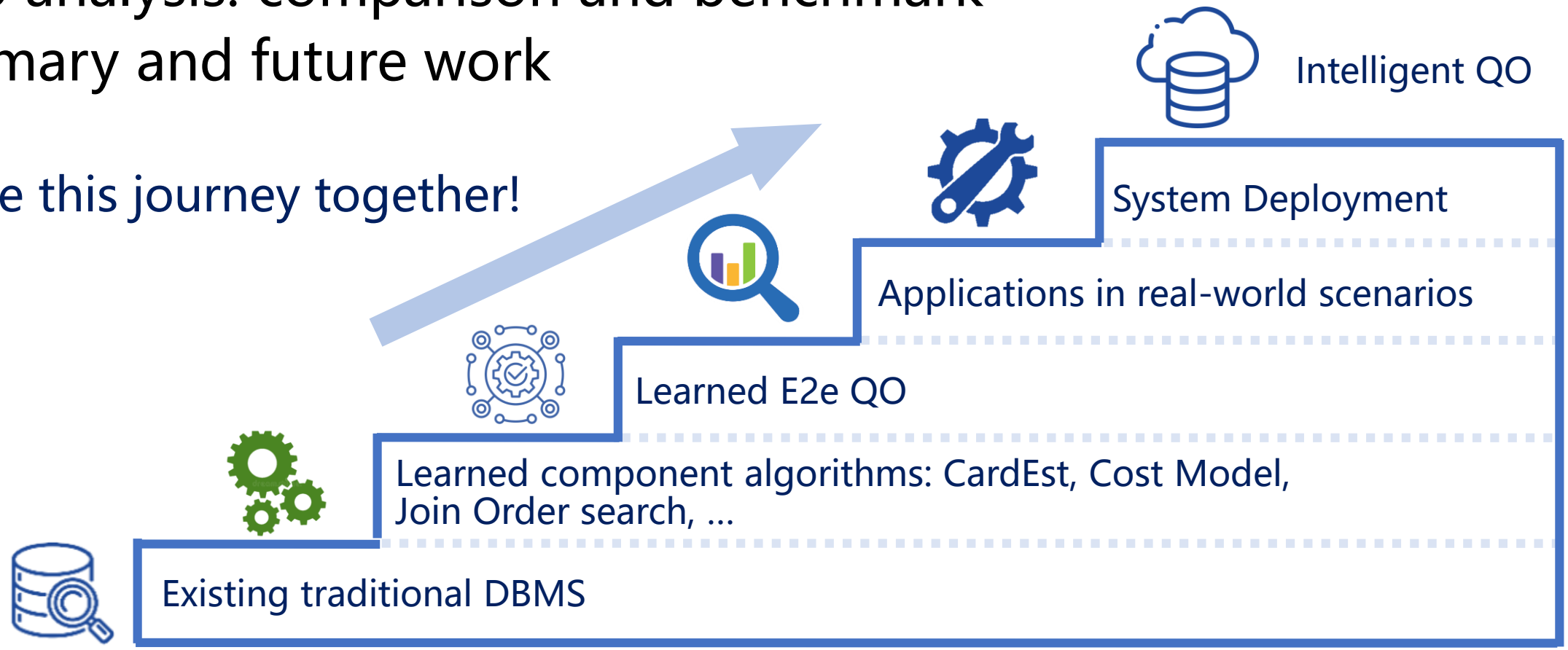
- QO is an experimental plot for learned techniques: more automatic, fine-grained and accurate solutions



Learned QO: Status and Opportunities

- Review on recent advances: 100+ papers recently
- Deep analysis: comparison and benchmark
- Summary and future work

Explore this journey together!



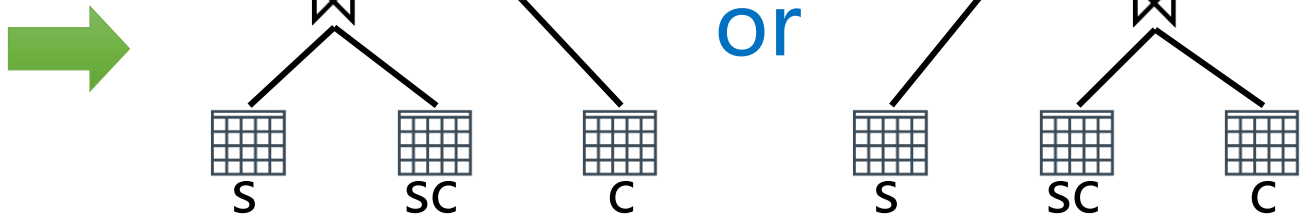
Part 2: Learned QO Components

- ❑ Cardinality Estimation (CardEst)
- ❑ Cost Model
- ❑ Join Order Search

Cardinality estimation

- CardEst: estimate the result size $C(Q)$ of the query Q without actual execution

```
SELECT COUNT(*)
FROM s, sc, c
WHERE s.sid = sc.sid AND sc.cid = c.cid
AND s.year < 2009 AND c.credit >= 3
```



- Lay foundations for cost estimation and join ordering selection
- A key component in QO and decides the query plan quality

CardEst methods overview

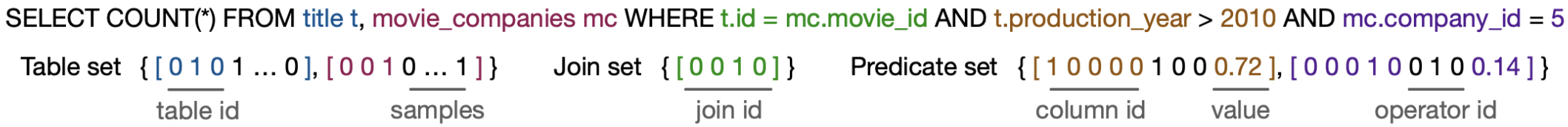
- Traditional methods: Histogram and Sampling
- Learned query-driven methods:
  - Analyze query workload, learn regression model mapping Q to C(Q)
- Learned data-driven methods:
  - Analyze data, learn $Pr_T(A)$ of table T with attributes A
  - $C(Q) = Pr_T(Q) * |T|$
- Bound-based methods:
  - Instead of estimating the cardinality, it provides an upper bound
  - Can avoid very expensive join orders and physical operators
  - Rigorously not CardEst (will not explain in detail)

Traditional methods

- Most widely used in modern DBMS
- Histogram: attribute independent assumption $\Pr(A) \approx \prod_i \Pr(A_i)$
  - Fast inference, low storage cost, high estimation error
  - Multi-dimensional Histogram
- Sampling:
  - Execute the query on a smaller sample of the data
  - Estimation accuracy and inference speed trade-off
  - Kernel-density estimation, join sampling
- Join uniformity assumption
  - Estimate join query $Q=A \bowtie B$ as $C(Q) = Pr_A(Q_A) * Pr_B(Q_B) * |A \bowtie B|$

Learned query-driven methods

- Build supervised model mapping Q to C(Q)
  - Query feature engineering is important



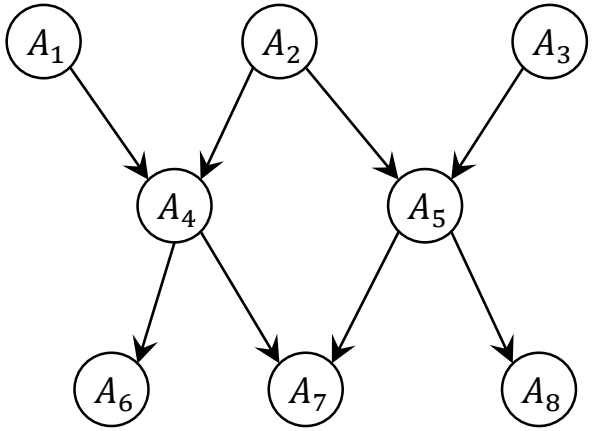
  - XGboost, Neural network, Tree-LSTM, deep ensemble
- Fast inference speed, versatile
- Estimation accuracy is highly dependent on the training query workload.
  - Requires excessive amount of training queries (unavailable to new DB)
  - Performance degrades severely for data and workload changes

Learned data-driven methods

- Build statistical models to capture data distributes $Pr_T(A)$
  - Deep auto-regression model (Neurocard)
  - Probabilistic graphical models: Bayesian network (BayesCard), Sum-product network (DeepDB), Factorized-sum-product network (FLAT)
  - Normalizing flow model (FACE)
- Use fanout-based method to handle join queries.
  - Produce accurate estimates
  - Inference time and model size are generally small but maybe large for databases with large number of tables
- Current state-of-the-art methods in improving query plans.

Distribution modeling techniques

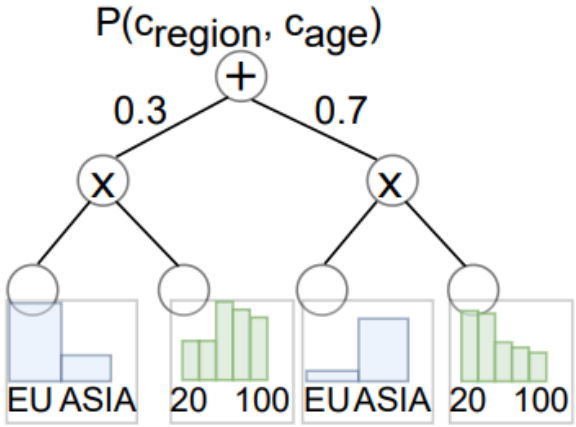
- Deep auto-regression model (Neurocard[1])
  - Fully factorize the distribution: for table A with attr $\{A_1, ..., A_n\}$
    $Pr_T(A) = Pr_T(A_1) * Pr_T(A_2|A_1) * Pr_T(A_3|A_2, A_1) * ... * Pr_T(A_n|A_{n-1}, ..., A_1)$
  - High accuracy, large model size, and slow inference
- Bayesian Network (BayesCard[2]): $\Pr(A) = \prod_i \Pr(A_i|A_{pa(i)})$
  - Conditional independence assumption: high accuracy
  - Explainable and compact model
  - Difficult in structure learning (NP-Hard)
  - BayesCard addresses the low inference of BN:
    JIT-compiled variable elimination,
    Progressive sampling.
  - High accuracy, small model size, and fast inference

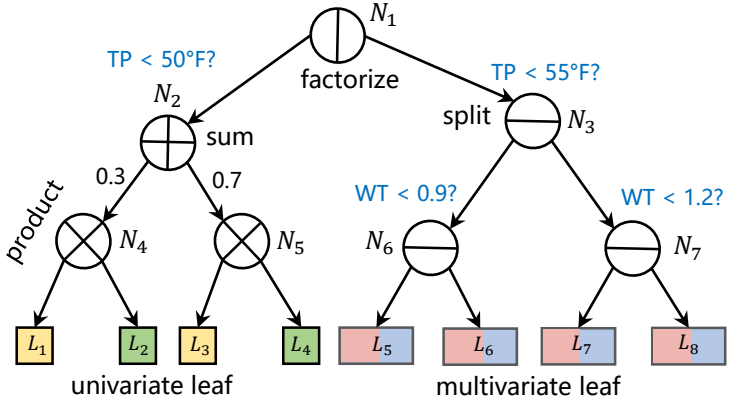


[1] Z. Yang, A. Kamsetty, S. Luan, E. Liang, Y. Duan, X. Chen, and I. Stoica. 2021. NeuroCard: One Cardinality Estimator for All Tables. PVLDB 14, 1 (2021), 61–73
[2] Z. Wu, A. Shaikhha, R Zhu, K Zeng, Y Han and J Zhou. BayesCard: A Unified Bayesian Framework for Cardinality Estimation. arXiv:2012.14743 (2021).

Distribution modeling techniques

- Sum-product network (DeepDB[3]):
  - Local independence assumptions
  - Split the data by rows to find local independence between attributes
  - Accuracy, inference speed, and model size are sensitive with attribute correlations.
- Factorized sum-product network (FLAT[4]):
  - Combining the techniques from BN and SPN
  - Adaptively process highly and weakly correlated attributes.
  - High accuracy and small model size. Inference can be slow for large # attrs.

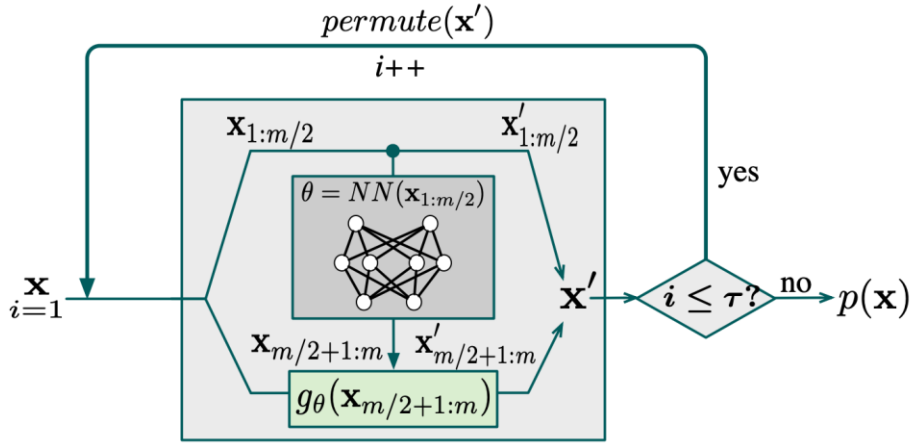




[3] B. Hilprecht, A. Schmidt, M. Kulessa, A. Molina, K. Kersting, and C. Binnig. 2019. DeepDB: learn from data, not from queries!. In PVLDB.
[4] R. Zhu, Z. Wu, Y. Han, K. Zeng, A. Pfadler, Z. Qian, J. Zhou, and B. Cui. FLAT: Fast, Lightweight and Accurate Method for Cardinality Estimation. VLDB 14, 9 (2021), 1489–1502.

Distribution modeling techniques

- Normalizing flow model(FACE[5]):
  - Continuous joint distribution
  - Dequantize discrete attributes to ensure continuity
  - High accuracy, inference speed, and compact model even for columns with large domain size.

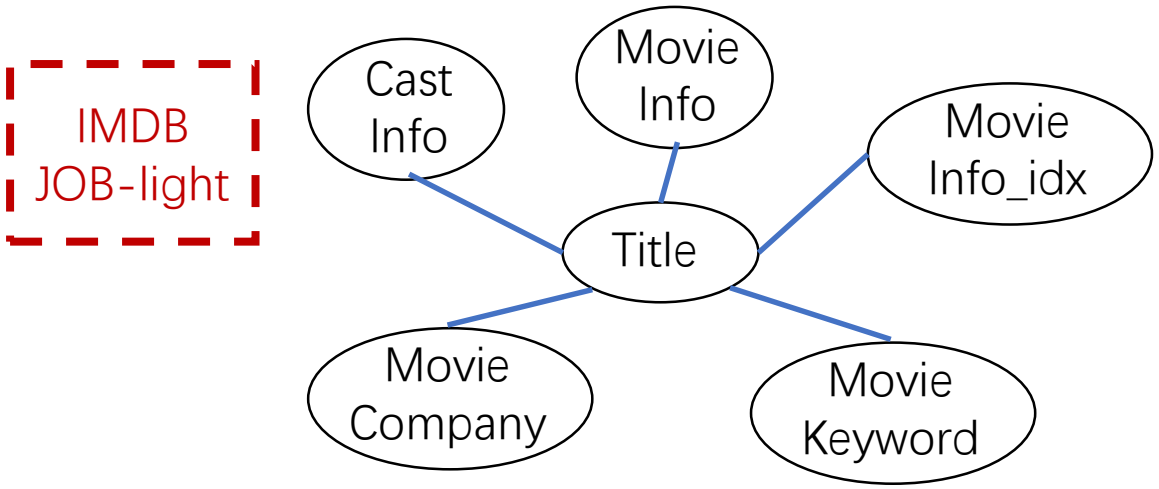


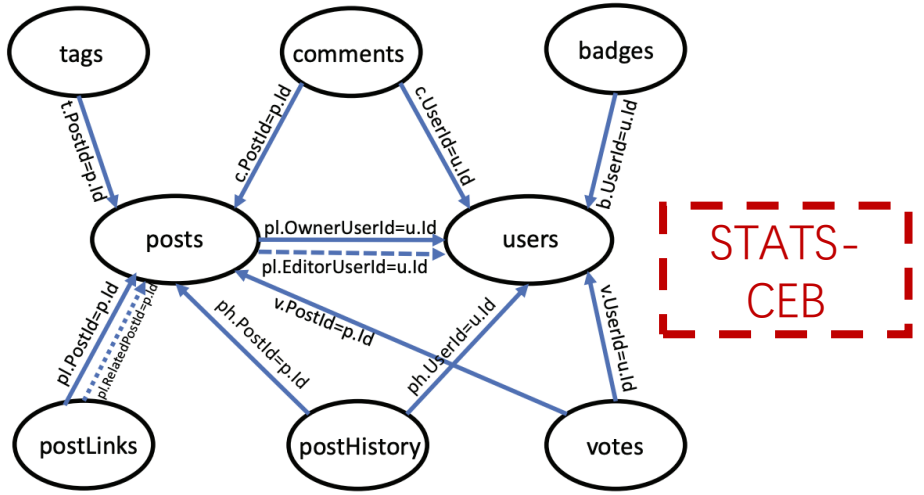
$$sel(\theta) = \int_{\mathbf{x} \in R_1 \times \cdots \times R_m} p(\mathbf{x})\, d\mathbf{x}.$$

[5] J. Wang, C. Chai, J. Liu, and G. Li. FACE: A normalizing ow based cardinality estimator. Proc. VLDB Endow., 15(1):72–84, 2021.

Benchmark evaluations

- IMDB JOB-light benchmark:
  - real-world data with complicated distributions
  - contains 6 tables, forming a star-shaped join





- Stats-CEB benchmark[5]:
  - real-world data with complicated distributions
  - More tables, more complicated join pattern, more attributes

[5] Y. Han, Z. Wu, P. Wu, R. Zhu, J. Yang, L. Tan, K。 Zeng, G. Cong, Y. Qin, A. Pfadler, Z. Qian, J. Zhou, J. Li, B. Cui, Cardinality Estimation in DBMS: A Comprehensive Benchmark Evaluation. VLDB 2022.

Benchmark evaluations

Category	Method	Workload					
		JOB-LIGHT			STATS-CEB		
		End-to-End Time	Exec. + Plan Time	Improvement	End-to-End Time	Exec. + Plan Time	Improvement
Baseline	PostgreSQL	3.67h	3.67h + 3s	0.0%	11.34h	11.34h + 25s	0.0%
	TrueCard	**3.15h**	**3.15h + 3s**	**14.2%**	**5.69h**	**5.69h + 25s**	**49.8%**
Traditional	MultiHist	3.92h	3.92h + 30s	−6.8%	14.55h	14.53h + 79s	−28.3%
	UniSample	4.87h	4.84h + 96s	−32.6%	$> 25h$	−−	−−
	WJSample	4.15h	4.15h + 23s	−13.1%	19.86h	19.85h + 45s	−75.0%
	PessEst	3.38h	3.38h + 11s	7.9%	6.10h	6.10h + 43s	46.2%
Query-driven	MSCN	3.50h	3.50h + 12s	4.6%	$8.13h$	8.11h + 46s	28.3%
	LW-XGB	4.31h	4.31h + 8s	−17.4%	$> 25h$	−−	−−
	LW-NN	3.63h	3.63h + 9s	1.1%	11.33h	11.33h + 34s	0.0%
	UAE-Q	3.65h	3.55h+356s	−1.9%	11.21h	11.03h+645s	1.1%
Data-driven	NeuroCard^E	3.41h	3.29h + 423s	6.8%	12.05h	11.85h + 709s	−6.2%
	BayesCard	3.18h	3.18h + 10s	13.3%	7.16h	7.15h + 35s	36.9%
	DeepDB	3.29h	3.28h + 33s	10.3%	6.51h	6.46h + 168s	42.6%
	FLAT	3.21h	3.21h + 15s	12.9%	5.92h	5.80h + 437s	47.8%
Query + Data	UAE	3.71h	3.60h + 412s	−2.7%	11.65h	11.46h + 710s	−0.02%

Learned CardEst: Summary

- Traditional method are most general, lightweight, fast to train and update, low latency, and perfect for system deployment.
- Learned data-driven methods have the state-of-the-art performance, but less general and can sometimes have large model and slow inference.
- Learned query-driven methods can perform well for static DB instances, but not suitable for new DBs or DBs that have frequent data update/workload shift.
- Bound-based methods provide us deeper understanding of the CardEst problem but may not be practical.

Part 2: Learned QO Components

- ❑ Cardinality Estimation (CardEst)
- ❑ Cost Model
- ❑ Join Order Search

The Relations of Card/Cost Estimation

- **Task Target**
  - Cost estimation is to approximate the execution-time/resource-consumption;
- **Correlations**
  - Cost estimation is based on cardinality
- **Estimation Difficulity**
  - Cost is harder to estimate than cardinality, which considers multiple factors (e.g., seq scan cost, cpu usage)

Learned Cost Estimation

Method Classification

Single Query Cost Estimation

- **Characteristic** : end2end, tree-structure plan encoding.
- **Key idea** : use previous plans to train a tree-structure neural network, which directly predicts the cost.

Concurrent Query Cost Estimation

- **Characteristic**: multile queries are considered, and performance of one plan varies due to the correlation between plans.
- **Key idea** : use a graph to represent the correlation between plans and use a network to predict the cost.

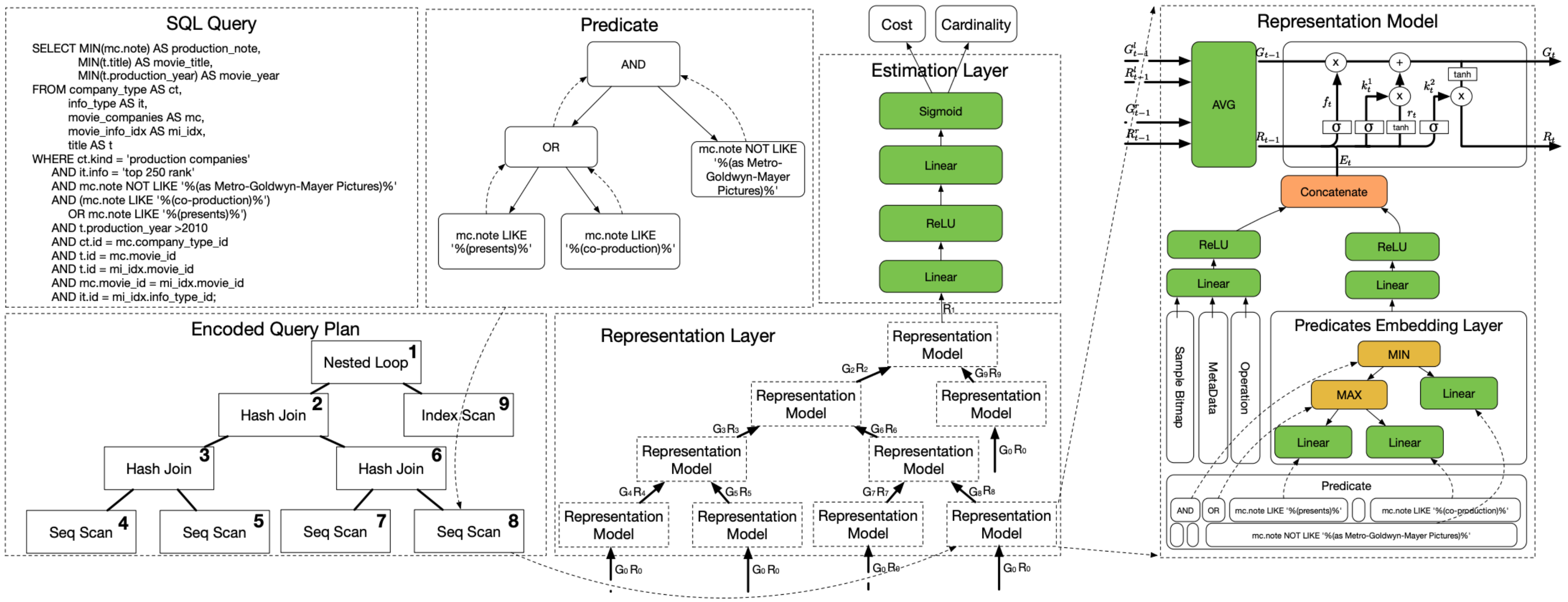
Single Query Cost Estimation

- Challenge
  - Build an end2end model of cost estimation to avoid the accumutative errors of cardinality estimation.
  - The learning model should capture the tree-structured information of the query plan
  - Hard to encode the predicate.

Tree-LSTM for Cost Estimation

◻ Model Construction

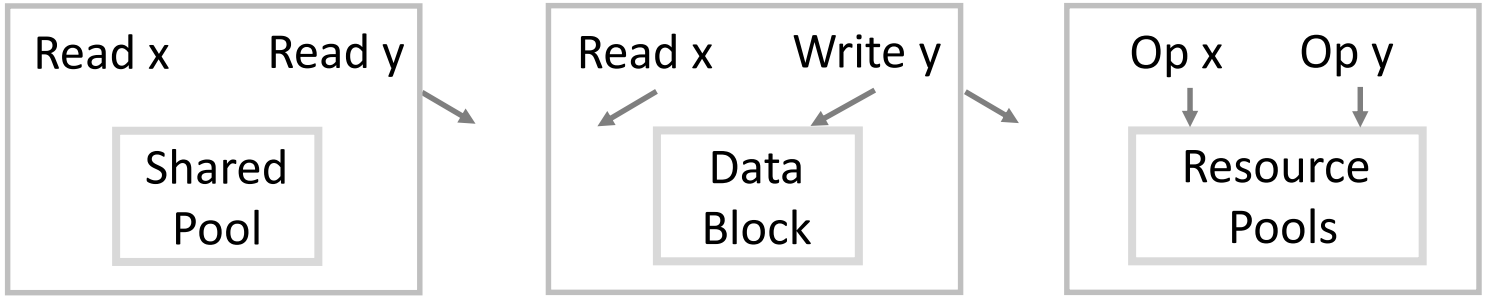
- ➢ The representation layer learns an embedding of each subquery
- ➢ **The estimation layer** outputs cardinality & cost simultaneously



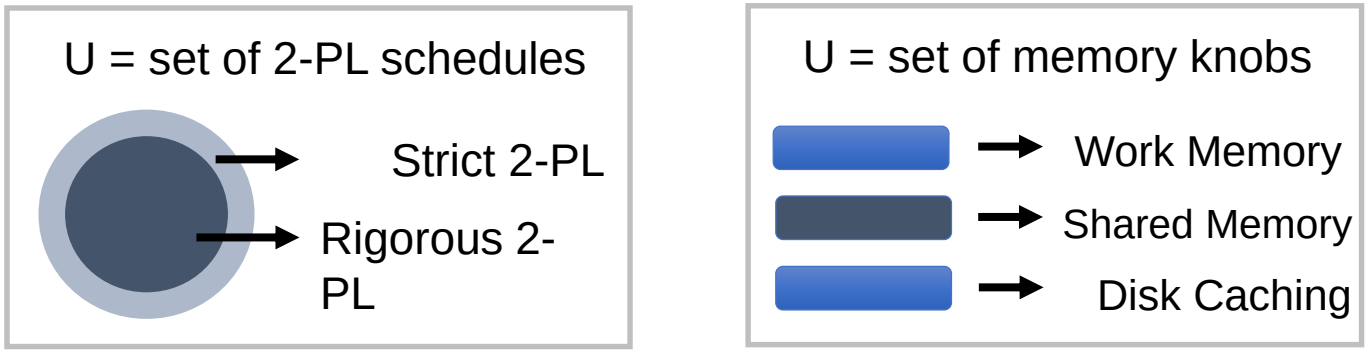
Ji Sun and Guoliang Li. An End-to-End Learning-based Cost Estimator. PVLDB 13, 3 (2019).

QPP for Concurrent Queries *troubles*

- Queries have complex correlations



- Constraints of DB configurations



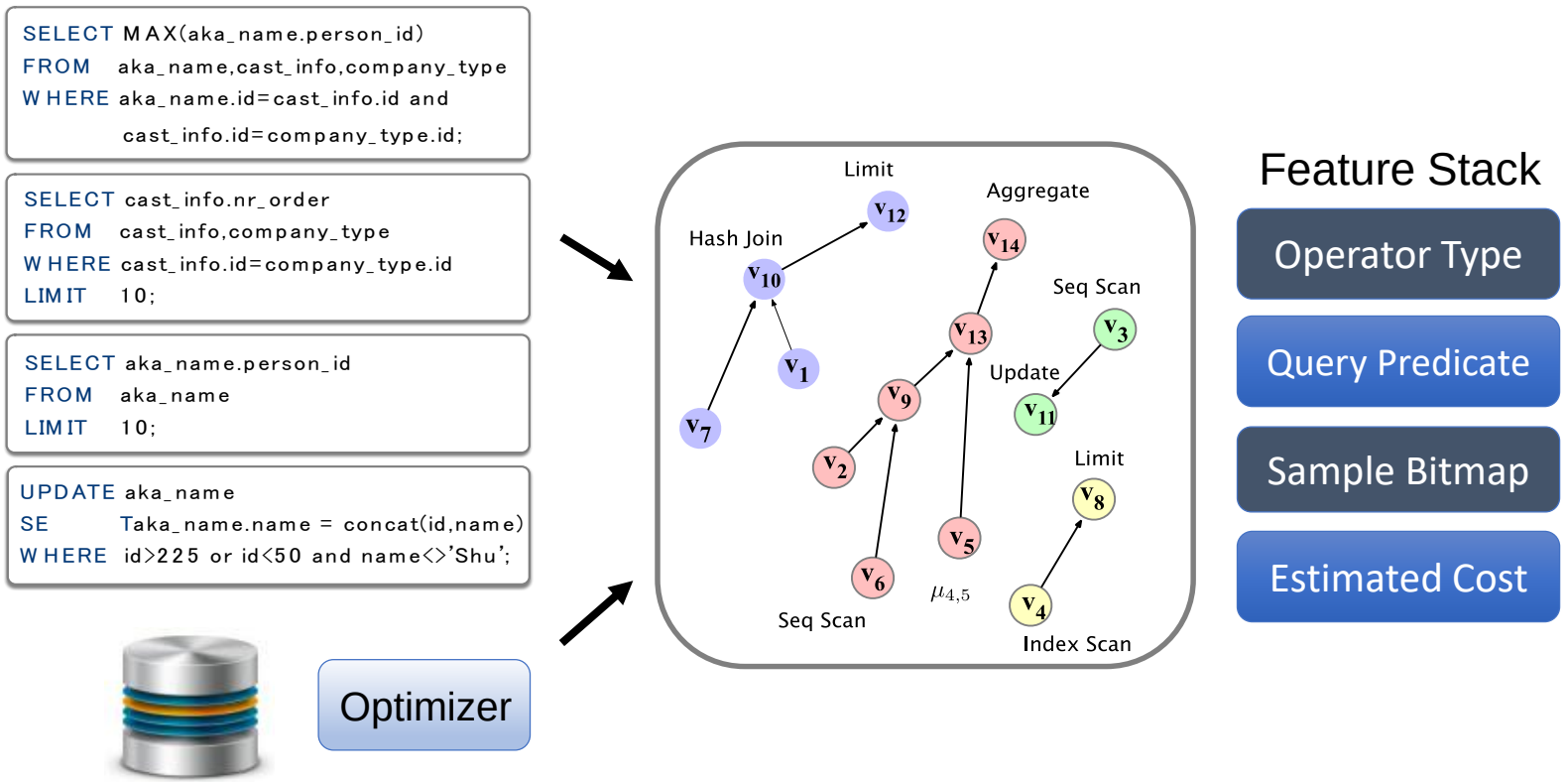
✓ **Queries and relations form a *graph***

✓ **DB configurations also take effects**

→ Predict Performance On a graph model

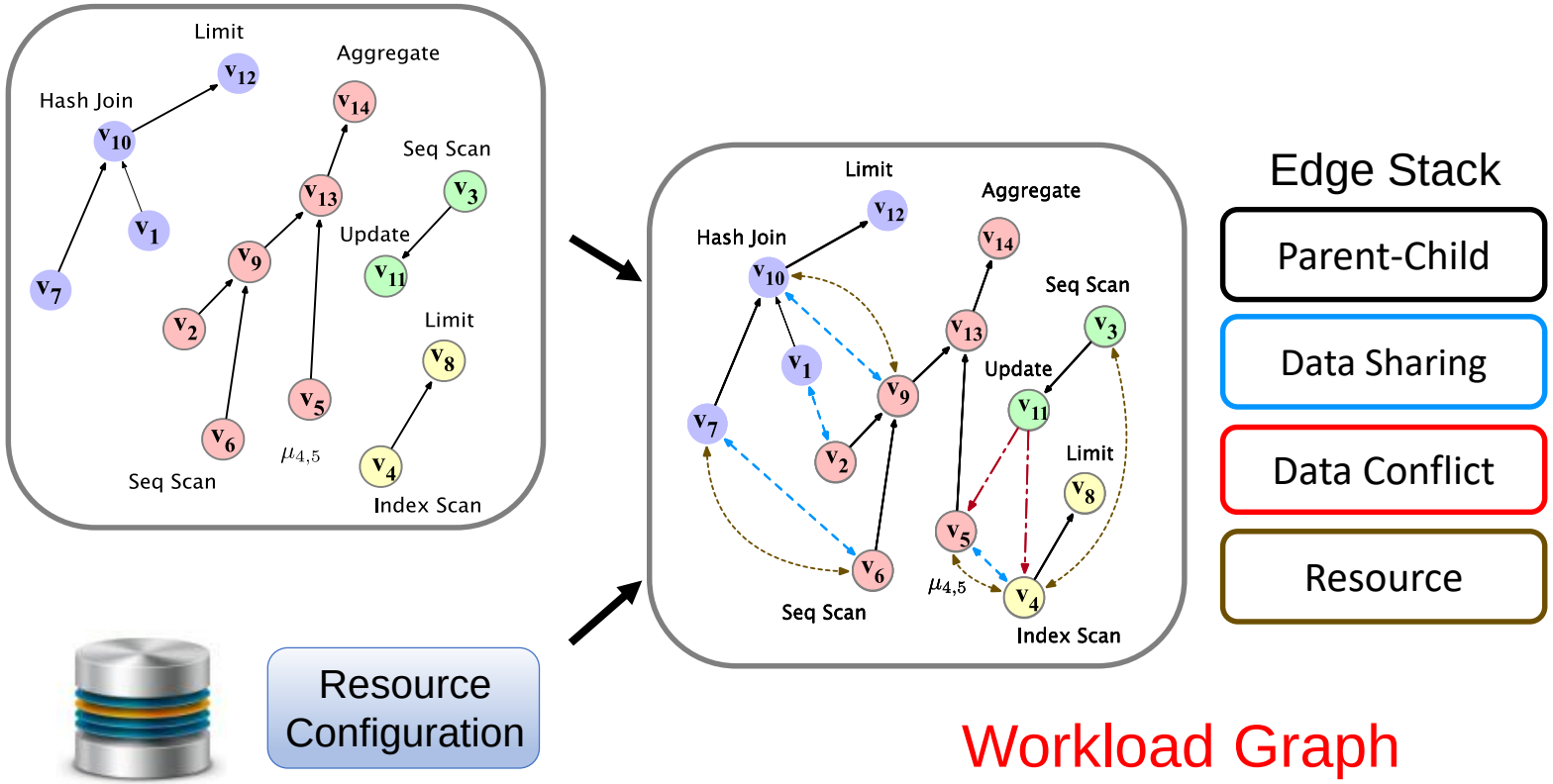
Graph Modeling for QPP

- **Vertex Modeling**: Obtain query plans; and extract operators from plans as vertex features



Graph Modeling for QPP

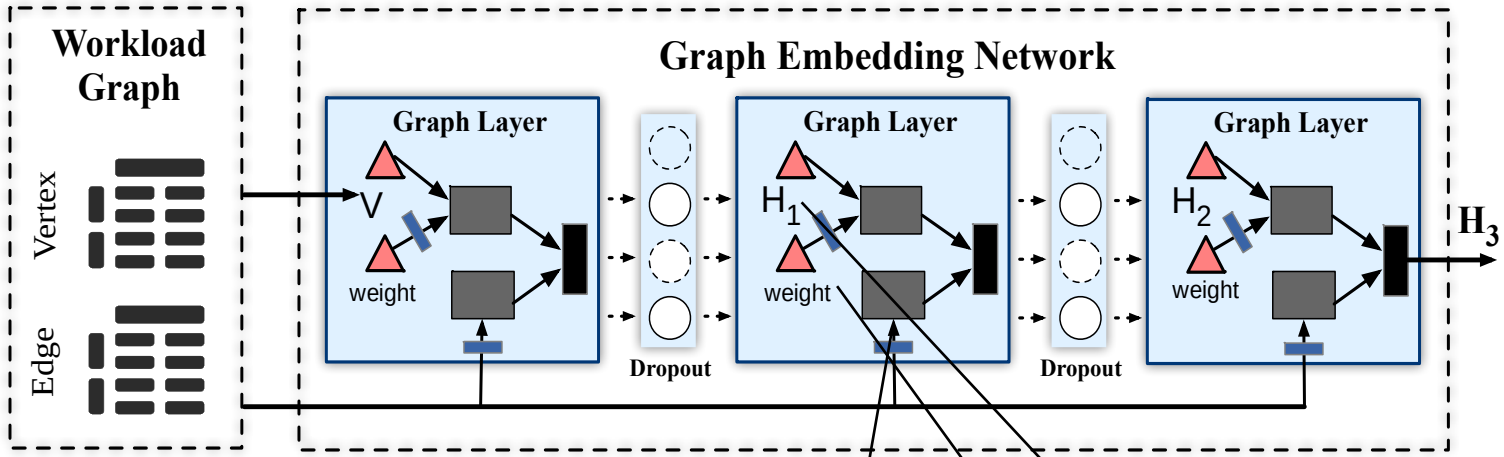
- **Edge Modeling**: Compute 4 types of correlations as edges



Graph Embedding for QPP

- ***Step* 1**: Embed graph features with learned weights

(in vertex level)



In each graph embedding layer:

$$H^l = \sigma^l\left(D^{-\frac{1}{2}} E' D^{-\frac{1}{2}} W^l H^{l-1}\right)$$

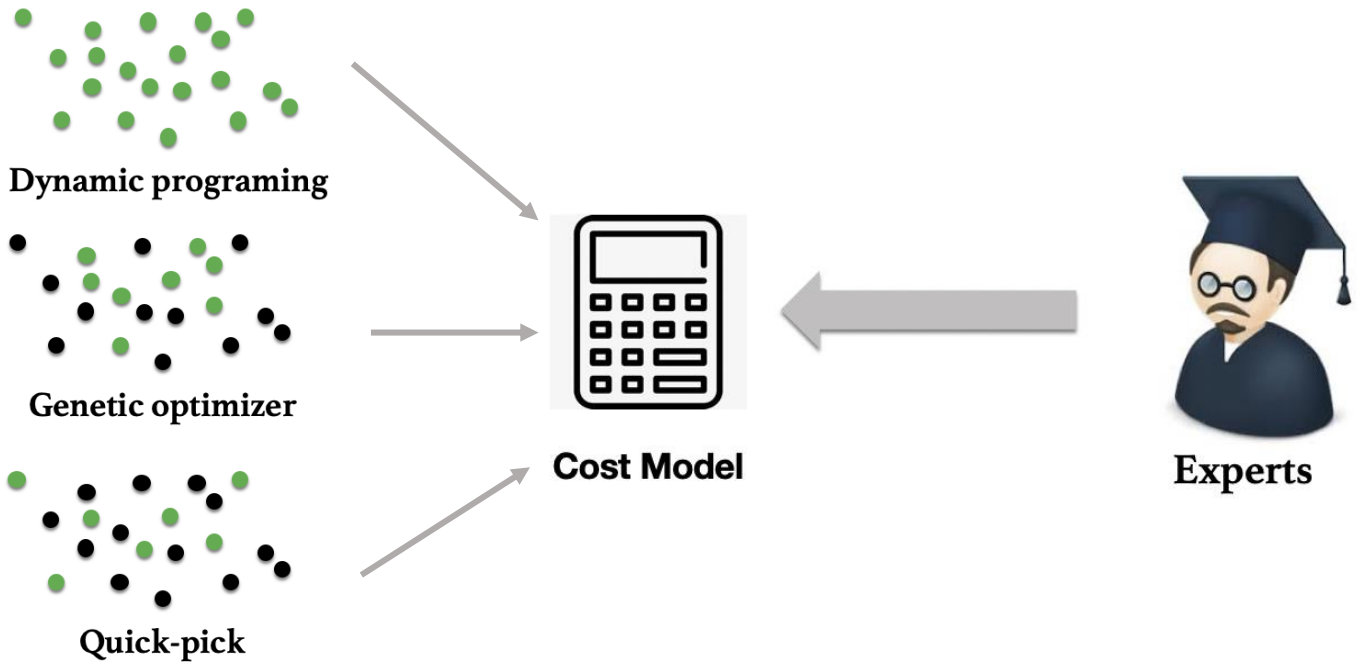
D: neighborhood vertices of every vertex

σ: activation function that conducts nonlinear transformations

Join Order Enumerator

☐Problem Definition

- Given a SQL query, a join ordering is captured by a binary tree, in which each leaf node represents a base relation. The aim is to select the "cheapest" ordering (according to the cost model) for execution.



Traditional Methods

Learning based Join Order Enumerator

Method Classification

Offline learning methods

- Characteristic : Based on the workload，use RL-based methods
- Key idea : Use existing workload to train a learned optimizer, which will predict the plan for future workload.

Online learning methods

- Characteristic: No workload provided, but relies on customized Database
- Key idea : The plan of a query can be changed during execution. The query can switch to another plan if it finds that current plan is bad. It learns when the database executes the query.

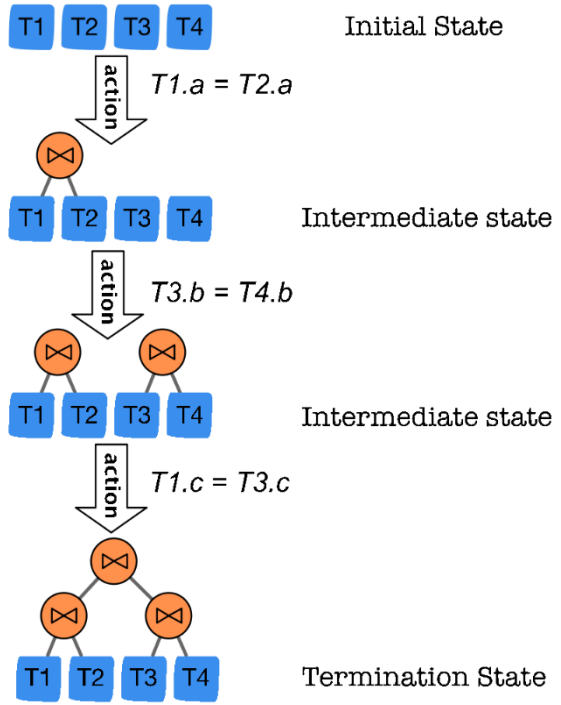
Offline learning(ReJoin)

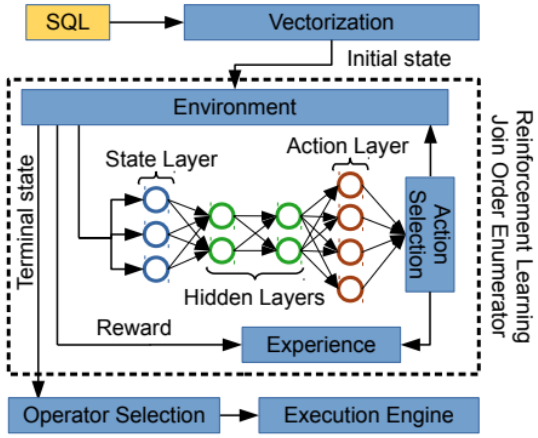
- **Background:**
  - The search space for join order is huge.
  - Traditional optimizer does not learn from previous examples.
- **Challenges:**
  - How to reduce the search space of join order.
  - How to select the best join order.

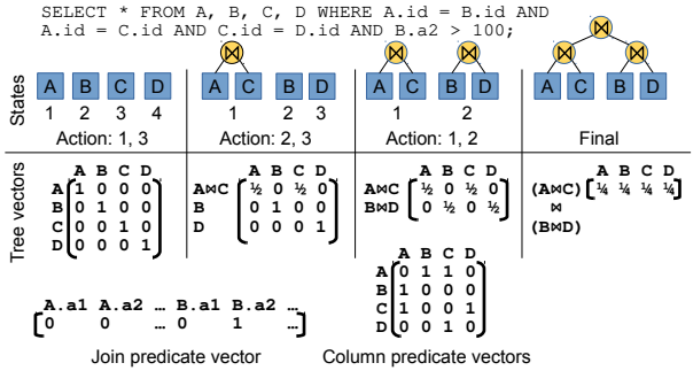
Offline learning(ReJoin)

- RL model
  - Agent : optimizer
  - Action: join
  - Environment: cost model, database
  - Reward： cost , latency
  - State : join order, O(n^2)

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







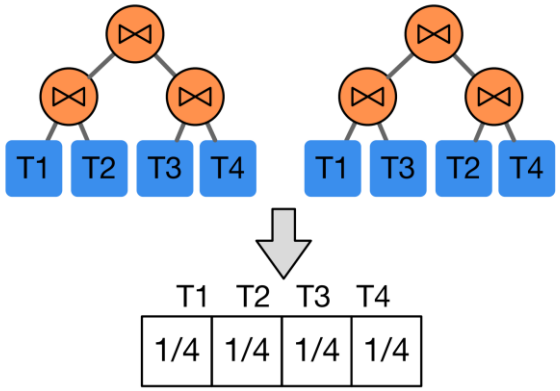
Offline learning(RTOS)

Background:

- Previous learning based optimizers give good cost, but they do not give good latency on test queries.
- Schema often changes in realworld database.

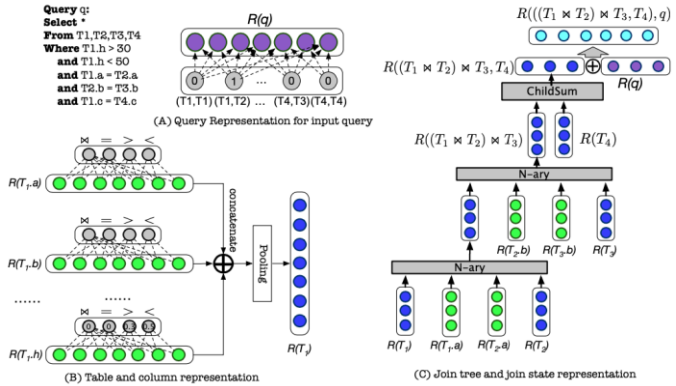
Challenges:

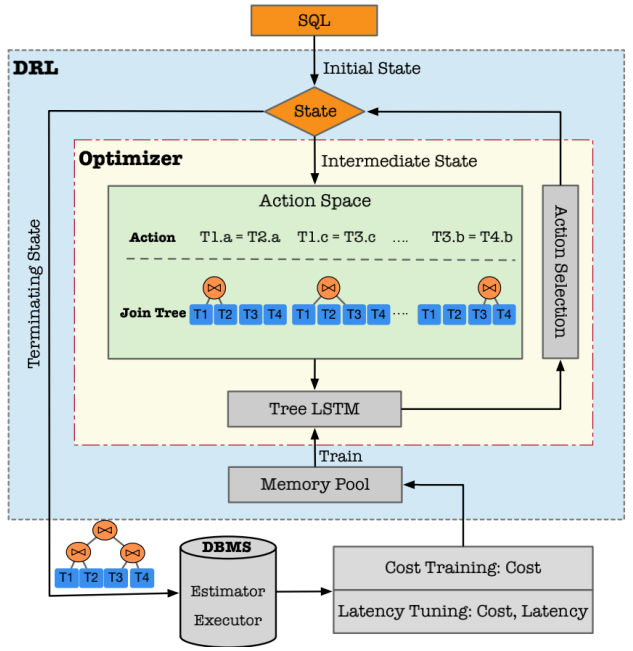
- The intermediate state of the rl is a forest which is hard to represent.
- The training time is huge when collecting latency as feedback.
- The schema change leads to the retraining.



Offline learning (RTOS)

- TreeLSTM based Q network
  - Use n-ary to represent the sub-trees
  - Use child-sum to represent the forest
- Two step training
  - Cost pretrain
  - Latency fine-tuning
- Dynamic neural network
  - DFS to build neural network for each plan





Online learning(SkinnerDB)

☐ Background:

- The workload varies in realworld database.
- Previous learning based optimizer need to give training queries and hard to give good plans to different workload.

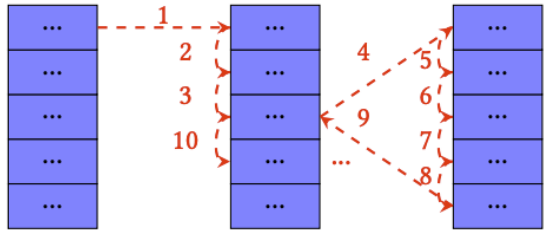
☐ Challenges:

- How to design a new working mechanism that allows the optimizer to learn and switch between different join orders online.
- How to evaluate and choose different join orders online.

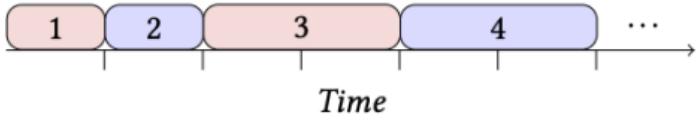
Online learning(SkinnerDB)

Eddies-style

- Divide the execution process into serveral time slices.
- N way join can support the plan switch.
- Select the plan for the next time slice based on the previous time slice

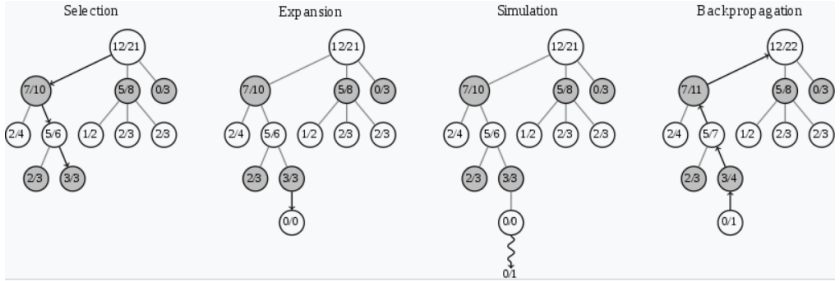


N way join



MCTS For JOS

- Learn and generate a plan in each time slice



Relys on Customize Database

- Switch plan in low latency

Join Order Enumerator

	Quality	Training Cost	Adaptive (workload)	Adaptive (DB Instance)
Traditional Methods [Genetic algorithms] [Dynamic Programming]	Low	Low	✓	High
Offline Optimization Methods [ReJoin aiDM 2018] [RTOS ICDE2020]	High	High	×	Medium
Online Optimization Methods [SkinnerDB SIGMOD2019]	Medium	Low	✓	Low

Learned CostEst and JoinSel: Summary

- They work together to generate an optimal query plan, i.e., JoinSel selects a good order based on CostEst.

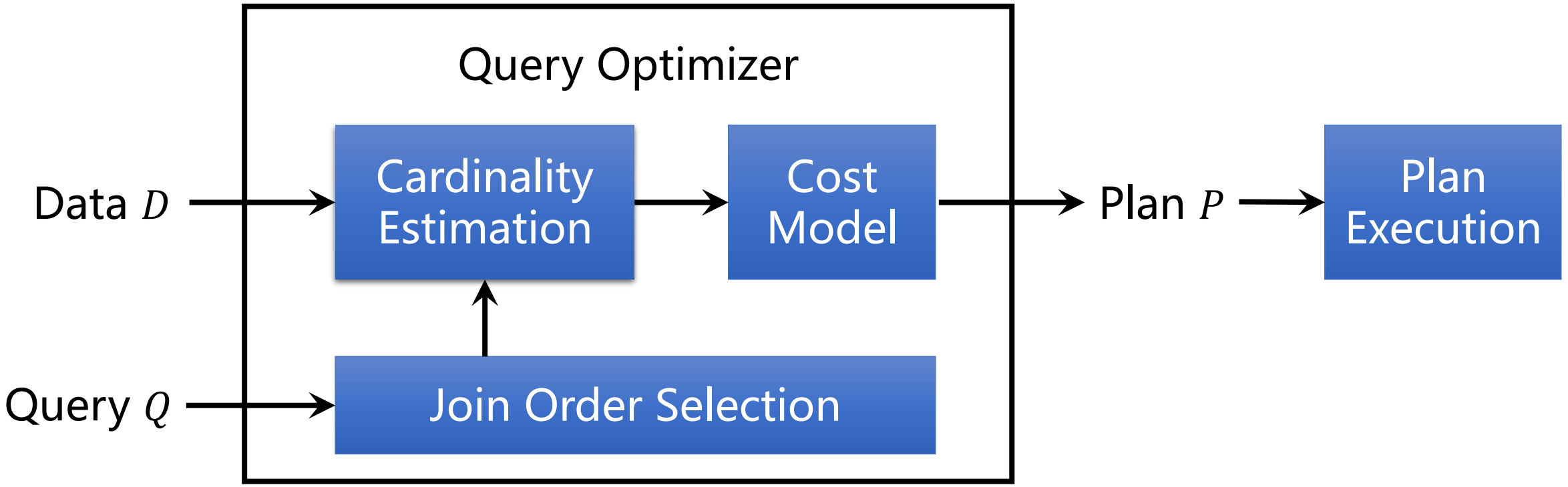
- Both can be optimized through ML techniques.
  - CostEst can leverage the neural network to encode the query and predict the cost, and the methods of CardEst can also be used.
  - JoinSel mostly relies on the RL-based methods, because it can be regraded as a decision making process and the DB can provide feedback as the reward.

Part 3: Learned Whole QO Module

- ❑ End-to-end learned QO: NEO
- ❑ Learn to steer QO: BAO
- ❑ One model for all: MTMLF
- ❑ Comparison and analysis

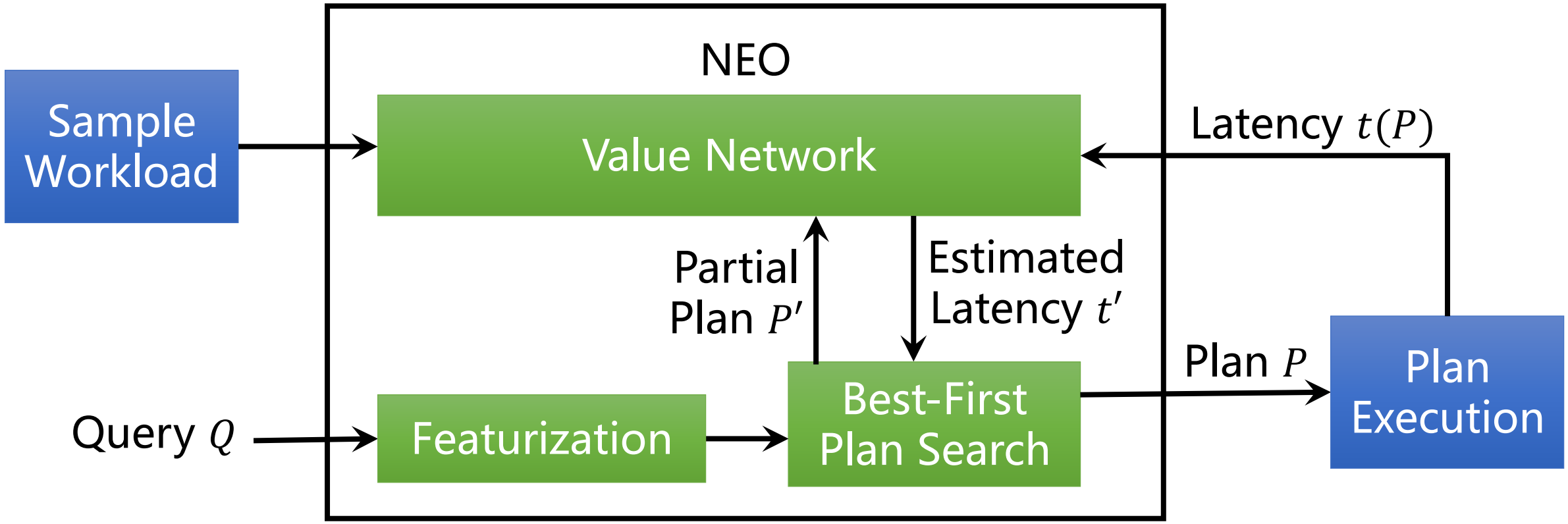
Learned QO Architecture: Revisit

- From learned components to learned QO module



NEO: E2E Learned QO Architecture

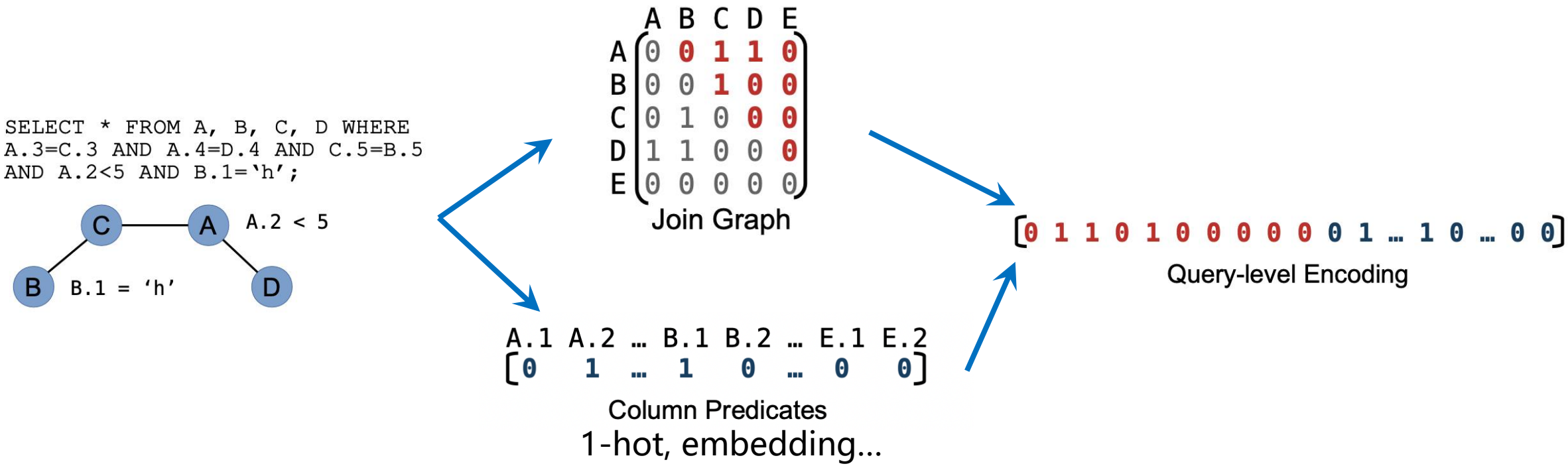
- From input query to executable plan



- Ryan Marcus, Parimarjan Negi, Hongzi Mao, Chi Zhang, Mohammad Alizadeh, Tim Kraska, Olga Papaemmanouil, Nesime Tatbul. *Neo: A Learned Query Optimizer*, *VLDB*, 2019. https://www.vldb.org/pvldb/vol12/p1705-marcus.pdf

NEO: Query Featurization

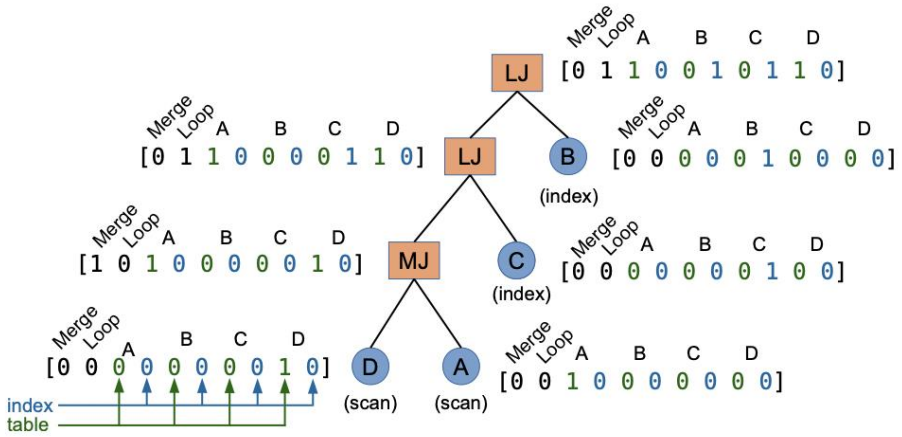
- Query encoding: join information + predicates information



Query Q → Featurization → Query Encoding

NEO: Value Network

- Learn the latency of the best possible plan from partial plan



Mapping function: $f(t'|Q, P')$

Value Network

Partial Plan P'

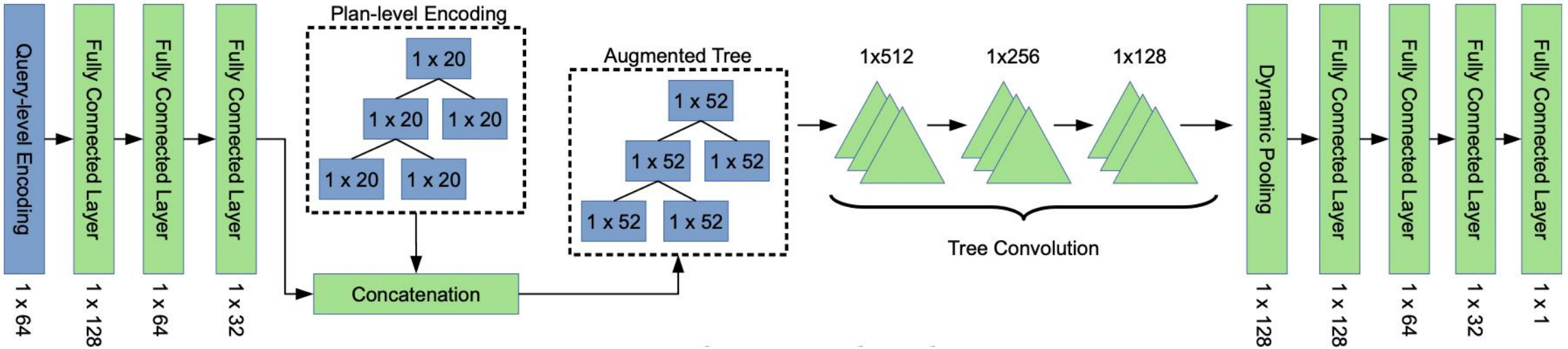
Estimated Latency

$t' = \min(t(P)|P' \subseteq P)$

Query Q → Featurization → Query Encoding

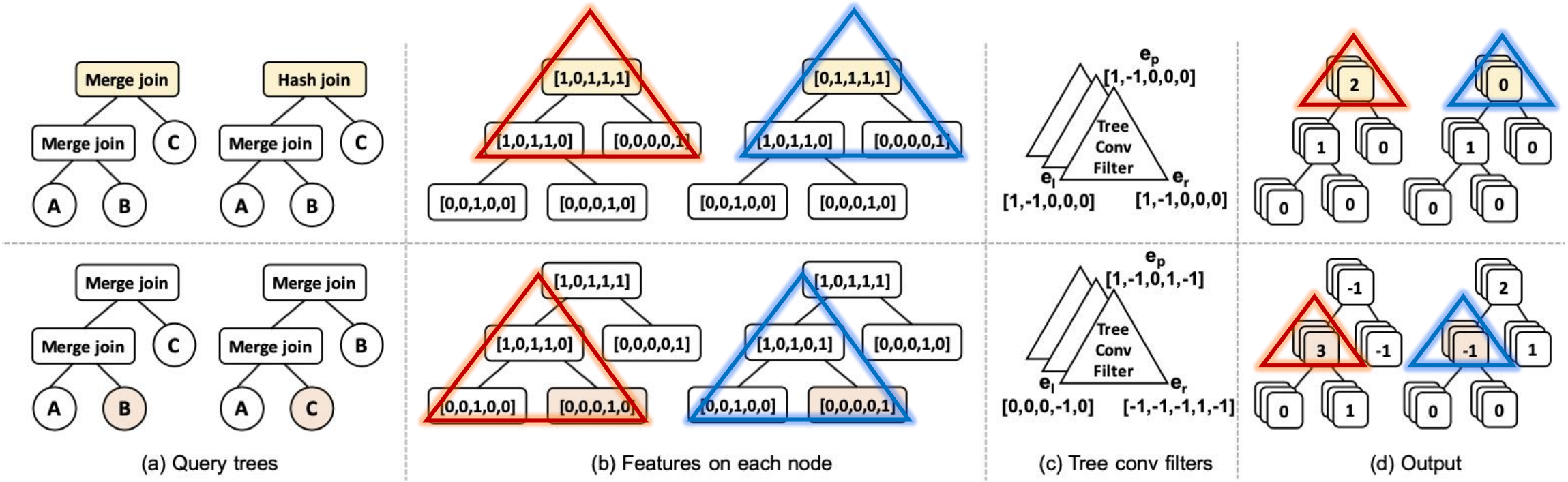
NEO: Value Network

- Network architecture: tree convolution
  - Inductive bias for tree-structured query plan



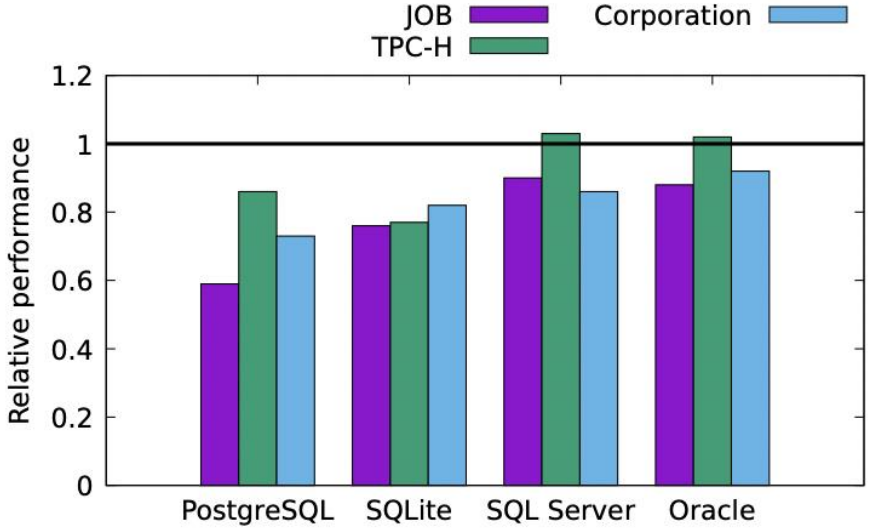
NEO: Value Network

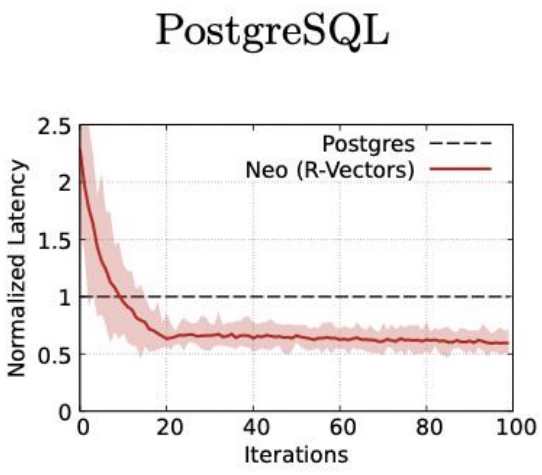
- Network architecture: tree convolution
  - Learn filter weights for different operations/tables automatically

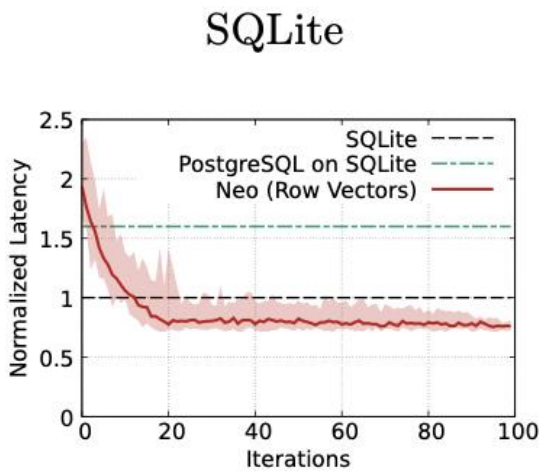


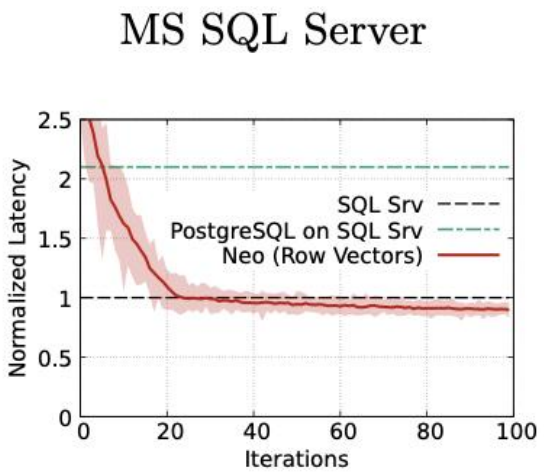
NEO: Evaluation Results

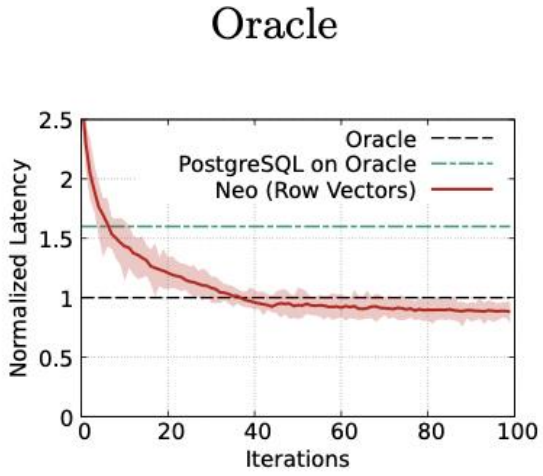
- NEO could outperforms or matches existing commercial query optimizers











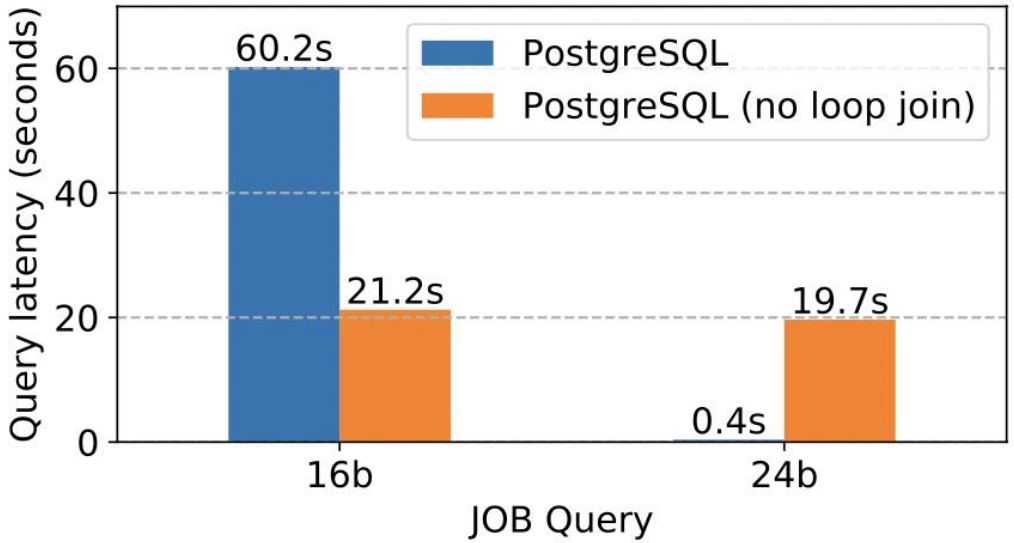
NEO: Summary

- Advantages
  - First claimed automatic e2e learned QO

- Disadvantages
  - Learn everything by itself → Long training time and heavy cold-start
  - Ad-hoc featurization for each DB → Low generalization to update
  - Replace but not modify → Can’ t reuse existing DBMS codes

BAO: Learn to Steer QO

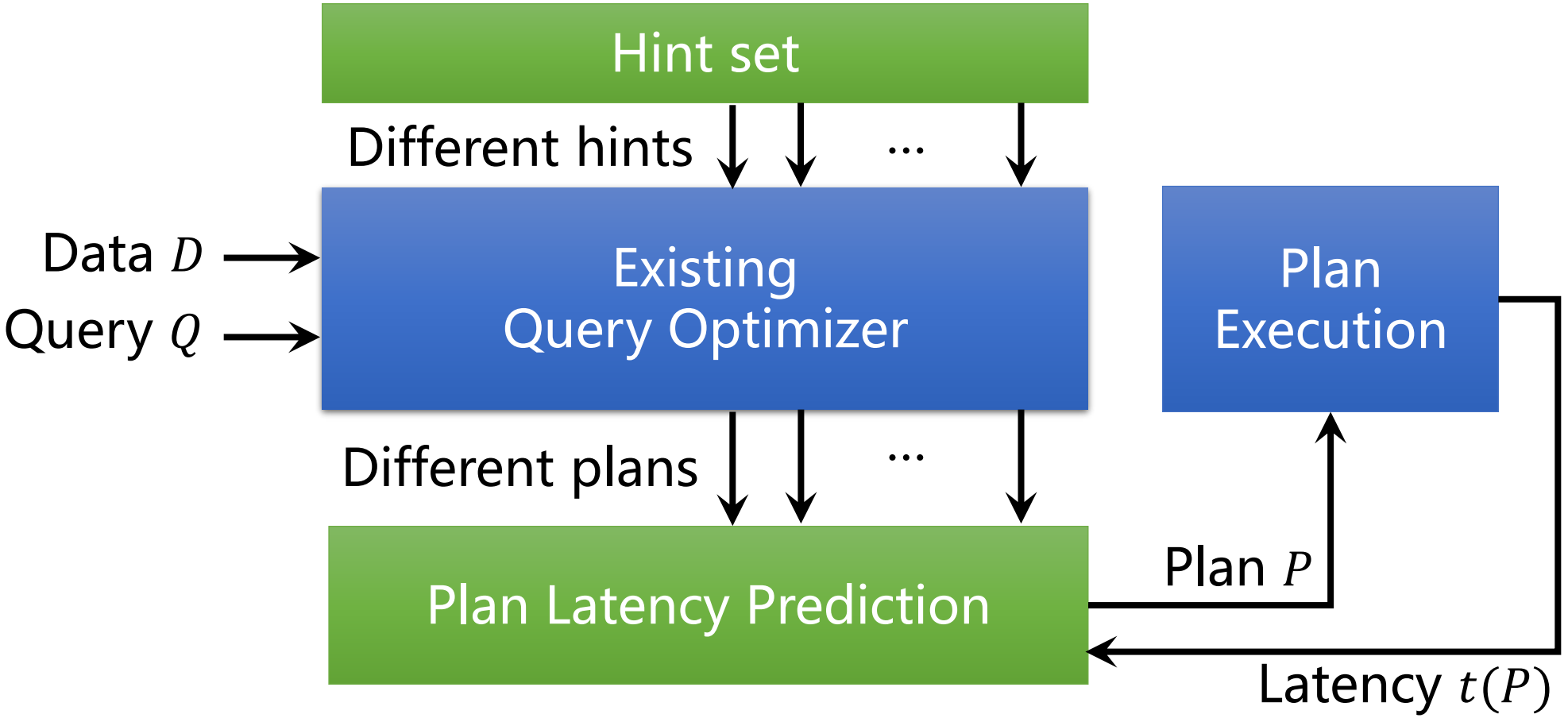
- Existing QO has different hint set: disable/enable certain types of operations, i.e., disable loop join
- For each query, tuning a good hint set may help to generate a good plan compensating its estimation error



- Ryan Marcus, Parimarjan Negi, Hongzi Mao, Nesime Tatbul, Mohammad Alizadeh, Tim Kraska. *Bao: Learning to Steer Query Optimizers*, *SIGMOD*, 2021. https://arxiv.org/abs/2004.03814

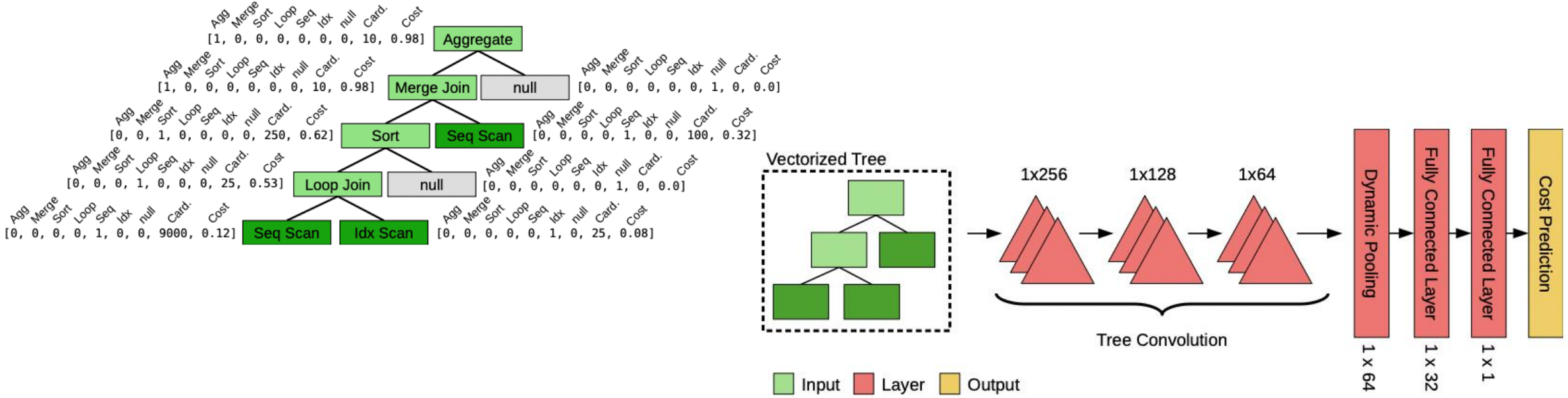
BAO: Learn to Steer QO Architecture

- Learn how to find the right hint set over existing QO



BAO: Latency Prediction Model

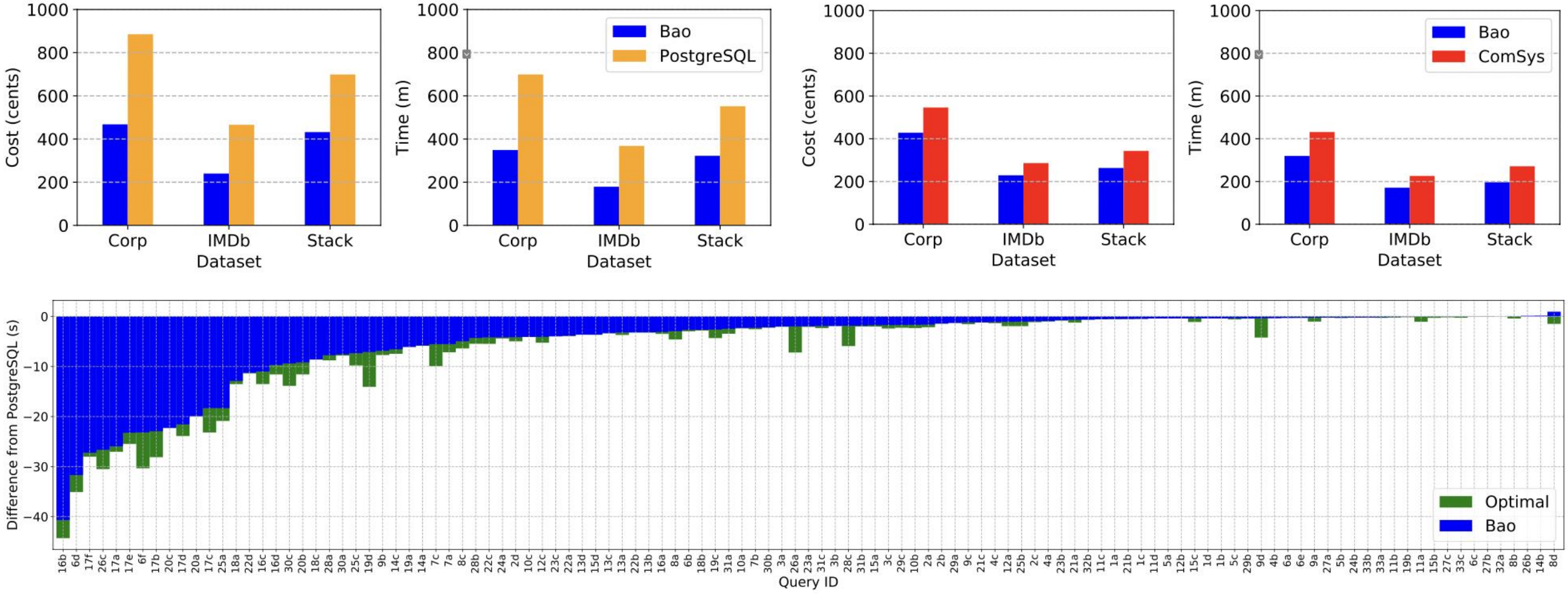
- Similar tree structure and convolution to NEO



Plan P → Plan Latency Prediction → Latency $t(P)$

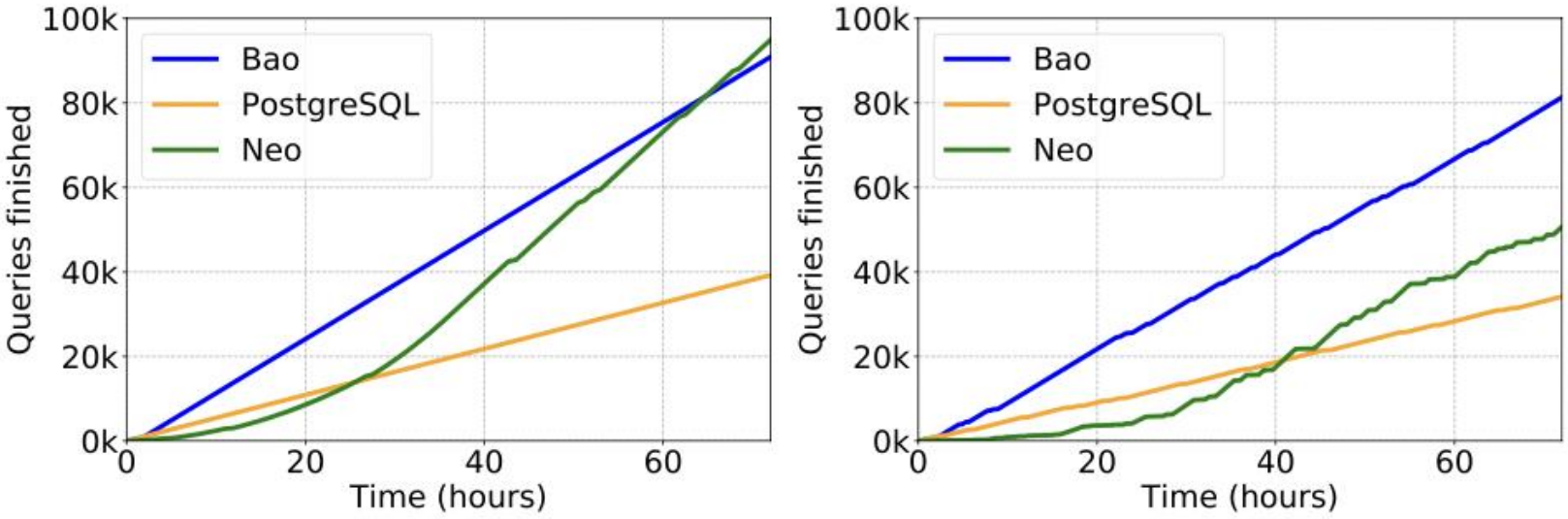
BAO: Evaluation Results

- BAO largely outperforms open-source and commercial DBMS



BAO: Evaluation Results

- NEO could overtake BAO after long time training due to more freedom on plan selection
- BAO coverages fast and easily adapts to dynamic workload



(a) Stable query workload (b) Dynamic query workloads

BAO: Summary

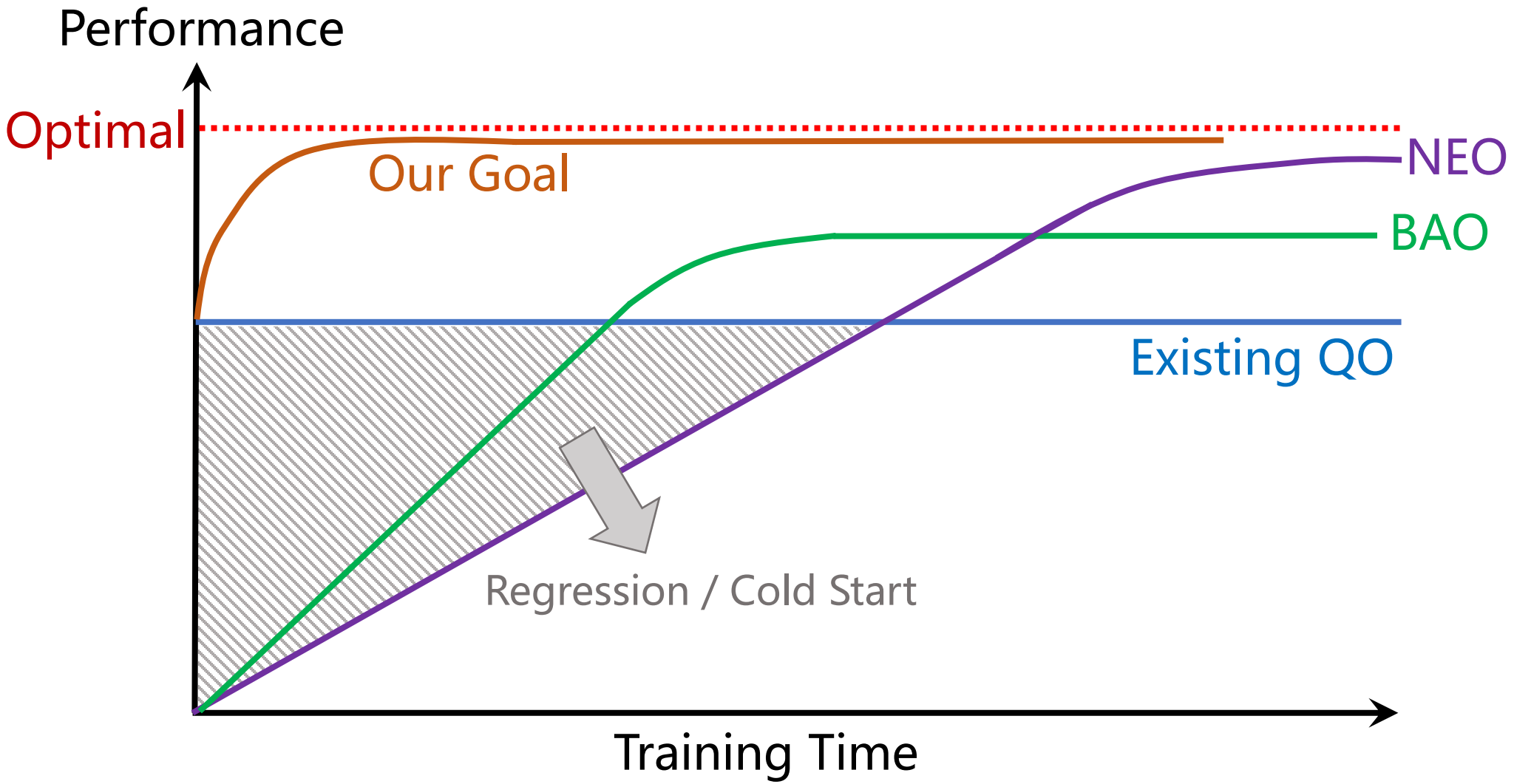
- Advantages
  - Steer but not replace → Reuse existing QO, easy to deploy
  - Easily adapts to data/query/system updates → Better generality
  - Smaller training time w.r.t. NEO

- Disadvantages
  - Less freedom of plan selection → Performance loss sometimes
  - Cold-start and regression problems stills exist

Learned QO: A Comparison

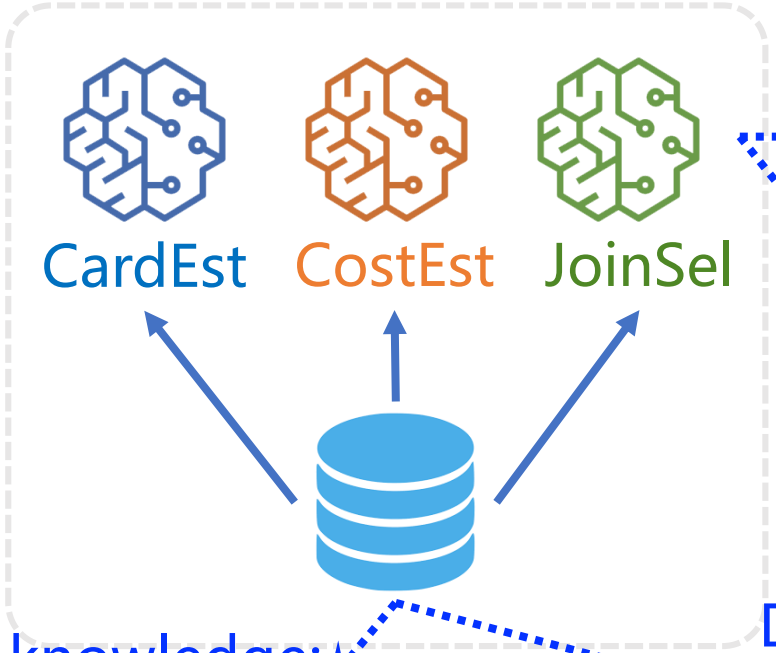
Items	Existing QO	NEO	BAO
Needs CardEst	Yes	No	Yes
Needs Cost Model	Yes	No	Yes
Learning Space	No	Large	Small
Plan Search	DP/Greedy	Best-first	DP/Greedy with hint set tuning
Plan Selection Freedom	High	High	Low
Query Encoding	No	Ad-hoc	Easy
Training Time	No	Long	Fast
Hands Update	Easy	Hard	Easy
Cold Start / Regression	No	Serious	Heavy
Deployment	Easy	Hard	Easy

Learned QO: Our Goal



MTMLF: One Model for All

- Basic idea: learned knowledge is decomposable



Task specific knowledge: method to solve a specific task, e.g. join ordering selection rules

Task shared knowledge: data and query representations used in all tasks

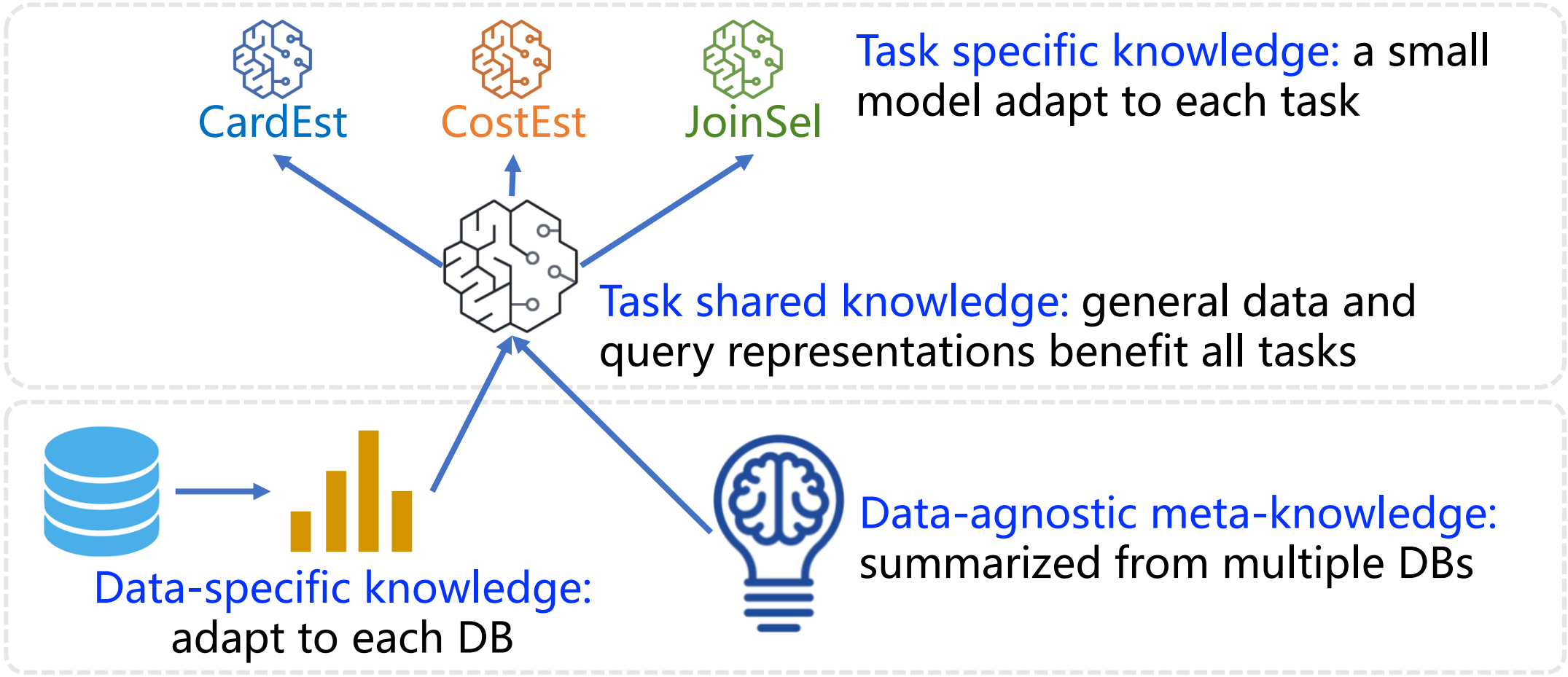
Data-agnostic meta-knowledge: common rules across DBs e.g., expert experiences and multi-table join rules

Data-specific knowledge: data unique, e.g. data and query workload distributions

- Ziniu Wu, Peilun Yang #, Pei Yu #, Rong Zhu, Yuxing Han , Yaliang Li, Defu Lian, Kai Zeng, Jingren Zhou. *A Unified Transferable Model for ML-Enhanced DBMS*, CIDR, 2022. https://arxiv.org/abs/2105.02418

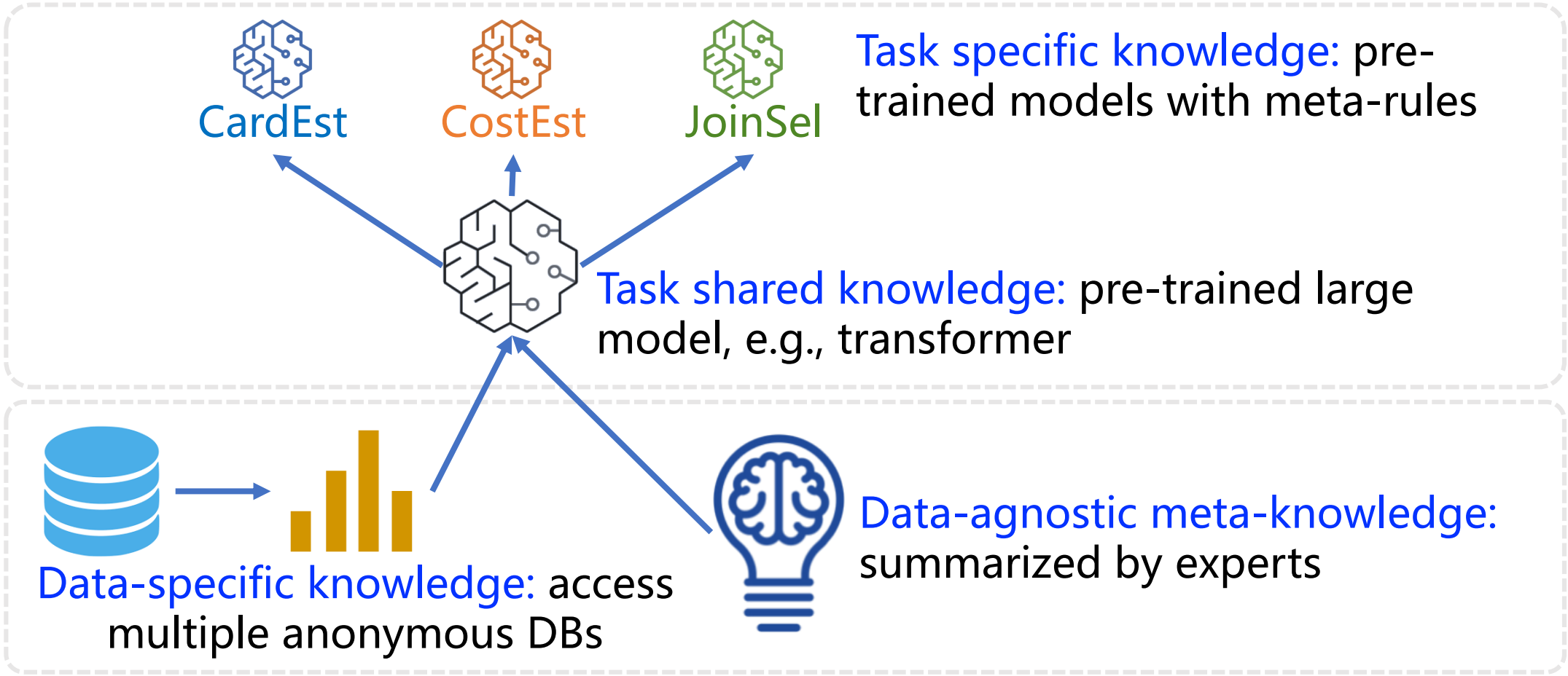
MTMLF: One Model for All

- Shared learning and specific adaption



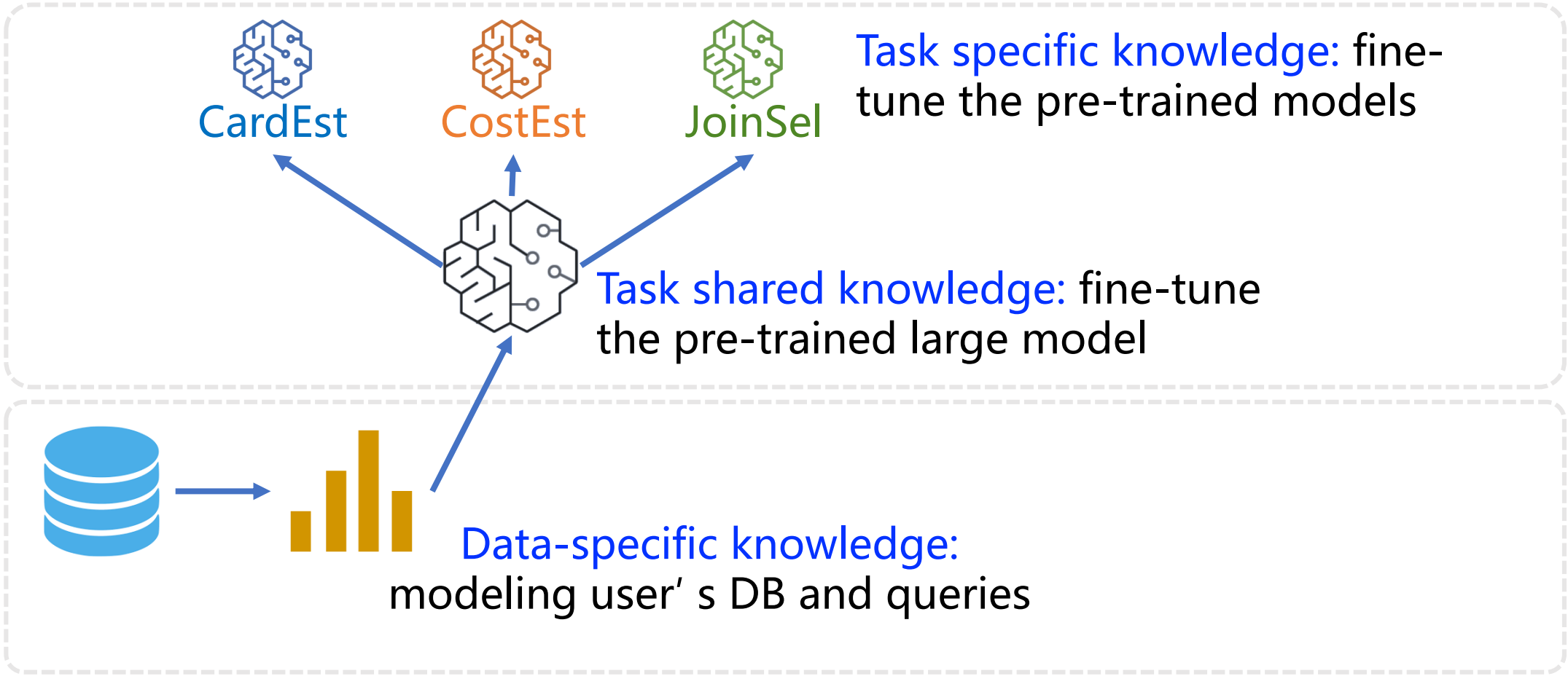
MTMLF: Service Provider Side

- Pre-trained large models as services



MTMLF: User Side

- Fine-tune pre-trained models to fit user data



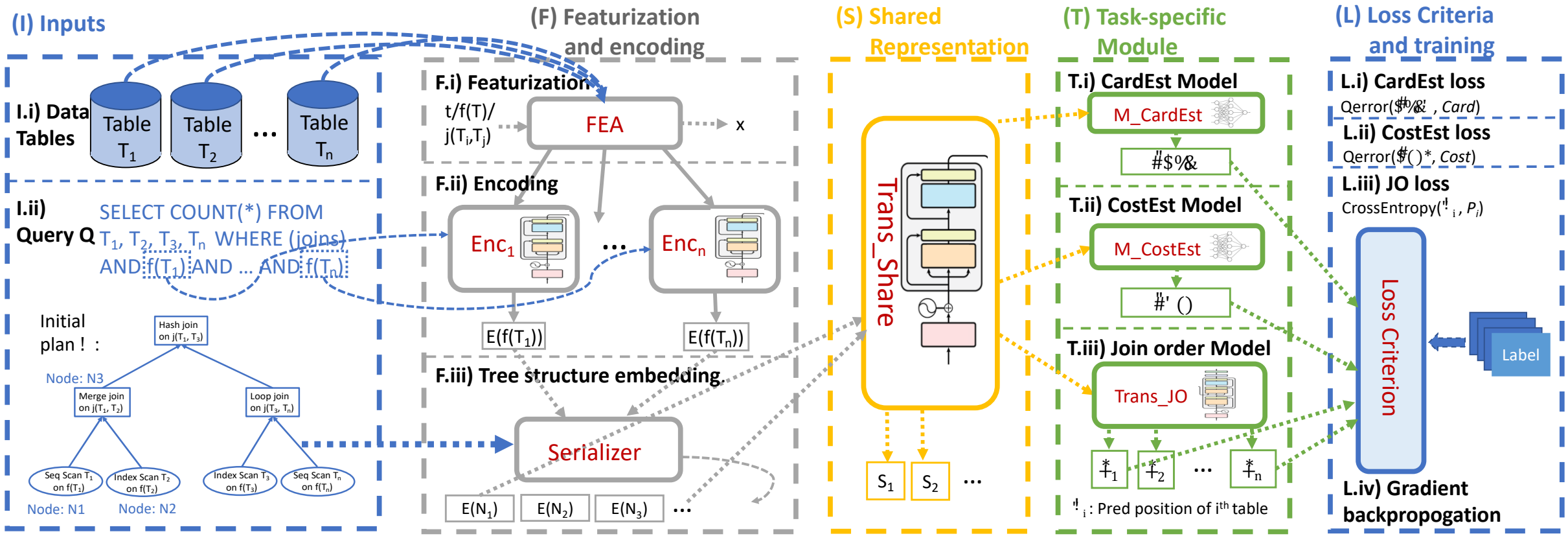
MTMLF: Advantages

- Architecture
  - More efficient training without redundant learning
  - More effective task modeling with posterior knowledge guidelines, e.g., CostEst to CardEst
  - Transferability to avoid cold start/regression

- Workflow
  - More green computation: pre-trained large models provided as fundamental tool
  - More in-depth optimization: see more data
  - Easy to evolve and manage

MTMLF: Detailed Architecture

- Jointly learn multiple components in QO together



MTMLF: Evaluation Results

- One model for all architecture is more accurate
  - Better than previous plug-in architecture
  - Multi-task learning > separate learning

Method	Cardinality Q-Error			Cost Q-Error		
	median	max	mean	median	max	mean
PostgreSQL	184	670,000	10,416	4.90	4,920	105
Tree-LSTM	8.78	696.29	36.83	4.00	290.35	15.01
Ours-QO	**4.48**	**614.45**	**28.69**	**2.10**	**37.54**	**4.20**
Ours-CardEst	5.12	804.48	36.66	---	---	---
Ours-CostEst	---	---	---	2.06	61.41	4.69

MTMLF: Evaluation Results

- One model for all architecture is effective
- Pre-train/fine-tune is transferable to new DBs

Method	Execution Time (min)	Improvement Ratio
PostgreSQL (Baseline)	1143.2	---
Optimal	209.1	81.7%
Ours-QO	318.3	72.2%
Ours-JoinSel	450.4	60.6%

Method	Execution Time (min)	Improvement Ratio
PostgreSQL (Baseline)	393.3	---
Ours-QO (No Finetune)	234.1	40.6%
Ours-QO (Retrained)	219.5	44.3%

Learned QO Module: Summary

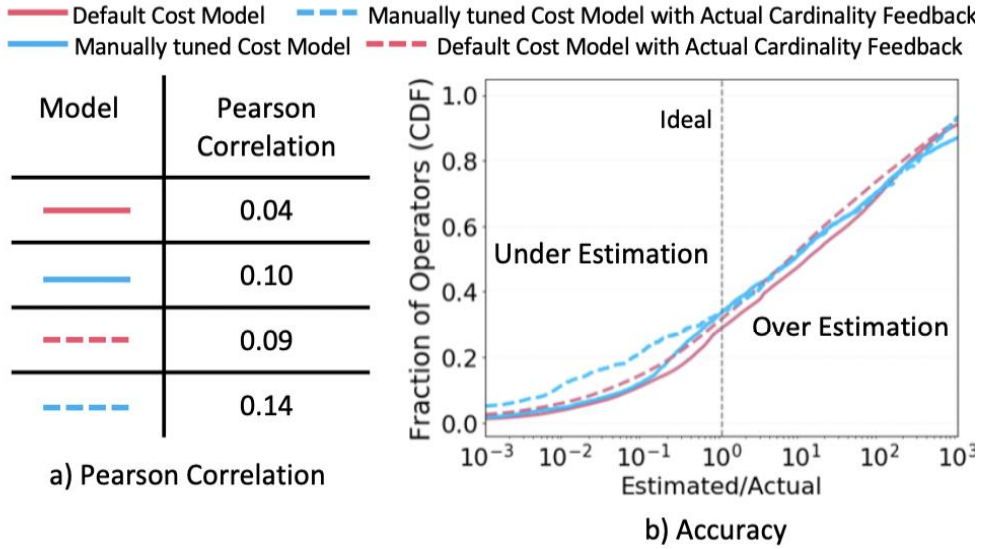
- NEO: Learn everything e2e by itself
  - Long training time, low generality and heavy cold-start
- BAO: learn to steer existing QO by tuning hint set
  - Easy to deploy and better generality
  - Performance loss and remaining cold-start problem
- MTMLF: pre-training + fine-tuning
  - A possible routine for a desirable learned QO
  - We still have a long way to go .....

Part 4: Applications and Deployment

- Application case studies of learned QO
  - Learned cost model for SCOPE
  - Learn to steer SCOPE
- Challenges of actual deployment
- Start-up system for deployment: Baihe

SCOPE and Its Cost Model

- SCOPE: the big data system for data analytics in Microsoft
  - Mainly consists of recurring jobs: same scripts, different data
  - Cost-based optimizer: wide gap between actual/estimated cost
  - Cost model gap exists even with actual cardinality

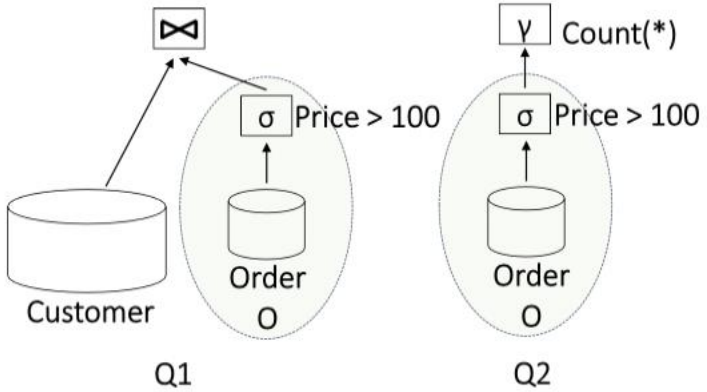


- Tarique Siddiqui, Alekh Jindal, Shi Qiao, Hiren Patel, Wangchao Le. *Cost Models for Big Data Query Processing: Learning, Retrofitting, and Our Findings*, *SIGMOD*, 2020. https://arxiv.org/pdf/2002.12393.pdf

Learned Cost Model

- A single global model: can’ t equally effective for all operators
  - Different operators have very different performance behavior
  - Performance of the same operator can vary drastically w.r.t. context

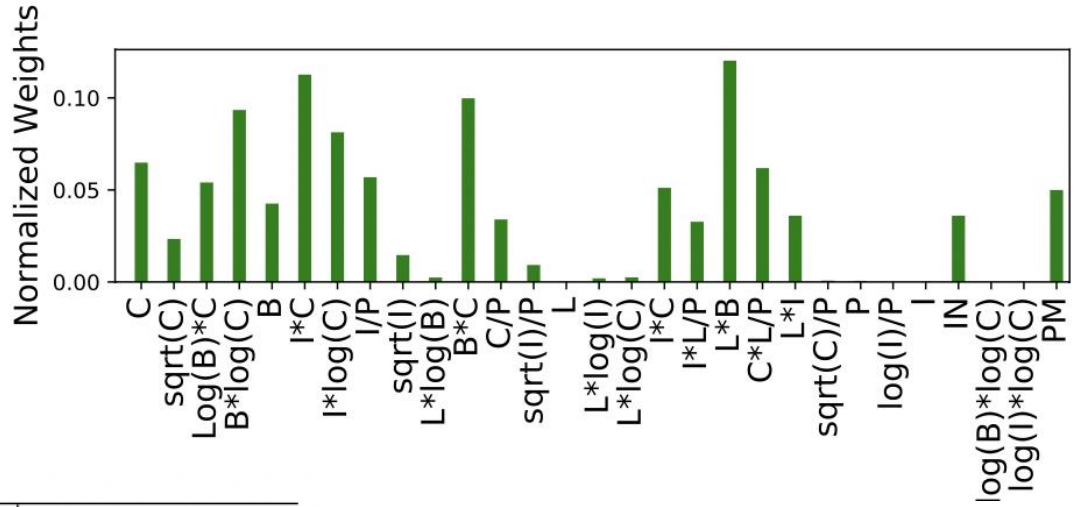
- Solution: many specialized small models, each for an operator-subgraph template
  - Common sub-expressions frequently occurring in recurring jobs



Learned Cost Model

- Operator-subgraph model: high accuracy
  - Low coverage: some operator subgraphs can’ t be covered

Feature	Description
Input Cardinality (I)	Total Input Cardinality from children operators
Base Cardinality (B)	Total Input Cardinality at the leaf operators
Output Cardinality (C)	Output Cardinality from the current operator
AverageRowLength (L)	Length (in bytes) of each tuple
Number of Partitions (P)	Number of partitions allocated to the operator
Input (IN)	Normalized Inputs (ignored dates, numbers)
Parameters (PM)	Parameters



Model	Correlation	Median Error
Default	0.04	258%
Neural Network	0.89	27%
Decision Tree	0.91	19 %
Fast-Tree regression	0.90	20%
Random Forest	0.89	32%
Elastic net	**0.92**	14%

Learned Cost Model

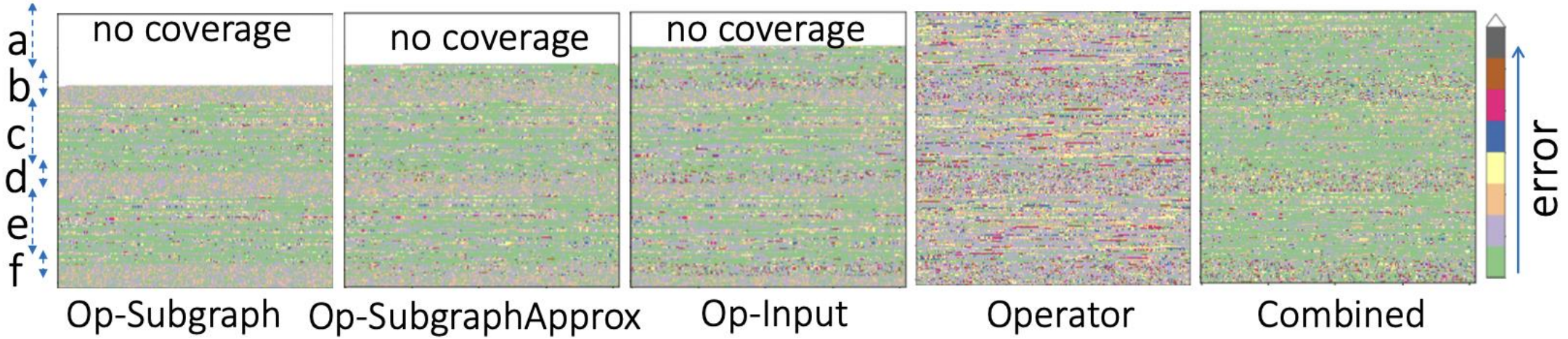
- Accuracy-coverage tradeoff
  - Another extreme: operator model
  - High coverage but poor accuracy, the behavior of an operator changes w.r.t. context
- Two trade-offs
  - Operator-input model: multiple per-operator models, each for jobs sharing similar inputs
  - Operator-subgraph Approx model: learn one model for all subgraphs sharing approximate subgraph structure

Learned Cost Model

- Combined models: a meta-ensemble

Model	Correlation	Median Error
Default	0.04	258%
Neural Network	0.79	31%
Decision Tree	0.73	41 %
FastTree Regression	**0.84**	**19%**
Random Forest	0.80	28%
Elastic net	0.68	64%

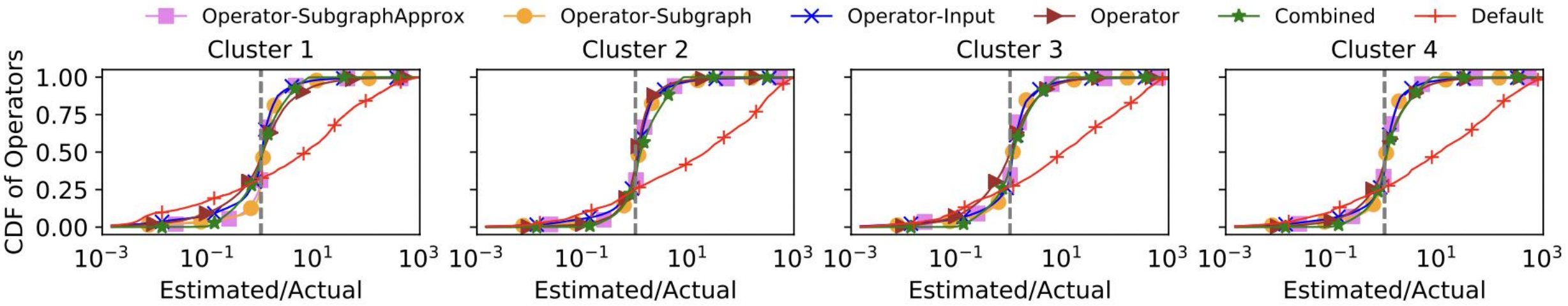
Model	Correlation	Median Error	Coverage
Default	**0.04**	**258%**	**100%**
Op-Subgraph	0.92	14%	54%
Op-Subgraph Approx	0.89	16 %	76%
Op-Input	0.85	18%	83%
Operator	**0.77**	**42%**	**100%**
Combined	**0.84**	**19%**	**100%**



Real-world Evaluation Results

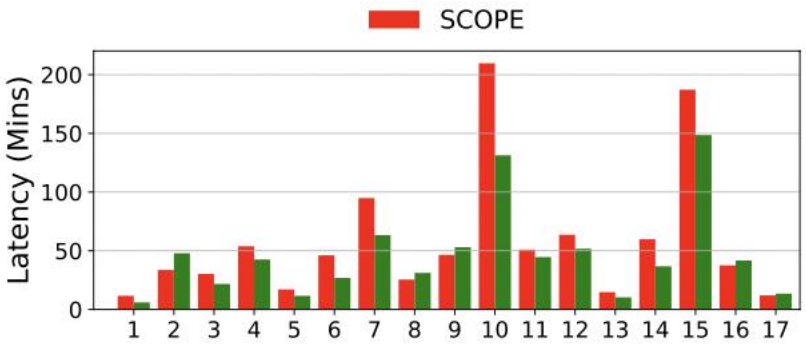
- Learned cost models are accurate and robust

	All jobs				Ad-hoc jobs			
	Correlation	Median Error	95%tile Error	Coverage	Correlation	Median Error	95%tile Error	Coverage
Default	**0.12**	**182%**	12512%	**100%**	**0.09**	**204%**	17791%	**100%**
Op-Subgraph	0.86	9%	56%	65%	0.81	14%	57%	36%
Op-Subgraph Approx	0.85	12 %	71%	82%	0.80	16 %	79%	64%
Op-Input	0.81	23%	90%	91%	0.77	26%	103%	79%
Operator	**0.76**	33%	138%	**100%**	**0.73**	**42%**	186%	**100%**
Combined	**0.79**	**21%**	112%	**100%**	**0.73**	**29%**	134%	**100%**



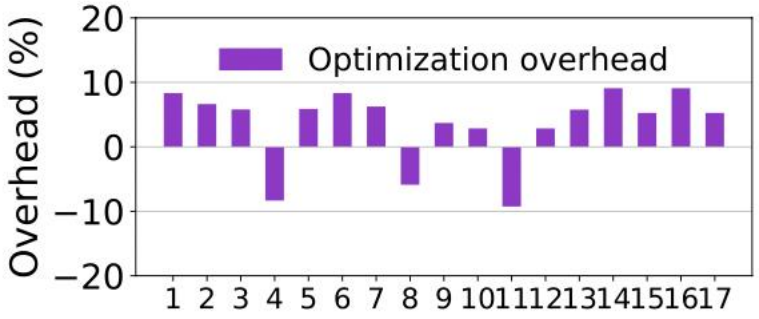
Real-world Evaluation Results

- Learned cost models could improve benchmark and production jobs plan quality

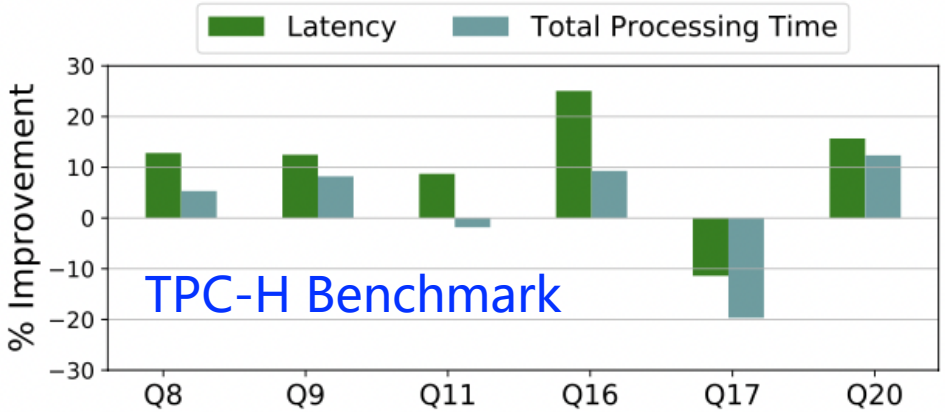


(a) Changes in Latency

(b) Changes in Total Processing Time

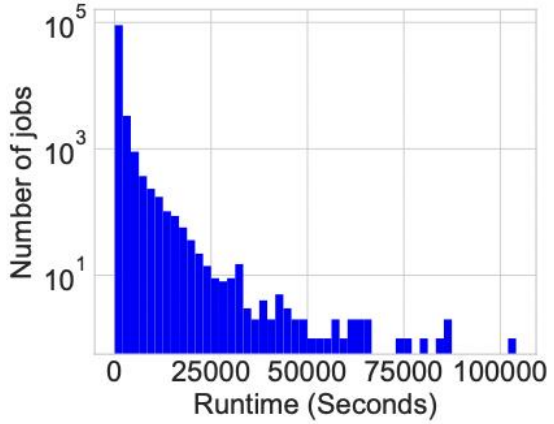


(c) Optimization Time Overhead

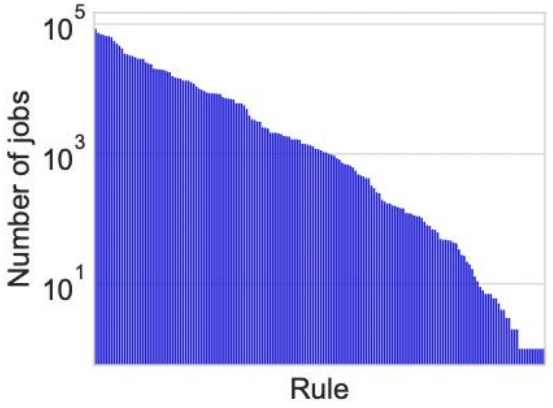


SCOPE and Its Rule Configuration

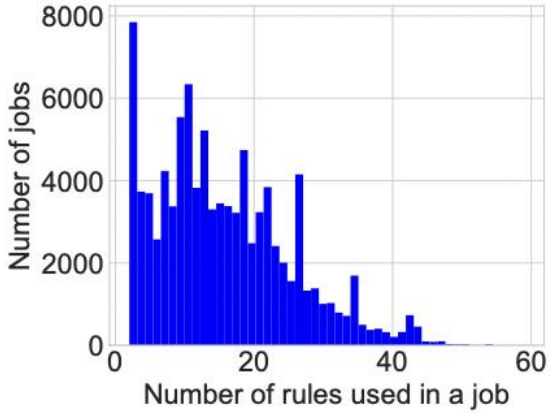
- SCOPE: query workload and different rules
  - Only 10% queries > 5mins, but they consume 90% containers
  - 256 rules in total, 100-150 rules are used frequently
  - Typically 10 – 20 rules are used in a single job



(a) Histogram of SCOPE job runtimes.



(b) Number of jobs using each of the rules.



(c) Histogram of number of different rules used in a job.

- Parimarjan Negi, Matteo Interlandi, Ryan Marcus, Mohammad Alizadeh, Tim Kraska, Marc Friedman, Alekh Jindal. *Steering Query Optimizers: A Practical Take on Big Data Workloads*, *SIGMOD*, 2021. https://dl.acm.org/doi/pdf/10.1145/3448016.3457568

SCOPE and Its Rule Configuration

- Problem: find suitable rules (hint set) for each job to steer QO
  - Large learning space: 219 rules
  - Expensive execution for model training

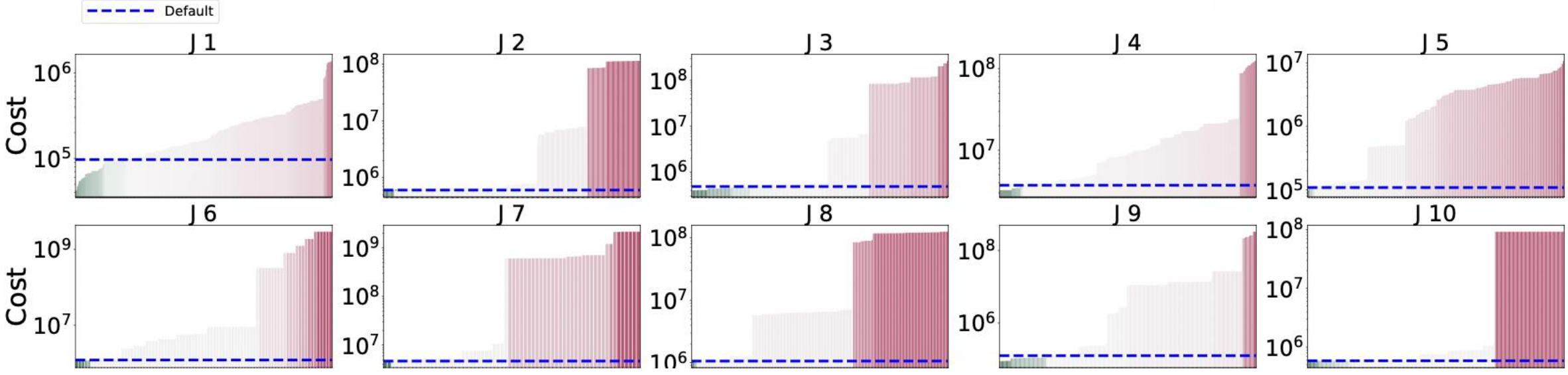
Category	#Rules	#Unused Rules	Rule Examples
Required	37	9	EnforceExchange, BuildOutput GetToRange, SelectToFilter
Off-by-default	46	36	CorrelatedJoinOnUnion1, GroupbyOnJoin
On-by-default	141	37	NormalizeReduce, CollapseSelect, SelectPartitions, SequenceProjectOnUnion
Implementation	32	4	HashJoinImpl1, JoinToApplyIndex1, UnionToVirtualDataset

Method Overview

- Discover rule configurations: which ones we should look at
- Candidate rule configuration running: do improve runtime
- Extrapolate to other jobs by learning models
- Deploy and test online

Rule Configuration Discovery

- Finding a number of interesting rule configurations that may change jobs' performance by randomized heuristic search
- Tuning rule configurations can find plans with estimated costs lower than the default rule configuration
  - Disabling rules would block certain code paths chosen by bad estimates or heuristics in the optimizer, thus nudge better plans



Rule Configuration Selection

- Plans performance could improve drastically using the best seen rule configuration

	Workload		
	A	**B**	**C**
# Queries	36	155	45
ΔRuntime	-1689s	-663s	-400s
ΔPercentage	-30%	-15%	-7%

- Examples on rule configuration difference
  - Enable off-by-default rules is not enough
  - Disable rules could prevent bad code paths
  - Alternate rules would be better

Job	**Runtime %change**	**Rules only in default plan**	**Rules only in best plan**
Q_{A1}	−90%	JoinImpl2 SelectOnProject GroupbyBelowUnionAll ...8 more rules	CorrelatedJoinOn -UnionAll2
Q_{A2}	−86%	HashJoinImpl1 SelectOnProject SelectPredNormalized ...3 more rules	-
Q_{A3}	−75%	UnionAllToUnionAll	UnionAlltoVirtual -Dataset
Q_{B1}	−96%	TopOnRestrRemap SelectOnTrue	CollapseSelects
Q_{B2}	−80%	JoinImpl2	HashJoinImpl1
Q_{B3}	−70%	ProcesOnnUnionAll UnionAllToUnionAll	UnionAlltoVirtual -Dataset

- How to extend to unseen jobs? Learning!

Rule Configuration Learning

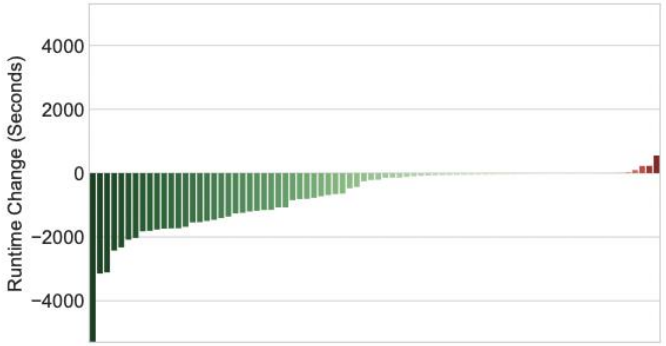
- Job group: all jobs having the same default rule configuration
  - Similar code paths to QO, could be optimized together

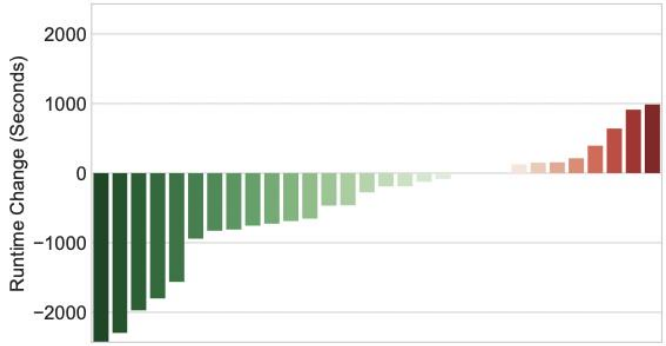
- Regression learning problem
  - Input features: job graph features + candidate rule configuration
  - Output: normalized runtime
  - Lightweight model with 1-layer NN

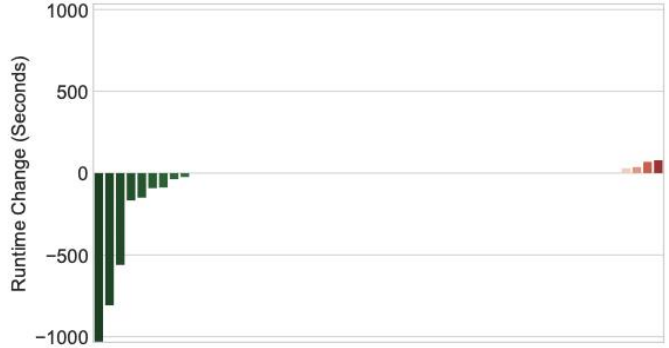
Rule Configuration Learning

- Online deployment: large improvements, small regressions

	1			2			3		
	Mean	90P	99P	Mean	90P	99P	Mean	90P	99P
Best	5458	14K	14.8K	19.8K	26K	27K	2966	13.8K	15.3K
Default	6461	16.3K	18.3K	20.7K	26.9K	28.9K	3304	14.7K	16.8K
Learned	**5724**	**14.7K**	**15.4K**	**20.2K**	**26.2K**	**27K**	**3252**	**14.6K**	16.8K







Actual Deployment: Challenges

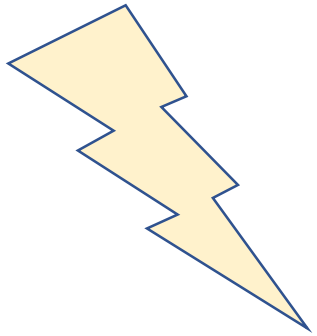
- ML +DBs sounds great, but...
  - It can be challenging to deploy in a real world production setting
  - Even more so in mission critical systems such as DBs!

- Key Issues
  - Brittleness of most models, hard to detect stochastic failure modes, model training is not a well-defined process
  - Dependencies (ML stacks, external services, ...)
  - Keeping track of training data and model versions

DBs and ML: A Tale of Two Cities

Databases

- Central and fundamental piece of IT infrastructure for almost any business
- Need rock-solid, reliable and stable behavior
- Deterministic



Machine Learning

- Produces predictions based on probabilistic methods
- Needs close supervision due to wide range of failure modes
- Stochastic

Need to resolve conflict through proper software engineering practices!

→ Requirements, design iterations, …

Deriving Requirements

High Level Design Philosophy

- Separation from the core system
- Minimal third party dependencies
- Robustness, stability and fault tolerance
- Usability and configurability.

Concrete ML-related requirements

- Support wide range of models and ML frameworks while not introducing too many dependencies
- Training outside of the core system, support iterative model development and evaluation process
- Well defined and robust model deployment procedure
- Fallbacks in case of model failures

Different User Perspectives

Actual User (both human and technical)

- Should not have to care about this

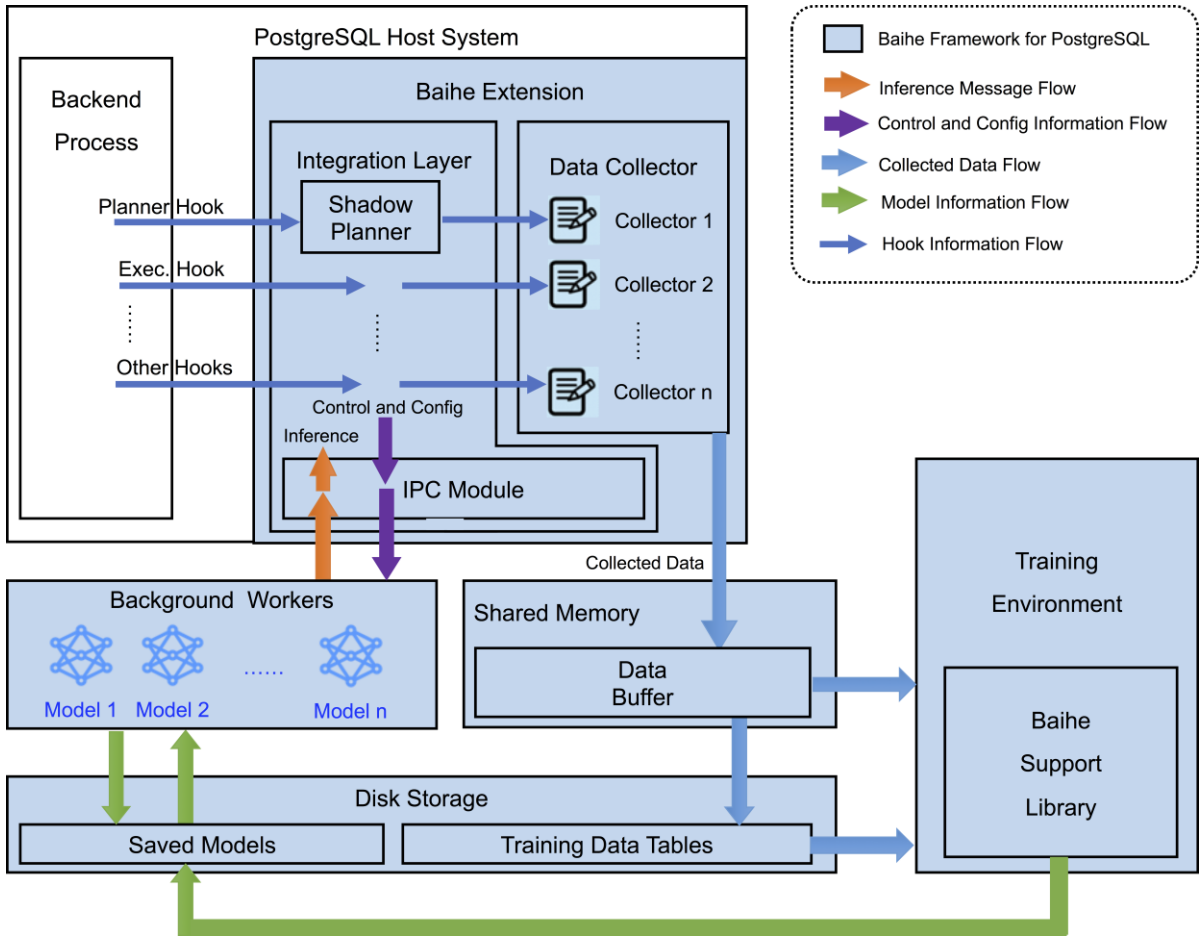
DBAs/OPs

- Should only have to learn a few config knobs and management commands (e.g. activate training data collection, configure which model to use, deploy a readily packaged model,…)
- Should not be responsible for managing extra external services which provide some sort of inference API
- Should not have to worry that some fancy new piece of tech breaks their system or security.

Model Developer / Researcher

- Easy access to training data, fast development iteration cycles
- Simple packaging and deployment (resp. handover to DBA)
- Develop models based on well defined environment (libs, packages, …) close to the usual ML/DS stack

Baihe: Design Blue Print and Implementation



Highlights

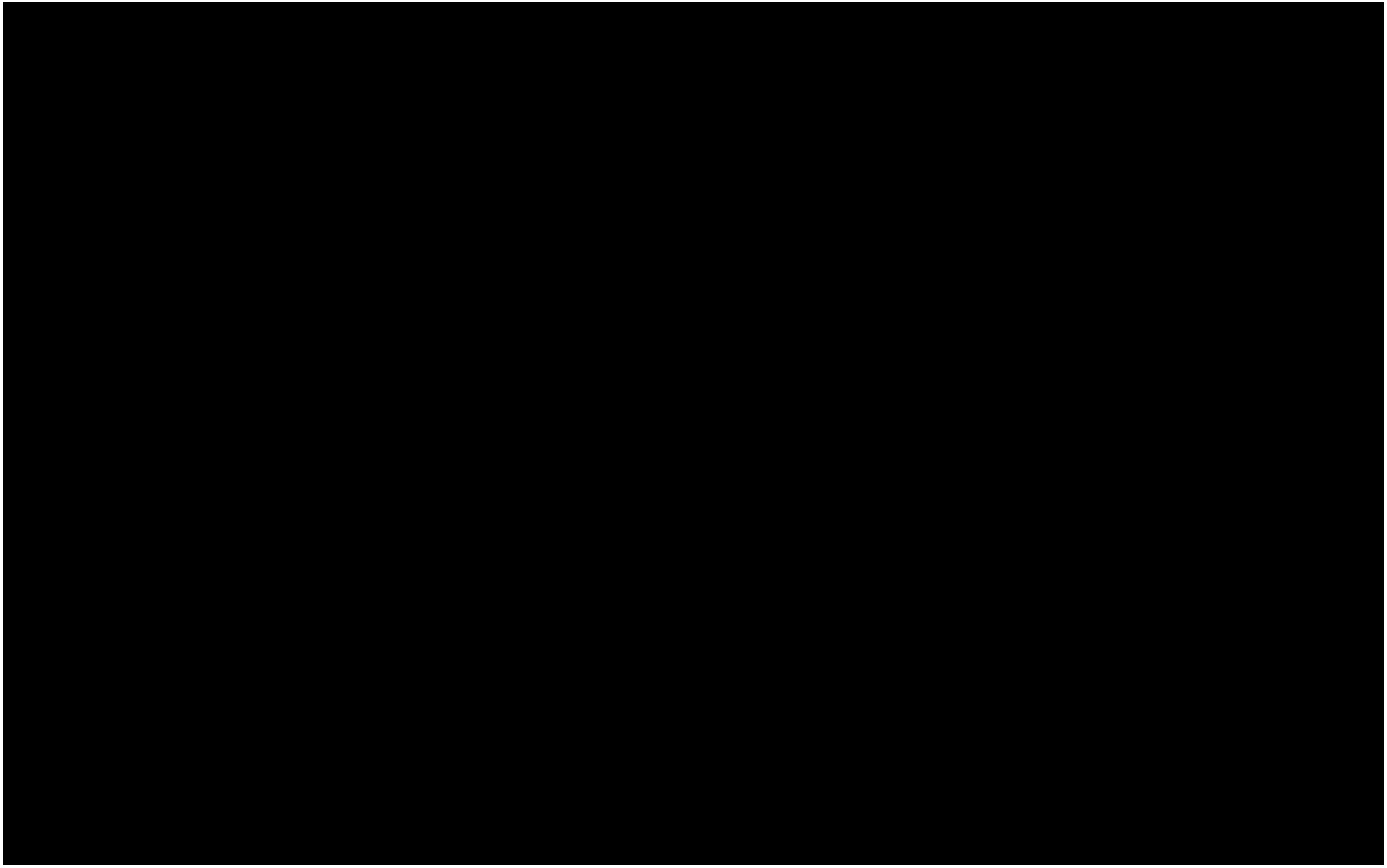
- Extend Postgres hooking mechanism through shadow planner component
- Make idiomatic use of Postgres functionality for shared memory, background workers, process management and IPC
- Simple model packaging and deployment for cardinality estimation and query runtime prediction
- Easy to use test bed for Ai4DB related research

Open Source Release

- Research oriented MVP: Available soon!
- Comments and discussions welcome!

Demo Time!

- Andreas Pfadler, Rong Zhu, Wei Chen, Botong Huang, Tianjing Zeng, Bolin Ding, Jingren Zhou. *Baihe: SysML Framework for AI-driven Databases*, *Arxiv*, 2022. https://arxiv.org/abs/2112.14460



Applications and Deployment: Summary

- Some attempts on customized systems and tasks
  - Learned cost model on SCOPE
  - Learn to steer QO on SCOPE
- General AI4DB deployment tool is very necessary
  - Challenges but also opportunities
- Our Baihe system takes the first step

Part 5: Summary and Future Work

- ❑ Summary
- ❑ Future Work

Summary

- QO is a suitable experimental plot for AI4DB → the prosperity of learned QO techniques in recent

- Learned QO components: CardEst, Cost Model, JoinSel
  - CardEst: data-driven, query-driven and bound-based methods
  - Cost Model: supervised learning
  - JoinSel: RL-based learning
  - Exhibit advantages, but still not ready for actual deployment

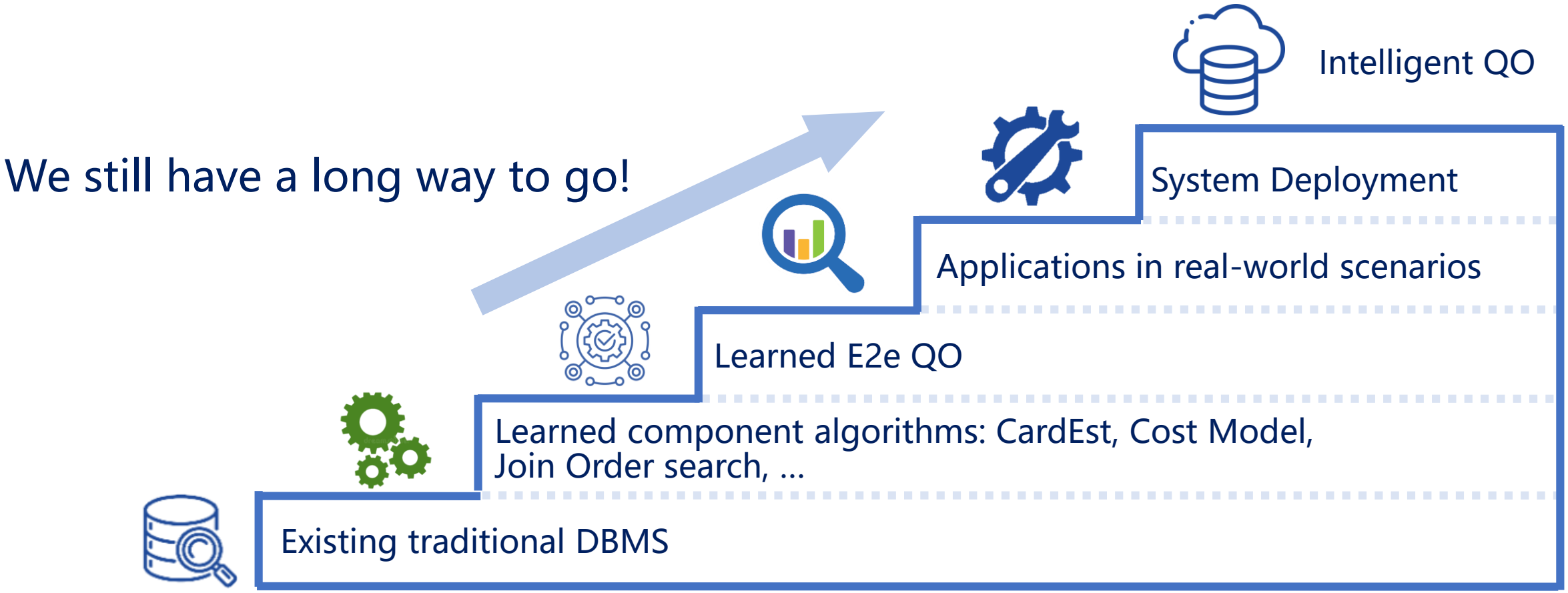
Summary

- Learned whole QO module
  - Learn to replace original QO vs. learn to steer existing QO
  - Far from a desirable learned QO
  - Some possible way is identified

- Applications and deployment
  - Some attempts on customized systems and tasks
  - General deployment tool is very necessary
  - The first step is already taken

Future Work

- Final goals: practical and intelligent QO and beyond



Future Work

- CardEst
  - Fusion of data-driven and query-driven methods
  - Adaptive methods: OLAP/OLTP, different data/workload,…
  - Performance improvement: update speed, accuracy on multi-table queries, aware of different sub-queries, …
- Cost Model and JoinSel
  - Automatically generate sufficient training data with large coverage
  - Robust model for dynamic workloads or different scenarios
  - Intelligent algorithms selection given a workload and datasets

Future Work

- Learned QO module
  - Pre-training + fine-tuning technique routine
  - New architecture to steer existing QO
  - New training and update strategy
- Applications and Deployment
  - General deployment tool
  - Customized tuning
- Beyond QO: extend to more AI4DB or even DB4AI tasks
  - Indexing, advisors, diagnosis, …

Q & A

Rong Zhu
Alibaba Group

Ziniu Wu
MIT

Chengliang Chai
Tsinghua University

Andreas Pfadler
Alibaba Group

Bolin Ding
Alibaba Group

Guoliang Li
Tsinghua University

Jingren Zhou
Alibaba Group

Contact:

red.zr@alibaba-inc.com
ziniuw@mit.edu
ccl@tsinghua.edu.cn