NANYANG TECHNOLOGICAL UNIVERSITY

SIGMOD
PODS
2024

Machine Learning for Databases: Foundations, Paradigms, and Open problems

Gao Cong, Jingyi Yang , Yue Zhao

Nanyang Technological University

Machine Learning for Databases (ML4DB)

- Machine learning techniques have been used to improve various aspects of a data management system
  - Core components: index, data layout, query optimizer, scheduler…
  - Advisors: knob tuning, index/view advisor, performance prediction…
  - Usability: text2SQL

- Existing tutorials on ML4DB
  - Survey ML solutions for different database problems
  - Focus on ML solutions for a particular problem

Tutorial Overview

- ML4DB Foundations (1 hour 15 mins)
  What are the key components for foundation in ML4DB?
- ML4DB Paradigms (1 hour 15 mins)
  “Replacement” approach vs. “ML-enhanced” approach
- ML4DB Open problems (30 mins)
  What are the open problems to be solved?

Tutorial Overview

- ML4DB Foundations
  - Query Plan Representation
  - Pretrained Models
- ML4DB Paradigms
  - Replace vs Enhance
  - Learned Index
  - Learned Optimizer
- ML4DB Open problems
  - What are the foundations of ML4DB?
  - Model efficiency
  - Generating training data of high quality
  - Handling data & workload shifts
  - Foundation models (Pretrained models) for Ml4DB
  - LLMs for ML4DB

ML4DB Foundations

- Can we have some foundations that are relevant to different ML4DB systems?

- We identify two key foundations
  - Query plan representation
  - Pretrained models

ML4DB Foundations

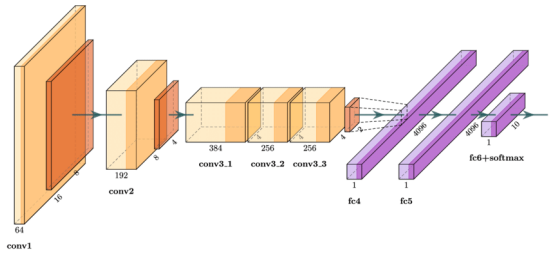
- Can we have some foundations that are relevant to different ML4DB systems?

- We identify two key foundations
  - Query plan representation
  - Pretrained models

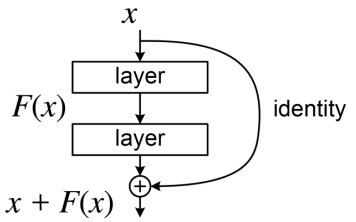
Foundations in other domains

- **Computer Vision**: as **input representation techniques** improve, all task performances get better.

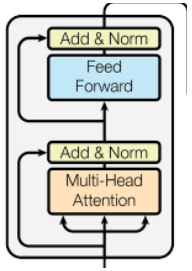
Input representation techniques



AlexNet (2012)



ResNet (2015)



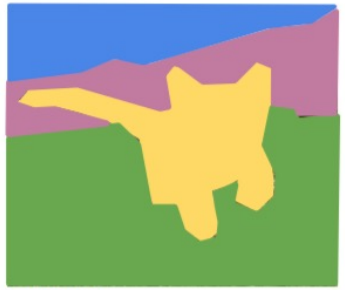
Vision Transformer (2020)

• • •

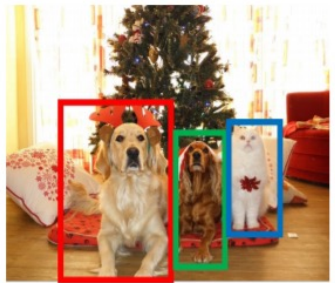
Apply to

Computer vision tasks

Classification



Semantic Segmentation



Object Detection

Instance Segmentation

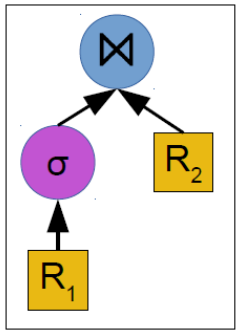
• • •

Picture from http://cs231n.Stanford.edu/slides/2020/lecture_12.pdf

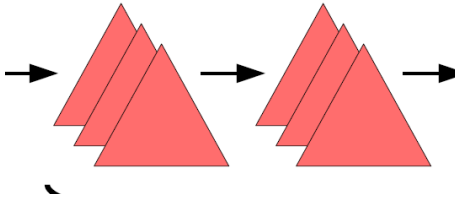
ML4DB Analogy

- Query plan representation techniques can be applied to various database tasks

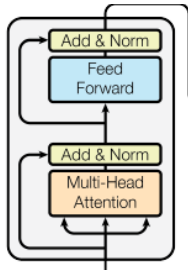
Query plan representation techniques



QPPNet



TreeCNN



Transformers

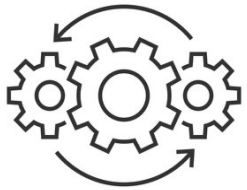
• • •

Apply to

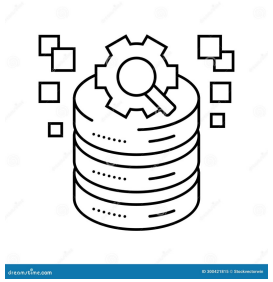
Database tasks

Index Tuner

Cost Estimator



View advisor



Optimizer

• • •

Physical query plans

- A query plan is a sequence of steps used to access data in a relational database management system.

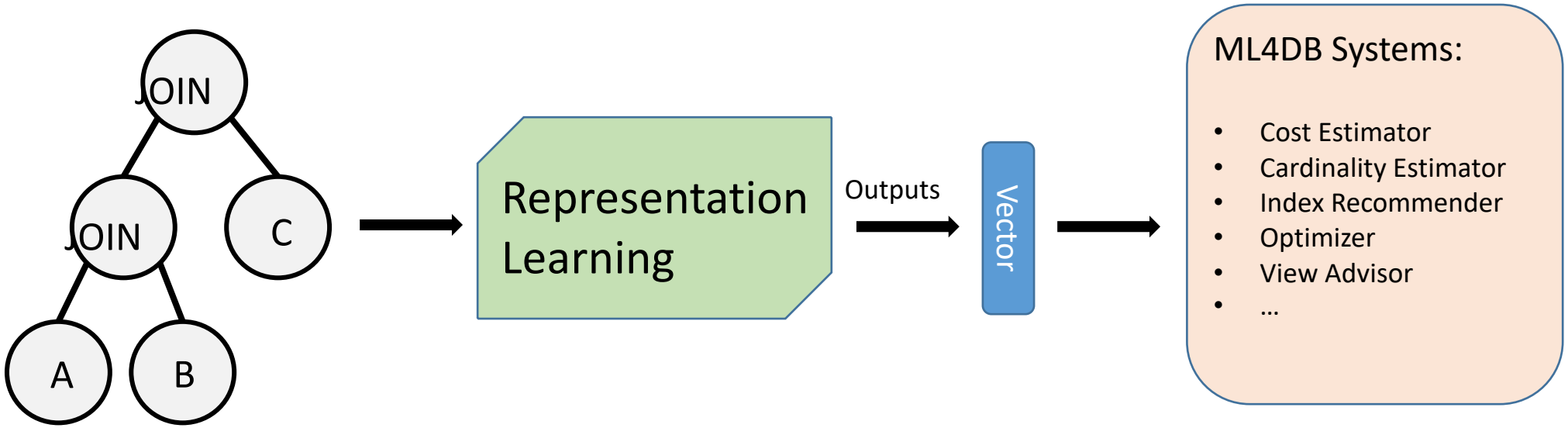
```
EXPLAIN SELECT *
FROM tenk1 t1, tenk t2
WHERE t1.unique1 < 100 AND
t1.unique2 = t2.unique2;
```

```
Merge Join  (cost=198.11..268.19 rows=10 width=488)
  Merge Cond: (t1.unique2 = t2.unique2)
  ->  Index Scan using tenk1_unique2 on tenk1 t1  (cost=0.29..656.28 rows=101 width=244)
        Filter: (unique1 < 100)
  ->  Sort  (cost=197.83..200.33 rows=1000 width=244)
        Sort Key: t2.unique2
        ->  Seq Scan on tenk t2  (cost=0.00..148.00 rows=1000 width=244)
```

Query plan from Postgres. https://www.postgresql.org/docs/current/using-explain.html.

Query Plan Representation

- **Query Plans** are used as inputs in many ML4DB systems
- **Query plan representation** is a key operation
- Research Problem: Given a *query plan*, learn a vector representation to be used as the input to a ML4DB system

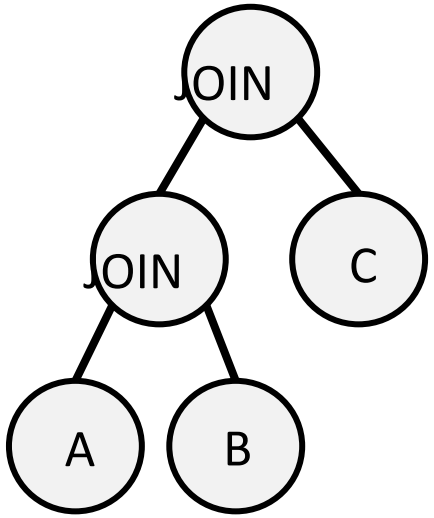


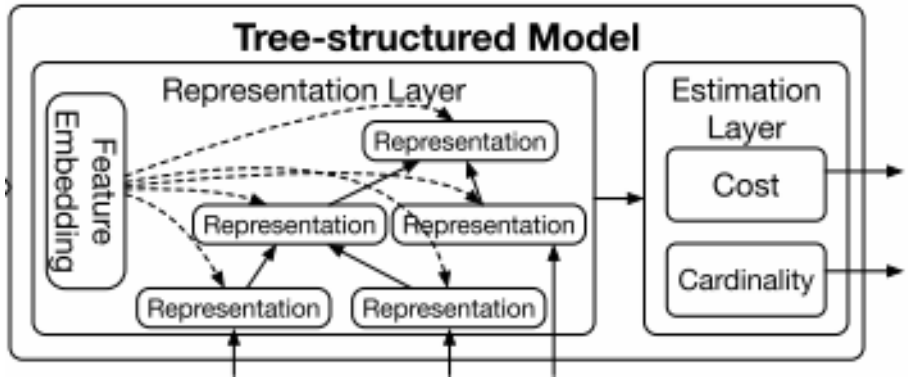
Zhao Y, et al. QueryFormer: a tree transformer model for query plan representation. VLDB 2022.

Example: Cost Estimation

- Cost and Cardinality Estimation
  - Uses Tree-LSTM to extract feature representation from a *query plan*
  - Uses MLP to predict cost and cardinality

Model Input





Model Output

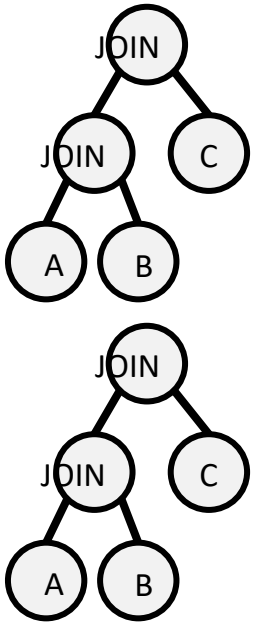
Est. Cost: 123 ms
Est. Cardinality: 12345

Ji Sun, Guoliang Li. An End-to-End Learning-based Cost Estimator. VLDB 2019.

Example: Index Recommendation

- Index Recommendation
  - Featurize a *query plan* by creating feature channels for each physical operator
  - Perform classification on *query plan* pairs

Model Input



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

Seek_Row_Serial	30
Scan_Row_Serial	30
HJ_Row_Serial	55
NLJ_Row_Serial	0
MJ_Row_Serial	0
...	...

EstNodeCost (P_1)

EstNodeCost ($P_2 - P_1$)

Seek_Row_Serial	20
Scan_Row_Serial	-50
HJ_Row_Serial	0
NLJ_Row_Serial	0
MJ_Row_Serial	0
...	...

EstNodeCost (P_2)

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

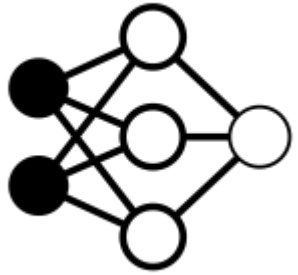
Seek_Row_Serial	1200
Scan_Row_Serial	1000
HJ_Row_Serial	6200
NLJ_Row_Serial	0
MJ_Row_Serial	0
...	...

LeafWeightEst RowsWeighted Sum (P_1)

LeafWeightEstRows WeightedSum (P_2 - P_1)

Seek_Row_Serial	1000
Scan_Row_Serial	-1000
HJ_Row_Serial	1600
NLJ_Row_Serial	0
MJ_Row_Serial	0
...	...

LeafWeightEst RowsWeighted Sum (P_2)



Classifier

Model Output

First plan is better

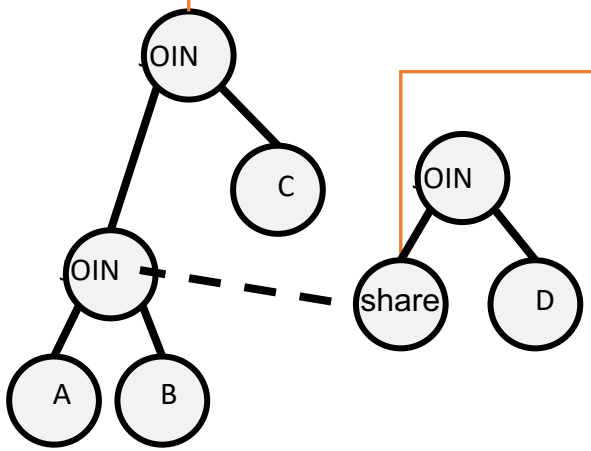
First index is better

Ding, B, et al. AI Meets AI: Leveraging Query Executions to Improve Index Recommendations. Sigmod 2019.

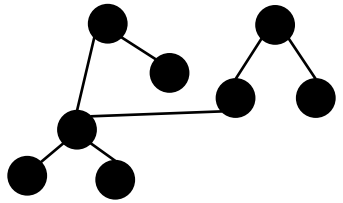
Example： Concurrent query optimization

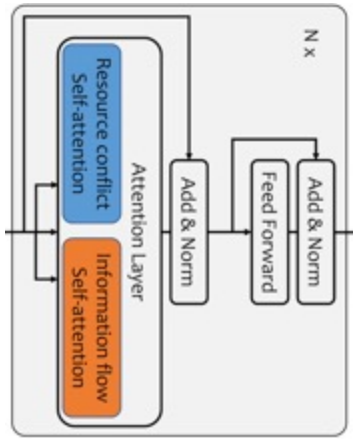
- Concurrent query optimization
  - Featurize *multiple (partial) query plans*
  - Featurize *SQL query* by extracting join graph and predicate information
  - Predict the cost for plan generation

Model Input



Join	Scan	Share	A	B	…
1	0	0	1	1	…
0	0	1	1	1	…
…	…	…	…	…	…





Cost estimation model

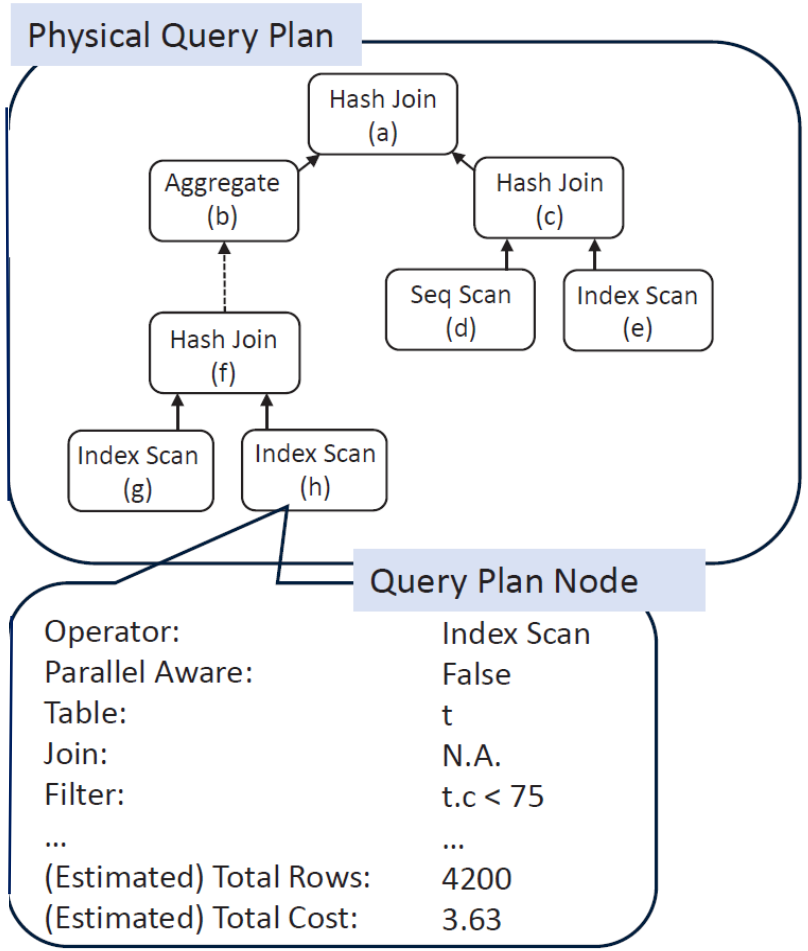
Model Output

cost

Mo, S, et al. Lemo: A Cache-Enhanced Learned Optimizer for Concurrent Queries. SIGMOD 2024

Query Plan Representation as a Foundation

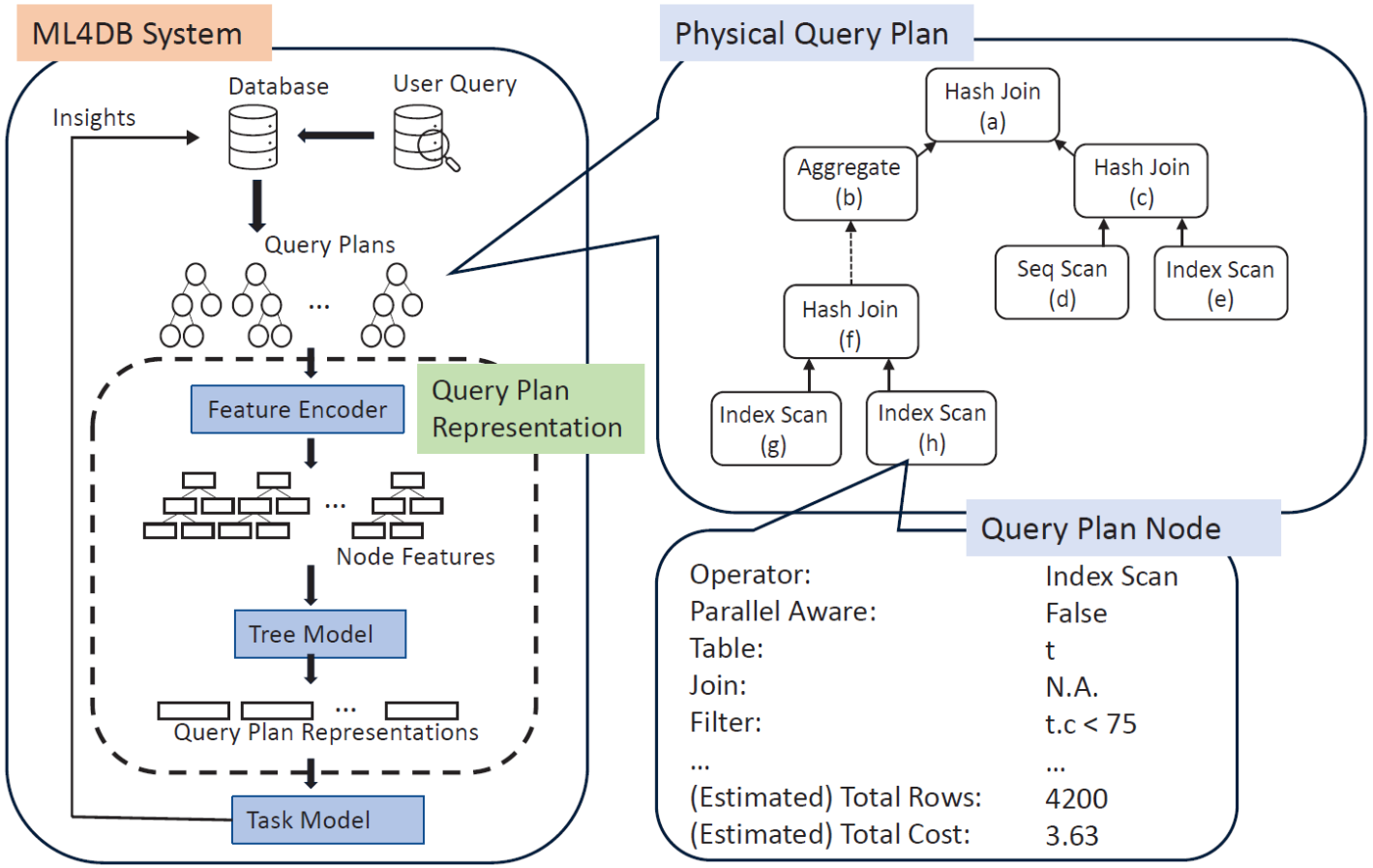
- Why is representation learning important?
  - Non-trivial to define features from a query plan
  - Difficult to deal with the tree structure of a query plan
  - A key factor to the performance of all these tasks



Zhao, Y, et al. A Comparative Study and Component Analysis of Query Plan Representation Techniques in ML4DB Studies. VLDB 2024.

Query Plan Representation in ML4DB Pipeline

- Can be broadly broken down to two components
  - Feature Encoder
  - Tree Model



Zhao, Y, et al. A Comparative Study and Component Analysis of Query Plan Representation Techniques in ML4DB Studies. VLDB 2024.

Query Plan Representation in ML4DB Pipeline

- Tree Models: *LSTM, TreeCNN, Transformer, TreeLSTM*, etc.
- Feature Encoding: *Operator, Predicate, Table*, Various *Database statistics*.

A summary of query plan representation methods and their components.

Method	Original Task	Tree Model	Encoding Information			
			Operator	Table	Predicate	Database Statistics
AVGDL [51]	View Selection	LSTM	Yes	Yes	Yes	-
AIMeetsAI [8]	Index Selection	Feature Vector	Yes	-	-	Estimates
ReJOIN [30]	Join Order Selection	Feature Vector	Yes	Yes	Column	-
BAO [28]	Optimizer	TreeCNN	Yes	-	-	Estimates
NEO [29]	Optimizer	TreeCNN	Yes	Yes	Column	Estimates
Prestroid [17]	Cost Estimation	TreeCNN	Yes	Yes	Yes	-
E2E-Cost [39]	Cost/Card Estimation	TreeLSTM	Yes	Yes	Yes	Sample
RTOS [50]	Join Order Selection	TreeLSTM	Yes	Yes	Yes	-
Plan-Cost [31]	Cost Estimation	TreeRNN	Yes	-	-	Estimates
QueryFormer [54]	General Purpose	Transformer	Yes	Yes	Yes	Sample & Histogram

Zhao, Y, et al. A Comparative Study and Component Analysis of Query Plan Representation Techniques in ML4DB Studies. VLDB 2024.

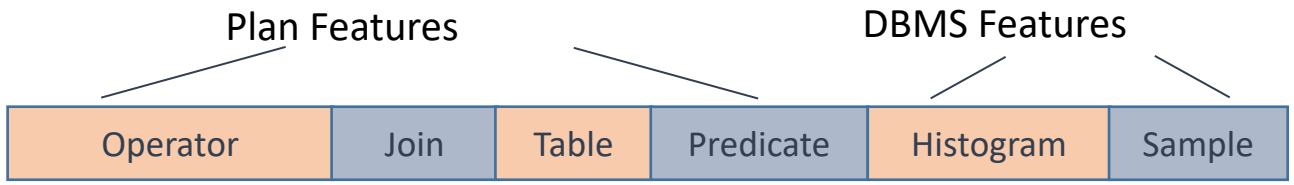
Feature Encoding

- Challenge:
  - How to select features to encode from a query plan node?

	Content	Example	Encoding
Plan Features	Operator	hash scan	One-hot / Embedding
	Table	title as t	One-hot / Embedding
	Column	t.production_year	One-hot / Embedding
	Predicate	t.production_year > 1965	<col, op, val> triplet
	Join	t.id = mc.movie_id	One-hot / Embedding
DBMS enhanced	Database Estimates	est. Rows: 400	Normalized value
	Samples	[1,0,0,0,1,0,0,1,0]	Bitmap
	Histogram	[0, 0, 0, 0, 0, 0, 1, 1, 1]	Bitmap/ Normalized value
	…	…	…

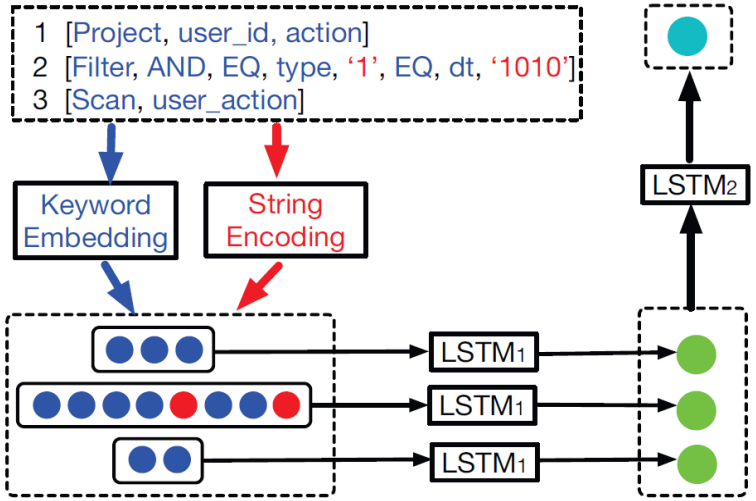
Feature Encoding

- How to combine the features?
  - Concatenate the features (most Popular)



  - Feature Channels [Ding, B, et al. SIGMOD 19]

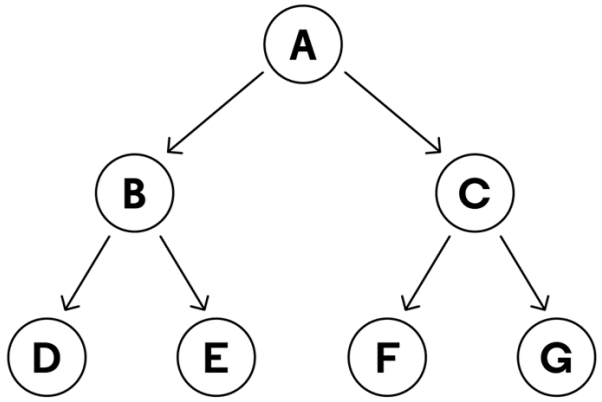
Seek_Row_Serial	10
Scan_Row_Serial	80
HJ_Row_Serial	55
NLJ_Row_Serial	0
MJ_Row_Serial	0
...	...

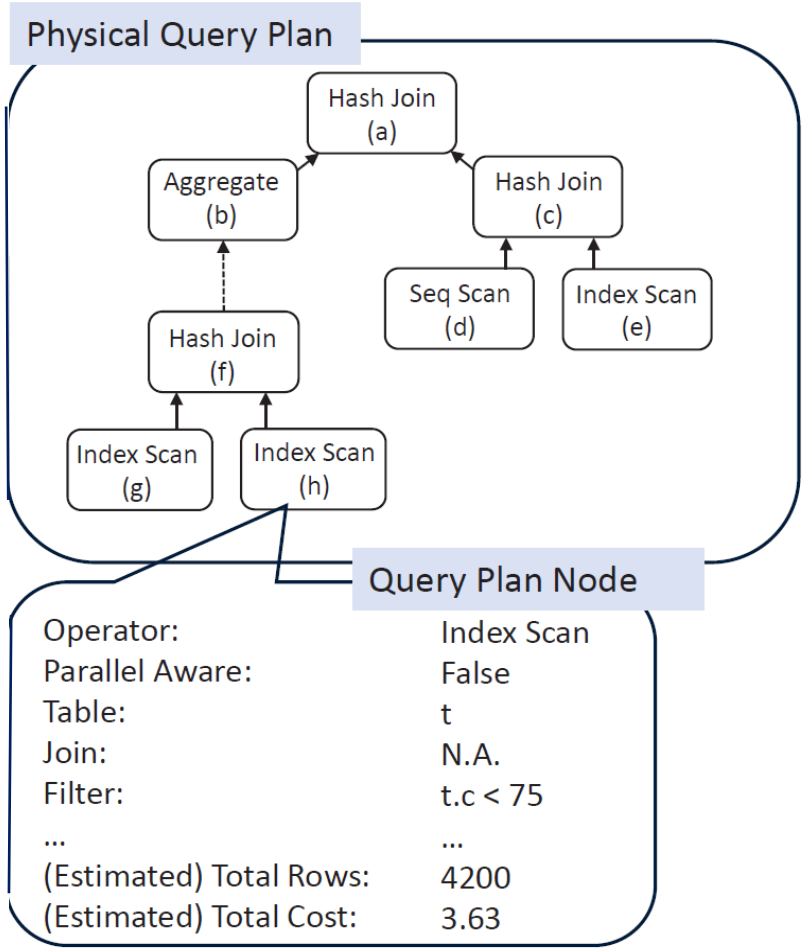


  - Deep Learning Models, i.e., LSTM, to join the encoded components [Yuan, H, et al. ICDE 2020]

Tree Models

- How to handle the tree structure of query plans?
  - Directional dependence
  - Irregular shapes

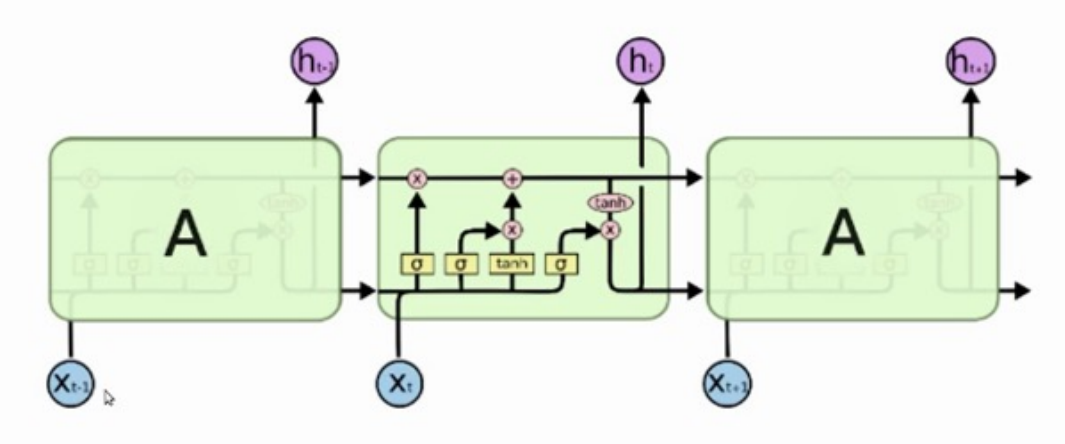




Flat sequence model, e.g., LSTM

- LSTM
  - Using gates to solve ‘gradient vanishing’ and ‘gradient explosion’

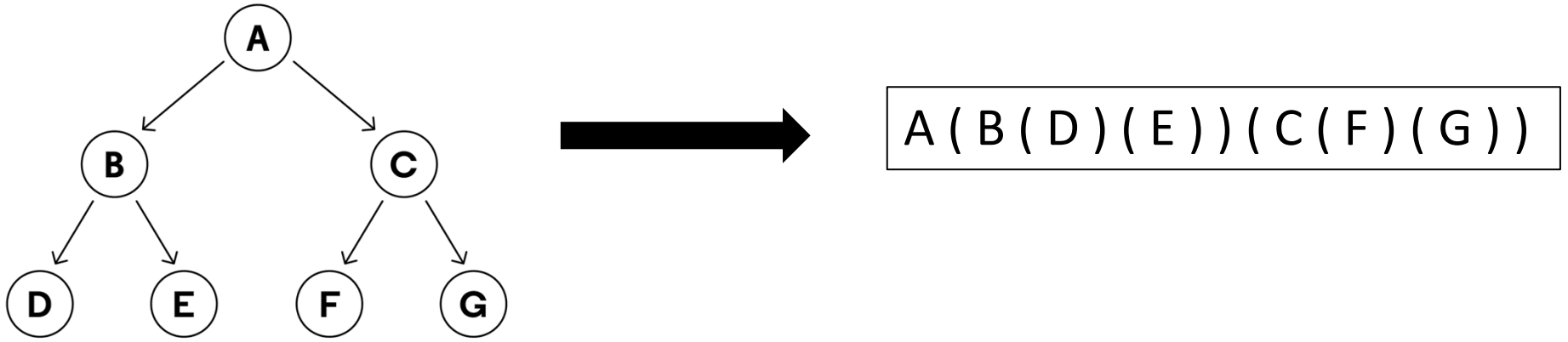
$$
\begin{aligned}
i_t &= \sigma\left(W^{(i)}x_t + U^{(i)}h_{t-1} + b^{(i)}\right), \\
f_t &= \sigma\left(W^{(f)}x_t + U^{(f)}h_{t-1} + b^{(f)}\right), \\
o_t &= \sigma\left(W^{(o)}x_t + U^{(o)}h_{t-1} + b^{(o)}\right), \\
u_t &= \tanh\left(W^{(u)}x_t + U^{(u)}h_{t-1} + b^{(u)}\right), \\
c_t &= i_t \odot u_t + f_t \odot c_{t-1}, \\
h_t &= o_t \odot \tanh(c_t),
\end{aligned}
$$



Hochreiter, S and Schmidhuber J. Long Short-Term Memory (LSTM). Neural Computation 1995.

Flat sequence model, e.g., LSTM

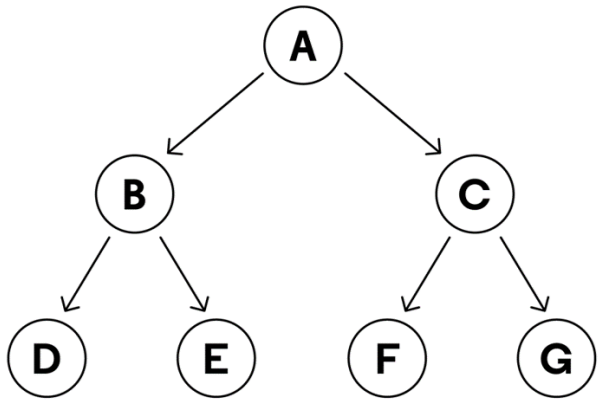
- Step 1: DFS to convert the tree to sequence
- Step 2: Apply LSTM to obtain output representation



Hochreiter, S and Schmidhuber J. Long Short-Term Memory (LSTM). Neural Computation 1995.

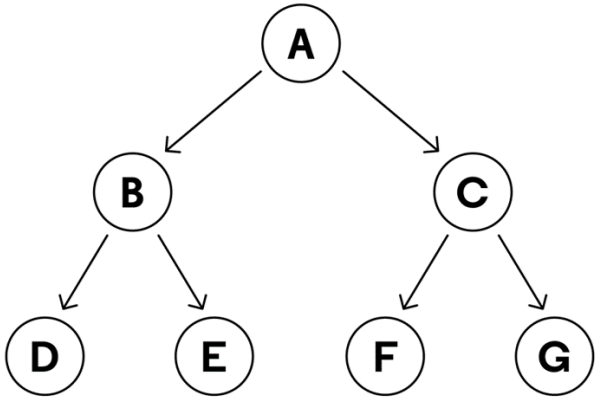
Tree RNN

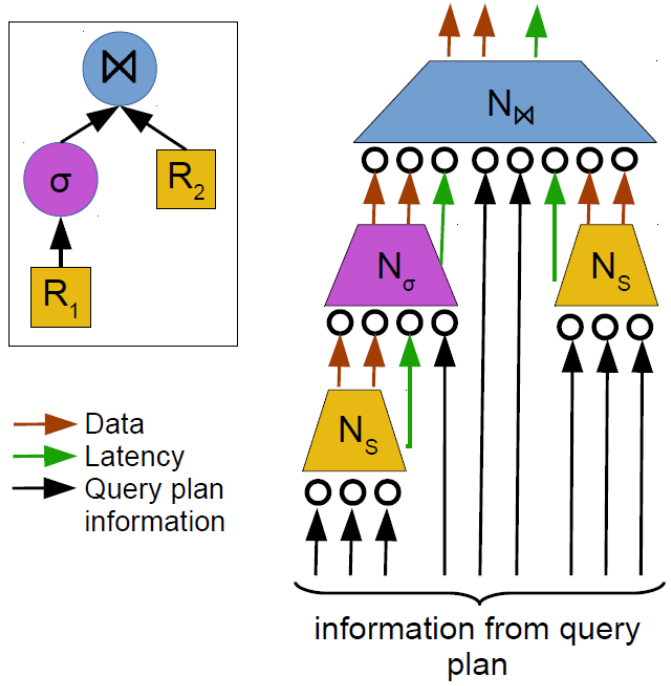
- Can we pass information bottom-up, just like how a query plan is processed?



Tree RNN

- Can we pass information bottom-up, just like how a query plan is processed?
  - Tree Structured RNN model
  - Pass the information regardless of tree shape

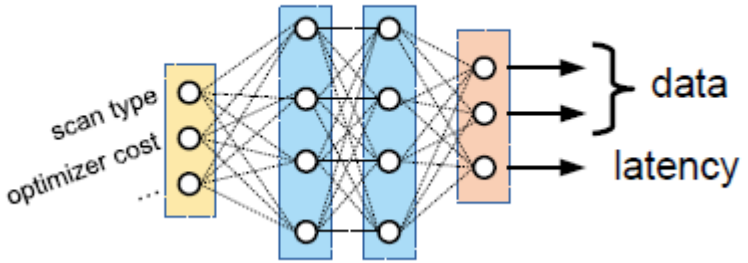


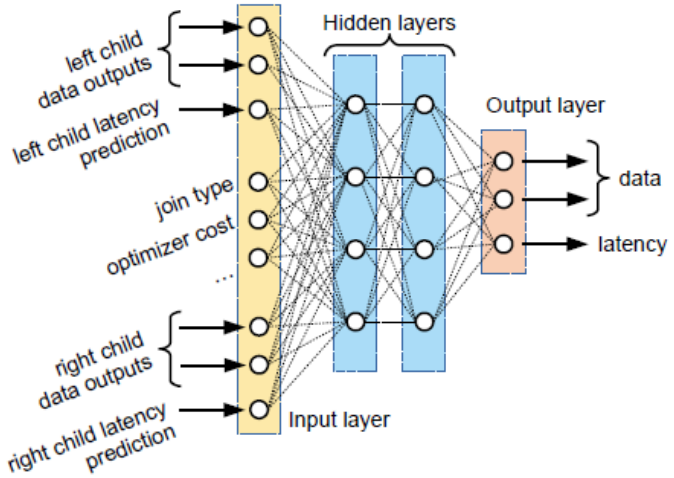


Ryan Marcus, Olga Papaemmanouil. Plan-Structured Deep Neural Network Models for Query Performance Prediction. VLDB 2019.

Tree RNN

- Two novel neural units:
  - Scan
  - Join





- How to perform batching?
  - Group query plans of the same shape

Ryan Marcus, Olga Papaemmanouil. Plan-Structured Deep Neural Network Models for Query Performance Prediction. VLDB 2019.

Tree LSTM

- Can we apply some advanced ML technique to Tree RNN setup?

Tree LSTM

- Can we apply some advanced ML technique to Tree RNN setup?
  - Use LSTM-cell at each step
  - First explored in NLP
  - Child-sum vs N-ary Tree LSTM

$$\hat{h}_j = \sum_{k \in C(j)} h_k$$

$$i_j = \sigma(W^{(i)} x_j + U^{(i)} \hat{h}_j + b^{(i)})$$

$$f_{jk} = \sigma(W^{(f)} x_j + U^{(f)} h_k + b^{(f)})$$

$$o_j = \sigma(W^{(o)} x_j + U^{(o)} \hat{h}_j + b^{(o)})$$

$$u_j = \tanh(W^{(u)} x_j + U^{(u)} \hat{h}_j + b^{(u)})$$

$$C_j = i_j \odot u_j + \sum_{k \in C(j)} f_{jk} \odot c_k$$

$$h_j = o_j \odot \tanh(c_j)$$

$$i_j = \sigma\left(W^{(i)} x_j + \sum_{\ell=1}^{N} U_\ell^{(i)} h_{j\ell} + b^{(i)}\right),$$

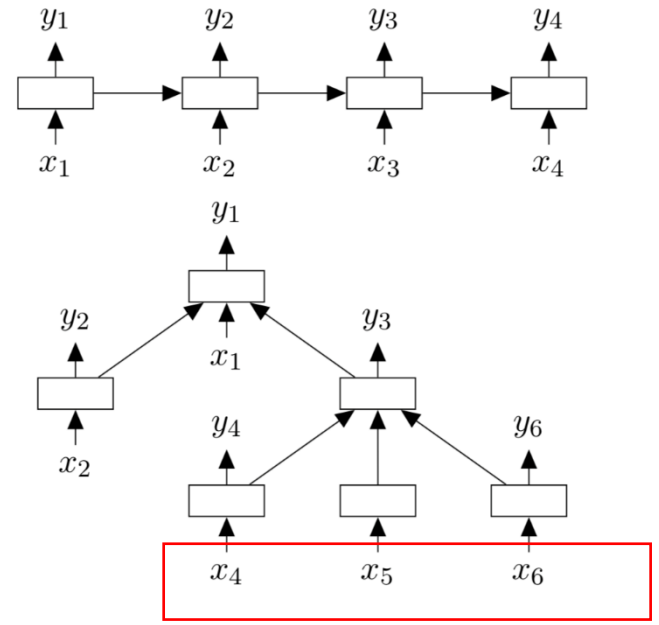
$$f_{jk} = \sigma\left(W^{(f)} x_j + \sum_{\ell=1}^{N} U_{k\ell}^{(f)} h_{j\ell} + b^{(f)}\right),$$

$$o_j = \sigma\left(W^{(o)} x_j + \sum_{\ell=1}^{N} U_\ell^{(o)} h_{j\ell} + b^{(o)}\right),$$

$$u_j = \tanh\left(W^{(u)} x_j + \sum_{\ell=1}^{N} U_\ell^{(u)} h_{j\ell} + b^{(u)}\right),$$

$$c_j = i_j \odot u_j + \sum_{\ell=1}^{N} f_{j\ell} \odot c_{j\ell},$$

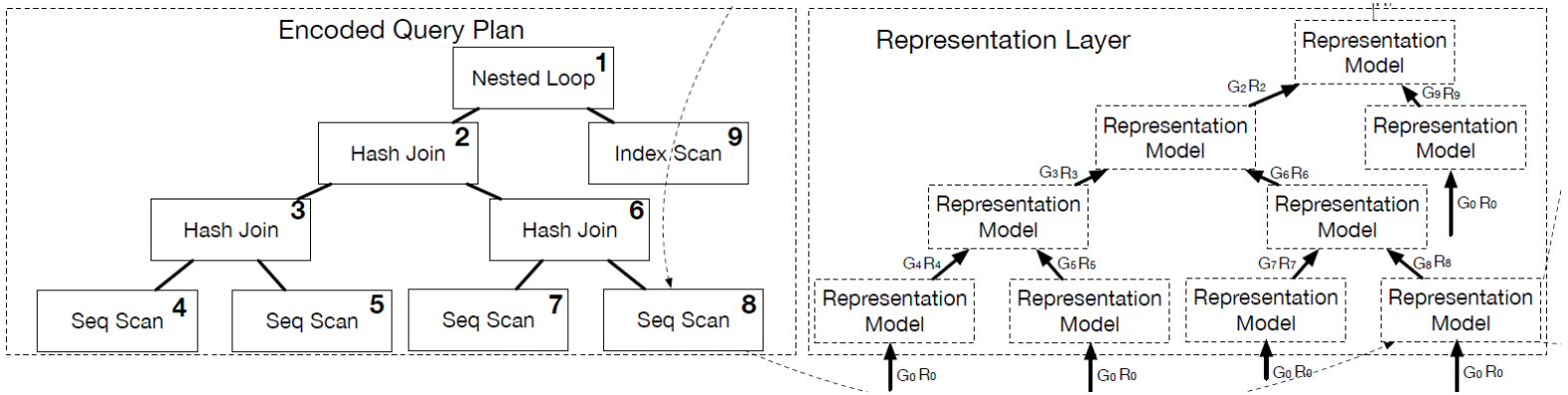
$$h_j = o_j \odot \tanh(c_j),$$



Tai, K S, et, al. Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks. ACL 2015.

Tree LSTM

- Apply Tree LSTM to query plan representation
  - Traverse the tree bottom up
  - Apply the LSTM cell at each step

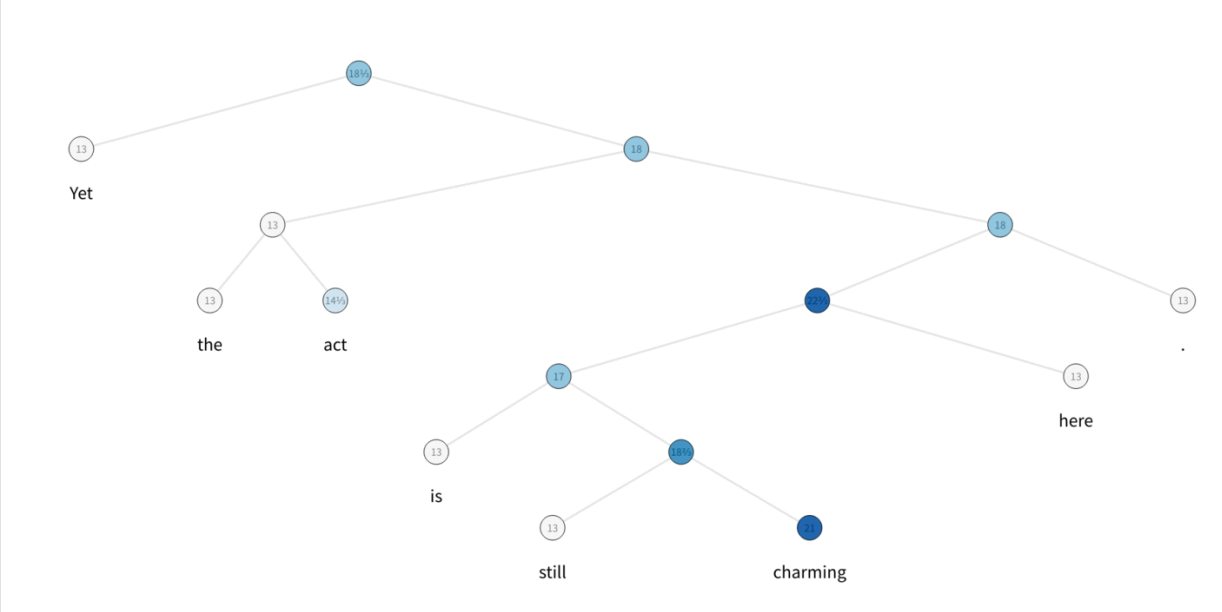


- How do we do batching now?

Ji Sun, Guoliang Li. An End-to-End Learning-based Cost Estimator. VLDB 2019.

Tree LSTM Batching

- Parent Node depends on children node results
  - Assign 'height' of nodes to the forest (batch of query plan trees)
  - Process nodes with height=0 (scan nodes)
  - Process nodes with height=1
  - ..

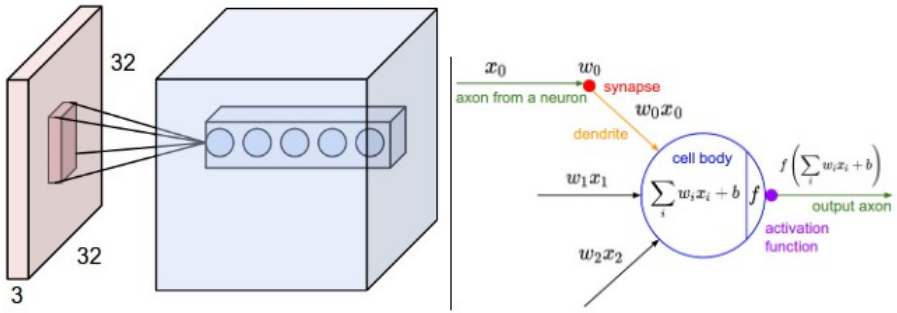


Picture from dgl documentation. https://docs.dgl.ai/tutorials/models/2_small_graph/3_tree-lstm.html

Ji Sun, Guoliang Li. An End-to-End Learning-based Cost Estimator. VLDB 2019.

Tree CNN

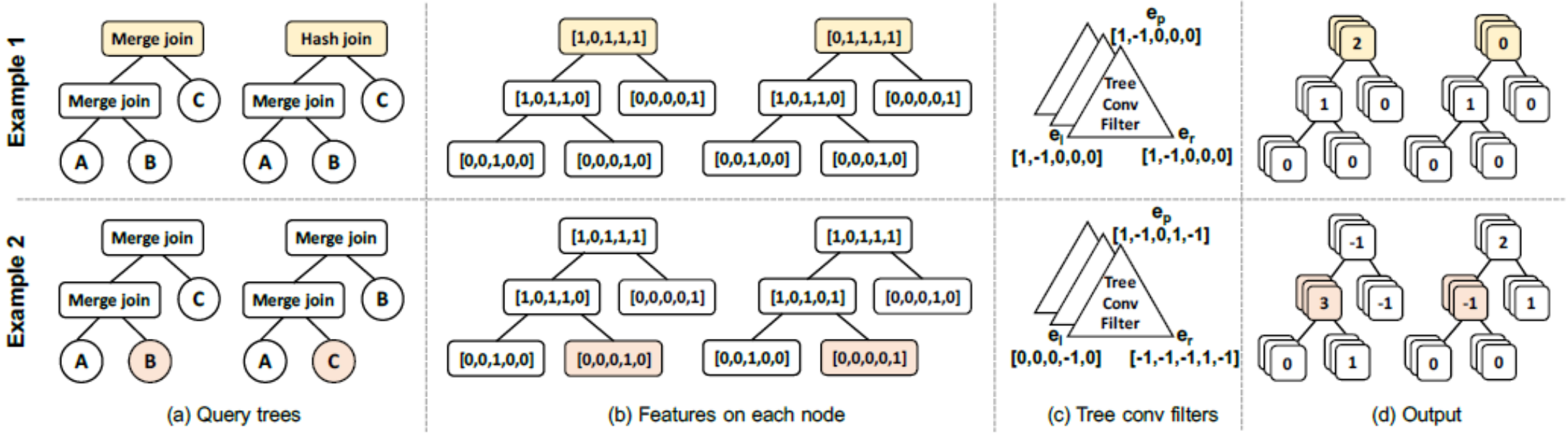
- Idea:
  - An extension to traditional CNN
  - Conv layers as pattern matching
    - Image: edges, shapes, etc.,
    - Query plans: nest loop over index scan, etc.,



Picture from https://cs231n.github.io/convolutional-networks/

Tree CNN

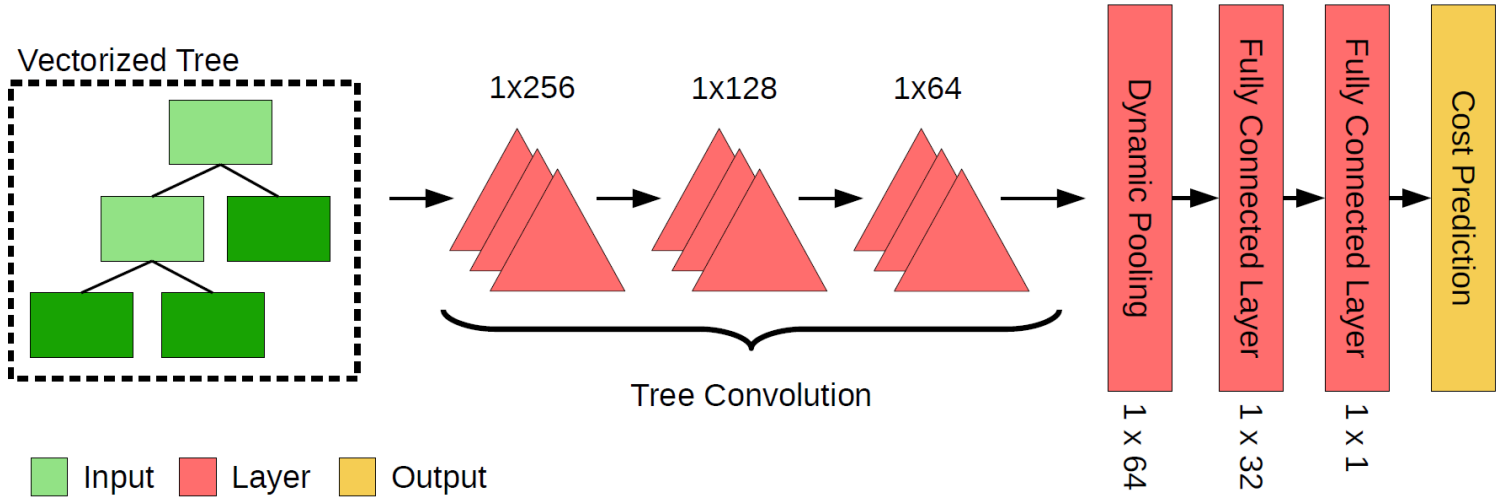
- Idea:
  - Triangular-shape (parent-child-child) convolution filters
  - ‘Slide’ over a query plan tree to match patterns, just like CNN to image



Marcus, R, et al. Neo: A Learned Query Optimizer. VLDB 2019.

Tree CNN

- Network Architecture
  - Stack of Tree Conv layers
  - Dynamic Pooling to unify output size

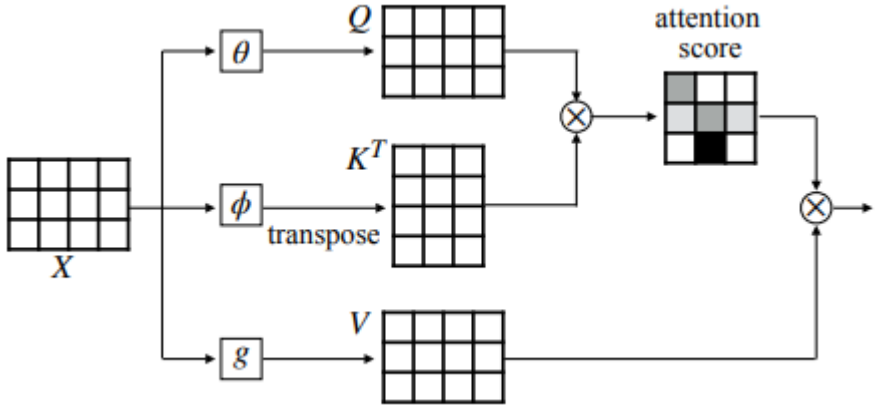


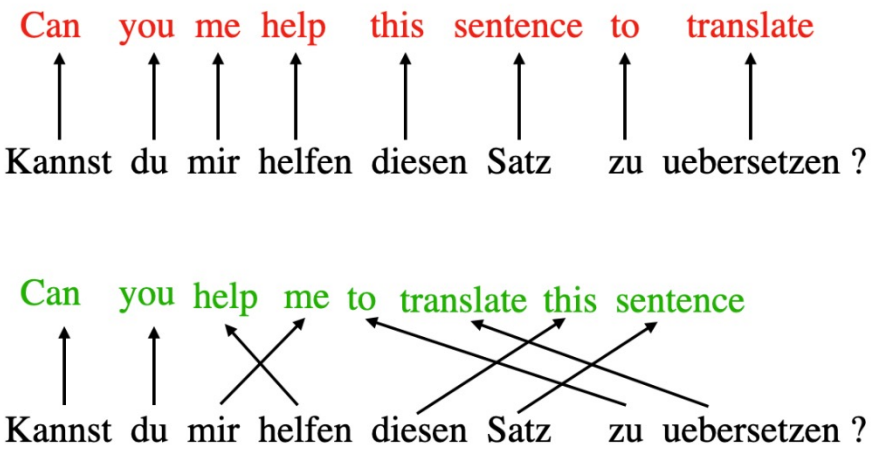
Marcus, R, et al. Neo: A Learned Query Optimizer. VLDB 2019.

Transformers

- Attention mechanism
  - To capture pair-wise dependency and interaction among elements

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V,$$

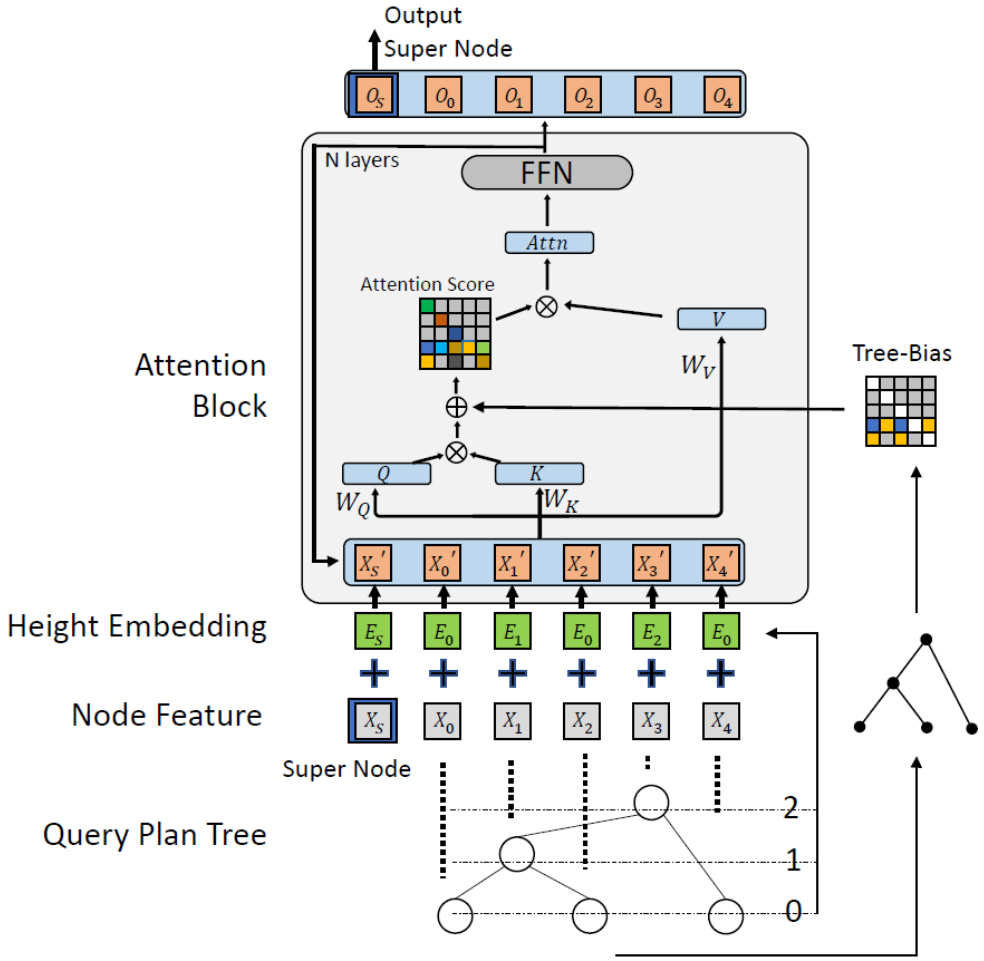




Picture from https://blogs.oracle.com/ai-and-datascience/post/multi-head-self-attention-in-nlp

QueryFormer

- Challenge
  - How to infuse the tree structure?
- Solution: Tree-biased attention
  - Bias the attention score by path distance d, specifically:
  - Contribution of j to node i: $A_{ij} = \frac{Q_i K_j^T}{\sqrt{d_k}}$
  - New attention: $A'_{ij} = A_{ij} + b_d$, where b_d is a learnable scalar, d is distance between i and j



Zhao, Y, et al, QueryFormer: a tree transformer model for query plan representation. VLDB 2022.

Summary

- A representation method: a combination of **tree model** and **feature encoding**.

A summary of query plan representation methods and their components.

Method	Original Task	Tree Model	Encoding Information			
			Operator	Table	Predicate	Database Statistics
AVGDL [51]	View Selection	LSTM	Yes	Yes	Yes	-
AIMeetsAI [8]	Index Selection	Feature Vector	Yes	-	-	Estimates
ReJOIN [30]	Join Order Selection	Feature Vector	Yes	Yes	Column	-
BAO [28]	Optimizer	TreeCNN	Yes	-	-	Estimates
NEO [29]	Optimizer	TreeCNN	Yes	Yes	Column	Estimates
Prestroid [17]	Cost Estimation	TreeCNN	Yes	Yes	Yes	-
E2E-Cost [39]	Cost/Card Estimation	TreeLSTM	Yes	Yes	Yes	Sample
RTOS [50]	Join Order Selection	TreeLSTM	Yes	Yes	Yes	-
Plan-Cost [31]	Cost Estimation	TreeRNN	Yes	-	-	Estimates
QueryFormer [54]	General Purpose	Transformer	Yes	Yes	Yes	Sample & Histogram

Zhao Y, et al, A Comparative Study and Component Analysis of Query Plan Representation Techniques in ML4DB Studies. VLDB 2024.

What to evaluate and how?

- Open Problems:
  - Are query plan representation methods interchangeable across tasks?
  - What is the best tree model design?
  - What is the best feature encoding?
- Experimental Methodology:
  - Interchange and compare query plan representation methods in 3 ML4DB systems
  - Fix feature encoding and compare tree models
  - Fix tree model and compare feature encoding

Zhao Y, et al, A Comparative Study and Component Analysis of Query Plan Representation Techniques in ML4DB Studies. VLDB 2024.

Experimental Settings

- Methodology:
  - Perform database tasks by replacing **query plan representation** of ML4DB systems.
    - Cost Estimation
    - Index Selection
    - (Steer) Query Optimizer
- Dataset: both synthetic and real workloads with different characteristics

Query Plan Sizes in datasets.

Dataset	Max #	Avg #	Skewness	Correlation
TPC-H	26	16.8	Uniform	Uncorrelated
TPC-DS	143	44.4	Uniform	Uncorrelated
Synthetic	10	4.9	Moderate	Moderate
JOB-Light	14	8.44	Moderate	Moderate
STATS	16	9.49	High	High
Job-Extend	73	21.2	Moderate	Moderate

Zhao Y, et al, A Comparative Study and Component Analysis of Query Plan Representation Techniques in ML4DB Studies. VLDB 2024.

Key Finding 1

Are query plan representation methods interchangeable across tasks?

- Yes. QueryFormer and BAO performs well across tasks.

Query Plan Representation Across Tasks

Method/ Component	Cost (ID)	Cost (OOD)	Index (ID)	Index (OOD)	Optimizer
AVGDL	++	-	++	-	++
AIMEETSAI	++	-	+	-	-
ReJOIN	-	-	-	+	- -
BAO	++	++	++	+	+
NEO	-	+	+	+	-
Prestroid	-	+	+	-	+
E2E-Cost	++	+	-	-	- -
RTOS	++	-	+	+	+
Plan-Cost	+	-	-	-	-
QueryFormer	++	++	++	+	+

Zhao Y, et al, A Comparative Study and Component Analysis of Query Plan Representation Techniques in ML4DB Studies. VLDB 2024.

Key Finding 2

What is the best tree model design?

- Surprisingly, tree model is **less** important when fixing feature encoding
  - It does not matter in cost estimation
  - When relative ranking matters (in index selection and optimization)
    - TreeCNN works best for in-distribution workload
    - LSTM works best for out-of-distribution workload

Table 4: Tree Model Comparison

Method/ Component	Cost (ID)	Cost (OOD)	Index (ID)	Index (OOD)	Optimizer
TreeCNN	++	+	++	+	+
TreeLSTM	++	+	+	-	-
LSTM	++	+	+	++	++
Transformer	++	+	+	+	-

Zhao Y, et al, A Comparative Study and Component Analysis of Query Plan Representation Techniques in ML4DB Studies. VLDB 2024.

Key Finding 3

What is the best feature encoding?

- *Database Estimate* is the most effective feature
- Despite individual feature effectiveness, different encoding strategies should be adopted depending on task requirement
  - To optimize **absolute error** (cost estimation), concatenate as much feature as possible
  - To optimize **relative ranking** (index selection, optimization), *Database Estimate* and *Histogram* are the best features

Table 5: Feature Encoding Comparison

Method/ Component	Cost (ID)	Cost (OOD)	Index (ID)	Index (OOD)	Optimizer
Pred	++	+	-	+	+
Est	++	+	++	++	++
Hist	-	-	+	+	++
Sample	+	-	+	-	-

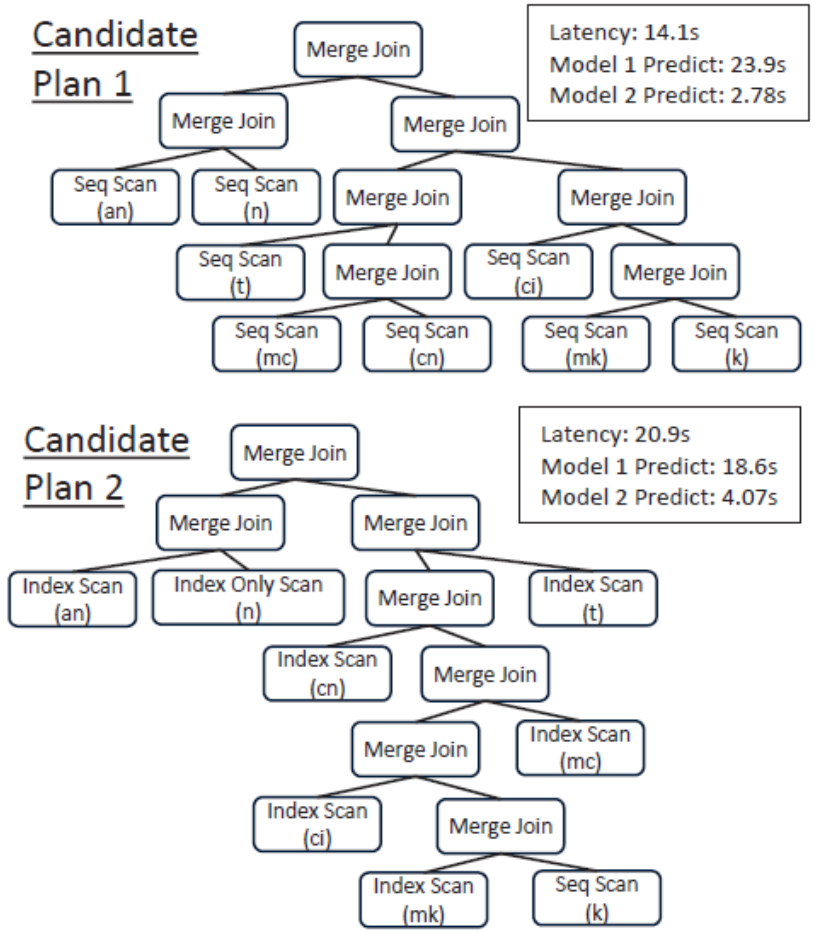
Zhao Y, et al, A Comparative Study and Component Analysis of Query Plan Representation Techniques in ML4DB Studies. VLDB 2024.

Why relative ranking differ from cost estimation?

- Case Study
  - Higher absolute accuracy does not guarantee better distinguishing power

QUERY

```
SELECT
  MIN(an.name) AS cool_actor,
  MIN(t.title) AS series_named
FROM
  aka_name AS an,
  cast_info AS ci,
  company_name AS cn,
  keyword AS k,
  movie_companies AS mc,
  movie_keyword AS mk,
  name AS n,
  title AS t
WHERE
  cn.country_code ='[us]' AND
  k.keyword ='character' AND
  t.episode_nr < 100 AND
  an.person_id = n.id AND
  n.id = ci.person_id AND
  ci.movie_id = t.id AND
  t.id = mk.movie_id AND
  mk.keyword_id = k.id AND
  t.id = mc.movie_id AND
  mc.company_id = cn.id AND
  an.person_id = ci.person_id AND
  ci.movie_id = mc.movie_id AND
  ci.movie_id = mk.movie_id AND
  mc.movie_id = mk.movie_id;
```



Zhao Y, et al, A Comparative Study and Component Analysis of Query Plan Representation Techniques in ML4DB Studies. VLDB 2024.

Take-aways for Query Plan Representation

- Query plan representation consists of **feature encoding** and **tree models**
- The optimal design should be selected based on task requirements
  - Feature encoding
    - *Histogram, Database Estimate* for relative rankings
    - The more the better for absolute numbers
  - Tree models
    - TreeCNN works best for in-distribution workload
    - LSTM works best for out-of-distribution workload

ML4DB Foundations

- Can we have some foundations that are relevant to different ML4DB systems?

- We identify two key foundations
  - Query plan representation
  - Pretrained models

Analogy to other domains

- LLMs: pretrained on large and diverse corpora

Chatbot, translation, text summarization, classification, etc., or as **zero-shot models**

Fine-tune/
RAG, etc.

Legal document analysis, customer support, technical document assistance, etc., or as **base models**

Analogy to other domains

- LLMs: pretrained on large and diverse corpora

Chatbot, translation, text summarization, classification, etc., or as **zero-shot models**

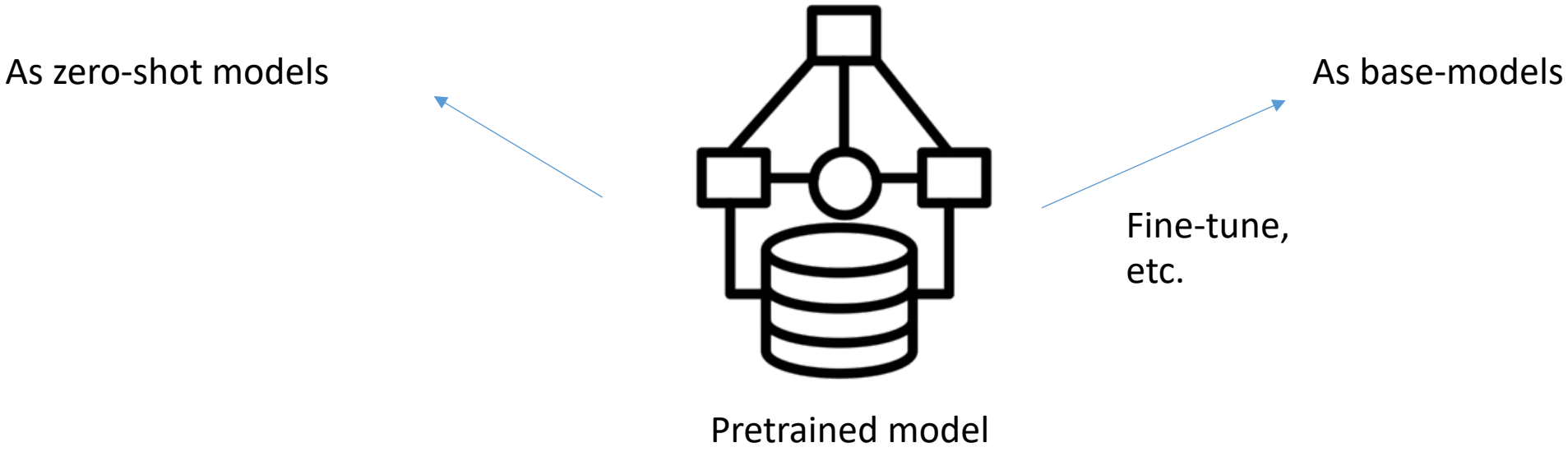
Fine-tune/ RAG, etc.

Legal document analysis, customer support, technical document assistance, etc., or as **base models**

- Benefits:
  - More resource efficient
  - Incredible performance

Can we do the same for ML4DB?

- Pretrained Models: pretrain on diverse databases, configurations, and workloads



- Benefits:
  - More resource efficient
  - Better performance

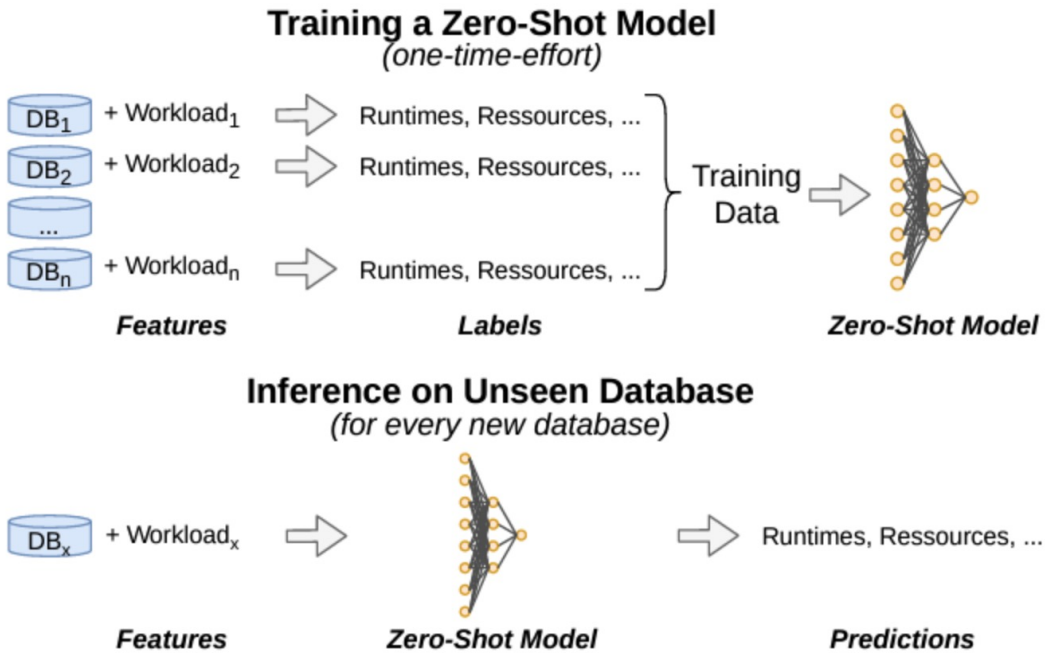
Zero-shot models

- Issues with workload-driven learning
  - Require running **huge** # of queries to get training data
  - Re-run training queries to support new databases, or even when data is updated
- Idea: use a pre-trained model that can generalize to **unseen databases** (with different data characteristics) **out-of-the-box**

Hilprechet & Binnig. One Model to Rule them All: Towards Zero-Shot Learning for Databases. CIDR 22.

Zero-shot models

- A pre-trained model that can generalize to **unseen databases** (with different data characteristics) **out-of-the-box**



Hilprechet & Binnig. One Model to Rule them All: Towards Zero-Shot Learning for Databases. CIDR 22.

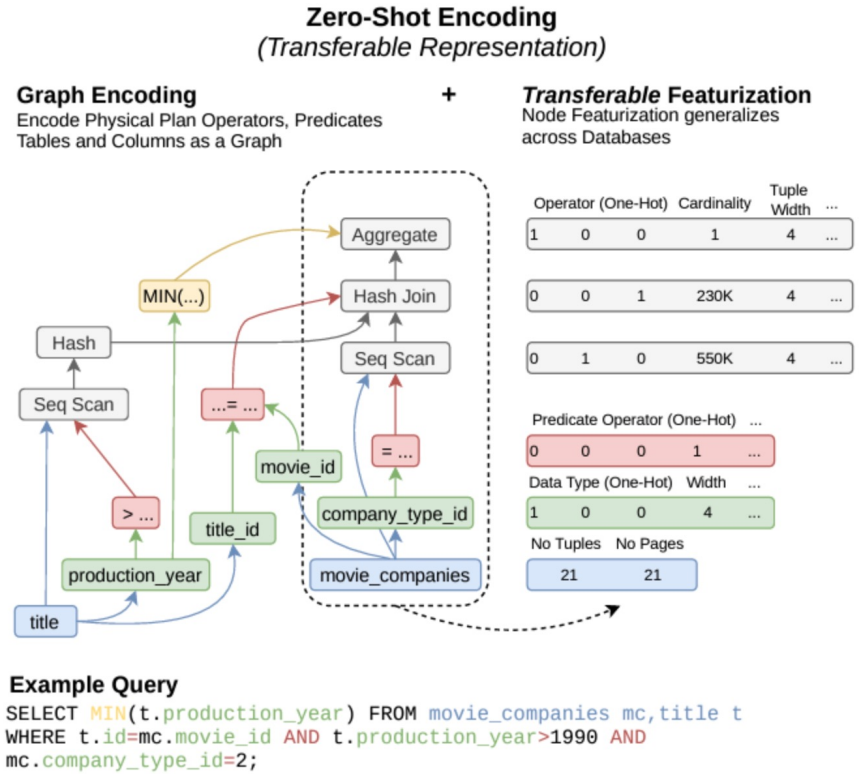
Zero-shot models

- Challenge: ML4DB systems typically encodes features specific to a database, i.e., a table, a column, etc., which are inconsistent for multiple databases

Hilprechet & Binnig. One Model to Rule them All: Towards Zero-Shot Learning for Databases. CIDR 22.

Zero-shot models

- Challenge: ML4DB systems typically encodes features specific to a database, i.e., a table, a column, etc., which are inconsistent for multiple databases
- Solution: Design transferrable (database-agnostic) features

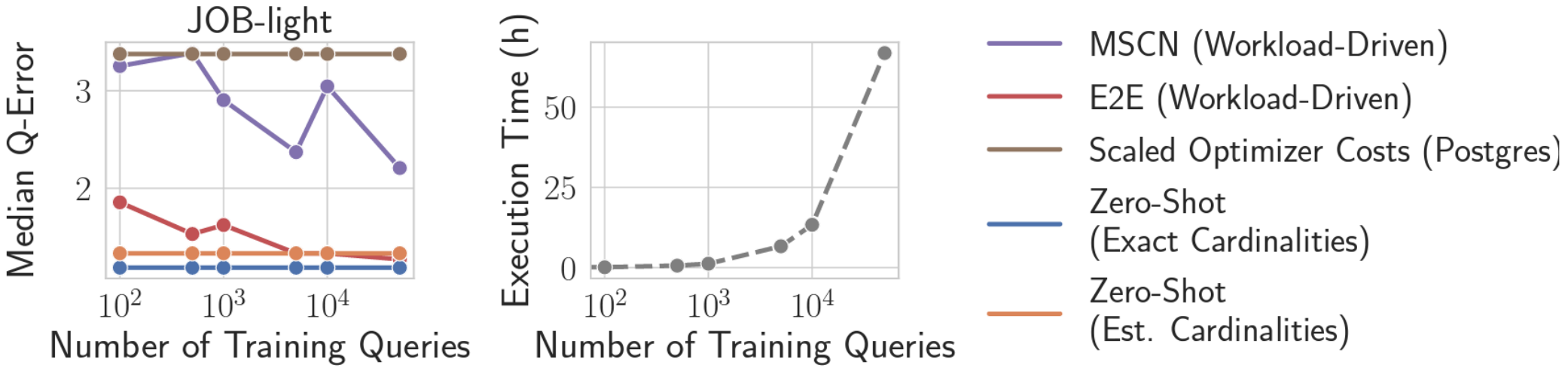


Transferrable Featurization
Data Type,Tuple Width, etc., instead of column name

Benjamin H, Carsten B. One Model to Rule them All: Towards Zero-Shot Learning for Databases. CIDR 22.

Zero-shot cost estimation

- Comparison with workload-driven learning
  - Workload-driven models require large execution time to match zero-shot performance



Hilprechet & Binnig. One Model to Rule them All: Towards Zero-Shot Learning for Databases. CIDR 22.

Other applications of Zero-shot models

- Physical design tuning
  - MV selection: predict benefit of MV candidates using zero-shot models
- Learned Optimizer
  - Enumerate plans and choose cheapest using zero-shot models
- Data Structure Design
- Device Placement
- …

Hilprechet & Binnig. One Model to Rule them All: Towards Zero-Shot Learning for Databases. CIDR 22.

MTMLF

- Existing ML4DB systems: one task at a time
  - Ignore relations between tasks
    - Redundant learning
    - Not effective
  - Cold start: not transferrable across tasks

Wu, Z , et al. A Unified Transferable Model for ML Enhanced DBMS. CIDR 22.

MTMLF

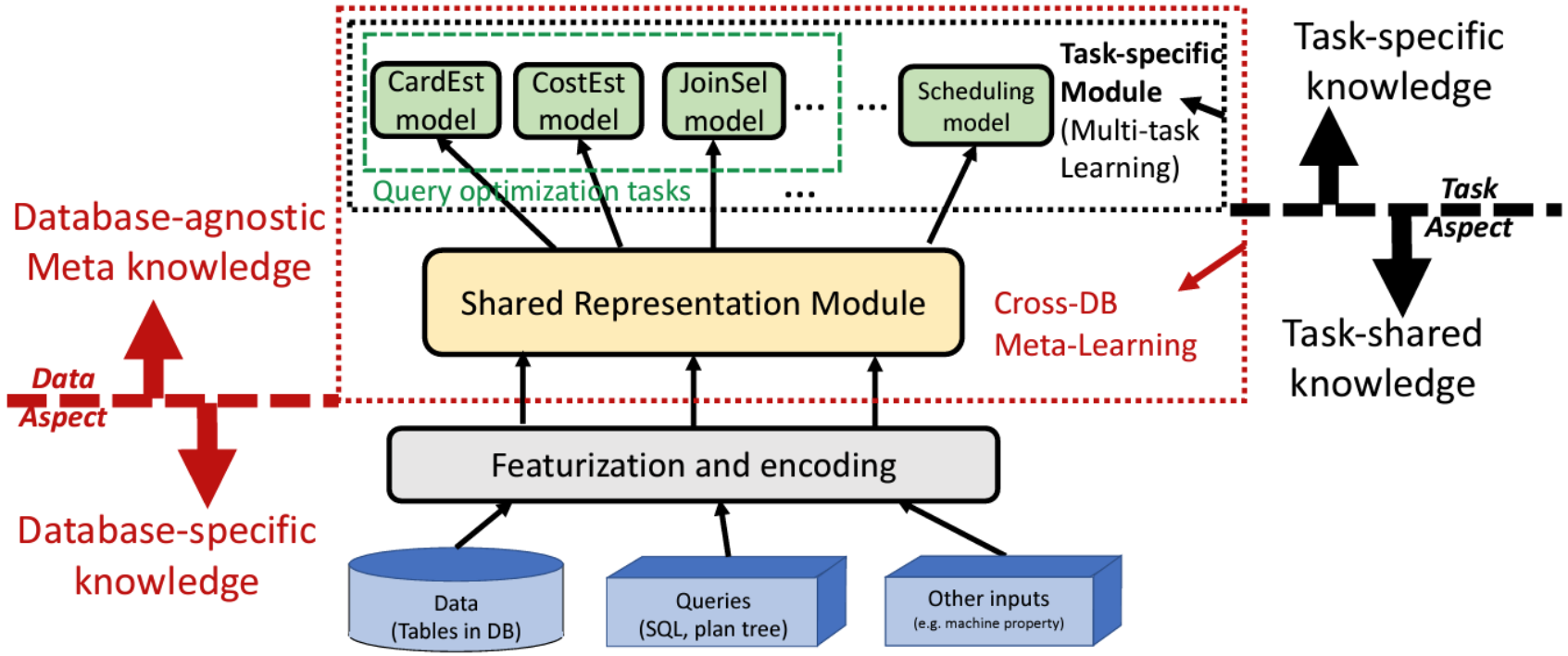
- Existing ML4DB systems: one task at a time
  - Ignore relations between tasks
    - Redundant learning
    - Not effective
  - Cold start: not transferrable across tasks

- Question
  - What knowledge can be re-used?
  - How do we re-use them?

Wu, Z , et al. A Unified Transferable Model for ML Enhanced DBMS. CIDR 22.

MTMLF

- Learned data and task knowledge decomposition



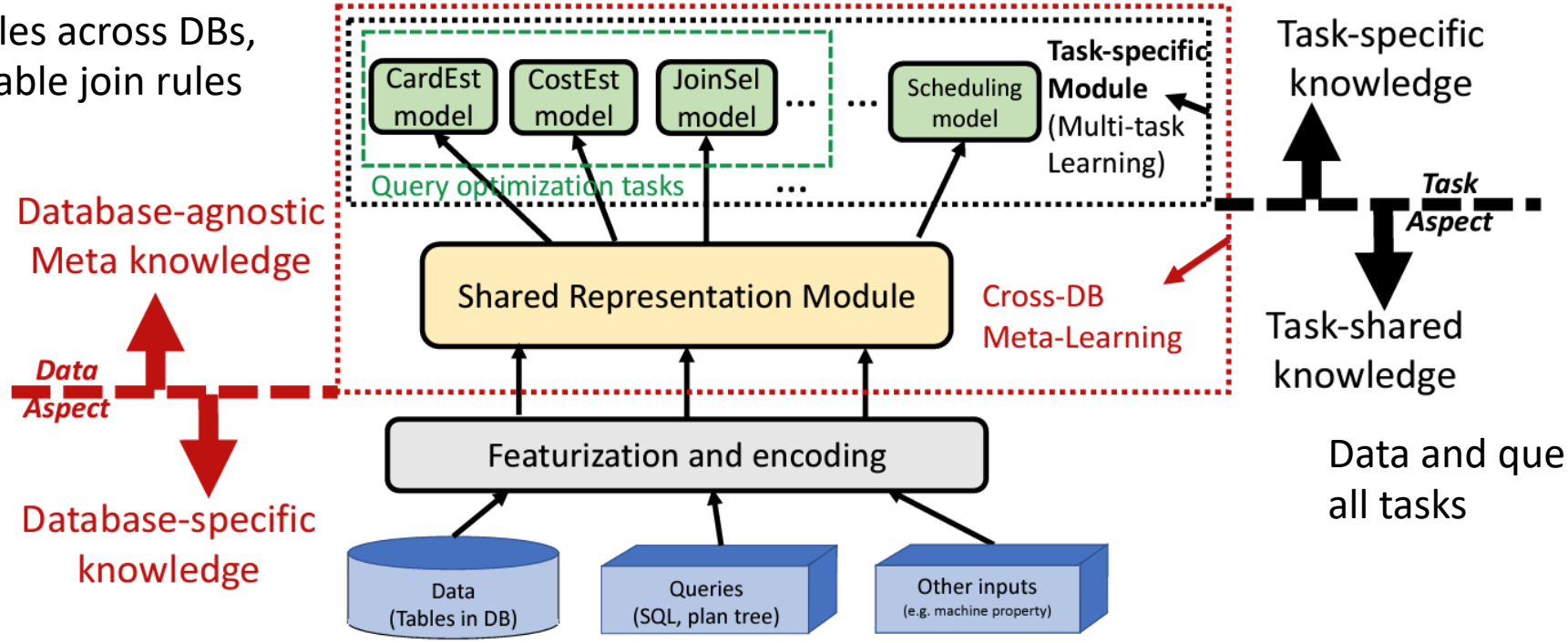
Wu, Z , et al. A Unified Transferable Model for ML Enhanced DBMS. CIDR 22.

MTMLF

- Learned data and task knowledge decomposition

Method to a specific task, e.g. join order selection rules

Common rules across DBs, e.g., multi-table join rules



Data and query representations in all tasks

Data distributions and join schema

Wu, Z , et al. A Unified Transferable Model for ML Enhanced DBMS. CIDR 22.

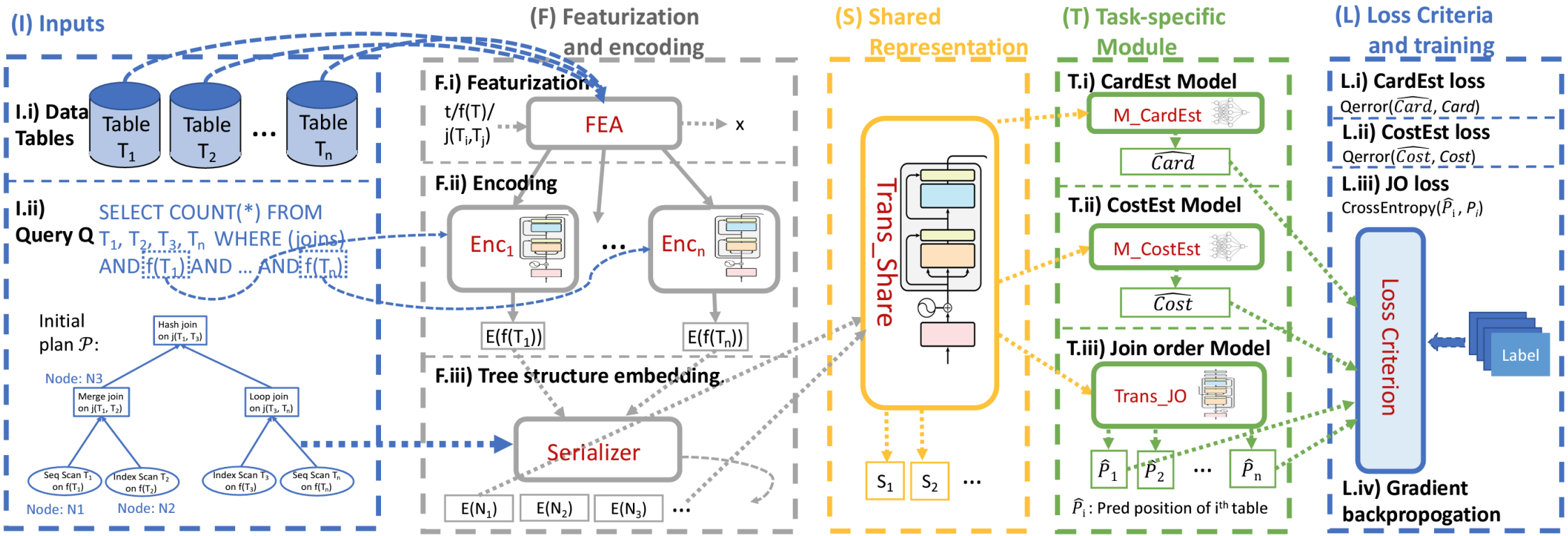
Envisioned DB service

- Service provider:
  - Train on multiple anonymized users DBs
  - Provide users the **shared representation** and **task specific** modules
  - Periodically collect anonymized information and update model as service upgrade
- Users:
  - Train **featurization** module for its own DB
  - Execute a small number of queries to fine-tune model
  - (Optionally) provide anonymized information to service provider

Wu, Z , et al. A Unified Transferable Model for ML Enhanced DBMS. CIDR 22.

Query Optimizer: A Case Study

- Jointly learn 3 tasks



Wu, Z , et al. A Unified Transferable Model for ML Enhanced DBMS. CIDR 22.

Case Study Result

- More effective than separate learning

Table 1: Q-errors on the JOB workload.

Method	Cardinality			Cost		
	median	max	mean	median	max	mean
PostgreSQL	184.00	670,000	10,416	4.90	4920	105.00
Tree-LSTM	8.78	696.29	36.83	4.00	290.35	15.01
MTMLF-QO	**4.48**	**614.45**	**28.69**	**2.10**	**37.54**	**4.20**
MTMLF-CardEst	5.12	804.48	36.66	\	\	\
MTMLF-CostEst	\	\	\	2.06	61.41	4.69

Table 2: Execution time with different join orders.

JoinOrder	Total Time	Overall Improvement Ratio
PostgreSQL	1143.2 min	\
Optimal	209.1 min	81.7%
MTMLF-QO	**318.3 min**	**72.2%**
MTMLF-JoinSel	450.4 min	60.6%

Wu, Z , et al. A Unified Transferable Model for ML Enhanced DBMS. CIDR 22.

Case Study Result

- Artificially generated 11 datasets with various distributions and number of tables
- Trained on 10, tested on 1

JoinOrder	Total Time	Overall Improvement Ratio
PostgreSQL	393.9 min	\
MTMLF-QO (MLA)	**234.1 min**	**40.6%**
MTMLF-QO (single)	219.5 min	44.3%

Wu, Z , et al. A Unified Transferable Model for ML Enhanced DBMS. CIDR 22.

Pretrained workload characterization

- Workload characterization in databases is important
  - Query plan performance depends on queries, data, and resources
- Challenges:
  - Diverse query structure, high modeling complexity, difficult domain adaptation, etc.

Debjyoti P, et al. Database Workload Characterization with Query Plan Encoders. VLDB 21

Pretrained workload characterization

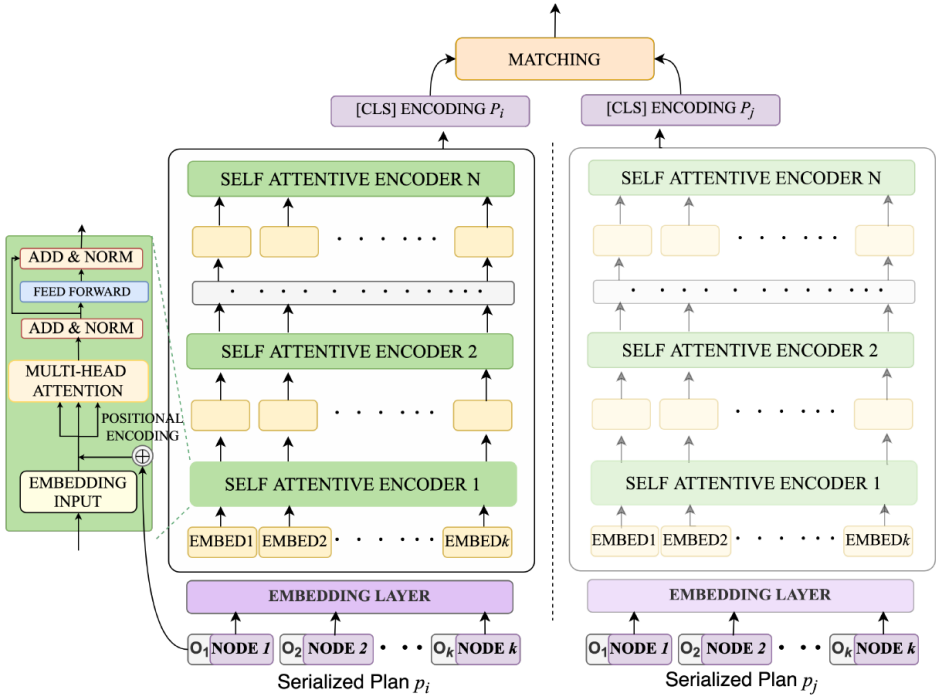
- Workload characterization in databases is important
  - Query plan performance depends on queries, data, and resources
- Challenges:
  - Diverse query structure, high modeling complexity, difficult domain adaptation, etc.
- Idea:
  - Develop and pre-train a query plan encoder
  - Fine-tune to arbitrary database and task

Debjyoti P, et al. Database Workload Characterization with Query Plan Encoders. VLDB 21

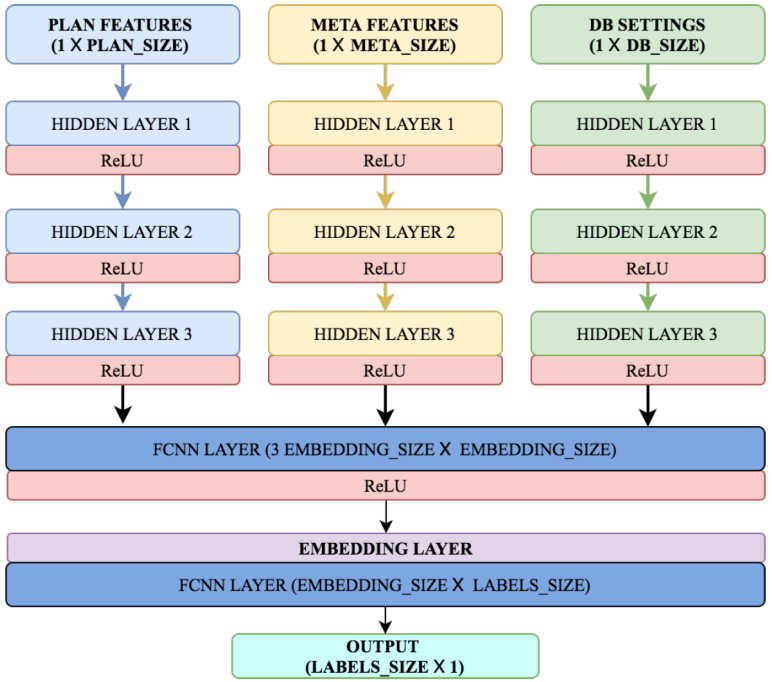
Separate Structural encoder and Performance Encoder

- Separate encoders for structure features and performance features

Train with matching loss



Train with label loss (time, cost, etc)



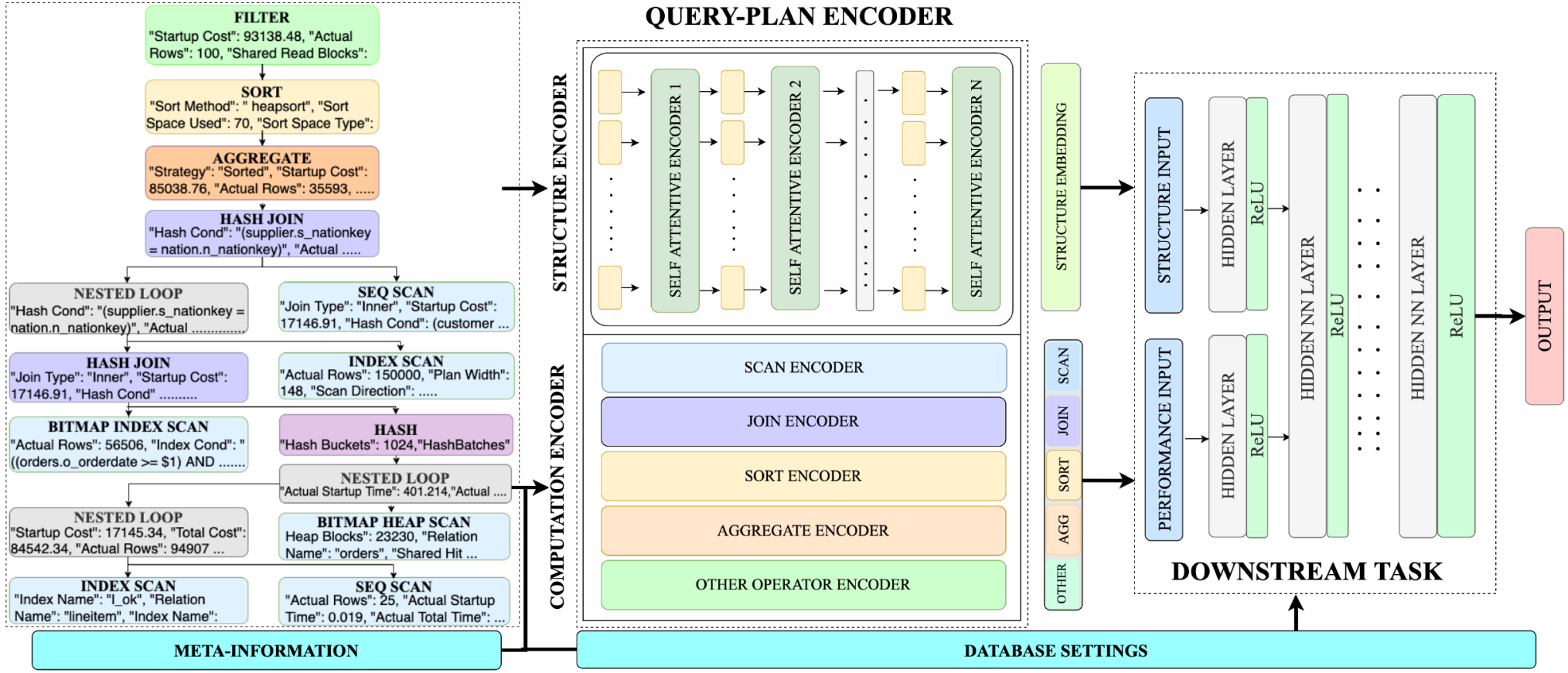
Structural Encoder

Performance Encoder

Debjyoti P, et al. Database Workload Characterization with Query Plan Encoders. VLDB 21

Separate Structural encoder and Performance Encoder

- Full architecture (Both encoder can be used together or separately)



Debjyoti P, et al. Database Workload Characterization with Query Plan Encoders. VLDB 21

Experiments

- Pretrain on various sources
  - Structure encoder: crowdsourced plan dataset
  - Performance encoder: TPC-H, TPC-DS
- Fine-tune to downstream tasks

Latency Prediction

Model	$R \leq 1.5$	$1.5 < R \leq 2.0$	$R > 2.0$
TAM	51%	22%	27%
SVM	68%	15%	17%
RBF	85%	6%	9%
QPPNet	89%	**7%**	4%
Plan Encoder	**91%**	**7%**	**2%**

Query Classification

Models	Development		Test	
	template	cluster	template	cluster
Structure only	0.2452	0.4670	0.1946	0.3847
Performance only	0.1645	0.2973	0.0977	0.1769
Both encoders	**0.2783**	**0.5573**	**0.2518**	**0.4647**
Both encoders 10% data	0.2000	0.4927	0.151	0.334
Both encoders 30% data	0.2555	0.5228	0.1843	0.3855

Debjyoti P, et al. Database Workload Characterization with Query Plan Encoders. VLDB 21

Take-aways for Pre-trained Models

- We can exploit pre-training techniques to reduce redundant computations by
  - Disentangle model components
  - Design more robust featurization methods
  - Employ multi-task learning techniques

- Early-stage idea
  - Proof-of-concept
  - How to collect pretraining dataset?

Tutorial Overview

- ML4DB Foundations
- ML4DB Paradigms
- ML4DB Open problems

ML4DB Paradigms

- “Replacement” Paradigm
  - Machine learning models replace the traditional database components
  - Example: replacing index, replacing query optimizer

- “ML-enhanced” Paradigm
  - Machine learning models improve the traditional database components
  - Example: learn better policies during index organization, generate helpful hints for expert optimizer

“Replacement” Paradigm

- Learned index: Replace the **B-tree Index** with a **cumulative density function (CDF)** model

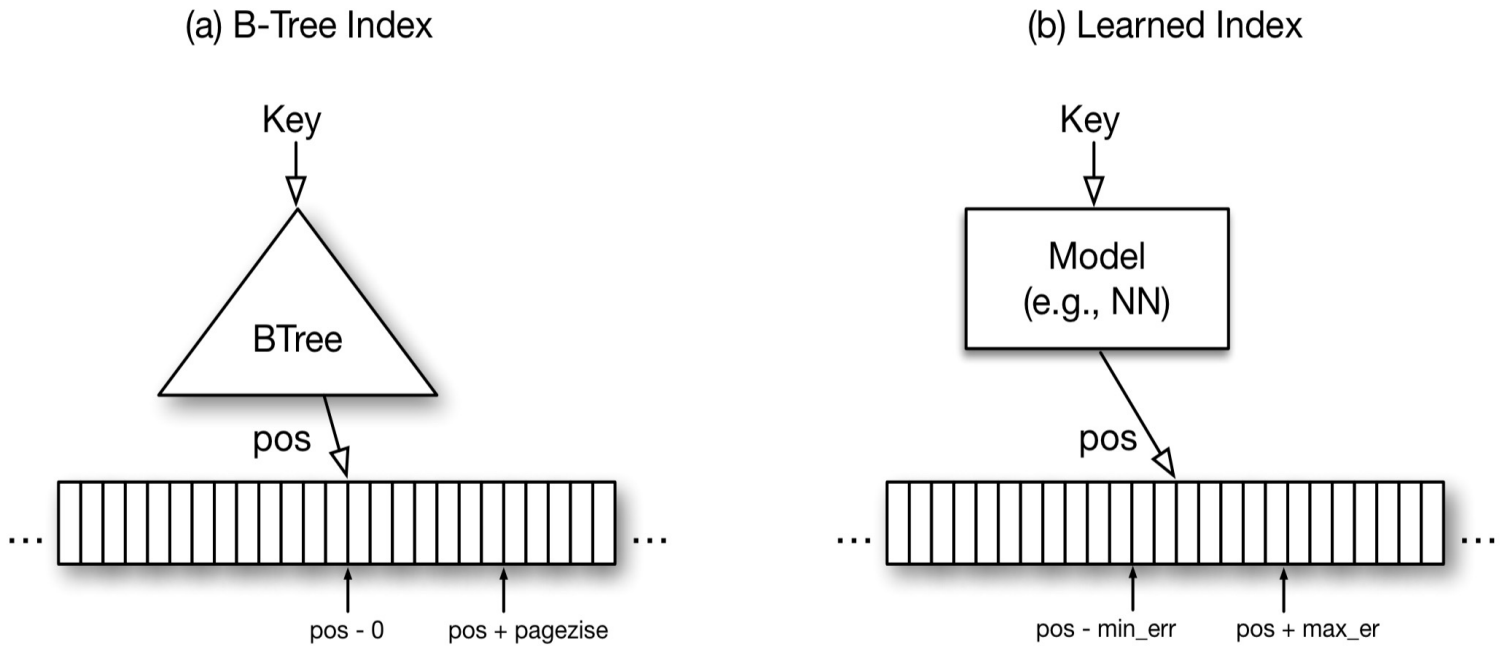


Figure from Tim Kraska et al. **The case for learned index structures.** SIGMOD’18

“ML-enhanced” Paradigm

- ML-enhanced index: Use machine learning to **improve** operations in **traditional R-tree index**, e.g., data insertion

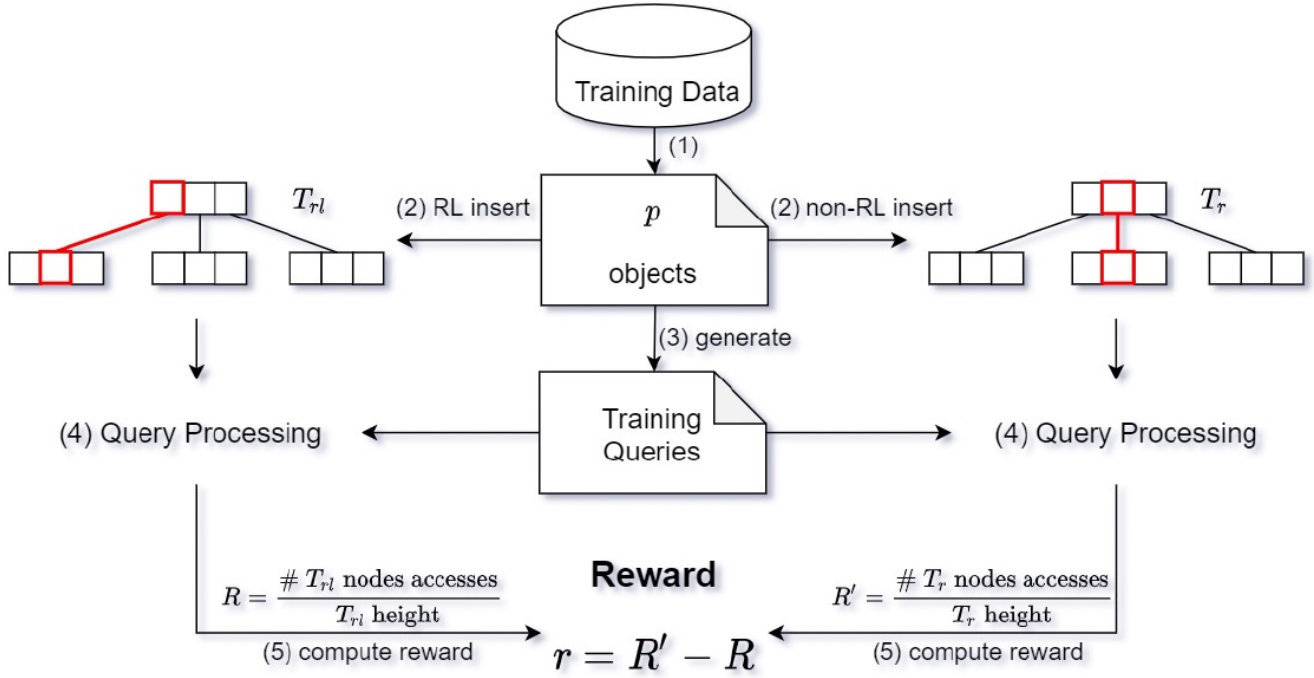


Figure from Tu Gu et al. **The RLR-Tree: A Reinforcement Learning Based R-Tree for Spatial Data.** SIGMOD’23

“Replacement” Paradigm

- Learned query optimizer: Replace **optimizer** with a **reinforcement learning** model

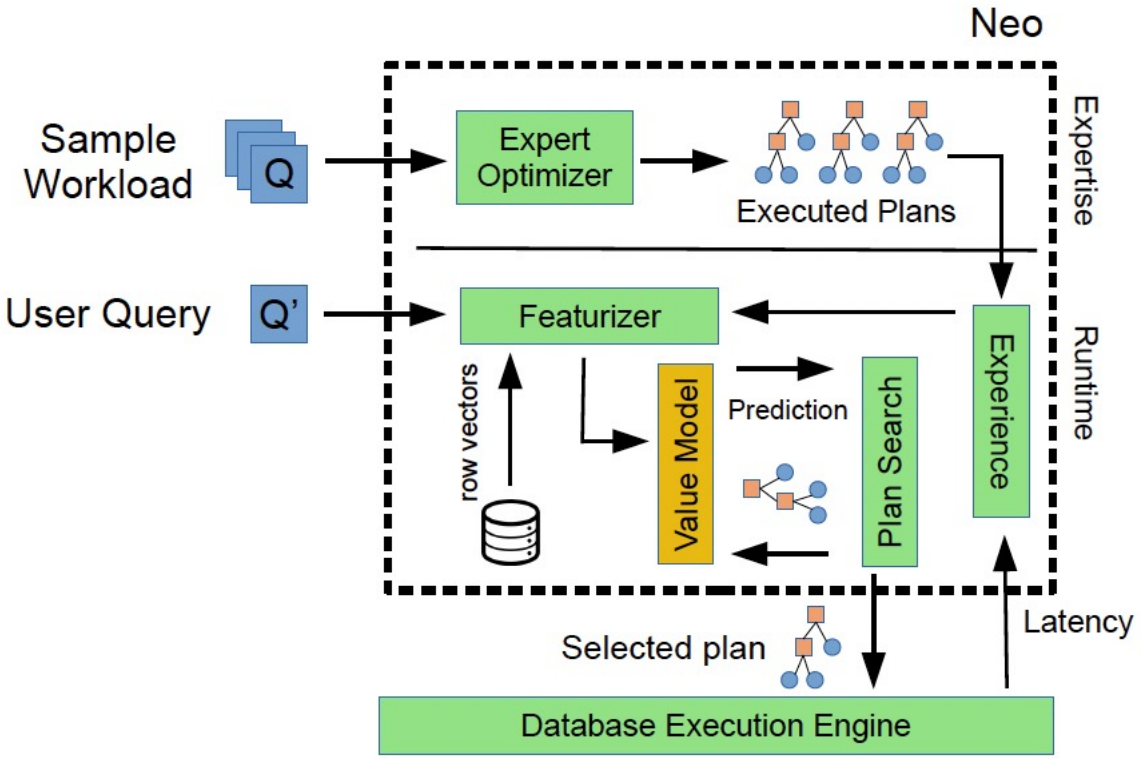


Figure from Ryan Marcus et al. **NEO: a learned query optimizer.** SIGMOD’19

ML4DB Paradigms

- ML-enhanced query optimizer: Use machine learning to **improve expert query optimizer**, e.g., provide a set of hints

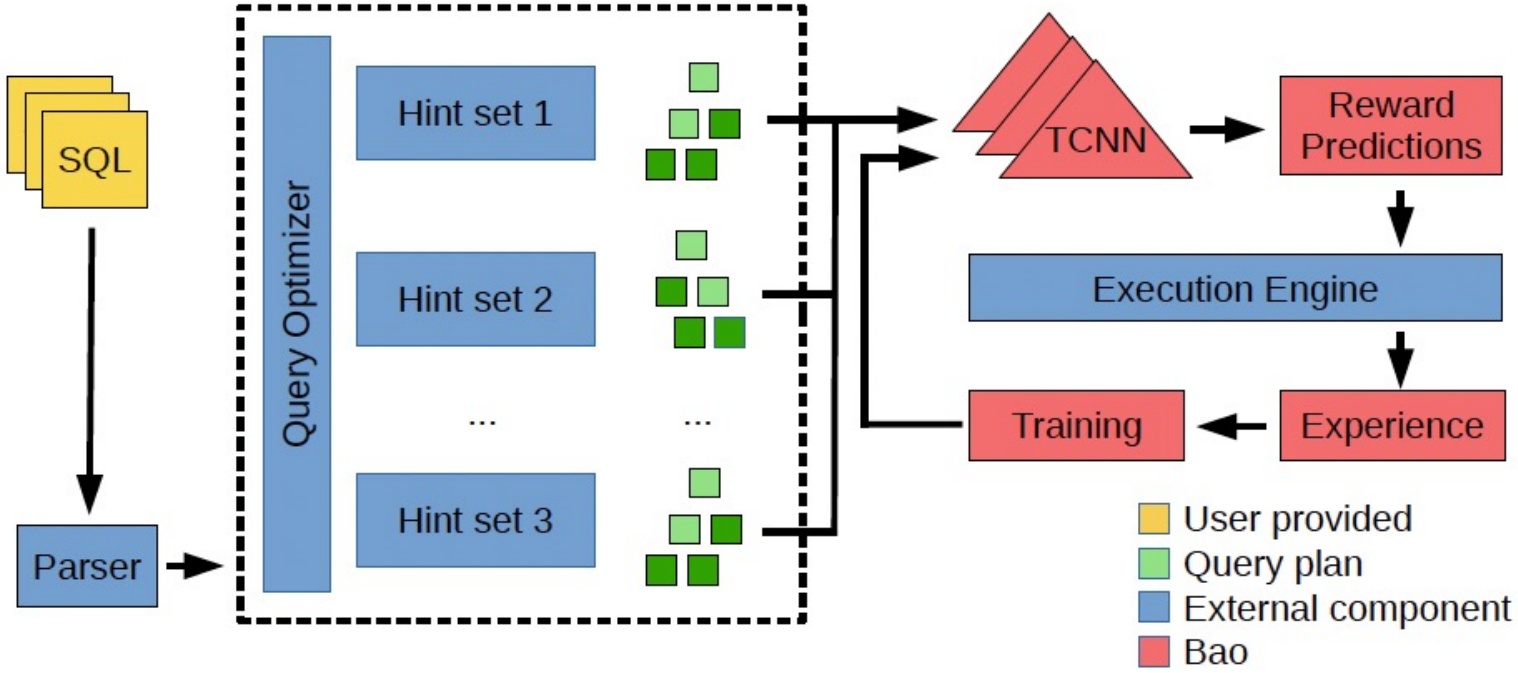
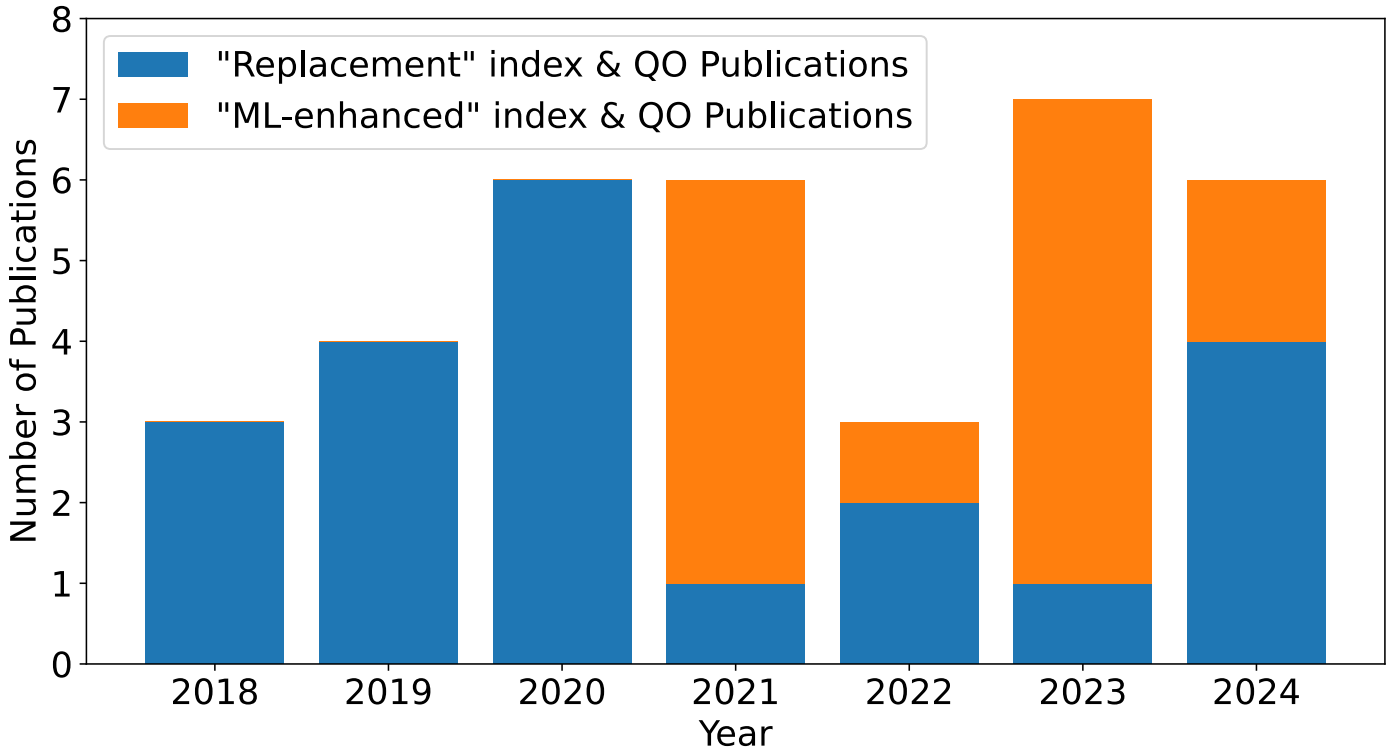


Figure from Ryan Marcus et al. **BAO: Making learned query optimization practical.** SIGMOD'21

Publication Trend

- We observe a noticeable shift from the “Replacement” paradigm to the “ML-enhanced” paradigm over the past years.



Publication on machine learning for index & Query Optimizer (QO) in SIGMOD and VLDB

ML4DB Paradigms

- We study ML4DB paradigms from the perspective of two important problems
  - Database indexing
  - Query optimization

ML4DB Paradigms

- We study ML4DB paradigms from the perspective of two important problem
  - Database indexing
  - Query optimization

ML4DB Paradigms in Database indexing

- Part 1: Preliminary
- Part 2: Learned index
- Part 3: ML-enhanced index
- Part 4: Summary

B+-tree Index

- B+-tree builds a balanced tree over the keys, which enables $O(\log n)$ key lookup.
- Support fast insertion and deletion of data - $O(\log n)$.

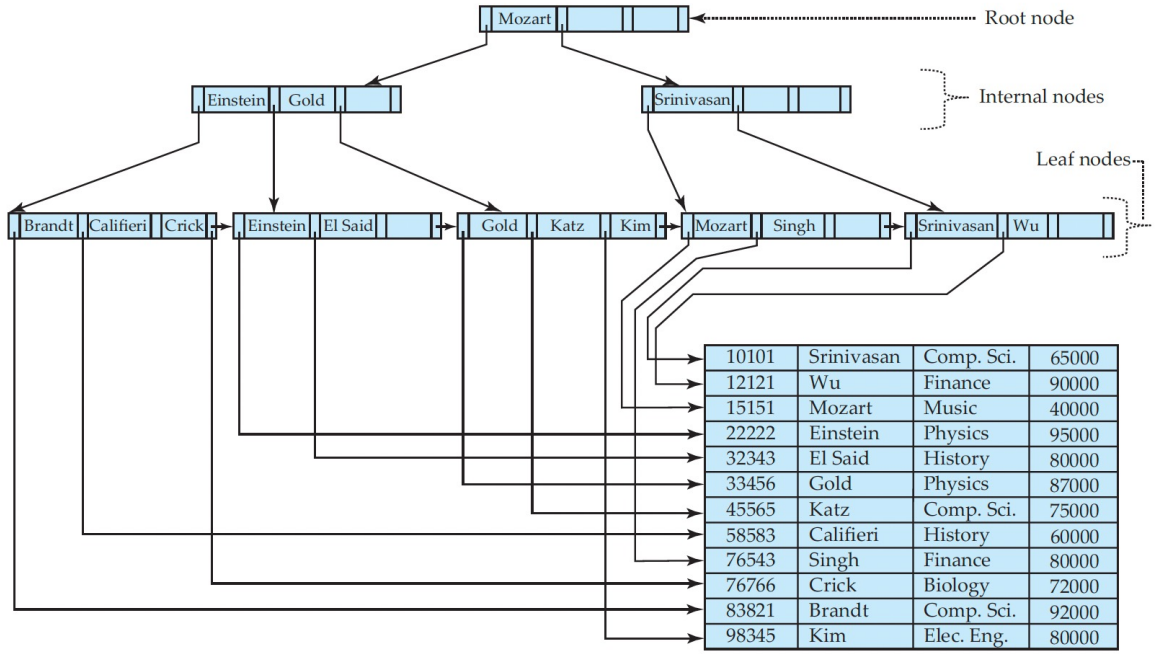
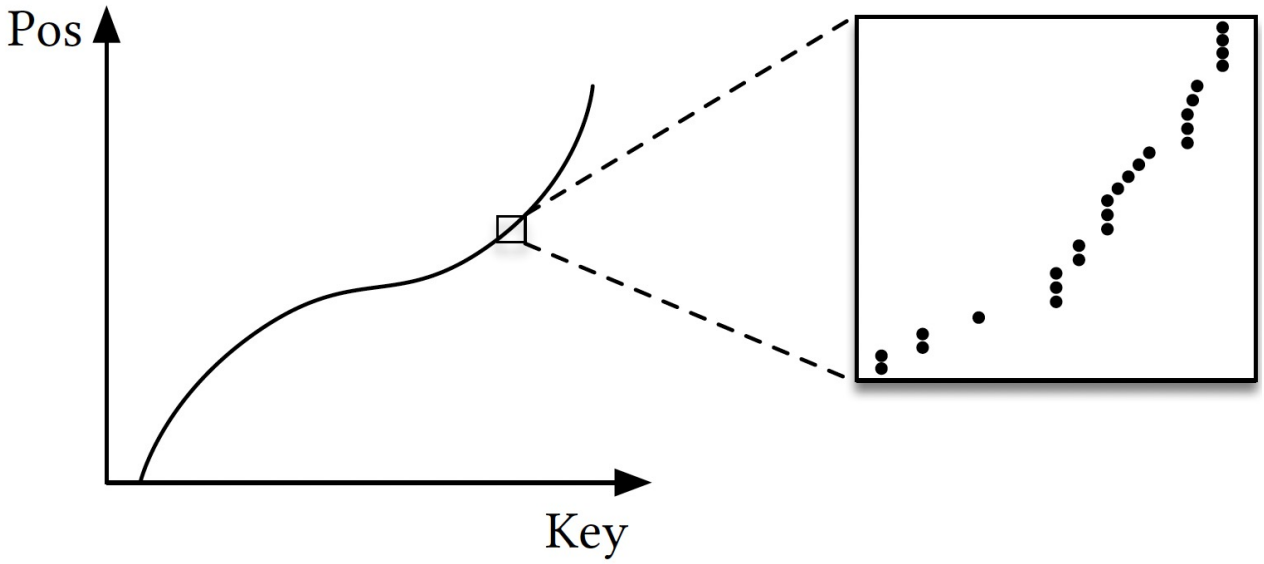


Figure from **Database System Concepts, 6th Edition.**

Learned index

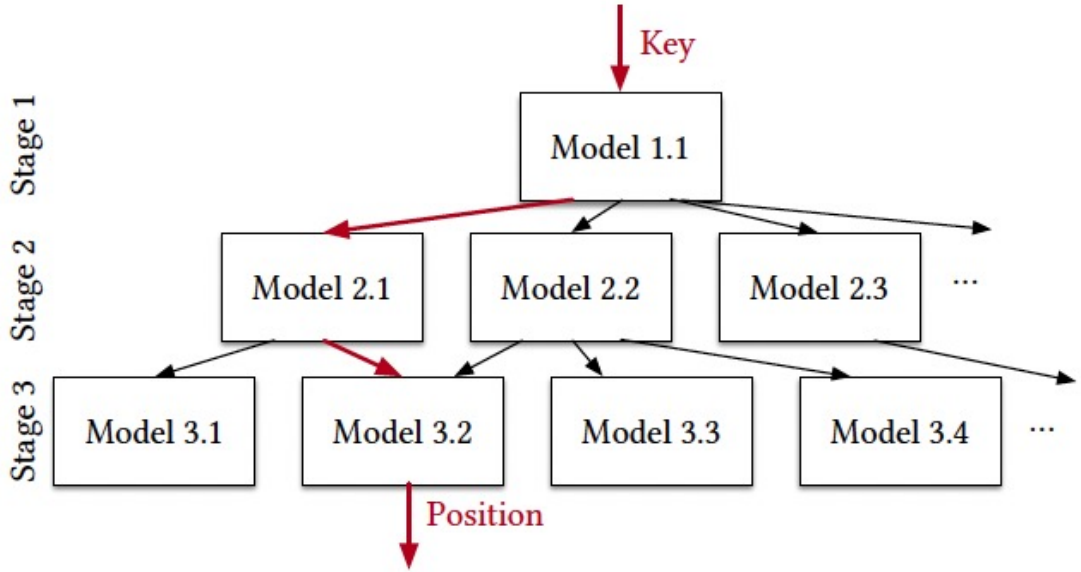
- Observation: B+-tree can be considered as a model.
  - Input: Search key
  - Output: Position of the record in a sorted file, i.e., page id
- B+-tree can therefore be replaced by a Cumulative Density Function (CDF) model
  - $p = F(key) * N$



Tim Kraska et al. **The Case for Learned Index Structures.** SIGMOD'18

Learned Index

- The Recursive Model Index (RMI) consists of a hierarchy of models.
- At each stage the model takes the key as an input and picks another model based on output.
- The final stage models predict the position.



Tim Kraska et al. **The Case for Learned Index Structures.** SIGMOD'18

Learned Index

- Improving lookup performance: $O(\log n) \rightarrow O(1)$
- Offload computation from the CPUs to GPUs.
- Space saving: Disk page -> Memory

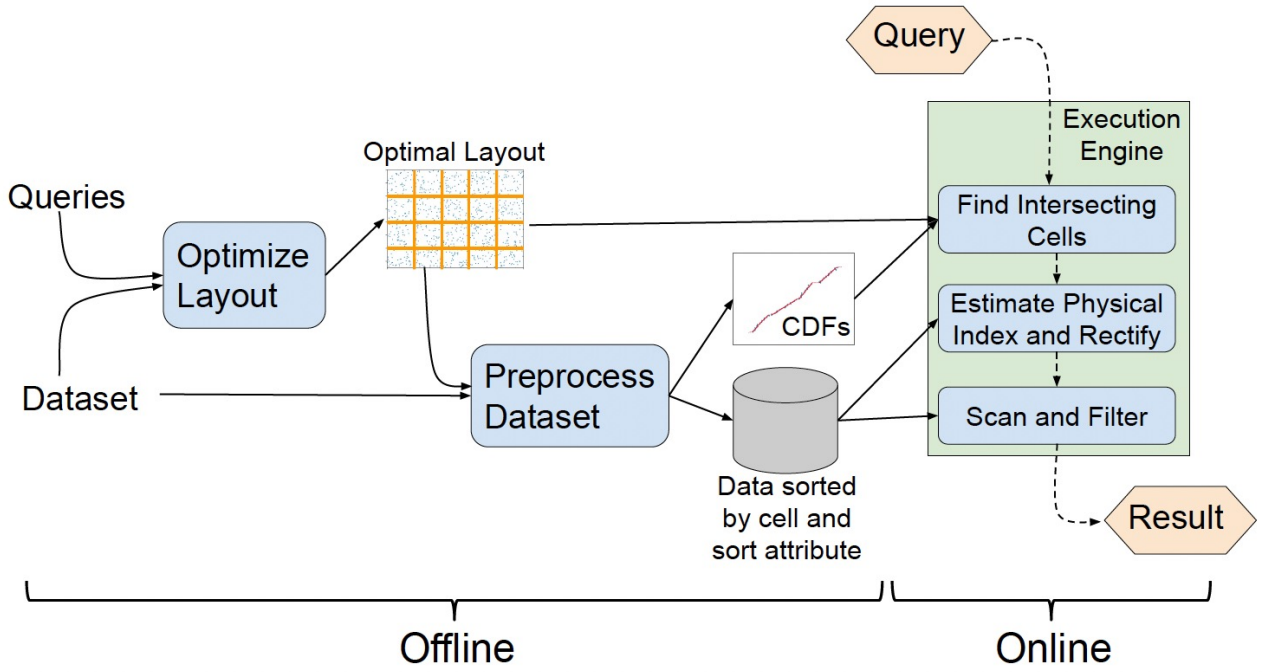
Tim Kraska et al. **The Case for Learned Index Structures.** SIGMOD'18

Learned Index

- A long line of work follows the RMI and improves it in various aspects
- Handling updates
  - Ding et al. ALEX: an updatable adaptive learned index. SIGMOD'20
- Provable bounds
  - Ferragina et al. The PGM-index: a fully-dynamic compressed learned index with provable worst-case bounds. VLDB'20
- Different storage medium
  - Lu et al. APEX: A High-Performance Learned Index on Persistent Memory. VLDB'21
- ……

Learned Multi-dimensional index

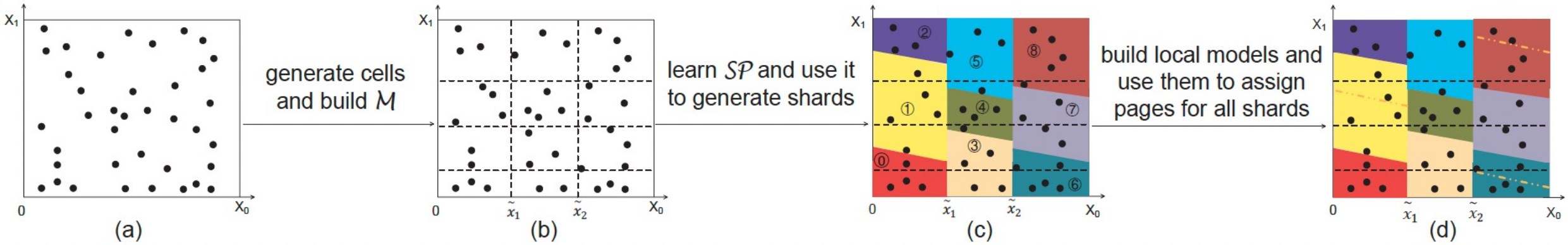
- Flood customizes the data layout to optimize query performance.
- Using empirical CDF models to flatten data into a more uniform space.



Vikram Nathan et al. **Learning Multi-dimensional Indexes.** SIGMOD'20

Learned Spatial Index

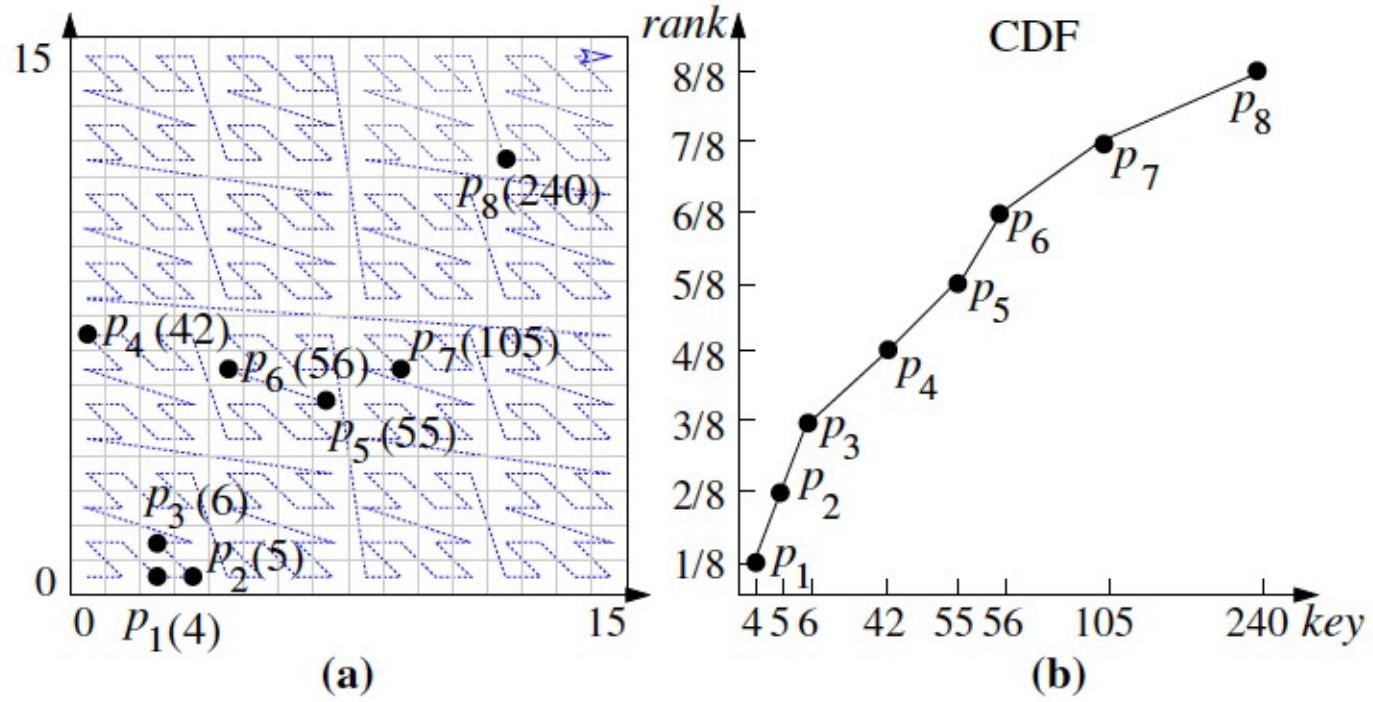
- Learned spatial indexes learns to map spatial coordinate → storage id.
- Specialized Models: a hierarchical structure
  - Grid cells
  - Shard Prediction Functions
  - Local models



Pengfei Li et al. **LISA: A learned index structure for spatial data.** SIGMOD'20

Learned Spatial Index

- Utilize a space-filling curve to linearize spatial coordinates into 1d value.
- Learn a CDF of the linearized values using a learned 1D index.



Jianzhong Qi et al. **Effectively learning spatial indices.** VLDB'20

Motivation for ML-enhanced Index

- **Robustness**:
  - Replace both the index structures and query processing algorithms currently used by the database systems **-->** hard to deploy in existing systems
- **Traditional index has better generalization capability**:
  - Support for more types of data
  - Support for more types of queries
  - Update and deletion handling

- Can we improve the traditional index without replacing the index structure or the query processing algorithm?

ML-enhanced Index

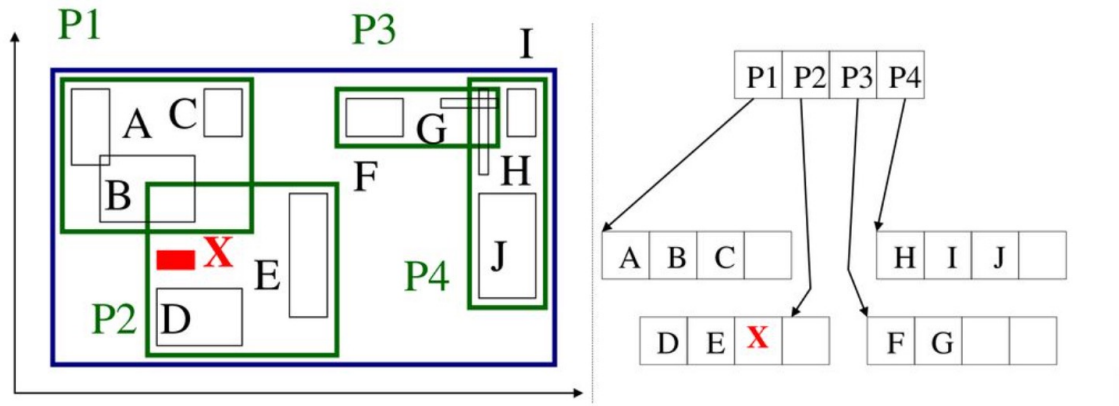
- We will discuss ML-enhanced methods that aid three types of index operations.
- **Insertion**: Building a better index for dynamic data.
  - RLR-tree: learning to improve R-tree insertion
  - BM-tree: learning to design space filling curve
- **Bulk-loading**: Constructing a better index through bulk-loading.
  - PLATON: R-tree packing with learned partition policy
- **Query**: Optimizing index search operation.
  - AI+R-tree: learning to accelerate R-tree search

ML-enhanced Index

- We will discuss ML-enhanced methods that aid three types of index operations.
- **Insertion**: Building a better index for dynamic data.
  - RLR-tree: learning to improve R-tree insertion
  - BM-tree: learning to design space filling curve
- **Bulk-loading**: Constructing a better index through bulk-loading.
  - PLATON: R-tree packing with learned partition policy
- **Query**: Optimizing index search operation.
  - AI+R-tree: learning to accelerate R-tree search

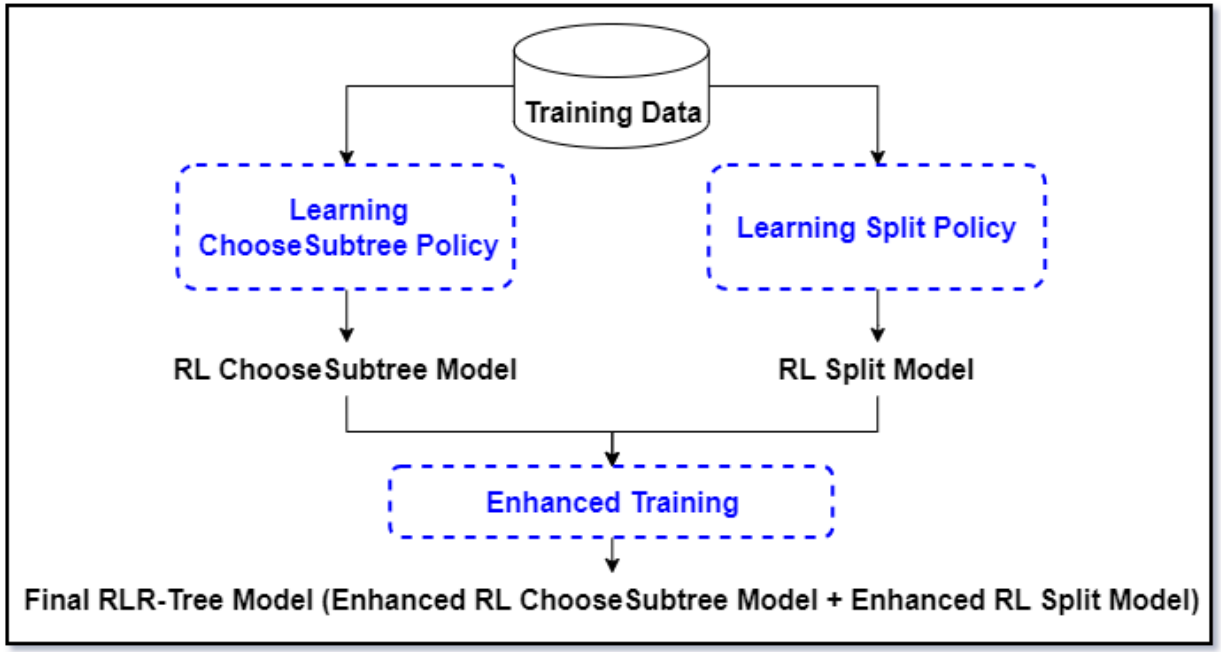
R-tree Insertion

- Two key operations of R-Tree, i.e., **ChooseSubtree** and **Split**.
  - **ChooseSubtree**: starting from the tree root, recursively choose which child node to insert the new data object, until a leaf node is reached.
  - **Split**: If the number of entries in a node exceeds the capacity, the Split operation is invoked to divide the entries into two groups.
- **Variants of R-tree have different hand-crafted heuristics. But no single heuristic rule is dominant**.



RLR-Tree Overview

- **RLR-Tree:** use **machine learning (ML) to construct a better R-Tree** for better query efficiency in a dynamic environment.
  - Model ChooseSubtree and Split as two Markov Decision Processes (MDPs) and train **reinforcement learning (RL) models** to learn optimal policies.



Tu Gu et al. **The RLR-Tree: A Reinforcement Learning Based R-Tree for Spatial Data.** SIGMOD'23

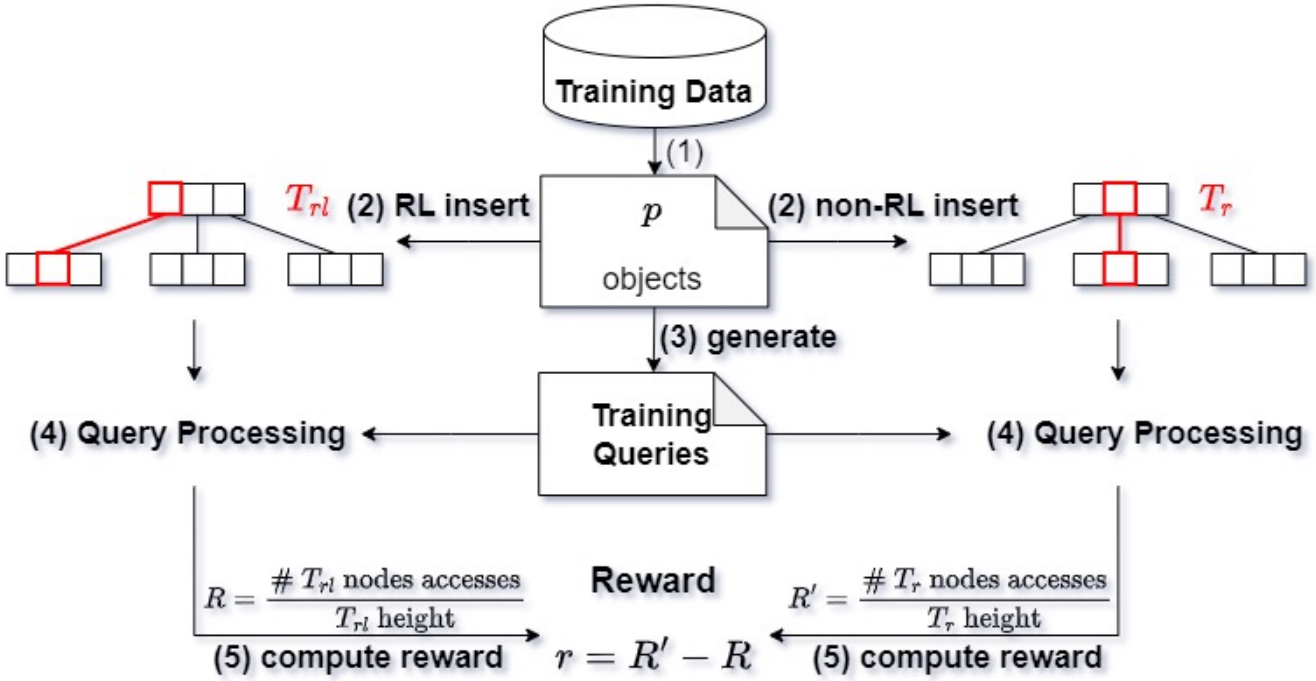
RLR-Tree ChooseSubtree

- Key Features of RL ChooseSubtree:
  - **Action Space**:
    - Setting existing hand-crafted heuristic strategies as actions is straightforward, but experimentally shown not to work.
    - Select the top k child nodes to form the action space. With "bad" actions removed, model training gets more efficient.
  - **State Space**:
    - Using the extreme values of the minimum bounding box of each node (action) as state features is straightforward, but experimentally shown not to work.
    - Instead, capture the change resulted from each action, such as increase in the area and the perimeter of the candidate child node.

Tu Gu et al. **The RLR-Tree: A Reinforcement Learning Based R-Tree for Spatial Data.** SIGMOD'23

RLR-Tree ChooseSubtree

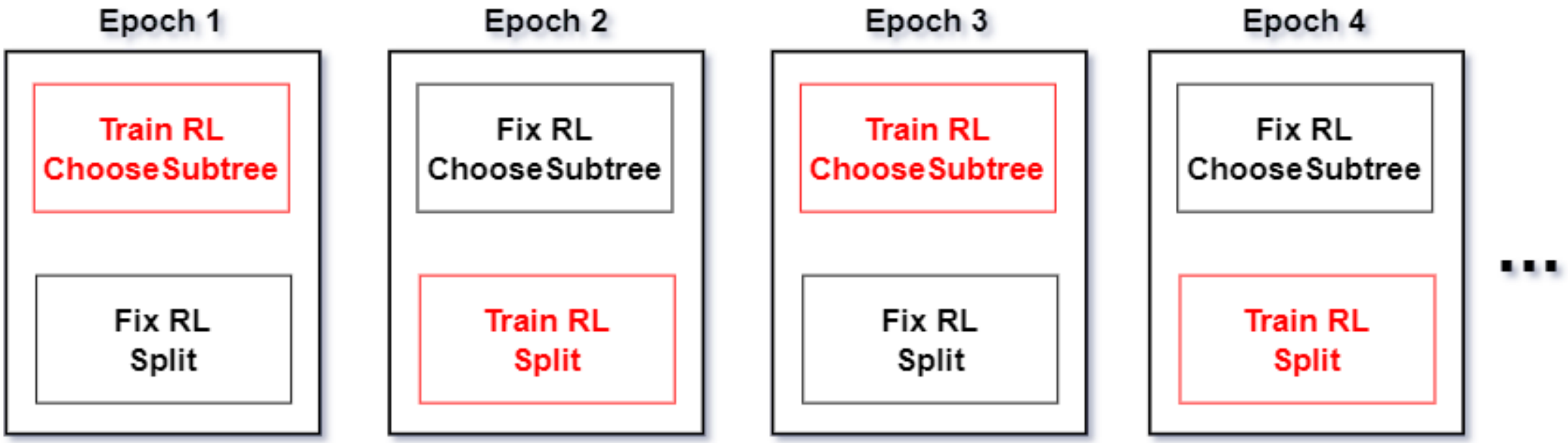
- Key Features of RL ChooseSubtree:
  - **Reward Computation:** construct an RLR-Tree (T_{rl}) and an R-Tree (T_r) and measure the difference in their query performance.



Tu Gu et al. **The RLR-Tree: A Reinforcement Learning Based R-Tree for Spatial Data.** SIGMOD'23

RLR-Tree Enhanced Training

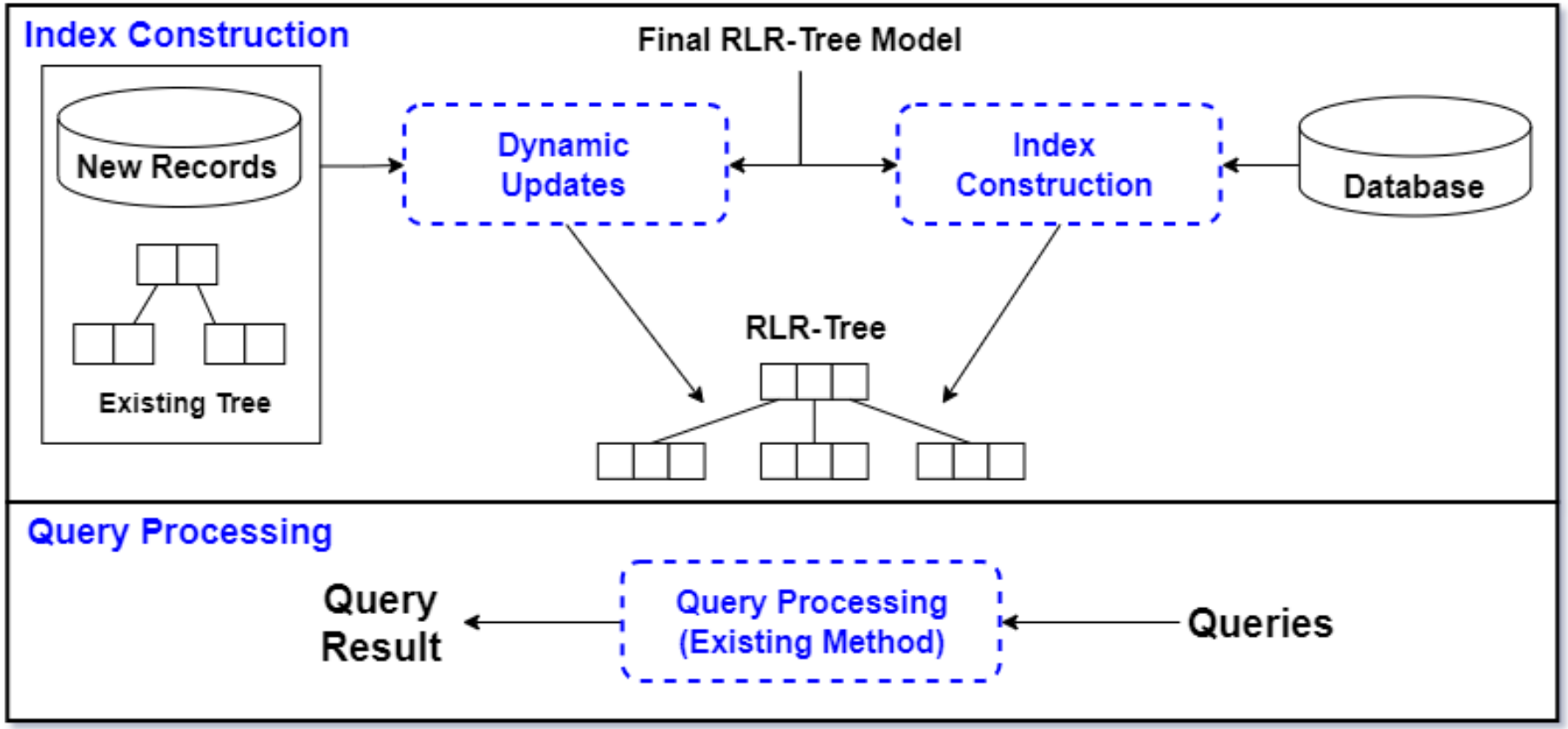
Train the RL agents for ChooseSubtree and Split together to further improve their performances.



Tu Gu et al. **The RLR-Tree: A Reinforcement Learning Based R-Tree for Spatial Data.** SIGMOD'23

RLR-Tree Overview

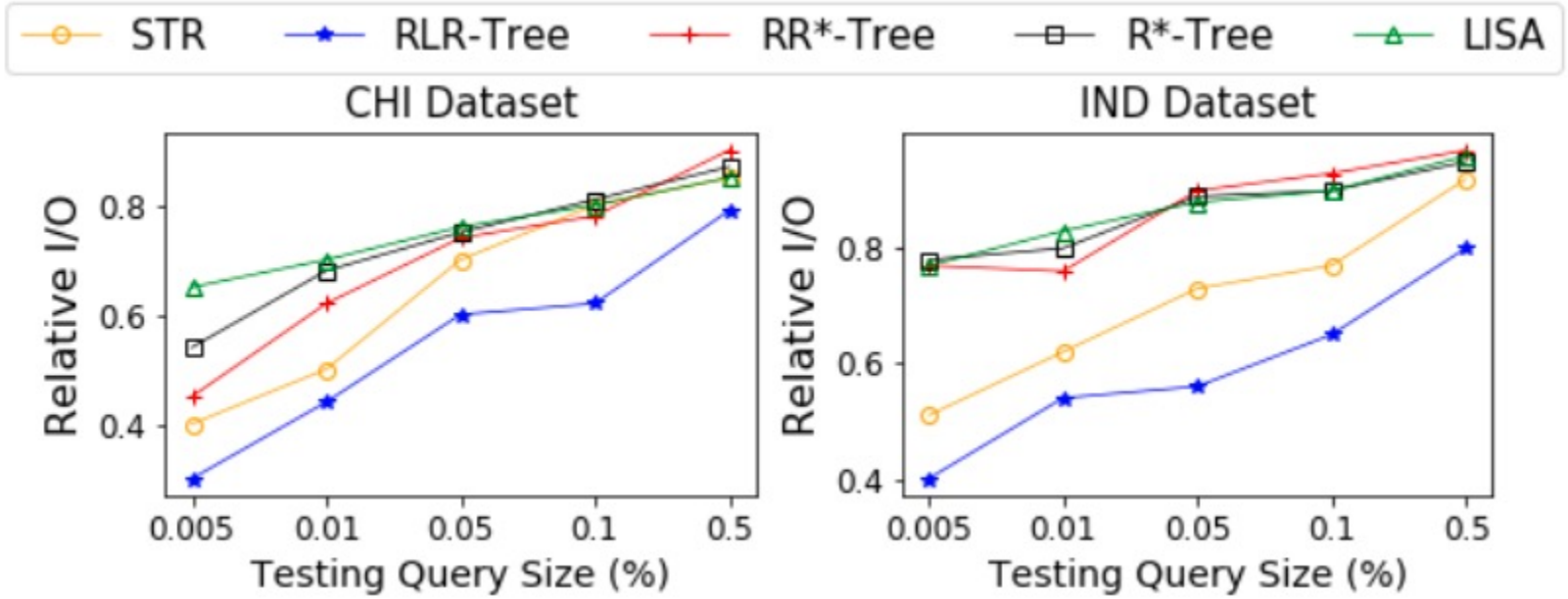
Overview (Index Construction & Query Processing):



Tu Gu et al. **The RLR-Tree: A Reinforcement Learning Based R-Tree for Spatial Data.** SIGMOD'23

Experimental Results

RLR-Tree outperform various R-tree variants on spatial queries.



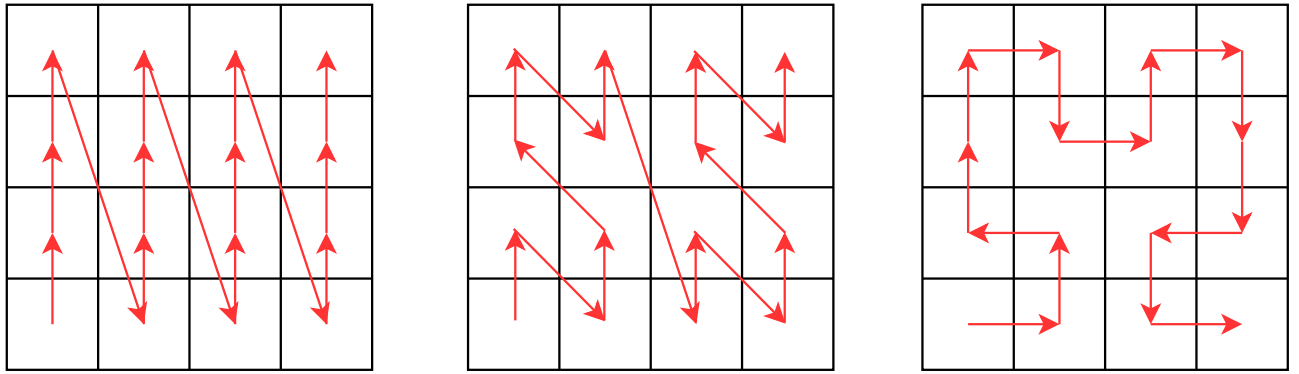
Tu Gu et al. **The RLR-Tree: A Reinforcement Learning Based R-Tree for Spatial Data.** SIGMOD'23

ML-enhanced Index

- We will discuss ML-enhanced methods that aid three types of index operations.
- **Insertion**: Building a better index for dynamic data.
  - RLR-tree: learning to improve R-tree insertion
  - BM-tree: learning to design space filling curve
- **Bulk-loading**: Constructing a better index through bulk-loading.
  - PLATON: R-tree packing with learned partition policy
- **Query**: Optimizing index search operation.
  - AI+R-tree: learning to accelerate R-tree search

Space-Filling Curve (SFC)

- A SFC is used to map a multi-dimensional data point to a 1d value
- Then a one-dimensional index can be used to index the mapped values
  - B+-tree index
  - Learned indexes



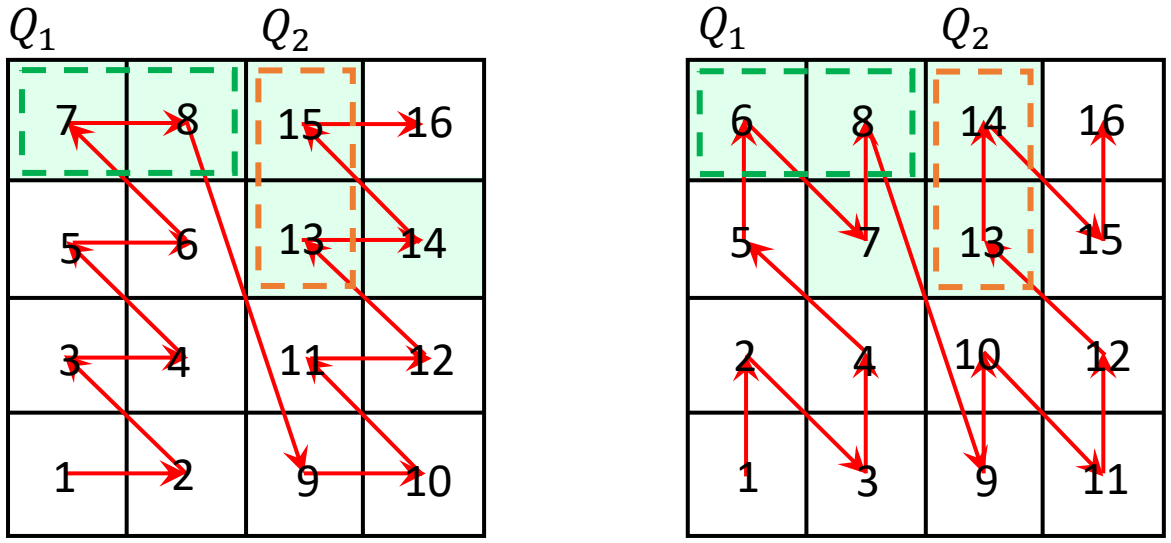
(a) C-curve (b) Z-curve (c) Hilbert curve

- Each type of SFC has a fixed mapping function
- May not fit with different datasets/queries.

Jiangneng Li et al. **Towards Designing and Learning Piecewise Space-Filling Curves.** VLDB'23

Design instance-optimized SFCs

- No single SFC can dominate the performance on all datasets and query workloads



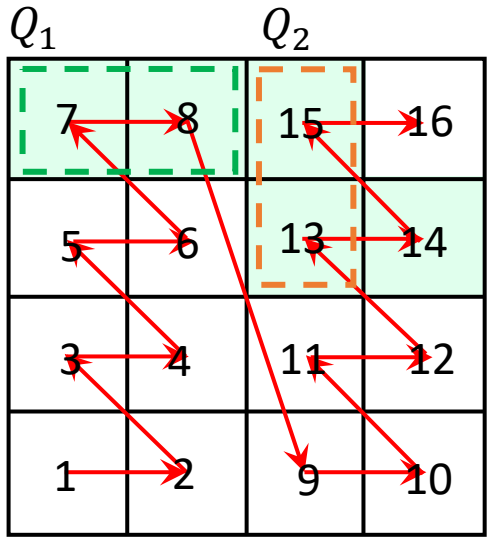
(a) SFC-1 works best for Q_1.

(b) SFC-2 works best for Q_2.

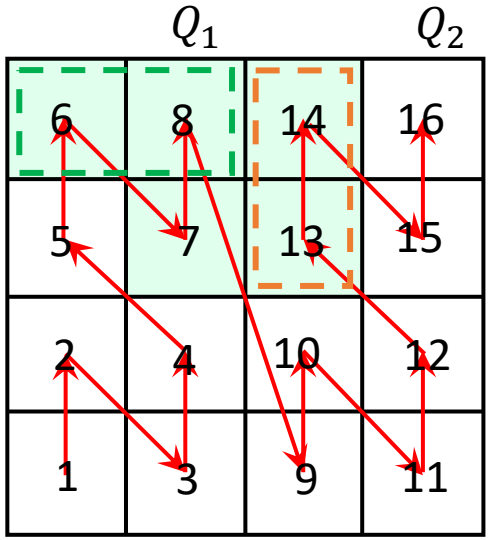
Jiangneng Li et al. **Towards Designing and Learning Piecewise Space-Filling Curves.** VLDB'23

Learning to design SFC

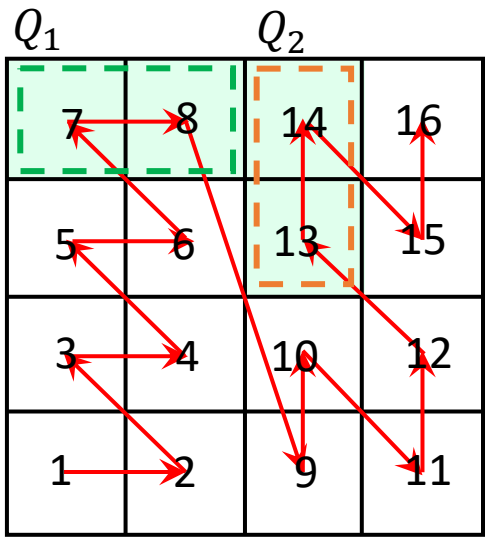
- Design a SFC that combines the advantage of multiple SFCs and thus reach to an optimized performance (**piecewise SFC**)



(a) SFC-1 works best for Q_1.



(b) SFC-2 works best for Q_2.



(c) SFC-3 combines SFC-1 and SFC-2, works best for both queries.

Jiangneng Li et al. **Towards Designing and Learning Piecewise Space-Filling Curves.** VLDB'23

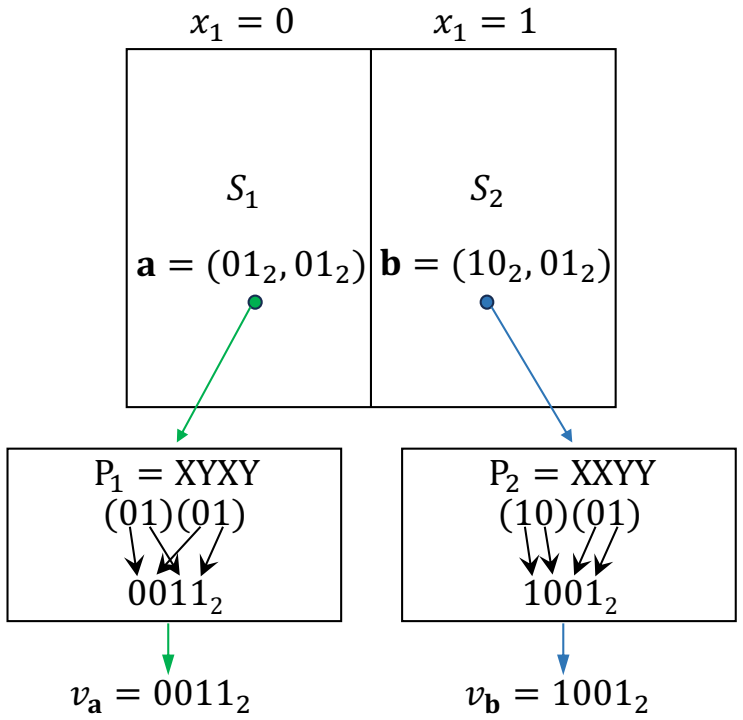
Problem Statement

- Database D
  - Each data point $\mathbf{x} \in D$, has n dimensions, denoted by $\mathbf{x} = (d_1, d_2, \ldots, d_n)$
- Query Workload Q
  - Each query $q \in Q$, $q = (x_{\min}, y_{\min}, x_{\max}, y_{\max})$
- Space-Filling Curve Design for Query Processing
  - Given a database D and a query workload Q, develop a **piecewise SFC**, aiming to optimize the performance of an index built on the SFC values of data points in D.

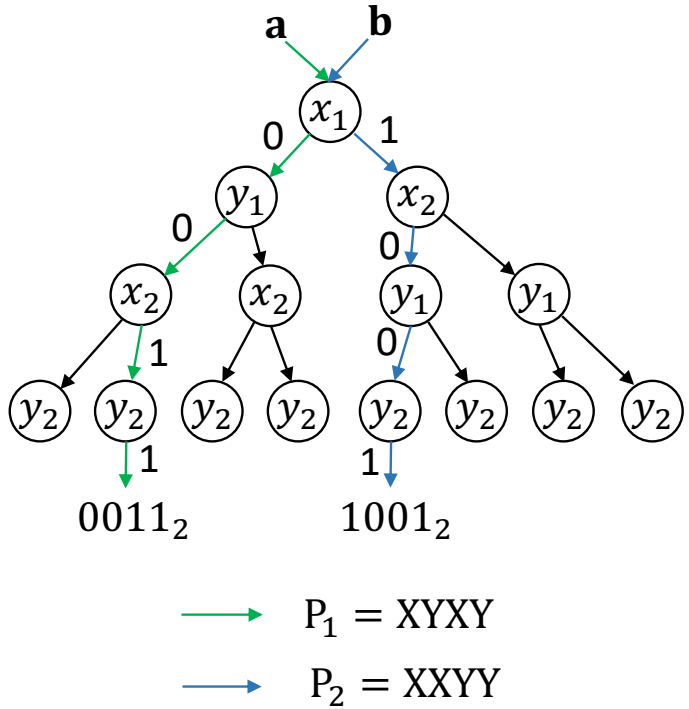
Jiangneng Li et al. **Towards Designing and Learning Piecewise Space-Filling Curves.** VLDB'23

Bit Merging Tree (BM-Tree) Overview

- **BM-Tree:** use **reinforcement to construct a piecewise space filling curve** for better query performance.



(a) Example of Piecewise SFC Design.

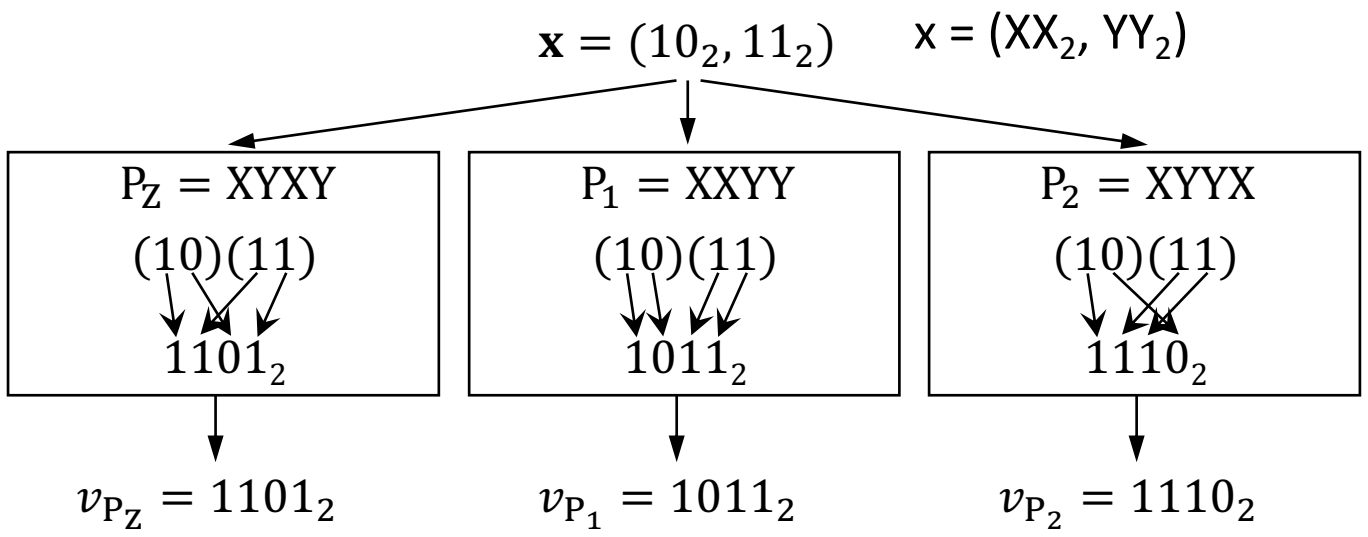


(b) Example of BMTree Structure.

Jiangneng Li et al. **Towards Designing and Learning Piecewise Space-Filling Curves.** VLDB'23

Bit Merging Pattern (BMP)

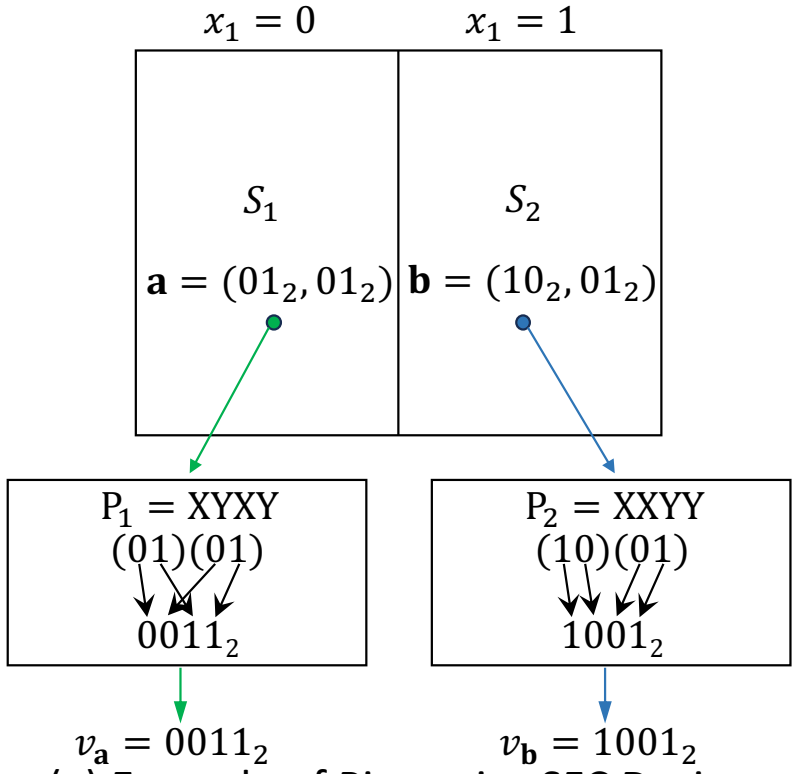
- The bit merging pattern (BMP, Nishimura & Yokota, SIGMOD'17) describes a set of bit merging-based SFCs.
  - The input data is first written as the binary form, then merge the bit according to the pattern (e.g., XYXY)



Jiangneng Li et al. **Towards Designing and Learning Piecewise Space-Filling Curves.** VLDB'23

Piecewise SFC Design

- A way of seamlessly integrating the subspace partitioning and BMP generation while ensuring the desired properties.



(a) Example of Piecewise SFC Design.

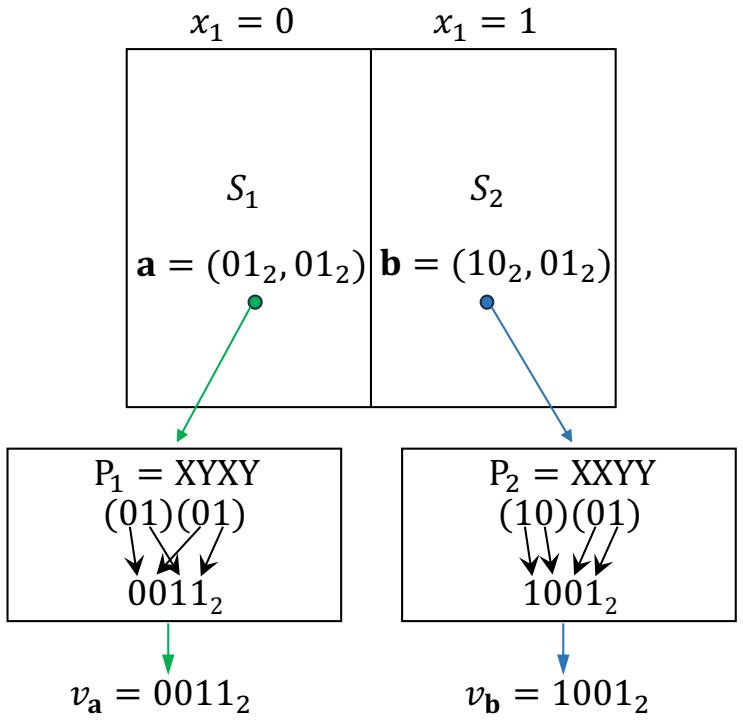
follow the left-to-right BMP design:

- choose the first bit x_1 for BMP P= X???.
- Then the whole data space is partitioned into two subspaces: Left subspace corresponds to $x_1 = 0$; right $x_1 = 1$
- Then separately design different BMPs for the two subspaces (S_1 and S_2).
- …

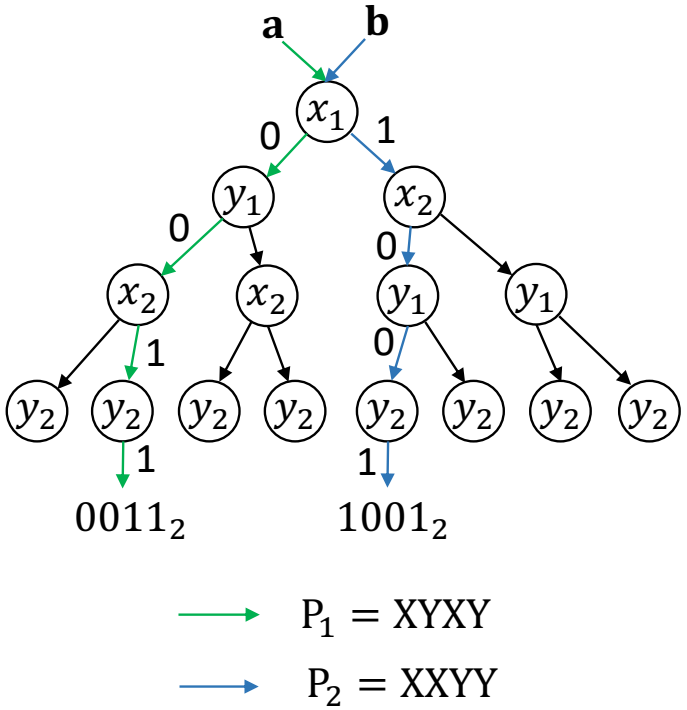
Jiangneng Li et al. **Towards Designing and Learning Piecewise Space-Filling Curves.** VLDB'23

Bit Merging Tree (BMTree)

- The BMTree is to model the partition and BMP design of a piecewise SFC.



(a) Example of Piecewise SFC Design.

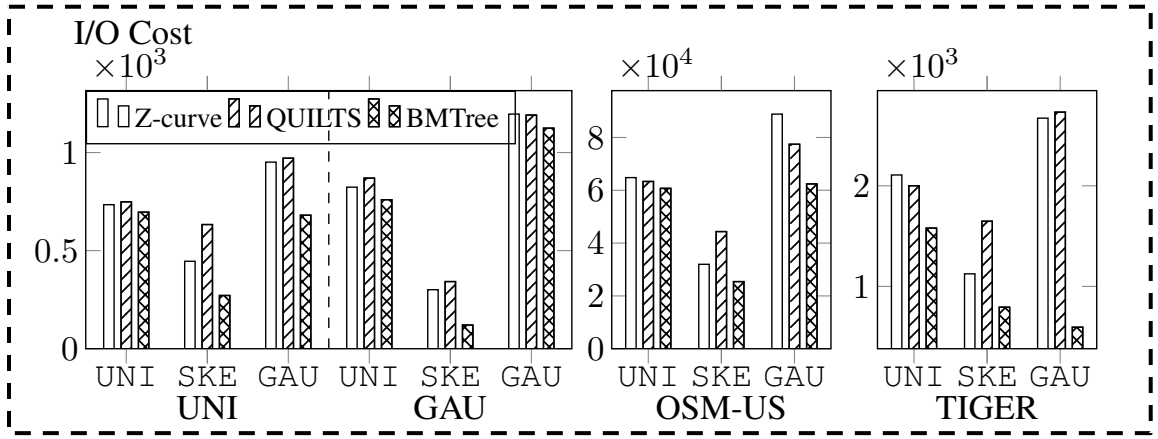


(b) Example of BMTree Structure.

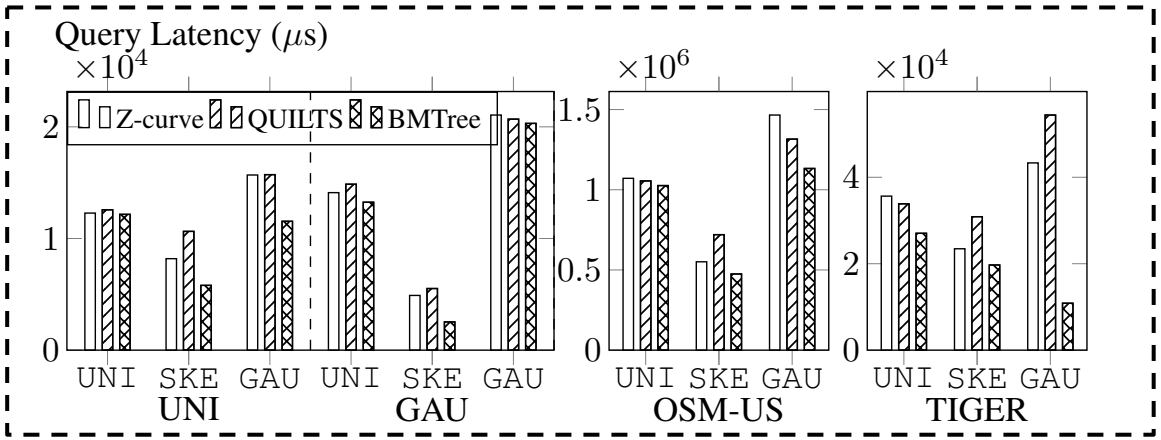
Jiangneng Li et al. **Towards Designing and Learning Piecewise Space-Filling Curves.** VLDB'23

BM-Tree performance

- Experiment on PostgreSQL.



(a) I/O Cost



(b) Query Latency

Jiangneng Li et al. **Towards Designing and Learning Piecewise Space-Filling Curves.** VLDB'23

ML-enhanced Index

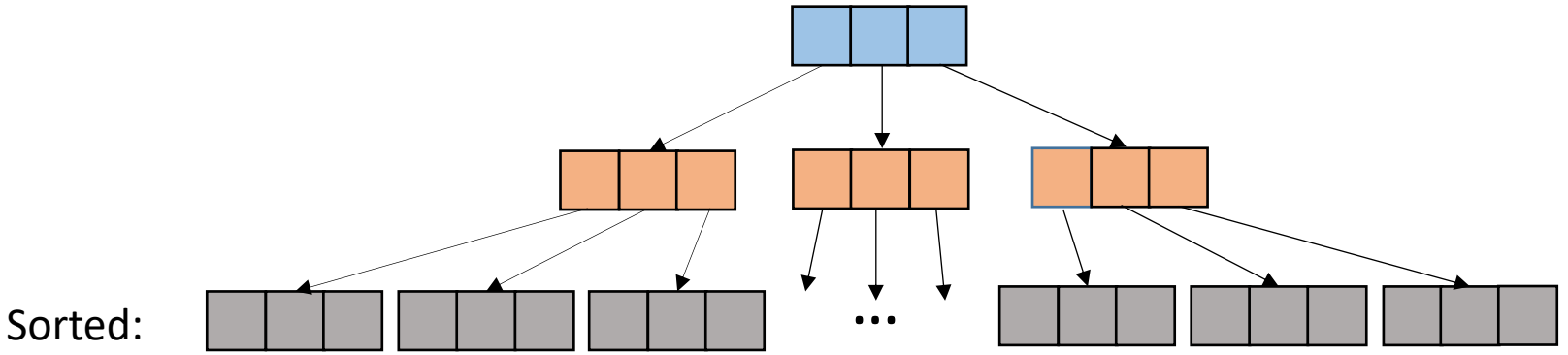
- We will discuss ML-enhanced methods that aid three types of index operations.
- **Insertion**: Building a better index for dynamic data.
  - RLR-tree: learning to improve R-tree insertion
  - BM-tree: learning to design space filling curve
- **Bulk-loading**: Constructing a better index through bulk-loading.
  - PLATON: R-tree packing with learned partition policy
- **Query**: Optimizing index search operation.
  - AI+R-tree: learning to accelerate R-tree search

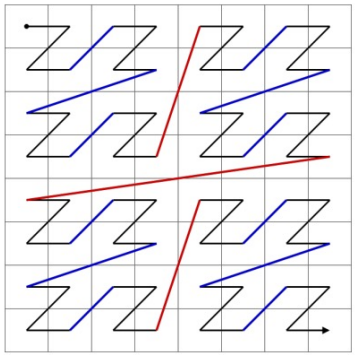
R-tree Bulk-loading/Packing

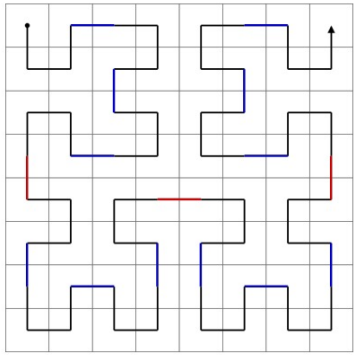
- R-tree packing is usually performed at index construction to build an R-tree with better **space utilization** and **query performance**.
  - **Bottom-up packing**: build the R-tree from leaf nodes upward by sorting the data objects in a heuristic order with a good clustering property.
  - **Top-down packing**: build the R-tree through recursive partitioning, with a partition policy that greedily optimizes a heuristic cost function.
- **The use of heuristic rules results in the constructed R-tree only performing well for certain data distribution and workload patterns.**

Bottom-up R-tree packing

- Bottom-up methods build the R-tree from leaf nodes upward by sorting the data objects in a heuristic order with a good clustering property.



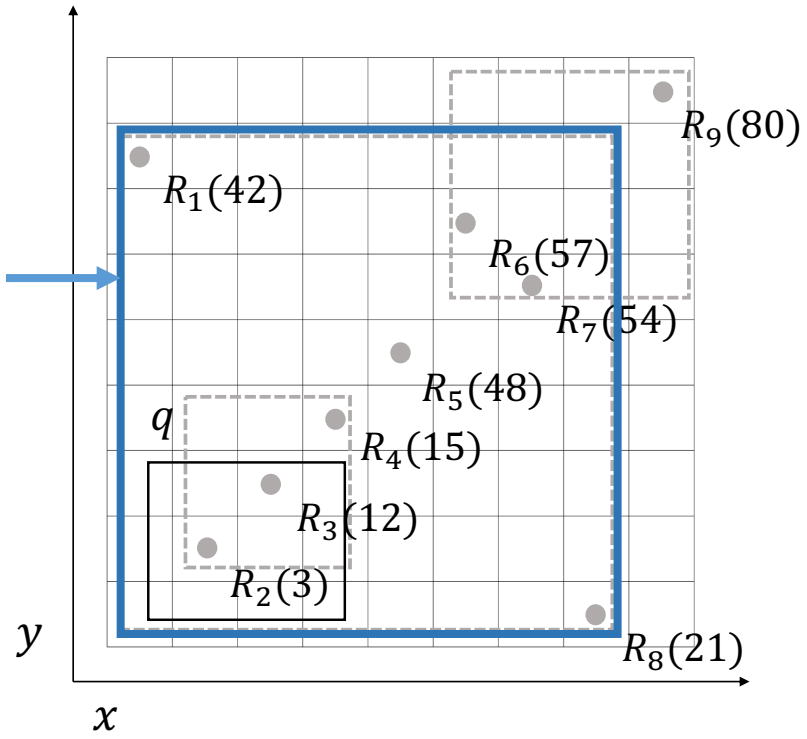




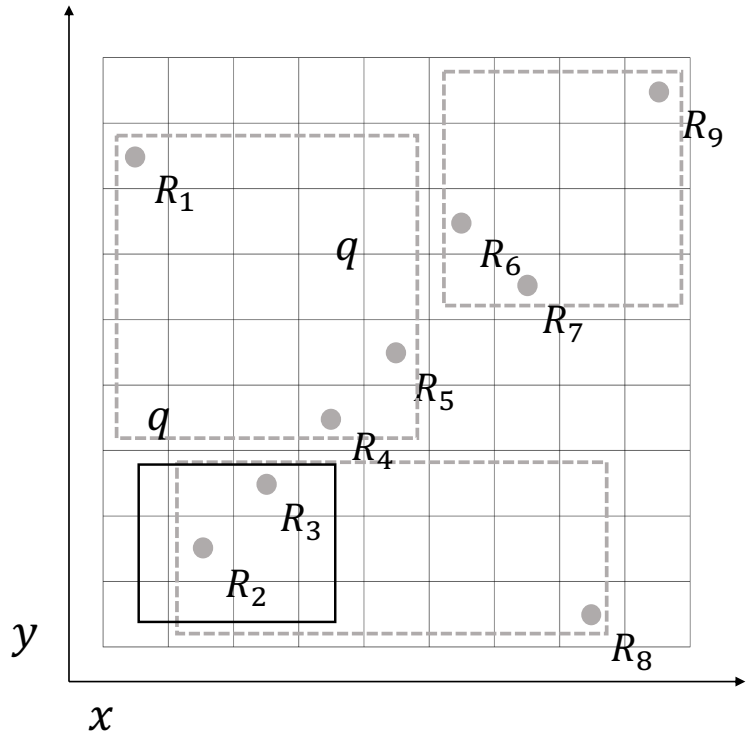
Bottom-up R-tree packing

- Heuristic sort order → only performing well for **certain data distributions and workload patterns**.

Bad nodes spanning a large space



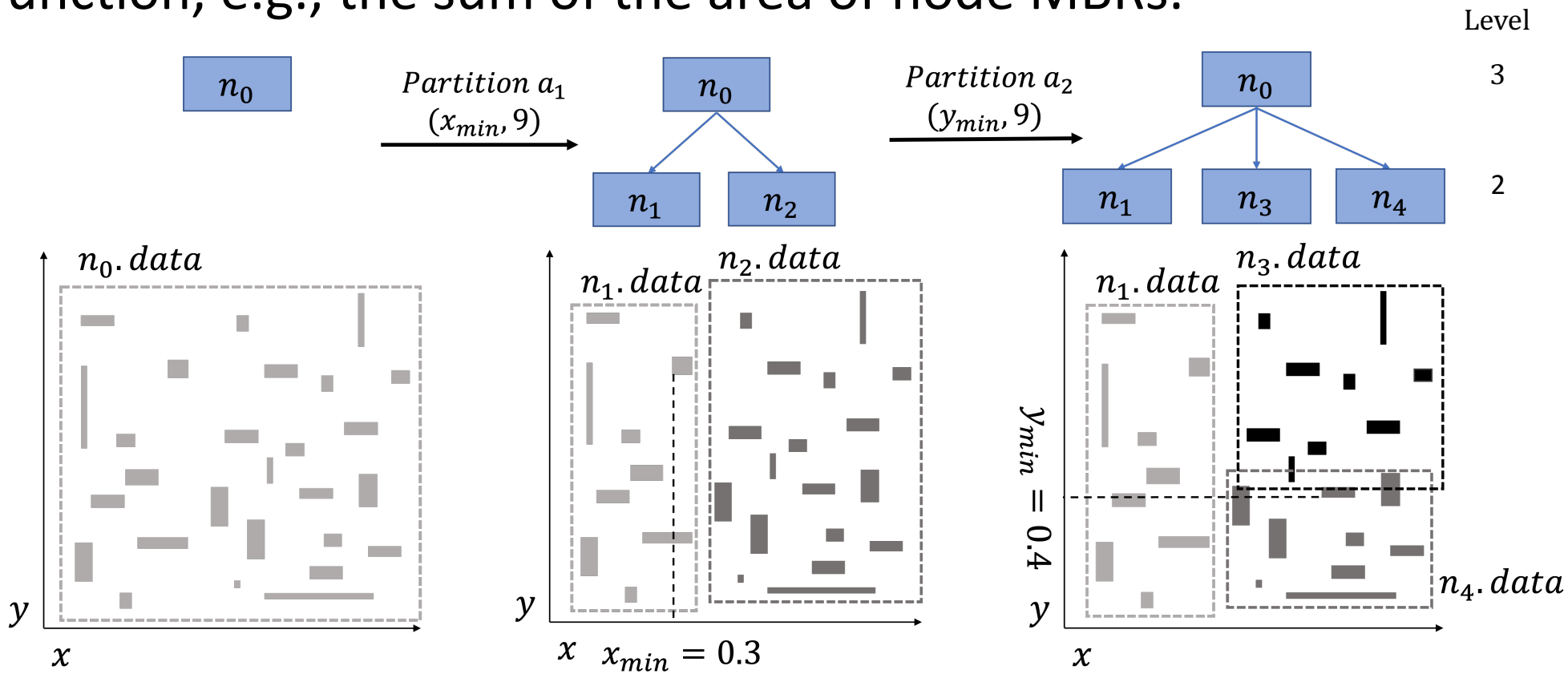
ZR



A better R-Tree

Top-down R-tree packing

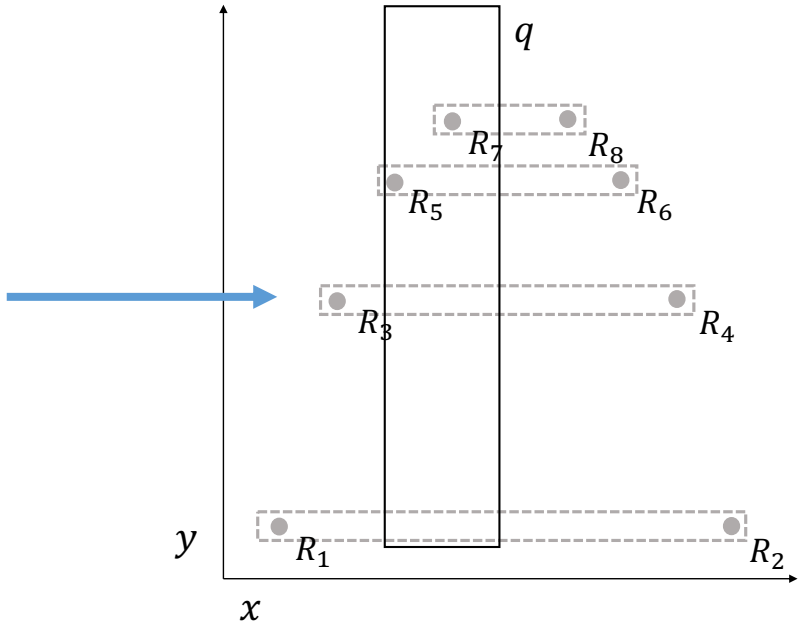
- Top-down methods build the R-tree through recursive partitioning, with a partition policy that greedily optimizes a heuristic cost function, e.g., the sum of the area of node MBRs.



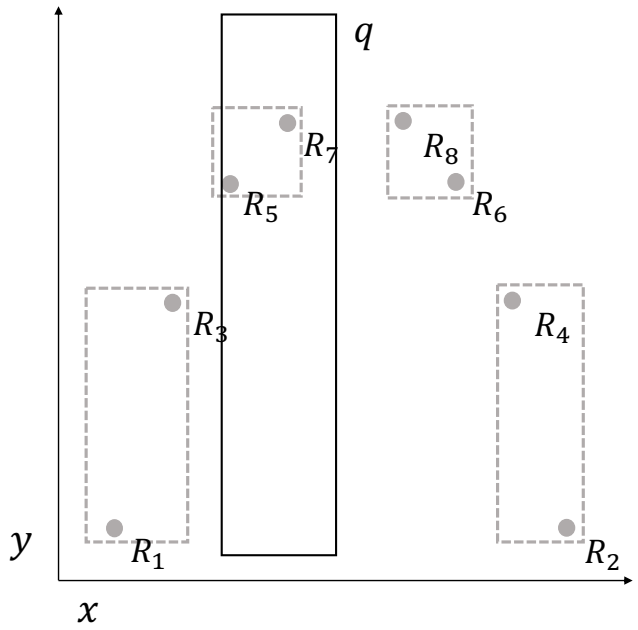
Top-down R-tree packing

- Heuristic cost function → not working well for **different data distributions and workload patterns**.
- The greedy partition policy also **ignores the dependencies** between the partition decisions on different nodes.

Long & thin stripes



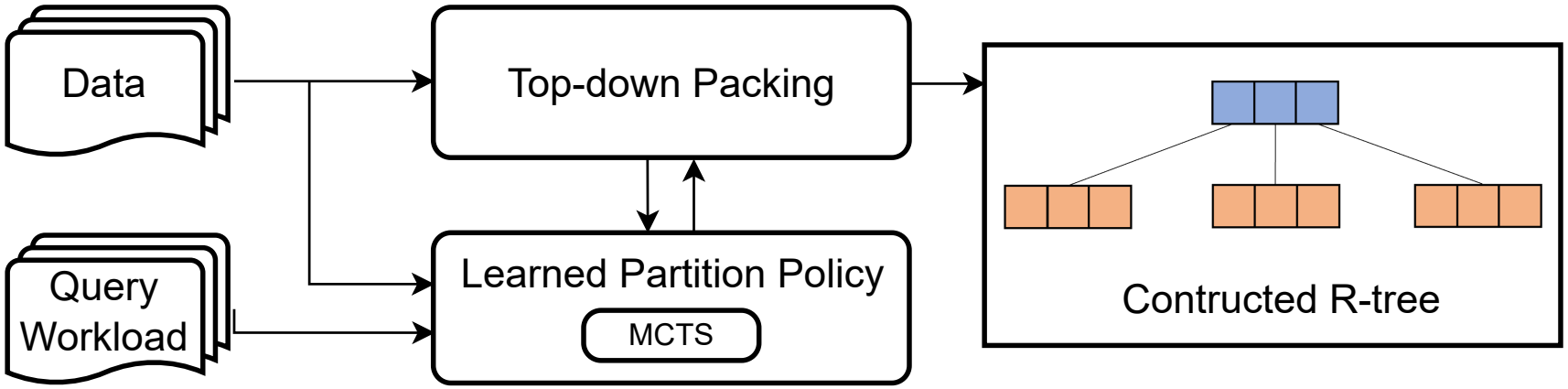
TGS



A better R-Tree

Learning to Pack R-tree

- **PLATON:** use machine learning to **optimize the partition policy** during **top-down packing** for better query performance.
  - PLATON adapts to different data distribution and workload patterns.
  - Light-weight model (Monte Carlo Tree Search) and performance-optimized algorithm



Jingyi Yang et al. **PLATON: Top-down R-tree packing with learned partition policy.** SIGMOD'24

Overview of PLATON

- PLATON is designed with the following goals.
  - Optimizing query performance.

  → Top-down packing as MDP, Monte Carlo Tree Search
  - Designing exploration and simulation strategy for better convergence.

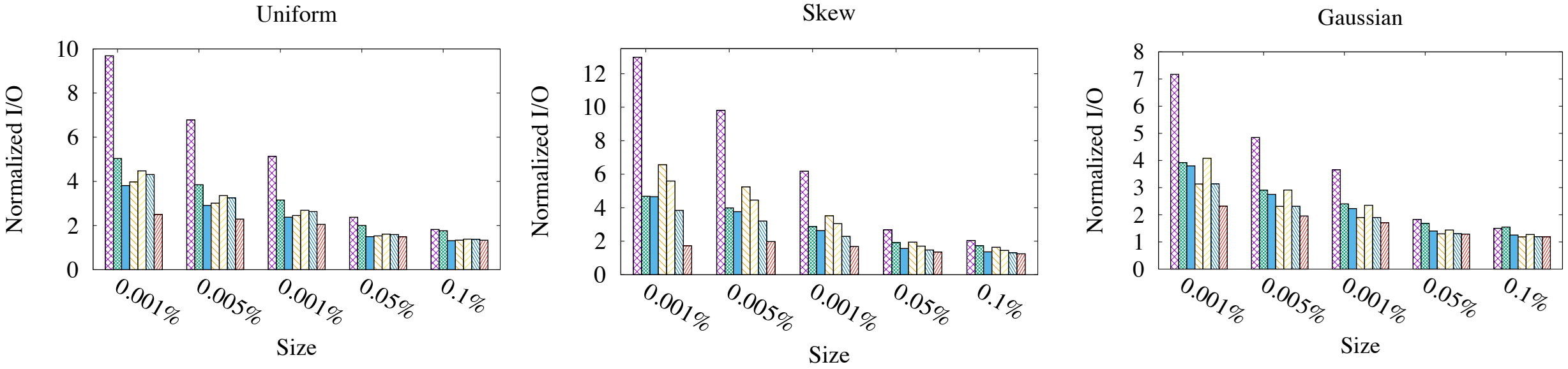
  → Incorporate domain knowledge into MCTS simulation
  - Efficiently learning the partition policy.

  → Divide and conquer strategy, two optimization techniques -> $O(n)$ $training$

Jingyi Yang et al. **PLATON: Top-down R-tree packing with learned partition policy.** SIGMOD'24

Comparison with R-tree variants

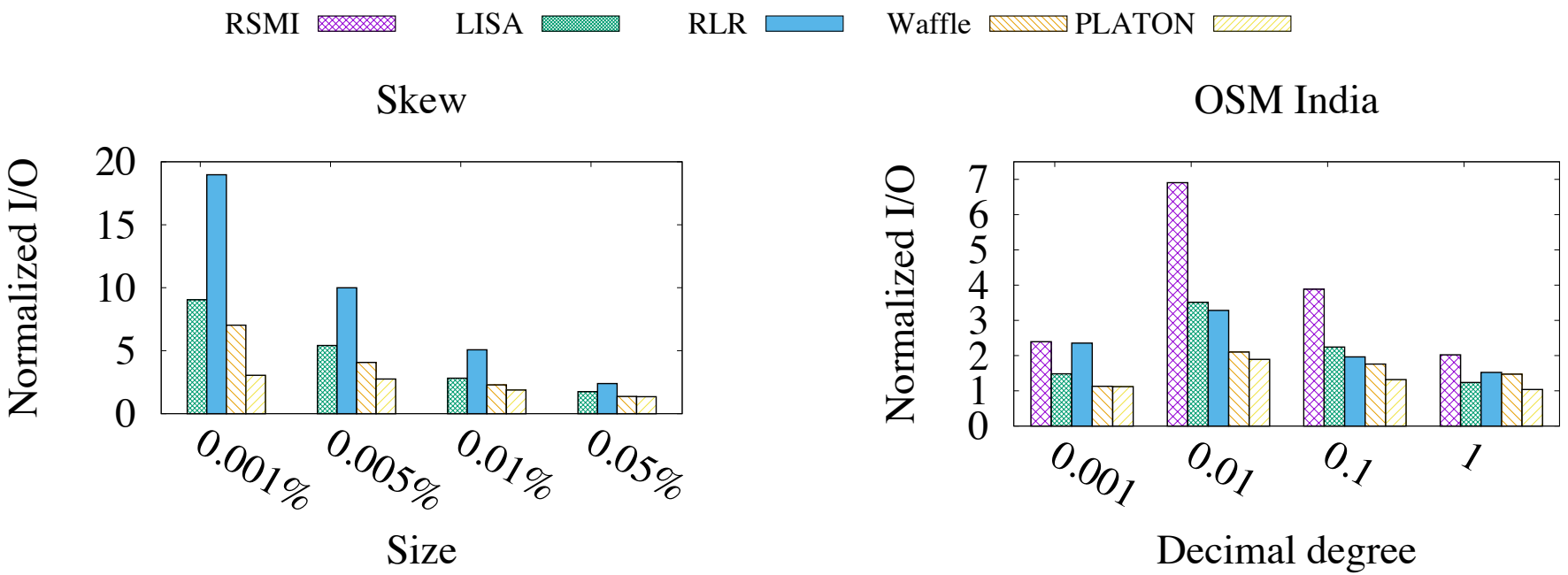
- PLATON significantly outperforms all existing R-tree variants for range query with varying sizes.
- It achieves a speed up of 1.52×, 2.70× and 1.35× compared to the best-performing baseline.



Jingyi Yang et al. **PLATON: Top-down R-tree packing with learned partition policy.** SIGMOD'24

Comparison with learned/workload-aware index

- PLATON achieves a speedup of up to 2.30× and 1.42× compared to the best performing baseline Waffle.



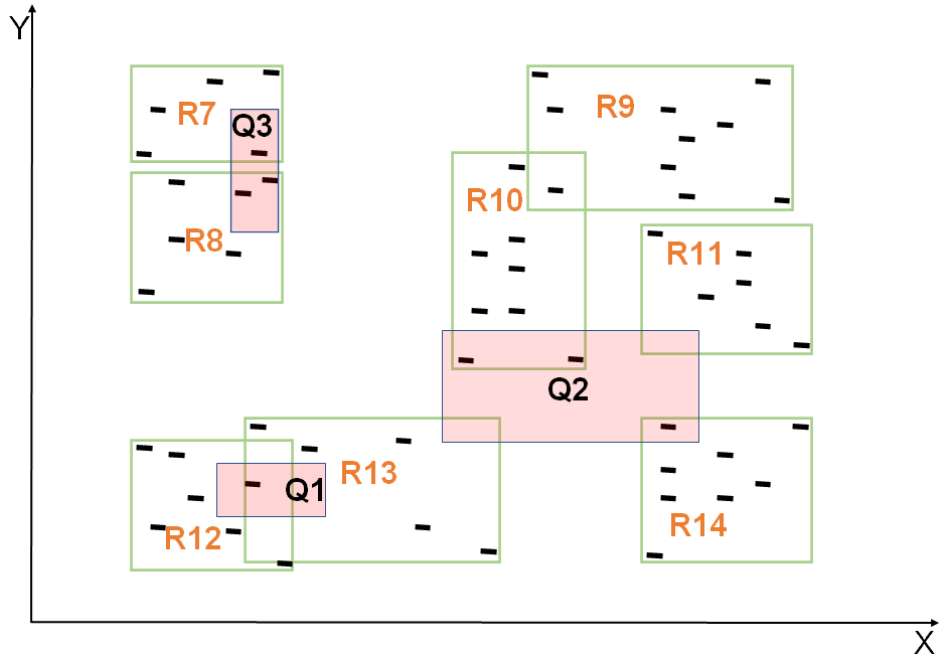
Jingyi Yang et al. **PLATON: Top-down R-tree packing with learned partition policy.** SIGMOD'24

ML-enhanced Index

- We will discuss ML-enhanced methods that aid three types of index operations.
- **Insertion**: Building a better index for dynamic data.
  - RLR-tree: learning to improve R-tree insertion
  - BM-tree: learning to design space filling curve
- **Bulk-loading**: Constructing a better index through bulk-loading.
  - PLATON: R-tree packing with learned partition policy
- **Query**: Optimizing index search operation.
  - AI+R-tree: learning to accelerate R-tree search

R-Tree Search

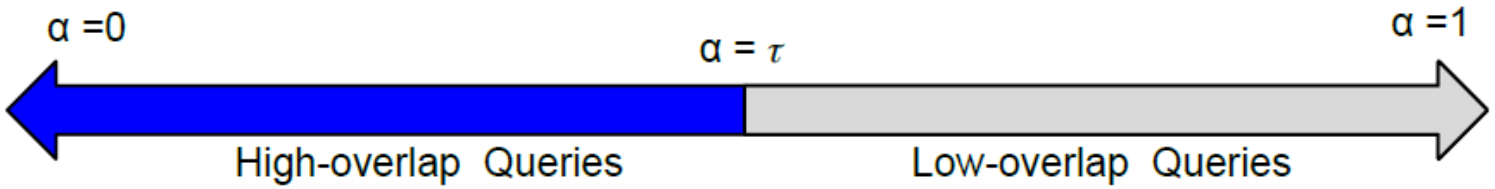
- In the R-tree, objects are stored using Minimum Bounding Rectangles (MBRs), and the MBRs of tree nodes can overlap in space.
- A query would access all leaf nodes with overlapping MBR, and some queries may access many extraneous node.
- **The excessive number of extraneous nodes accessed by high-overlap queries leads to high I/O cost.**



Query	Visited leaf nodes	True leaf nodes
Q1	R12, R13	R12
Q2	R10, R11, R13, R14	R10, R14
Q3	R7, R8	R7, R8

Overlap Ratio

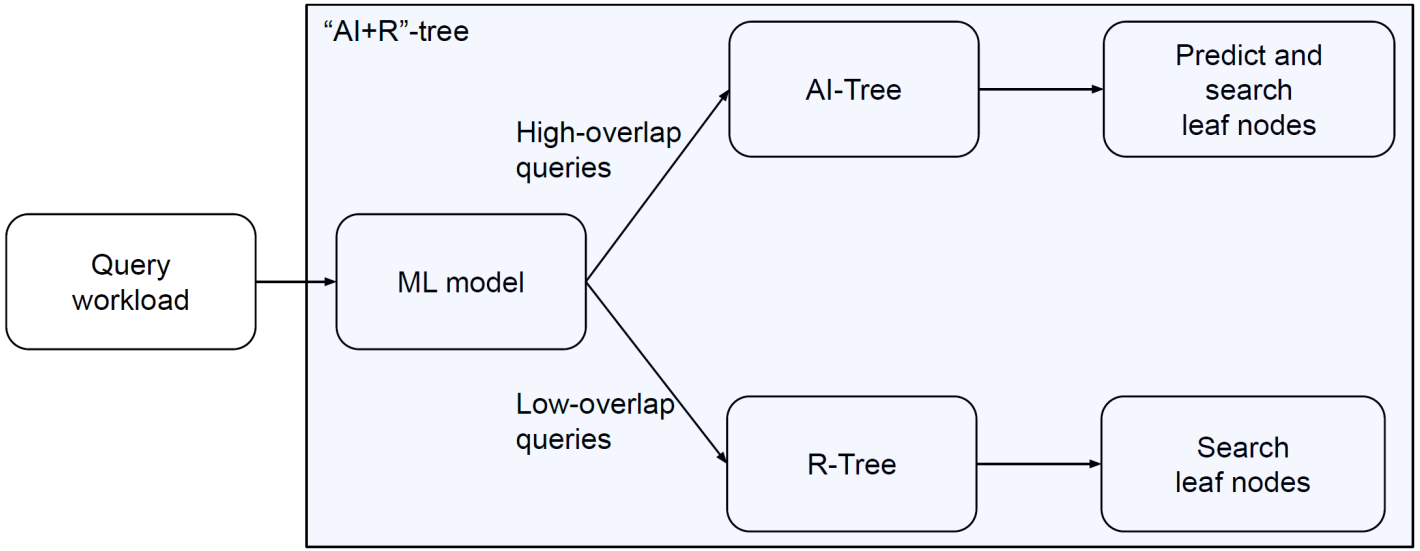
- Define an overlap ratio α to quantify the degree of extraneous leaf node accesses required by a range query.
- For a range query:
  - $\alpha = \frac{Number\ of\ true\ leaf\ nodes\ that\ actually\ contains\ the\ output\ data\ objects}{Number\ of\ leaf\ nodes\ searched\ by\ the\ R{-}Tree}$
  - α is in the range [0,1]
- High-overlap Queries:
  - A range query is declared as a "high-overlap query" if the value of α is less than a pre-defined threshold τ.



Mamun et al. **The "AI+R" tree: An instanced optimized R-tree.** MDM'22

AI+R-Tree: learning to accelerate R-tree search

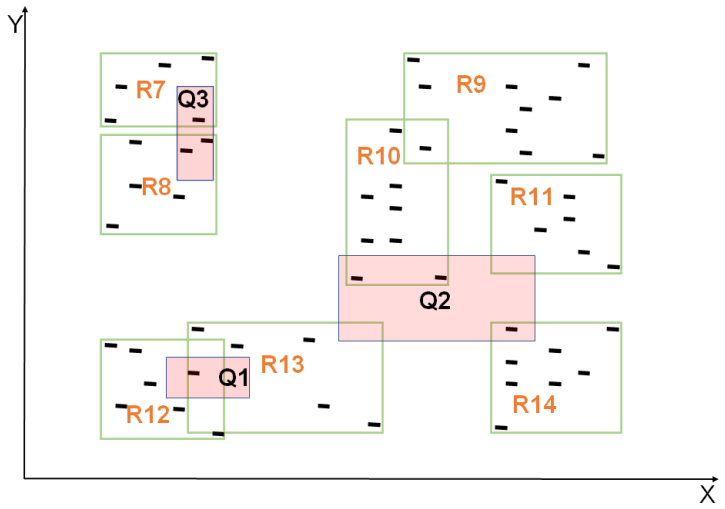
- Leveraging a binary classifier to automatically differentiate between high- and low-overlap queries.
- Process the high-overlap queries using the AI-tree.
- Process the low-overlap queries using the R-tree.



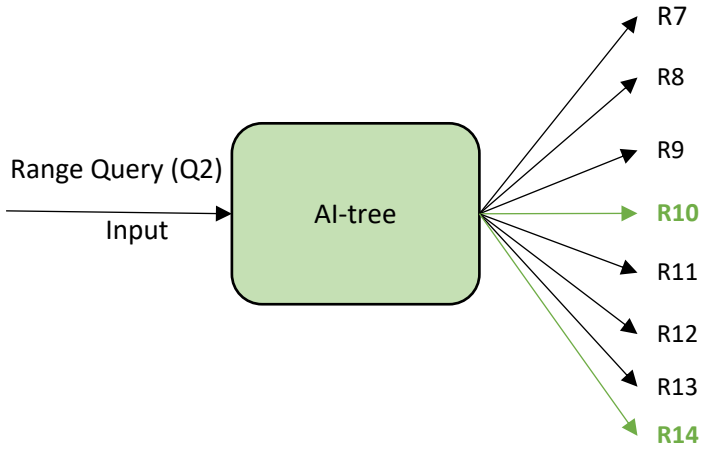
Mamun et al. **The "AI+R" tree: An instanced optimized R-tree.** MDM'22

AI+R-Tree: learning to accelerate R-tree search

- AI-Tree: predicting the node id that contains qualifying data objects
- Improve prediction accuracy with multiple space-partitioned models



An example of leaf nodes in an R-tree



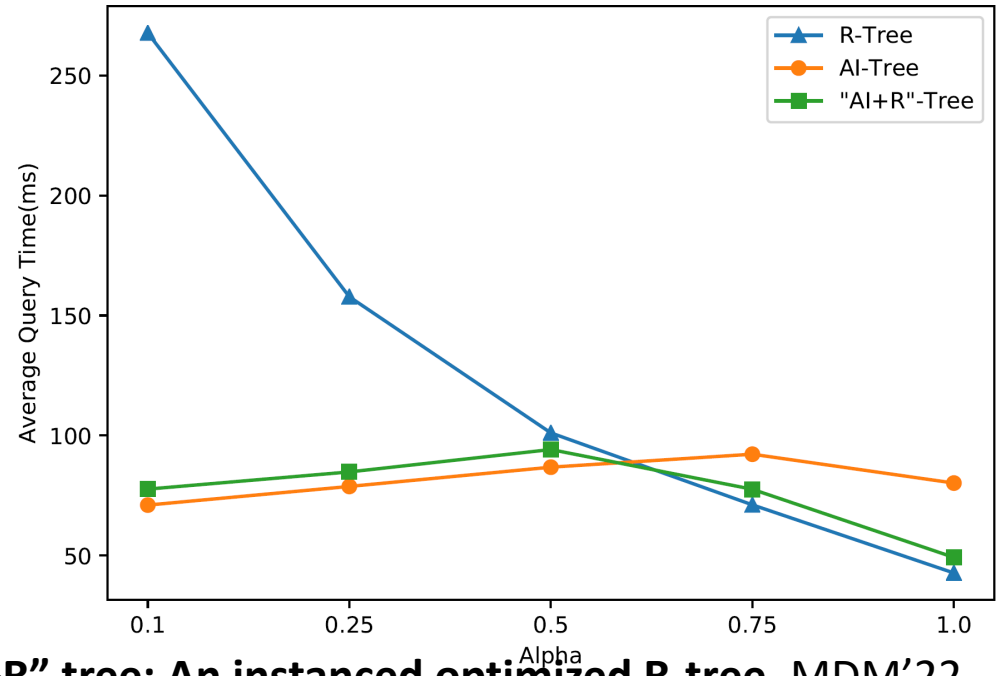
Total number of classes > 2
For an input, the number of output classes ≥ 0

M1 Q1	M2	M3	M4
M5	M6	M7	M8
M9	M10 Q2	M11	M12
M13	M14	M15	M16

Mamun et al. **The "AI+R" tree: An instanced optimized R-tree.** MDM'22

AI+R-Tree: learning to accelerate R-tree search

- For the high-overlap queries, the AI-tree performs better.
- For the low-overlap queries, the traditional R-tree performs better.
- The combined AI+R tree have more consistent performance.



Mamun et al. **The "AI+R" tree: An instanced optimized R-tree.** MDM'22

Summary

- ML-enhanced indexes are motivated by the need for more robust and generalizable learning-based indexes.

- Many of the index organization policies (implicitly) involve the use of heuristic rules, which can be formulated into a decision problem and optimized with ML/RL.
  - Node choosing and splitting policy in a multi-dimensional R-tree index
  - Data partition policy underlying a space-filling curve
  - Data sorting/partition policy in index bulk-loading

- Using Machine Learning model as “shortcuts” for inefficient operations.

ML4DB Paradigms

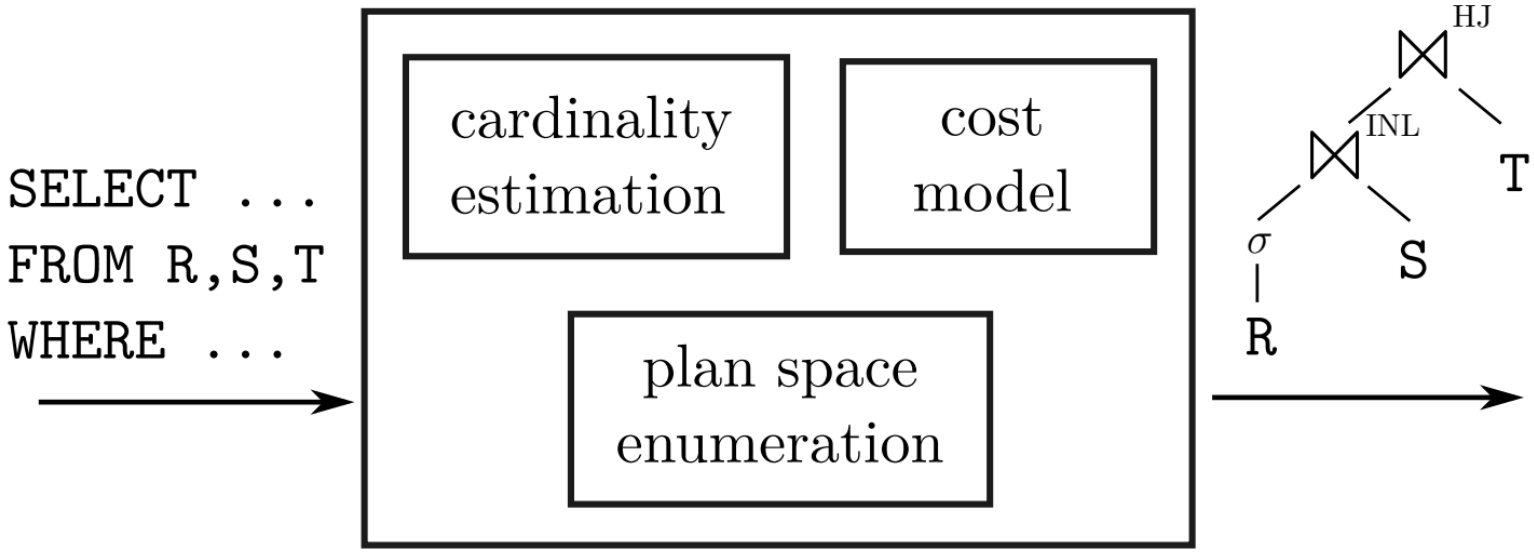
- We study ML4DB paradigms in the context of two important problem
  - Database indexing
  - Query optimization

ML4DB Paradigms in Query Optimization

- Part 1: Preliminary
- Part 2: Learned query optimizer
- Part 3: ML-enhanced query optimizer
- Part 4: Summary

Query optimizer

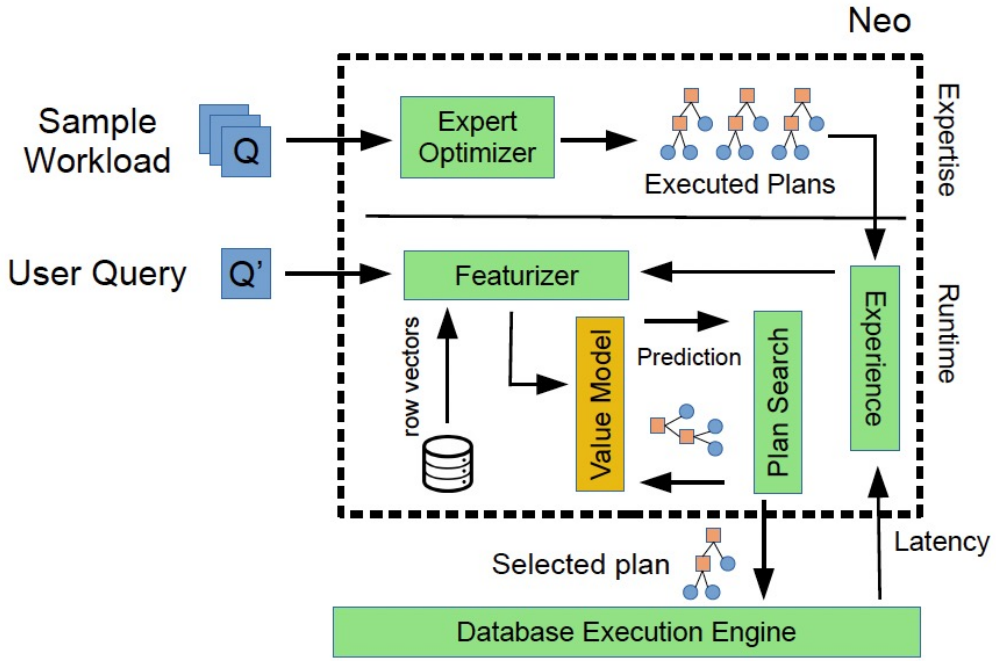
- A traditional database query optimizer consists of three components
  - Cardinality estimation: histogram
  - Cost estimation: formula-based
  - Plan enumeration: dynamic programming



Victor Leis et al. **How Good Are Query Optimizers, really?** VLDB'15

Learned query optimizer: NEO

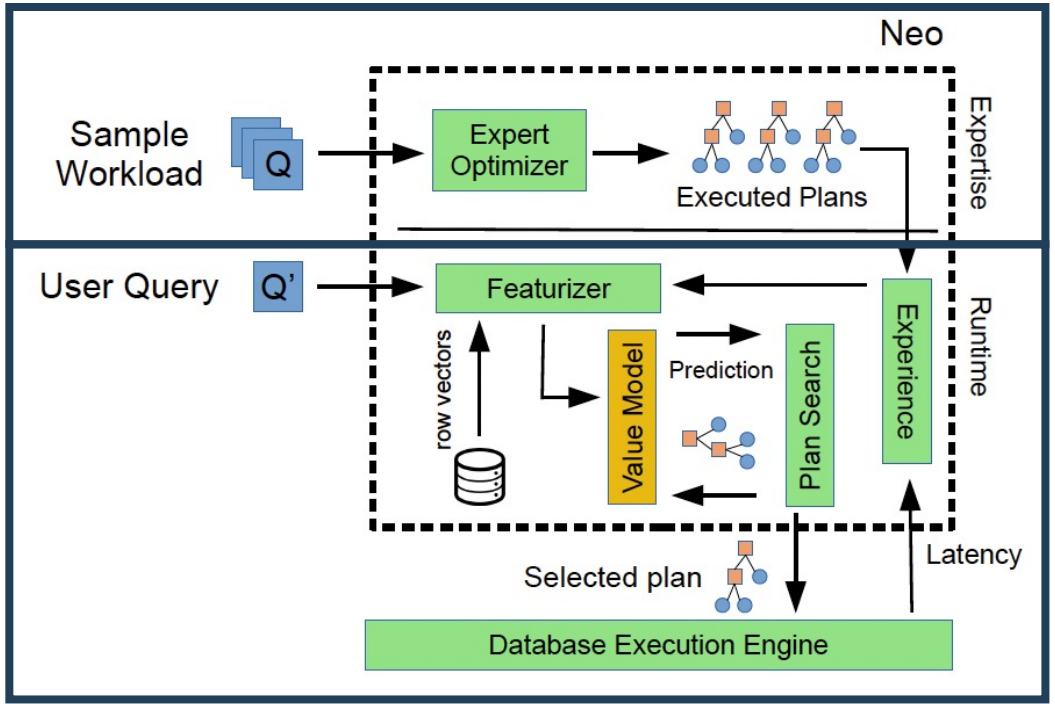
- Motivation: Query optimizers require a great deal of hand-tuning for specific workloads and datasets.
- NEO: Tailor the query optimizer to user's DB instance and workloads.



Ryan Marcus et al. **NEO: a learned query optimizer.** SIGMOD'19

Learned query optimizer: NEO

- Two phases
  - Initial phase: Learn a value network from an expert optimizer
  - Runtime phase: Plan search guided by the value network

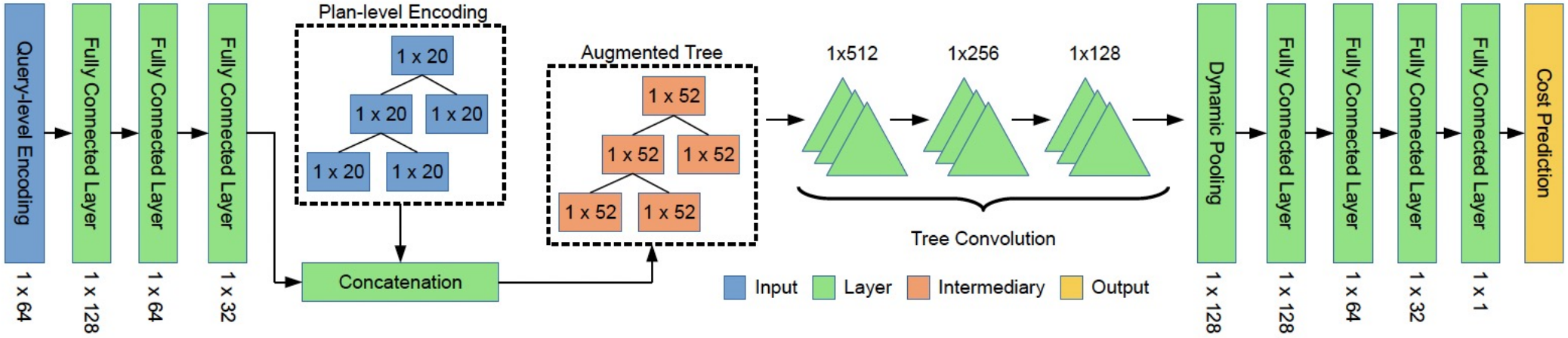


Ryan Marcus et al. **NEO: a learned query optimizer.** SIGMOD'19

Value Network

- Given a partial execution plan P_i, the value network predicts the best-possible query latency achievable by a complete execution plan P_f such that $P_i \subset P_f$

$$M(P_i) \approx \min\{C(P_f) \mid P_i \subset P_f \wedge P_f \in E\}$$



Ryan Marcus et al. **NEO: a learned query optimizer.** SIGMOD'19

Value network guided plan search

- Initialize an empty min heap to store partial execution plans
- Add partial execution plan with an unspecified scan for each relation into the heap
- At each search iteration
  - Remove the subplan P_i at the top of the min heap
  - Enumerate $P_i's$ Children, score them with the value network, add to heap
- Terminate when a complete plan is found, or when a time threshold is reached

Learned query optimizer: NEO

- Revisit the three components in NEO
  - Cardinality estimation: Tree convolution neural network (implicit)
  - Cost estimation: Tree convolution neural network
  - Plan enumeration: value network guided search

- The three components are jointly optimized in an end-to-end reinforcement learning pipeline.

Motivation of ML-enhanced query optimizer

- **Robustness**
  - Cold start problem: the performance of the model largely depends on the amount and diversity of training query plans, which are costly to gather
- **Complexity**
  - Huge plan space, hard to optimize, long training time
- **Generalization Capability**
  - Type of query: Select-Project-Join (SPJ) only
  - Learning-based optimizers are trained on a specific database with specific hardware and software configuration

- Can we improve the query optimizer without getting rid of the expert query optimizer in existing databases, e.g. PostgreSQL?

ML-enhanced Query Optimizer

- We will discuss three types of ML-enhanced query optimizer.

- **Bandit Optimizer**: Learning to steer query optimizer through hint-set.
- **Lero**: A Learning-to-Rank Query Optimizer
- **ParamTree**: Learning hyperparameters for formula-based cost model.

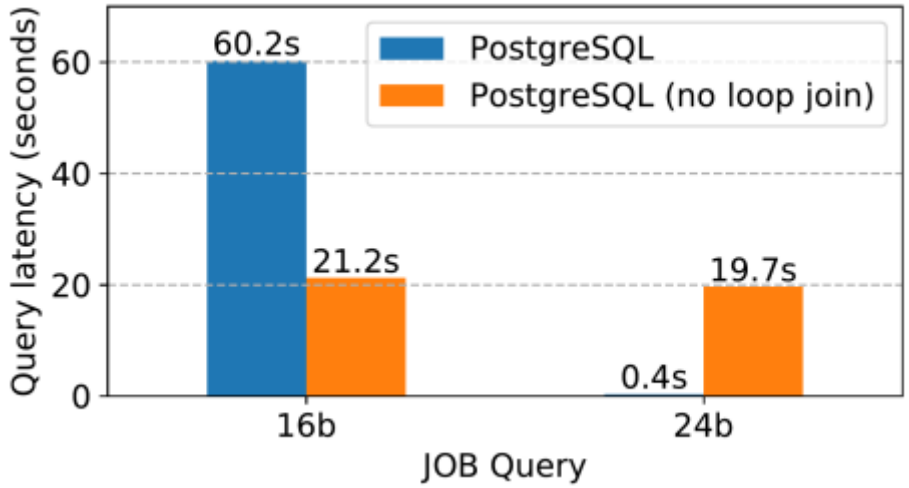
ML-enhanced Query Optimizer

- We will discuss three types of ML-enhanced query optimizer.

- **Bandit Optimizer**: Learning to steer query optimizer through hint-set.
- **Lero**: A Learning-to-Rank Query Optimizer.
- **ParamTree**: Learning hyperparameters for formula-based cost model.

Bandit Optimizer

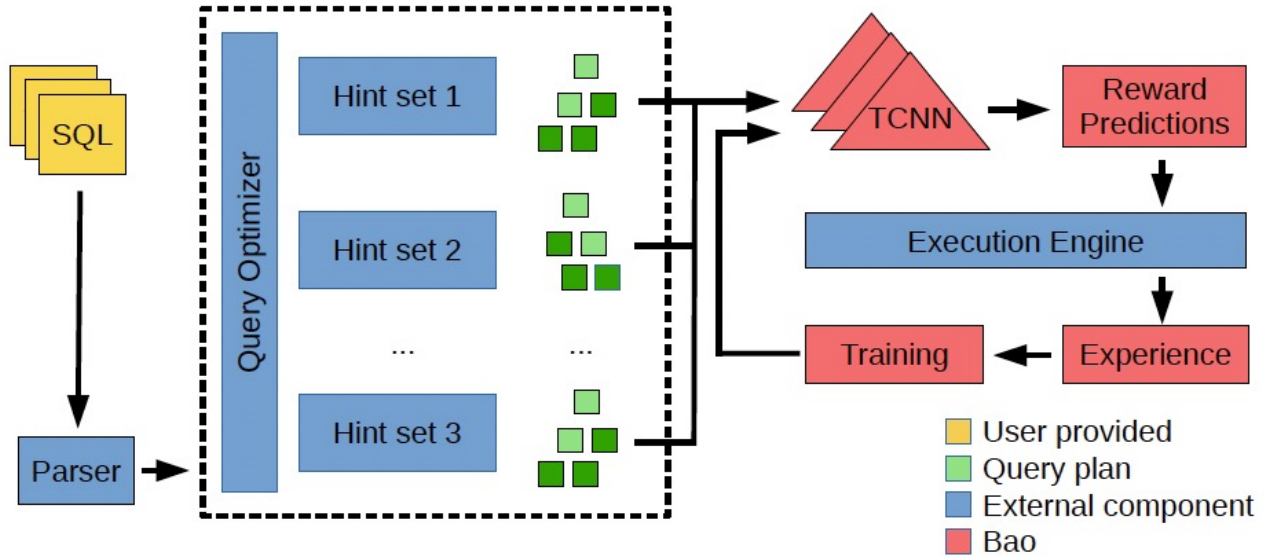
- Observation: Coarse-grained hints can improve the performance for certain types of queries
- Example: Cardinality under-estimates frequently prompt the query optimizer to select loop joins when other methods (merge, hash) would be more effective



Ryan Marcus et al. **BAO: Making learned query optimization practical.** SIGMOD'21

Bandit Optimizer

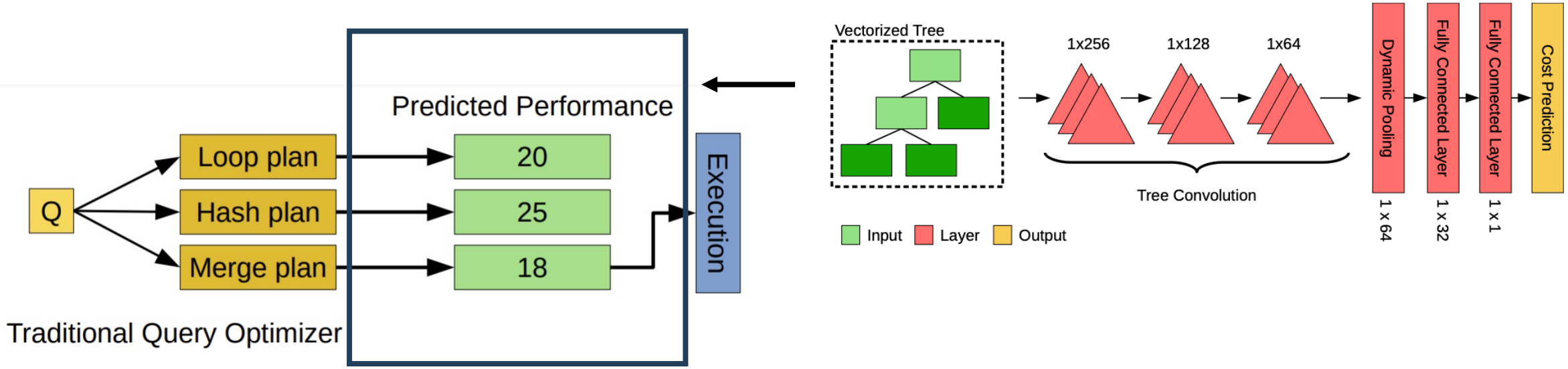
- Offload alternative plan enumeration
  - Only consider the best plan generated by expert optimizer for a hint set
- Model Plan Selection as a Contextual Multi-armed bandit



Ryan Marcus et al. **BAO: Making learned query optimization practical.** SIGMOD'21

Enumerate Query Hints

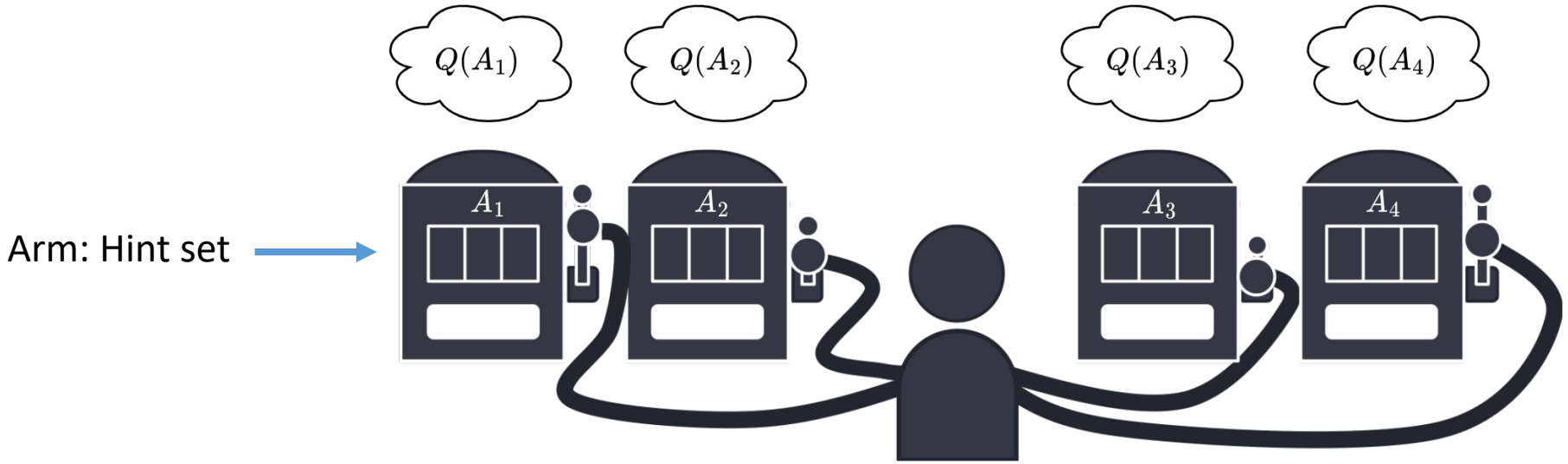
- Query Hint Set: A combination of configuration knobs
- Query plans are enumerated using different query hint sets
- Bao determines the optimal hint with a Tree CNN based predictive model



Ryan Marcus et al. **BAO: Making learned query optimization practical.** SIGMOD'21

Contextual Multi-armed bandit

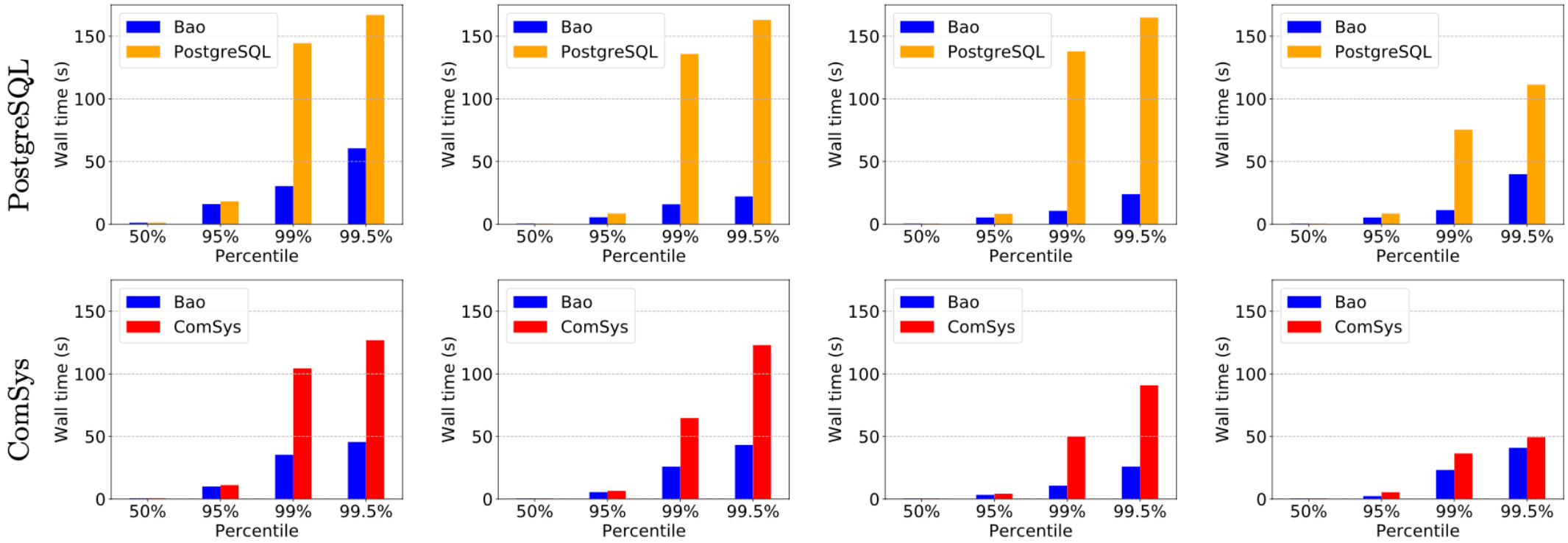
- Context: The set of query plans produced by the underlying optimizer given each hint set
- Exploration-exploitation tradeoff



Ryan Marcus et al. **BAO: Making learned query optimization practical.** SIGMOD'21

Bandit Optimizer

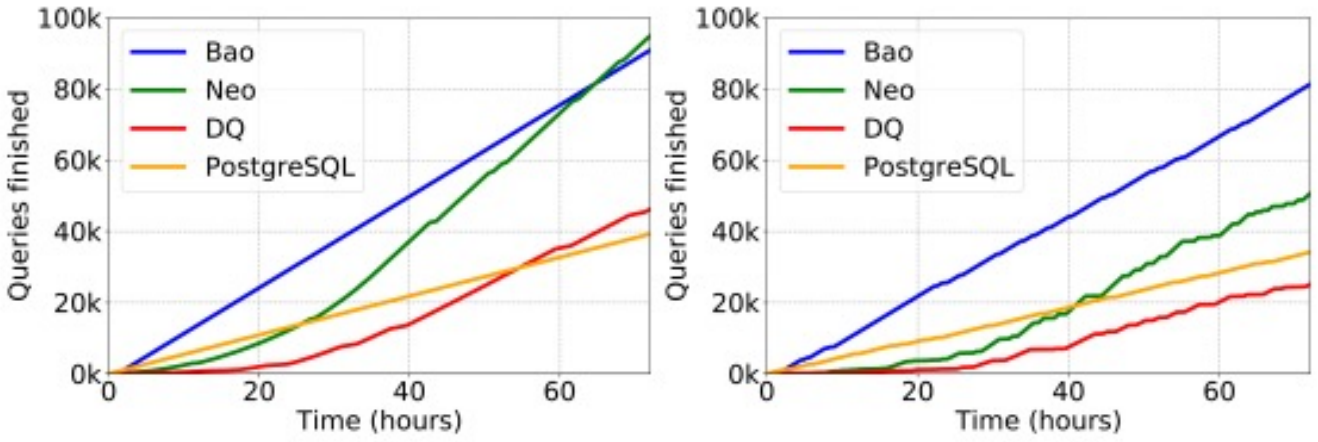
- BAO drives significant improvement in runtime, especially at tail



Ryan Marcus et al. **BAO: Making learned query optimization practical.** SIGMOD'21

Bandit Optimizer

- With a stable workload, BAO converges much faster than NEO
- BAO better adapts to change compared to learned query optimizers



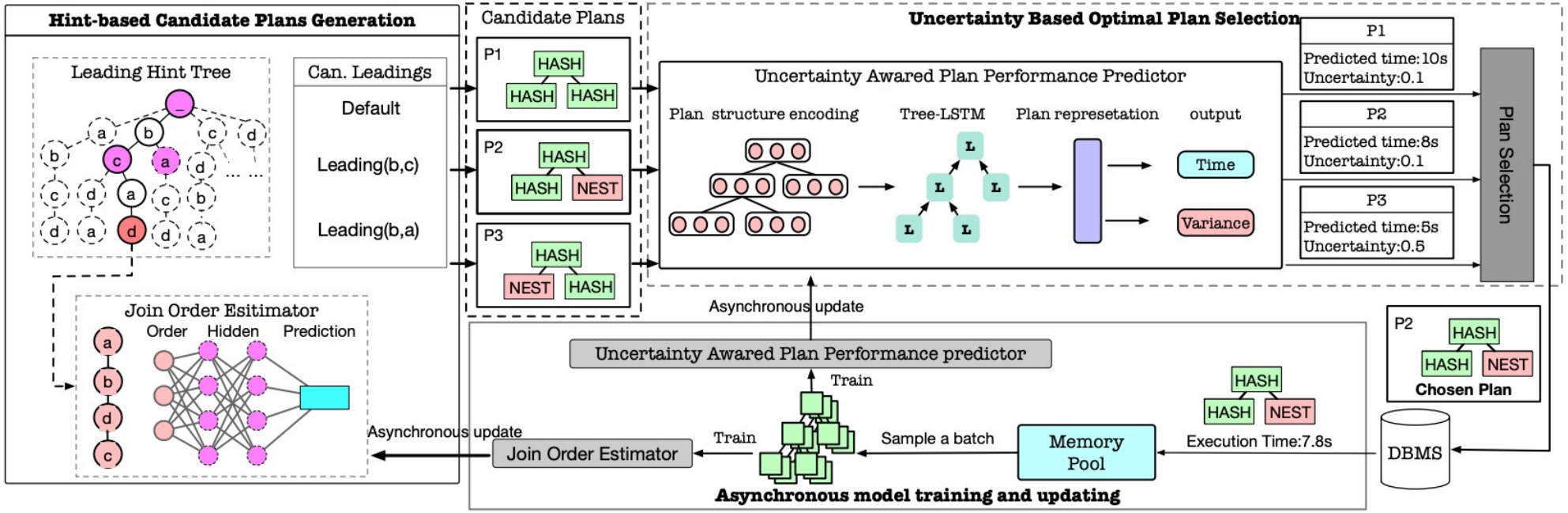
Ryan Marcus et al. **BAO: Making learned query optimization practical.** SIGMOD'21

Bandit Optimizer

- Revisit the three components in BAO
- Cardinality estimation: not needed
- Cost estimation: Tree CNN
- Plan enumeration: hint set + Expert query optimizer

Hybrid Query Optimizer - HYBRIDQO

- Select leading hint through machine learning, i.e. MCTS
- Uncertainty based model for plan selection



Xiang Yu et al. **Cost-based or Learning-based? A Hybrid Query Optimizer for Query Plan Selection.** VLDB'23

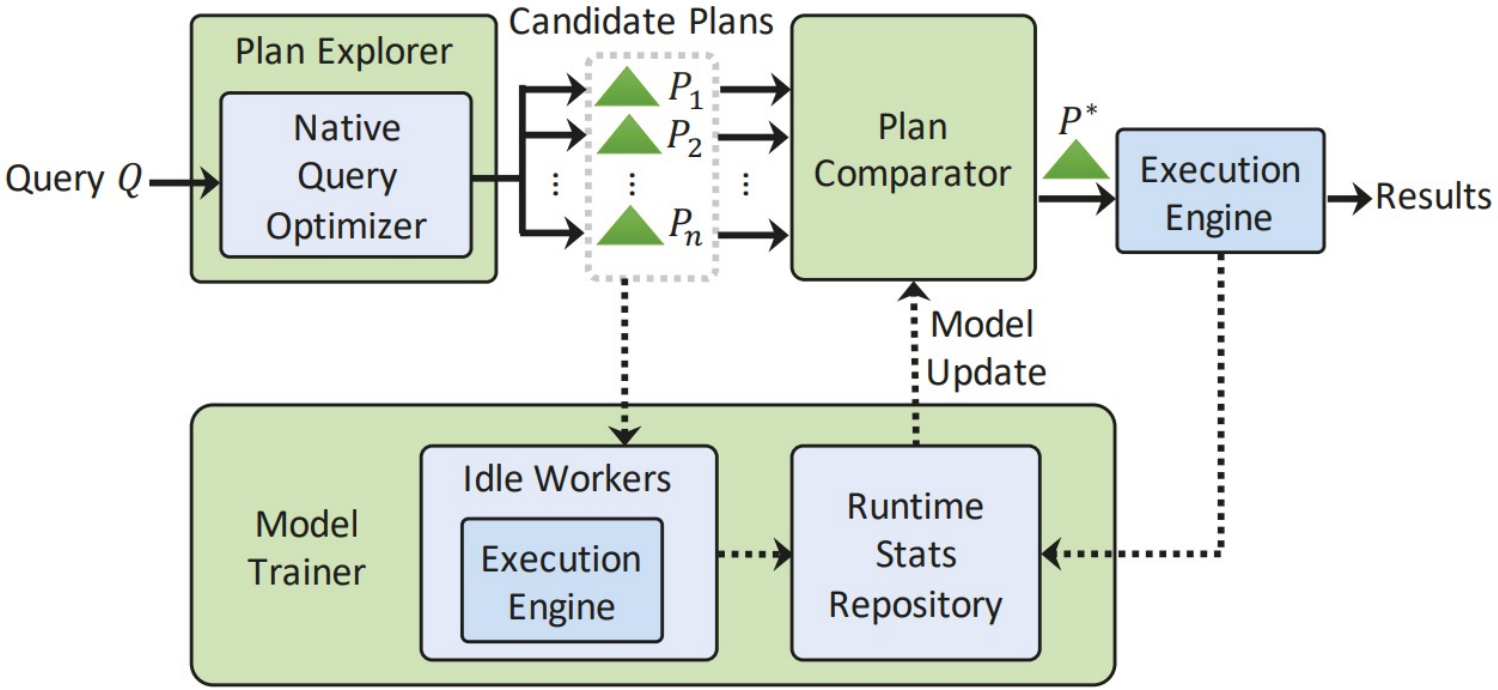
ML-enhanced Query Optimizer

- We will discuss three types of ML-enhanced query optimizer.

- **Bandit Optimizer**: Learning to steer query optimizer through hint-set.
- **Lero**: A Learning-to-Rank Query Optimizer.
- **ParamTree**: Learning hyperparameters for formula-based cost model.

Lero: Learning to Rank Plans

- Lero enumerate candidate plans by adjusting subplan cardinalities
- Using a pairwise plan comparator model to choose the best plan

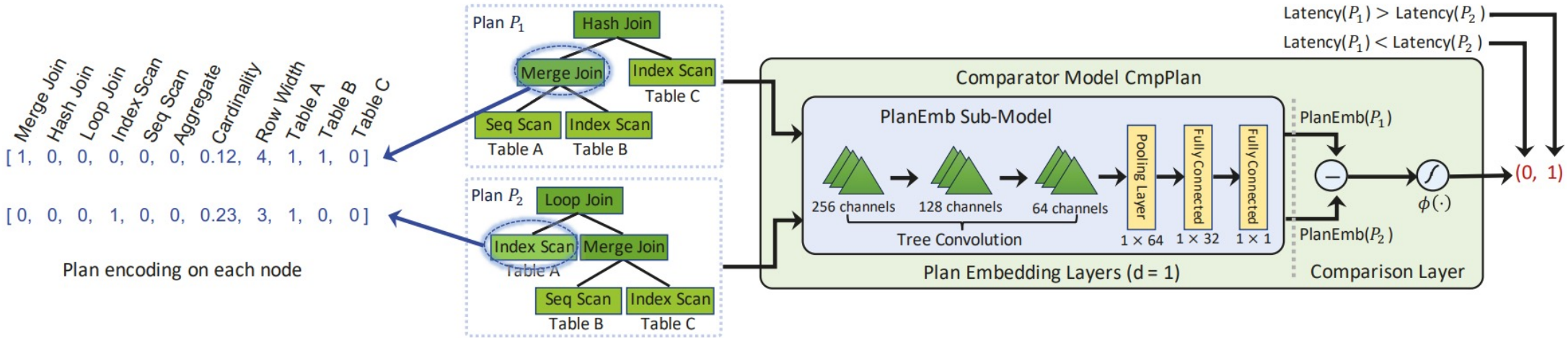


Rong Zhu et al. **Lero: A Learning-to-Rank Query Optimizer.** VLDB'23

Lero: Learning to Rank Plans

- Plan Comparator Model

$$CmpPlan(P_1, P_2) = \begin{cases} 0 \; if \; Latency(P_1) < Latency(P_2) \\ 1 \; if \; Latency(P_1) > Latency(P_2) \end{cases}$$



Rong Zhu et al. **Lero: A Learning-to-Rank Query Optimizer.** VLDB'23

Cardinality as Knob for Plan Explorer

- Lero explore plans by adjusting the sub-plan cardinalities
- Advantages of tuning cardinality:
  - Cardinality directly affect the query plan, introduce diversity
  - Platform-independent

Algorithm plan_explorer(Q, α, Δ)

1: Priority queue candidate_plans $\leftarrow \varnothing$
2: **for** each $f \in F_\alpha^\Delta$ in the increasing order of $|\log f|$ **do**
3: **for** $k \leftarrow 1$ to q (the number of tables in Q) **do**
4: **Inside query optimizer:** let $\mathrm{C}()$ be the default cardinality estimator in native query optimizer
5: $\widetilde{\mathrm{C}}(Q') \leftarrow f \cdot \mathrm{C}(Q')$ for size-k sub-queries $Q' \in \mathrm{sub}_k(Q)$
6: $\widetilde{\mathrm{C}}(Q') \leftarrow \mathrm{C}(Q')$ for sub-queries $Q' \in \mathrm{sub}(Q) - \mathrm{sub}_k(Q)$
7: feed cardinality $\widetilde{\mathrm{C}}()$ into the cost model to generate a plan P
8: candidate_plans $\leftarrow$ candidate_plans $\cup \{P\}$
9: **return** candidate_plans

Rong Zhu et al. **Lero: A Learning-to-Rank Query Optimizer.** VLDB'23

ML-enhanced Query Optimizer

- We will discuss three types of ML-enhanced query optimizer.

- **Bandit Optimizer**: Learning to steer query optimizer through hint-set.
- **Lero**: A Learning-to-Rank Query Optimizer.
- **ParamTree**: Learning hyperparameters for formula-based cost model.

Formula-based cost model

- **Formula-based cost model**: empirical formulas summarized by database researchers and developers over a long time.
- These formulas reflect how a database system interacts with the operating system and hardware components such as memory, CPU, and disk.
- Formula-based cost models typically require database administrators (DBAs) to tune certain hyperparameters (e.g., ratio to normalize I/O and CPU cost)

Jiani Yang et al. **Rethinking Learned Cost Models: Why Start from Scratch?** SIGMOD'24

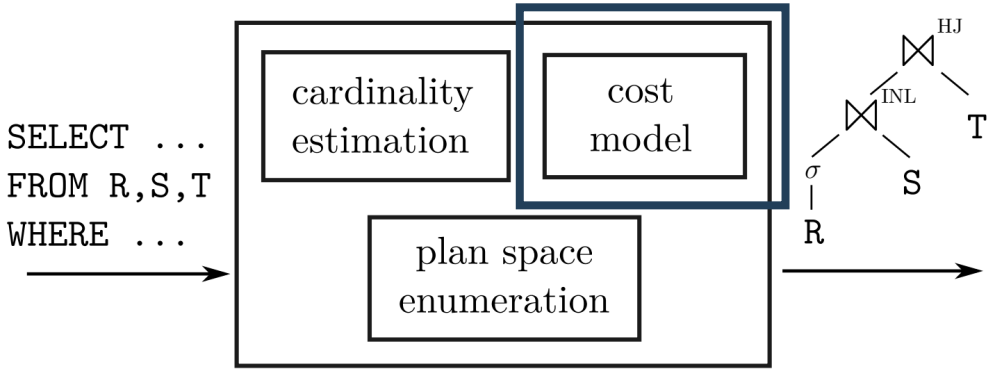
Formula-based cost model

Operator	Cost Type	Cost Formula	
SeqScan	startup_cost	0	
	runtime_cost	cpu	$(r_t + r_o \times N_{qp}) \times tuples\_num$
		disk	$r_s \times table\_pages$
IndexOnlyScan	startup_cost	$\lceil \log_2 index\_tuples \rceil \times r_o + (index\_tree\_height + 1) \times 50 \times r_o$	
	runtime_cost	cpu	$indexSelectivity \times index\_tuples \times (r_t + r_i + r_o \times N_{cp})$
		disk	$\lceil indexSelectivity \times index\_pages \rceil \times r_r$
IndexScan	startup_cost	$\lceil \log_2 index\_tuples \rceil \times r_o + (index\_tree\_height + 1) \times 50 \times r_o$	
	runtime_cost	cpu	$indexSelectivity \times index\_tuples \times (r_i + r_o \times N_{cp} + r_t + r_o \times N_{qp})$
		disk	$\lceil indexSelectivity \times index\_pages \rceil \times r_r + max\_IO\_cost +$ $indexCorrelation^2 \times (min\_IO\_cost - max\_IO\_cost)$ $(max\_IO\_cost = r_r \times \Phi_{Mackert-lohman},$ $min\_IO\_cost = (\lceil indexSelectivity \times table\_pages \rceil - 1) \times r_s + r_r)$
Sort	startup_cost	cpu	$C_t^L + 2 \times r_o \times tuples\_num \times \log_2 tuples\_num$
		disk	**When input_bytes > work_mem:** $2 \times \lceil \frac{input\_bytes}{BLCKSZ} \rceil \times \max(1, \lceil log_{mergeorder} \frac{input\_bytes}{c_w} \rceil) \times (0.75 r_s + 0.25 r_r)$
	runtime_cost	cpu	$r_o \times tuples\_num$
HashJoin	startup_cost	cpu	$C_s^L + C_T^R + R_r \times (r_o \times N_{cp} + r_t)$
		disk	$\text{sgn}(nbatches - 1) \times (r_s \times P_r)$
	runtime_cost	cpu	$C_t^L - C_s^L + r_o \times N_{cp} \times R_l + N_{cp} \times r_o \times R_l \times R_r \times innerbucketsize \times 0.5$ $+R_o \times (r_t + r_o \times N_{qp})$
		disk	$\text{sgn}(nbatches - 1) \times (r_s \times (P_r + 2 \times P_l))$

Jiani Yang et al. **Rethinking Learned Cost Models: Why Start from Scratch?** SIGMOD'24

ParamTree: Learned formula-based cost model

- **Insight**: conventional formula-based model can still provide a comparable estimation result to the learning-based one, if its hyperparameters are properly tuned based on hardware.
- Solution: use machine learning to tune the hyper-parameter of the formula-based models.
- Advantage: low cost, generalization, **explainability**



Jiani Yang et al. **Rethinking Learned Cost Models: Why Start from Scratch?** SIGMOD'24

ParamTree: Learned formula-based cost model

- Two types of parameters:
  - R-params: weight parameter
  - C-params: configuration parameters

HashJoin	startup_cost	cpu	$C_s^L + C_T^R + R_r \times (r_o \times N_{cp} + r_t)$
		disk	$\text{sgn}(nbatches - 1) \times (r_s \times P_r)$
	runtime_cost	cpu	$C_t^L - C_s^L + r_o \times N_{cp} \times R_l + N_{cp} \times r_o \times R_l \times R_r \times innerbucketsize \times 0.5$ $+R_o \times (r_t + r_o \times N_{qp})$
		disk	$\text{sgn}(nbatches - 1) \times (r_s \times (P_r + 2 \times P_l))$

Jiani Yang et al. **Rethinking Learned Cost Models: Why Start from Scratch?** SIGMOD'24

Objective

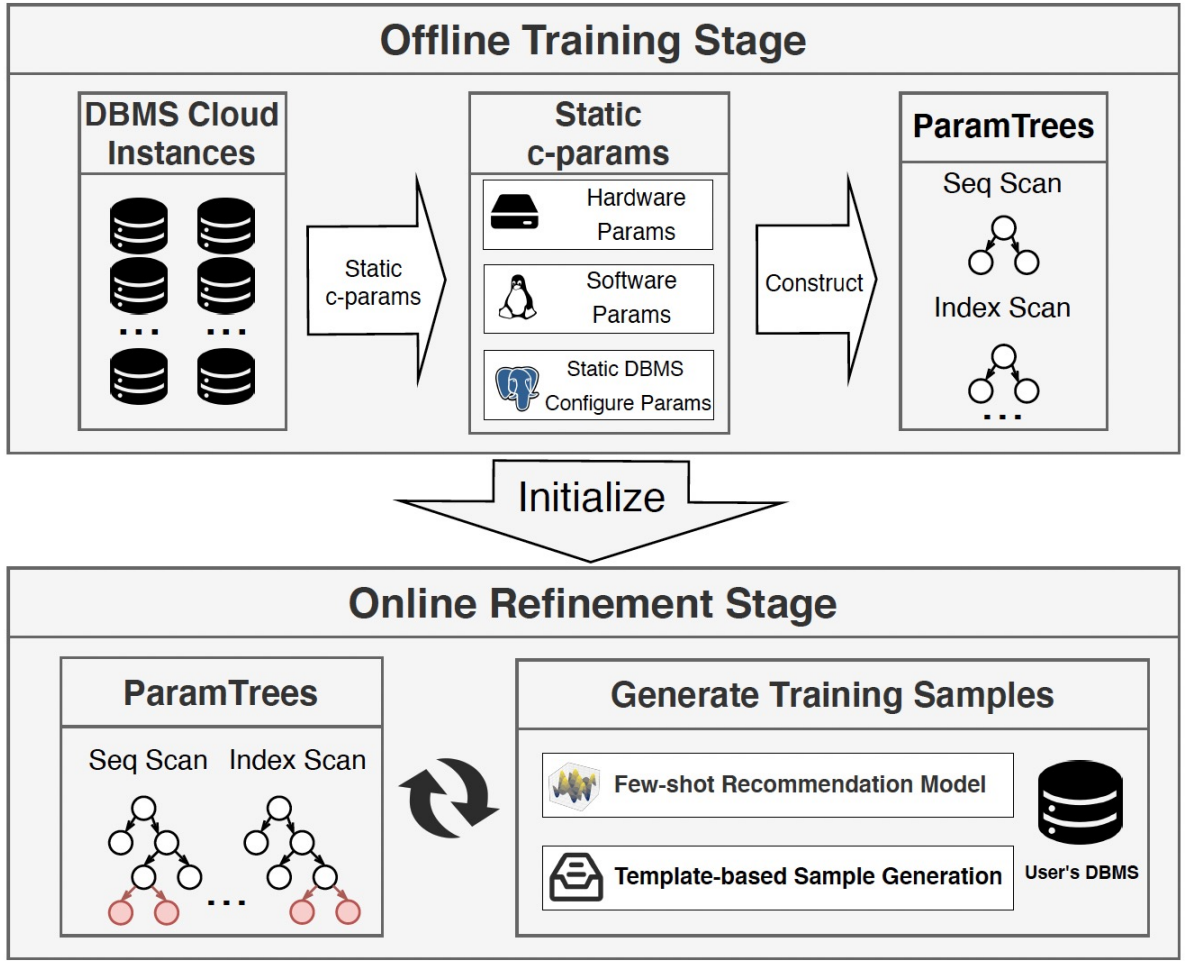
- The objective is to determine optimal choices of R-params in the built-in cost formulas of major databases.
- To accomplish this goal, the paper propose to learn a mapping function from C-params to R-params, to capture their dependencies for a given physical operator *op*.

$$G : C \xrightarrow{\text{op}} R$$

Jiani Yang et al. **Rethinking Learned Cost Models: Why Start from Scratch?** SIGMOD'24

Two-stage training and refinement

- **Offline training stage**: generate an initial decision tree that captures the relationship between R-params and C-params.
- **Online refinement stage**: Online tree expansion is invoked to split the leaf nodes if the **ParamTree** cannot provide a precise estimation for more than λ queries.



Jiani Yang et al. **Rethinking Learned Cost Models: Why Start from Scratch?** SIGMOD'24

ParamTree

- Revisit the three components in ParamTree
- Cardinality estimation: histogram
- Cost estimation: formula-based + learned hyper-parameters
- Plan enumeration: dynamic programming

Summary

- Review the three components in different query optimizers

Component	Traditional query optimizer	Learned Query optimizer	ML-enhanced query optimizer		
			BAO	LERO	ParamTree
Cardinality estimation	Histogram	Tree CNN (implicit)	_	Cardinality as knob	Histogram
Cost estimation	Formula-based	Tree CNN	Tree CNN	Ranking model	Formula-based with learned hyperparameter
Plan enumeration	Dynamic Programming	Value network guided search	Curated Hint set + Expert optimizer	Adjusting cardinality + Expert optimizer	Dynamic Programming

Tutorial Overview

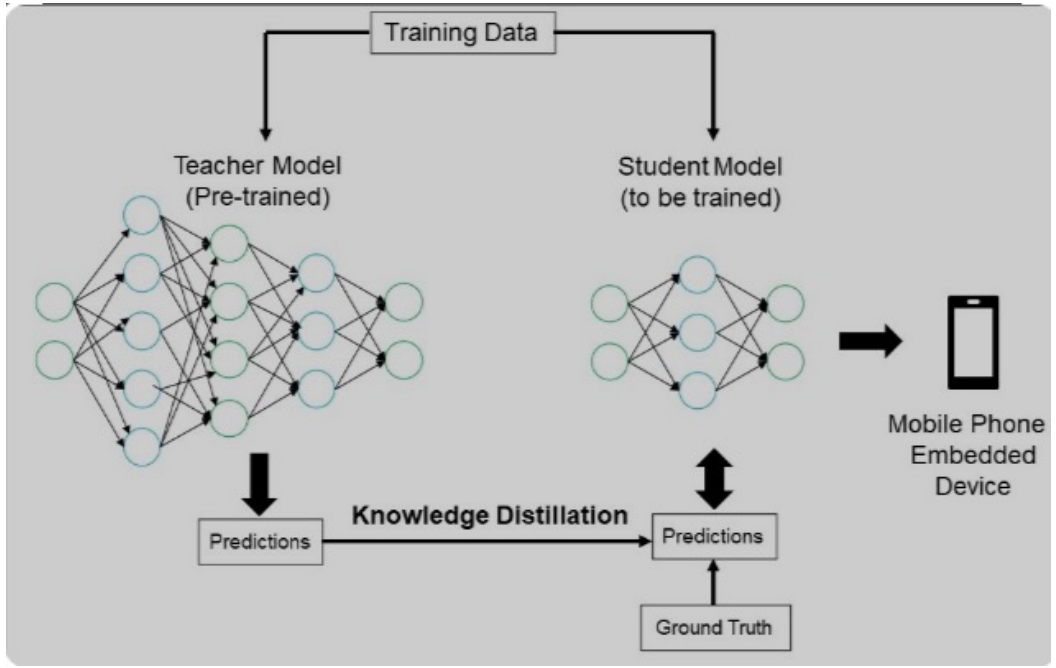
- ML4DB Foundations
- ML4DB Paradigms
- ML4DB Open problems & Opportunities
  1) What are the foundations of ML4DB?
  2) Model efficiency
  3) Generating training data of high quality
  4) Handling data & workload shifts
  5) Foundation models (Pretrained models) for Ml4DB
  6) LLMs for ML4DB

Opportunity 1: What are the foundations of ML4DB?

- Separate ML4DB components are developed. But it might not work to integrate them into DB systems.
- What are the foundations?
  - Query plan representation
  - Pre-trained models
  - Anything else?

Opportunity 2: Model Efficiency

- Ideal model: efficiency in both training & inference
- Potential ML techniques:
  - Light-weight AI models
  - Ad-hoc optimizations
  - Knowledge distillation

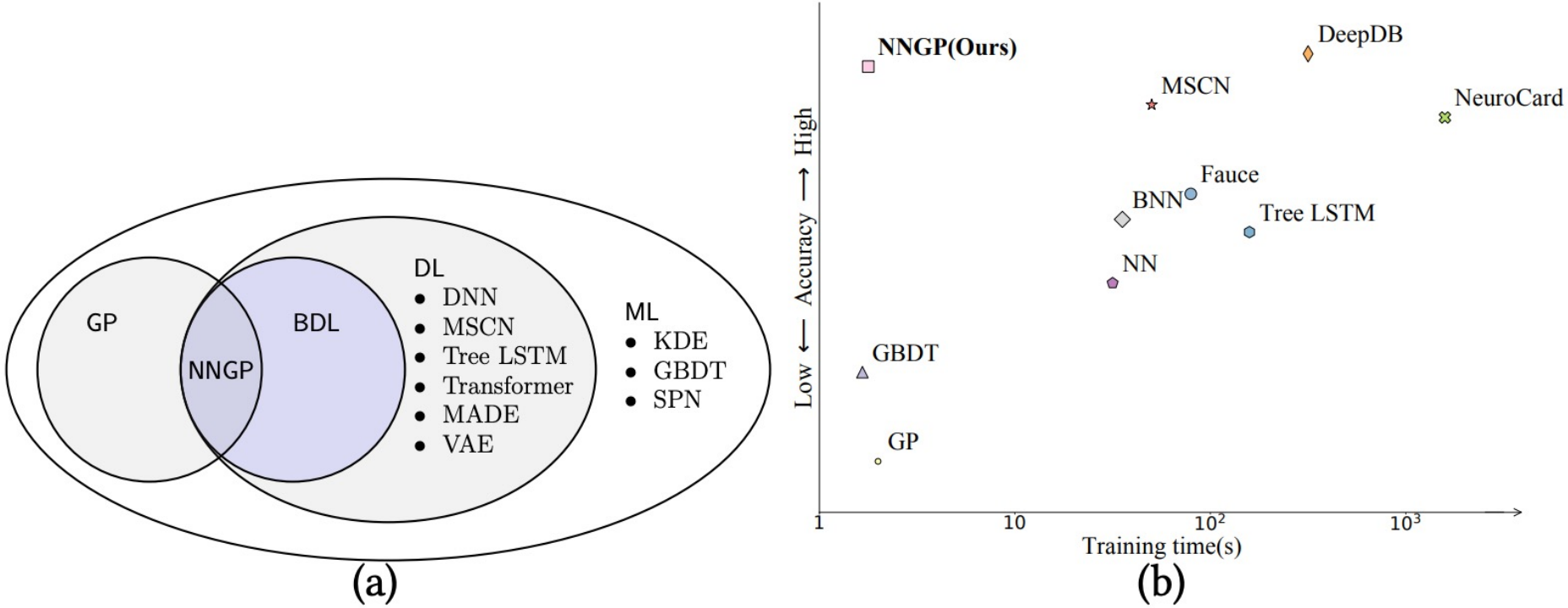


Picture from https://www.analyticsvidhya.com/blog/2022/01/knowledge-distillation-theory-and-end-to-end-case-study/

- Example: Some initial attempts at improving model efficiency for
  - Cardinality estimation
  - Query optimization

Model Efficiency: Cardinality Estimation

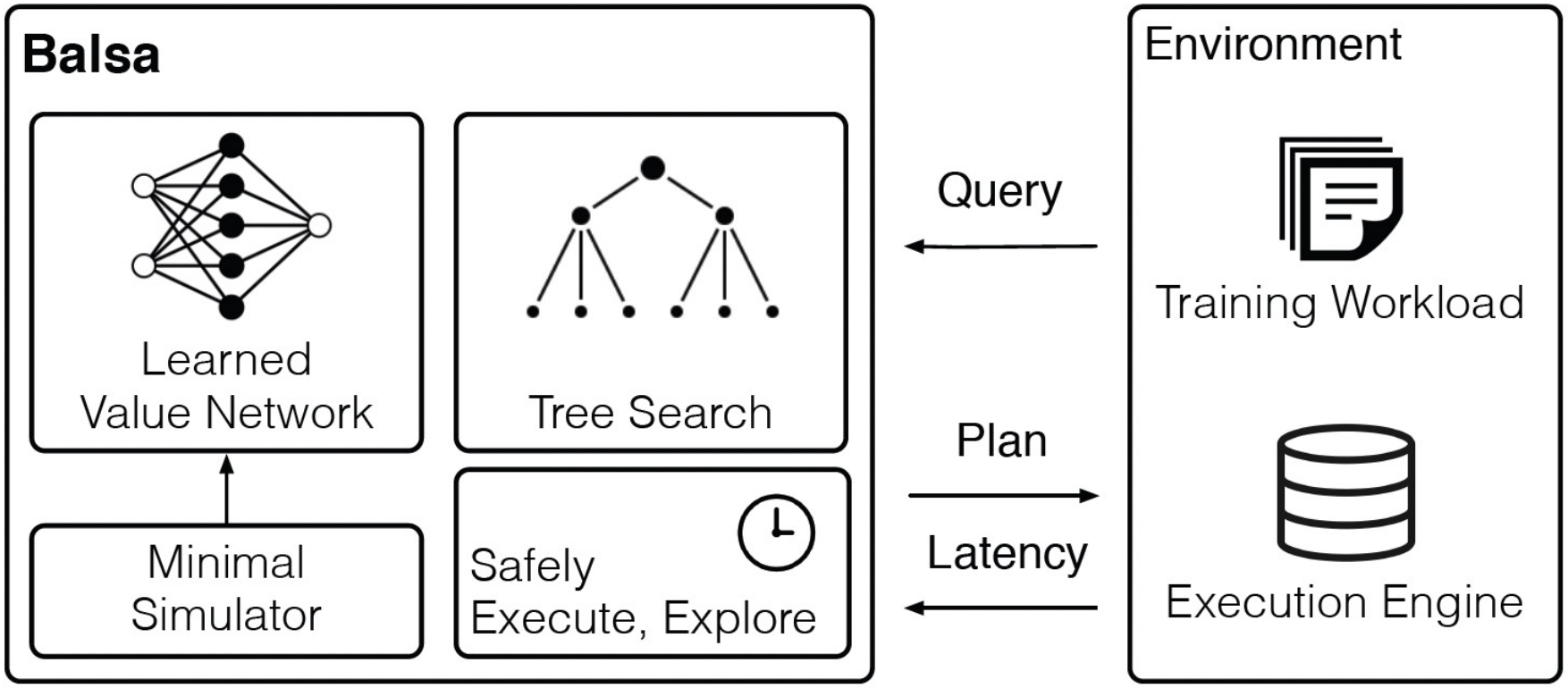
- NNGP (SIGMOD'22): Neural network gaussian process for cardinality estimation



Kangfei Zhao et al. **Lightweight and Accurate Cardinality Estimation by Neural Network Gaussian Process?** SIGMOD'22

Model Efficiency: Query optimization

- Balsa (SIGMOD'22): Simulation-to-reality learning and safe execution



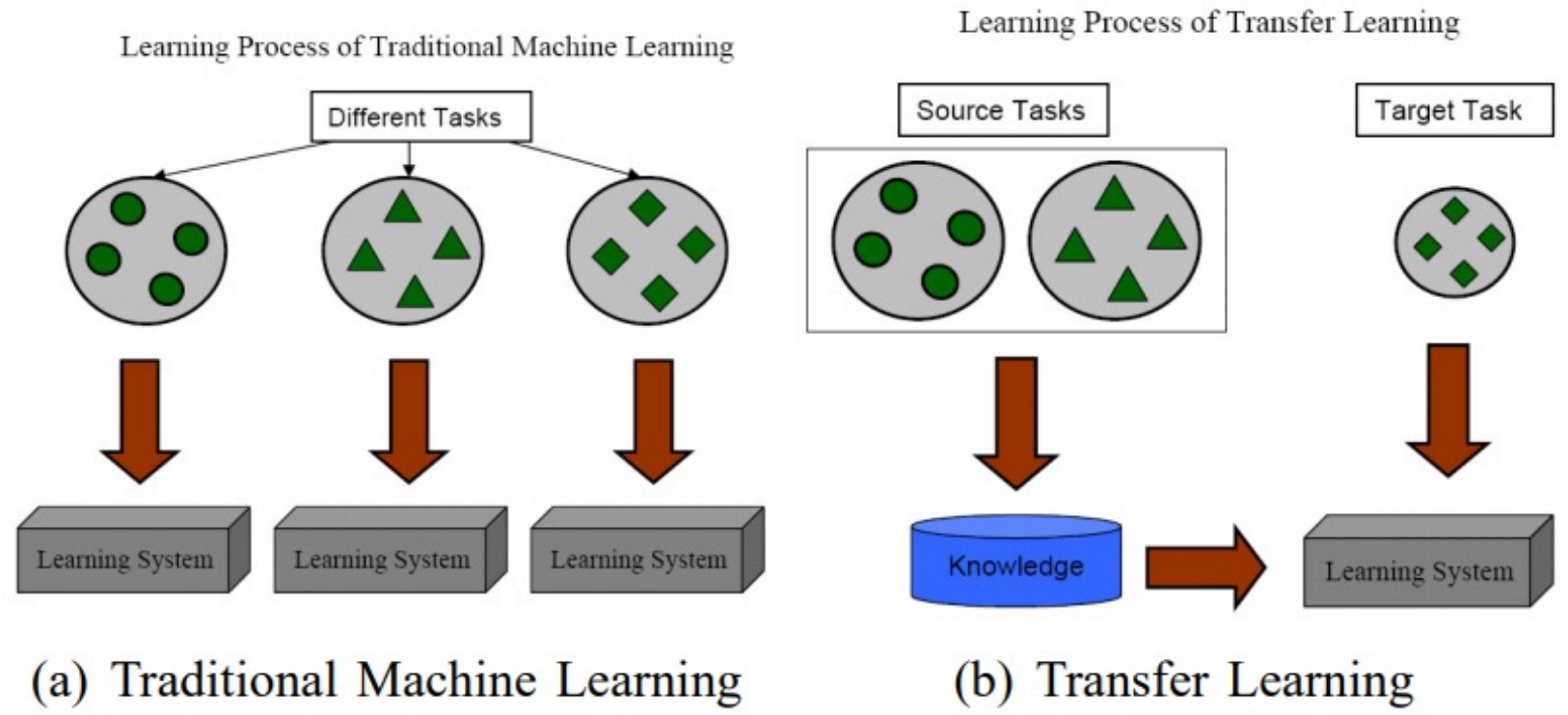
Yang et al. **Balsa: Learning a Query Optimizer Without Expert Demonstrations** SIGMOD'22

Opportunity 3: Handling data & workload shifts

- Most machine learning models require retraining in face of data & workload shifts, which can be very costly.
- Potential ML techniques
  - Transfer learning
  - Meta-learning
  - Domain generalization and adaptation
- Some models that adapt to data & workload shift
  - Cardinality estimation (Beibin Li et al., SIGMOD'22; Parimarjan Negi et al., SIGMOD'23; Zilong Wang et al., VLDB'23)
  - Data generation

Transfer Learning

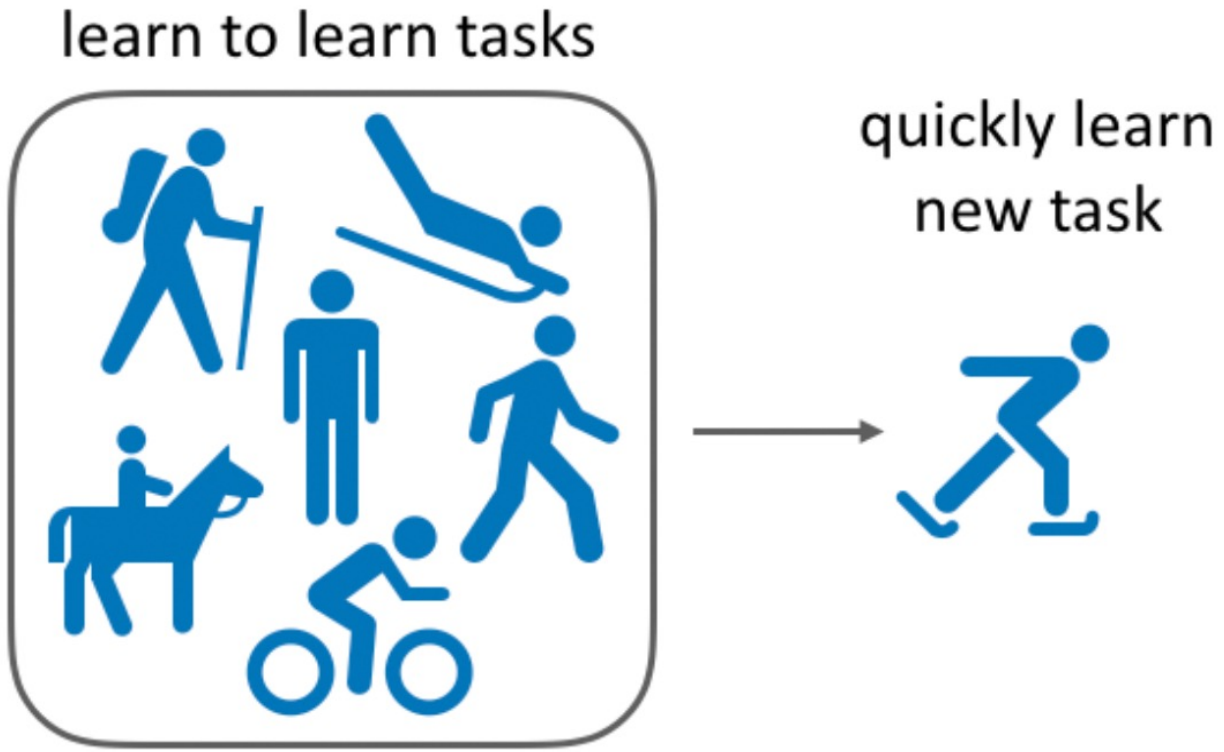
- Improve the target learners on target domains by transferring the knowledge contained in different but related source domains.



Pictures from Hosna, A et al, Transfer learning: a friendly introduction. Journal of Big Data (2022)

Meta-learning

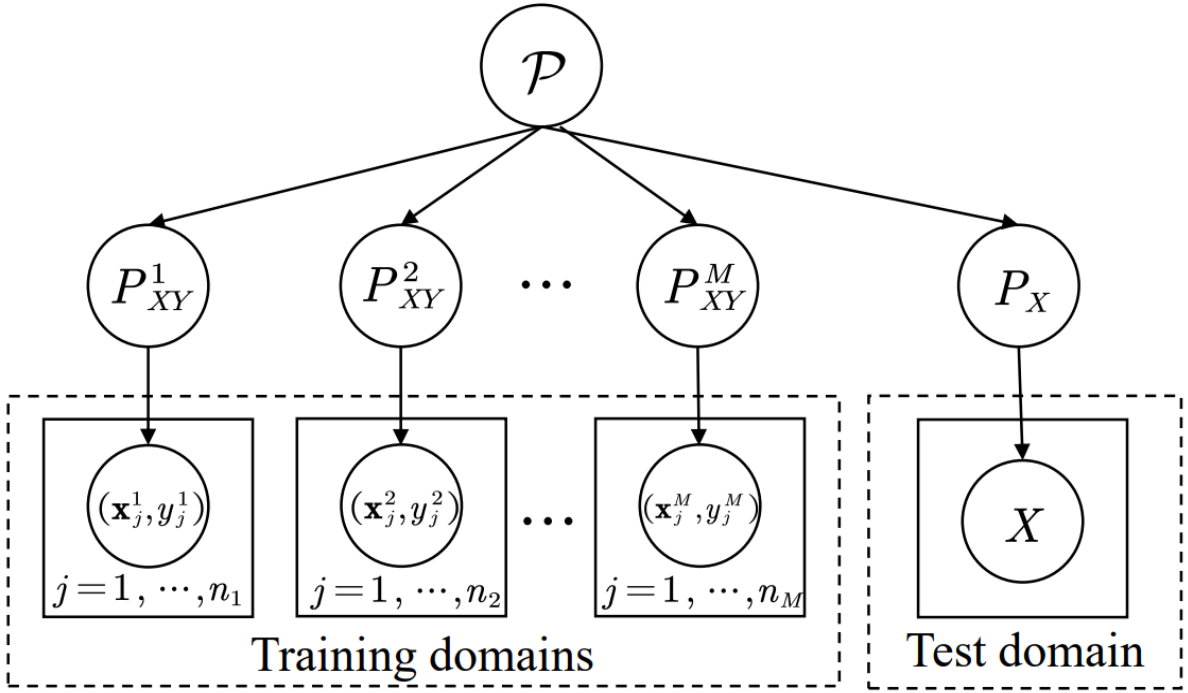
- Meta-learning aims to learn the learning algorithm itself by learning from previous experience or tasks, i.e., learning-to-learn.



Pictures from Huisman, Mike, Jan N. Van Rijn, and Aske Plaat. "A survey of deep meta-learning." Artificial Intelligence Review 54.6 (2021): 4483-4541.

Domain generalization

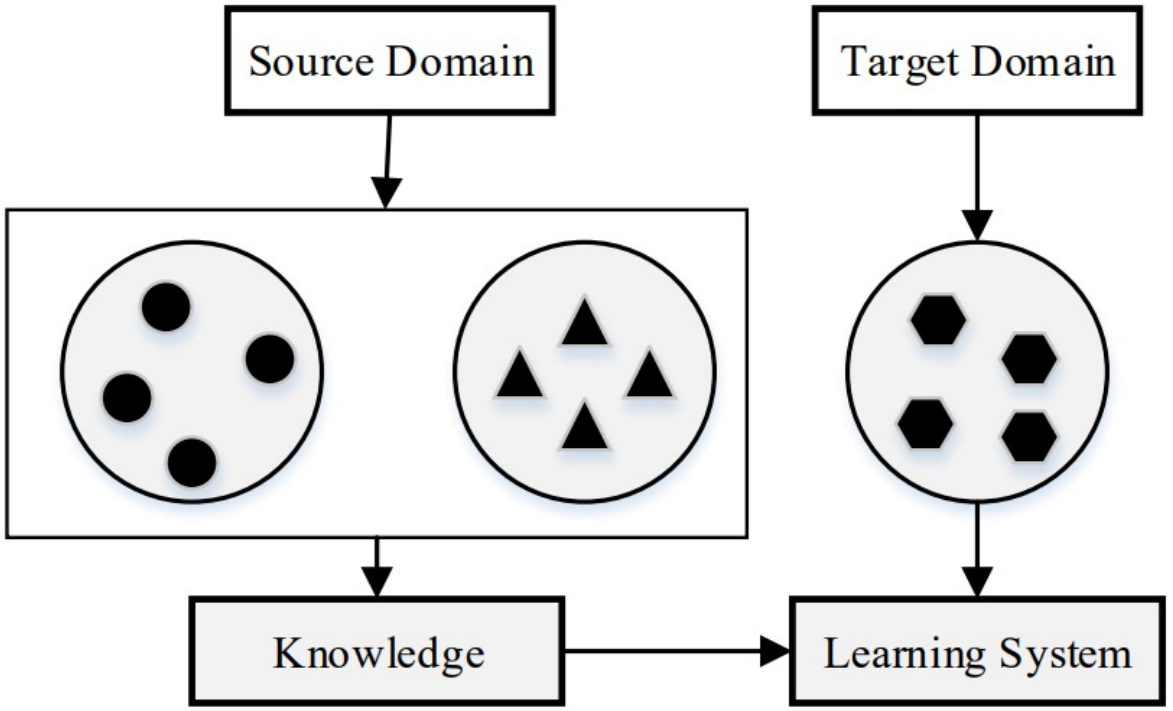
- Domain generalization leverage common knowledge learned from multiple training domains to generalize to unseen test domains.



Pictures from Wang, Jindong, et al. "Generalizing to unseen domains: A survey on domain generalization." TKDE 35.8 (2022): 8052-8072

Domain adaptation

- Domain adaptation aims to maximize the performance on a given target domain using existing training source domain(s).



Pictures from Wang, Mei, and Weihong Deng. "Deep visual domain adaptation: A survey." Neurocomputing 312 (2018): 135-153

Handling data & workload shifts: cardinality estimation

- Warper (SIGMOD'22): generate additional queries when limited examples are available and pick the queries to update the model

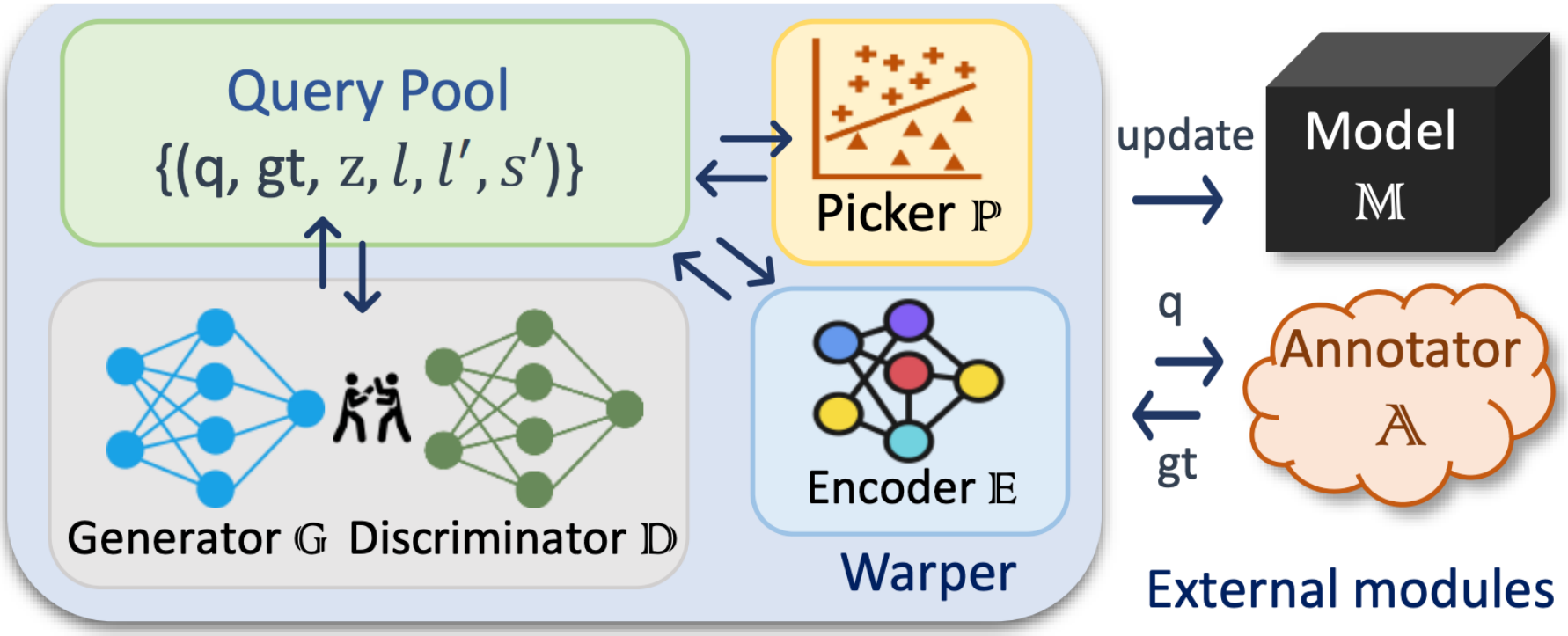


Figure from Beibin Li et al., **Warper: Efficiently adapting learned cardinality estimators to data and workload drifts.** SIGMOD'22

Handling data & workload shifts: data generation

- DDUp (SIGMOD'23): Statistical test for detecting data shifts, and incrementally update the model through knowledge distillation

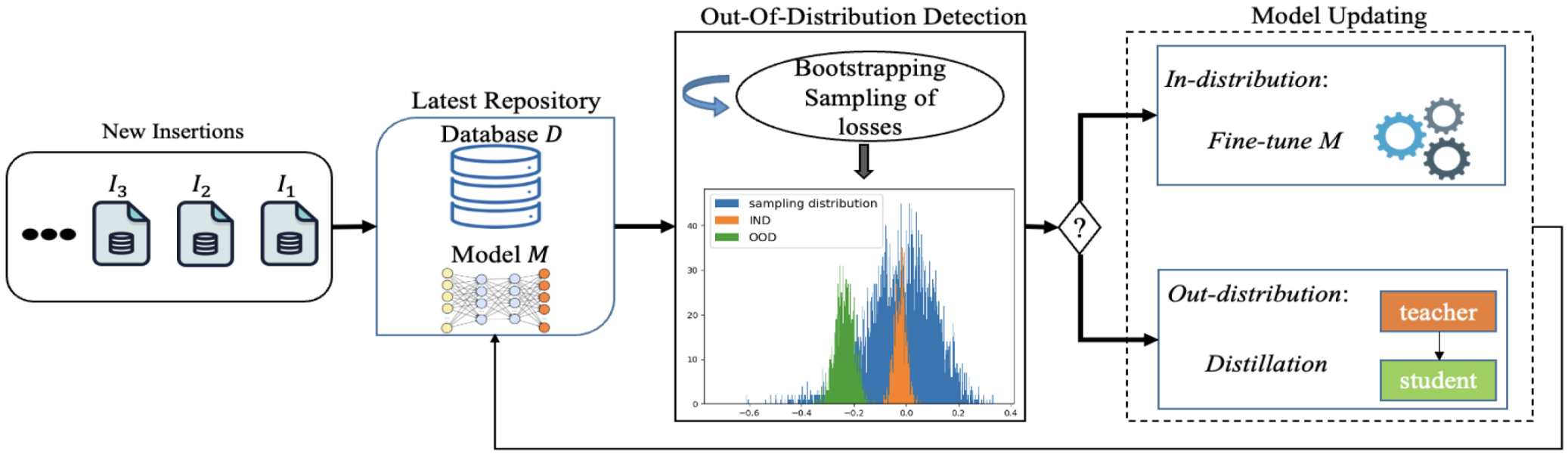


Figure from Meghdad Kurmanji et al., **Detect, Distill and Update: Learned DB Systems Facing Out of Distribution Data.** SIGMOD'23

Opportunity 4: Training data of high quality

We want to generate both datasets and query workload. Ideally,

1. Diverse Workloads
   - Include a variety of queries, from simple SELECT statements to complex operations.
2. Comprehensive Coverage
   - Include data from various domains (e.g., finance, healthcare, e-commerce) with different underlying distribution.
3. Annotated with Performance Metrics
   - Include detailed performance metrics such as query execution times, resource usage, throughputs, etc.
   - Annotate data with configuration settings and their impact on performance.

   ….

- But, even for a single ML4DB task, training data are generated by executing queries on a DBMS to collect the physical plan and corresponding cost. Very expensive !!!

Opportunity 5: Foundation (Pretrained) models for ML4DB tasks

- ML4DB models are typically trained on a specific database with specific hardware/software configuration
- Retraining is required when transferring to a new database or different hardware/software environment or a new task
- Can we have a large pretrained model for ML4DB?
  - Capability to generalize to different data
  - Capability to generalize across tasks
  - Capability to generalize across hardware/software

Opportunity 5: Foundation (Pretrained) models for ML4DB tasks

- Challenges
  - Lack of large collection of **databases and query workload, and training data**
  - Designing **scalable** pre-training methods
  - Identifying and integrating the most relevant **features**
  - Designing models that support **incremental** learning
  - **Integrating** pre-trained models with current database management systems
- Some pioneering attempts
  - Hilprechet & Binnig CIDR21, VLDB22
  - Paul et al VLDB22,
  - Wu et al, CIDR22

Opportunity 6: LLMs for ML4DB tasks

- What can LLMs do for DBs?
- Some pioneering attempts
  - Text2SQL (Many works)
  - Database tunning [1, 2]
  - DBA [3]
  - Query rewriting [4]
  - Interaction with data systems [5]
  - ...

Huang, X et al, LLMTune: Accelerate Database Knob Tuning with Large Language Models. arXiv 2024. [1]
Lao, J et al. GPTuner: A Manual-Reading Database Tuning System via GPT-Guided Bayesian Optimization. VLDB 2024. [2]
Zhou, X et al. LLM As DBA. arXiv 2023. [3]
Li, Z et al. LLM-R2: A Large Language Model Enhanced Rule-based Rewrite System for Boosting Query Efficiency. arXiv 2024 [4]
Xue, S et al. Demonstration of DB-GPT: Next Generation Data Interaction System Empowered by Large Language Models. arXiv 2024. [5]

THANK YOU!

The slides will be made available on our group website