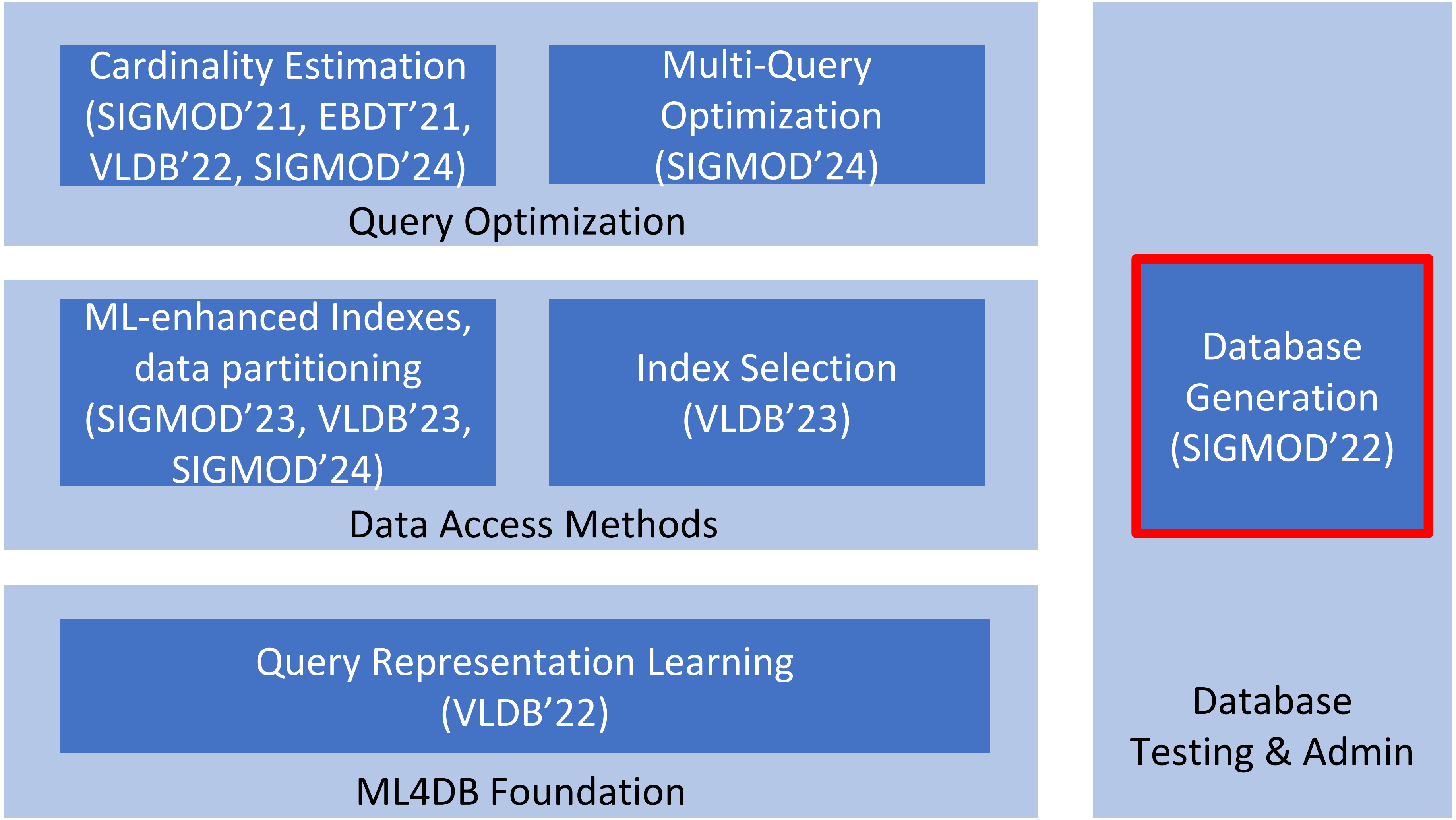
Empowering Database Systems with Machine Learning

Gao Cong

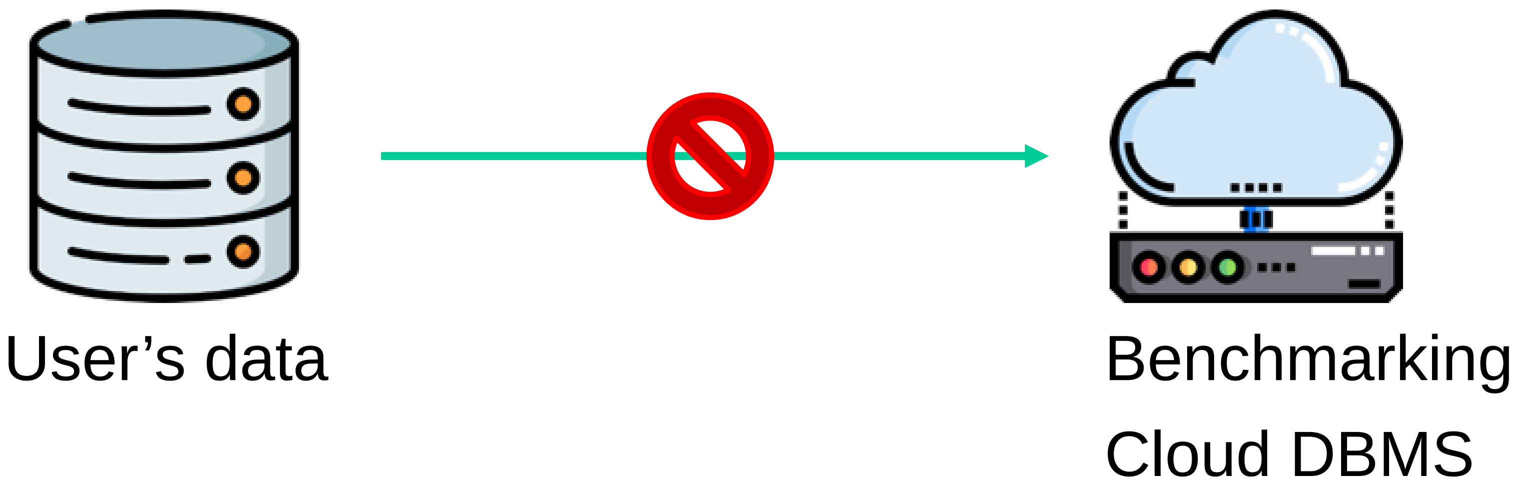
Nanyang Technological University

Overview of our Research on Machine Learning for Database (ML4DB)



SAM: Database Generation from Query Workloads

- Before migrating data from local to cloud, cloud providers need to benchmark different DBMS to recommend a product.
- **Problem**: Cloud Provider usually do not have access to the user's database.



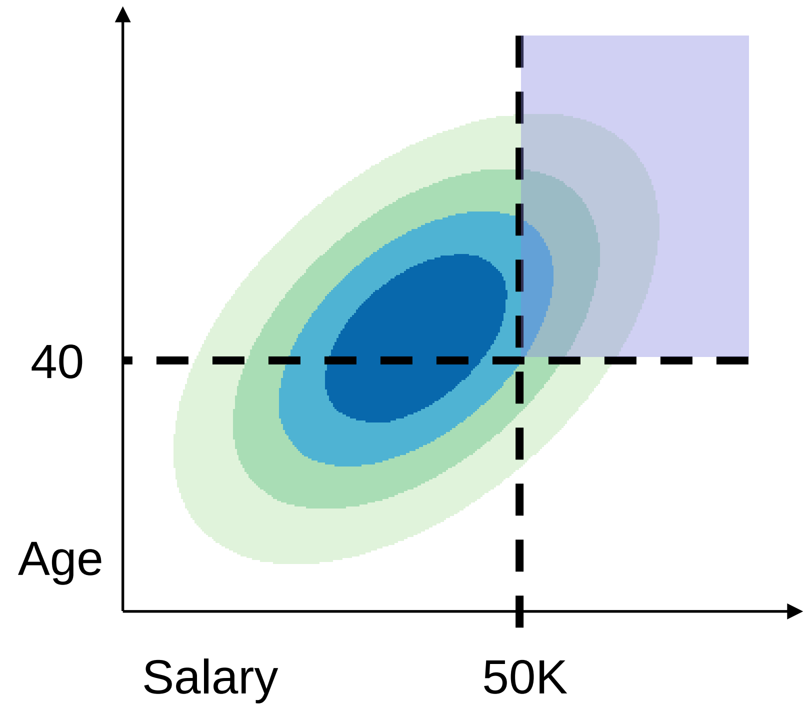
Jingyi Yang, Peizhi Wu, Gao Cong, Tieying Zhang, Xiao He. SAM: **Database Generation** from Query Workloads with Supervised Autoregressive Models. SIGMOD 2022

SAM: Database Generation from Query Workloads

- On the other hand, cloud providers may have access to the user's query logs and collect a set of queries & the result cardinalities.
- **Observation**: Queries and the result cardinalities provide information on the data distribution.

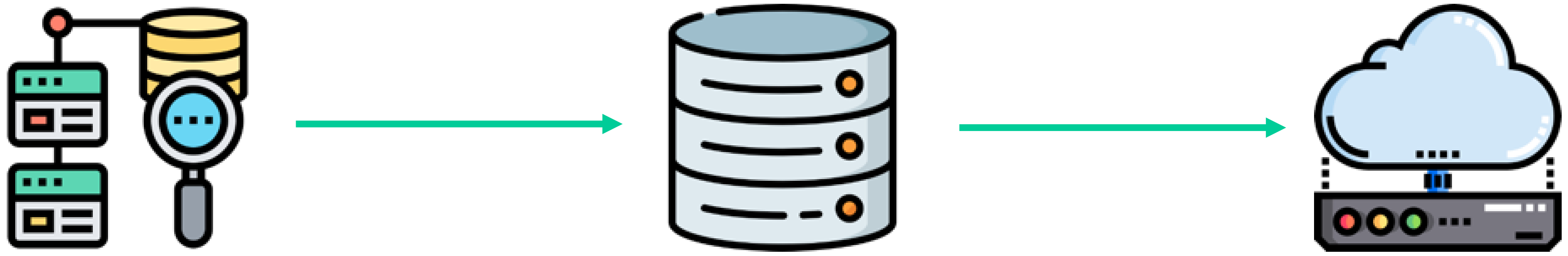
```
SELECT * From census WHERE
age > 40 and salary > 50K
```

```
Cardinality: 26992
```



SAM: Database Generation from Query Workloads

- Given a query workload with cardinalities, we aim to generate a synthetic database that satisfies the cardinality constraints and is close to the original database.
- Benchmarking can be conducted on the synthetic database.



User's query workload

Synthetic Database

Benchmarking Cloud DBMS

SAM: Database Generation from Query Workloads

- Another use case is stress testing for databases with strict access controls.
- For example, core user database of a social media or e-commerce platform, where replication is highly restricted.

Query workload of core database

Synthetic Database

Stress Testing

Problem Setup

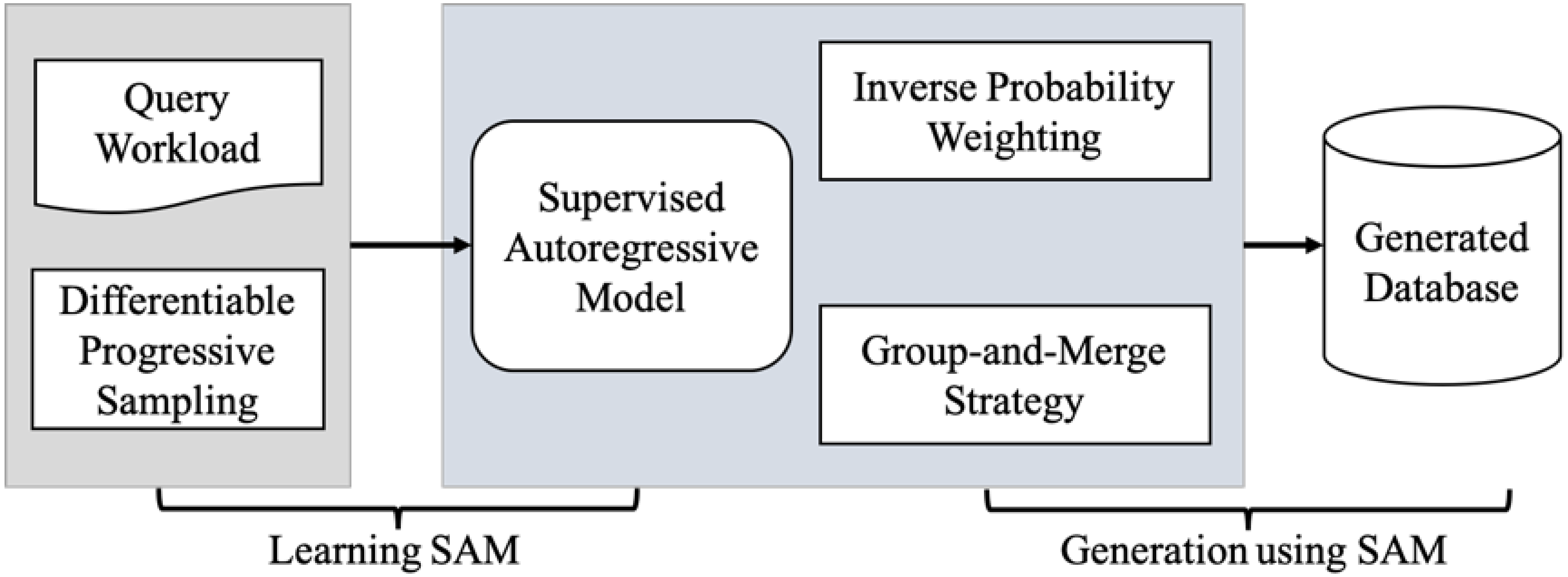
Database Generation From Query Workloads:

- Consider a set of n queries Q and their cardinalities collected on a database D.
- Aim to generate a database that satisfies the cardinality constraints and is close to the original database.
- Cross entropy between the discrete data distribution of the generated relation $\hat{T}$ and original relation T as a measure of closeness.

$$H(T, \hat{T}) = -\mathbb{E}_{x \sim T}[log(\widehat{Sel}(x))]$$

Workflow of SAM

- We propose SAM, a query-aware database generator based on autoregressive models:
  - Learning stage: **Efficiently** and **accurately** learn the join data distribution
  - Generation stage: Generate a **high-fidelity** database from the AR model



Evaluation on Closeness

- SAM generates a database that is closer to the original database.
- SAM can well generalize to unseen queries, achieving **300X** less mean error on IMDB.

Model	Census				DMV			
	Median	75th	90th	Mean	Median	75th	90th	Mean
PGM	46.00	872.0	3461	1097	646.0	$1 \cdot 10^5$	$1 \cdot 10^6$	$4 \cdot 10^5$
SAM	1.31	1.76	2.70	1.97	1.16	1.54	3.11	4.05

Table 5: Q-Error of test queries

Model	Median	75th	90th	Mean	Max
PGM	232.7	$6 \cdot 10^4$	$1 \cdot 10^6$	$9 \cdot 10^5$	$3 \cdot 10^7$
SAM w/o Group-and-Merge	38.67	$1 \cdot 10^5$	$3 \cdot 10^6$	$5 \cdot 10^6$	$3 \cdot 10^8$
SAM	2.29	5.39	27.78	2776	$2 \cdot 10^5$

Table 6: Q-Error of JOB-light queries on IMDB

Model	Census	DMV	IMDB
PGM	29.37	39.49	12.45
SAM	28.68	23.22	6.14

Table 7: Cross entropy of the generated relation

Evaluation on efficiency

- Processing time scales as a high-degree polynomial for PGM, but linearly for SAM.
- Therefore, SAM can process query workloads of a much larger scale.

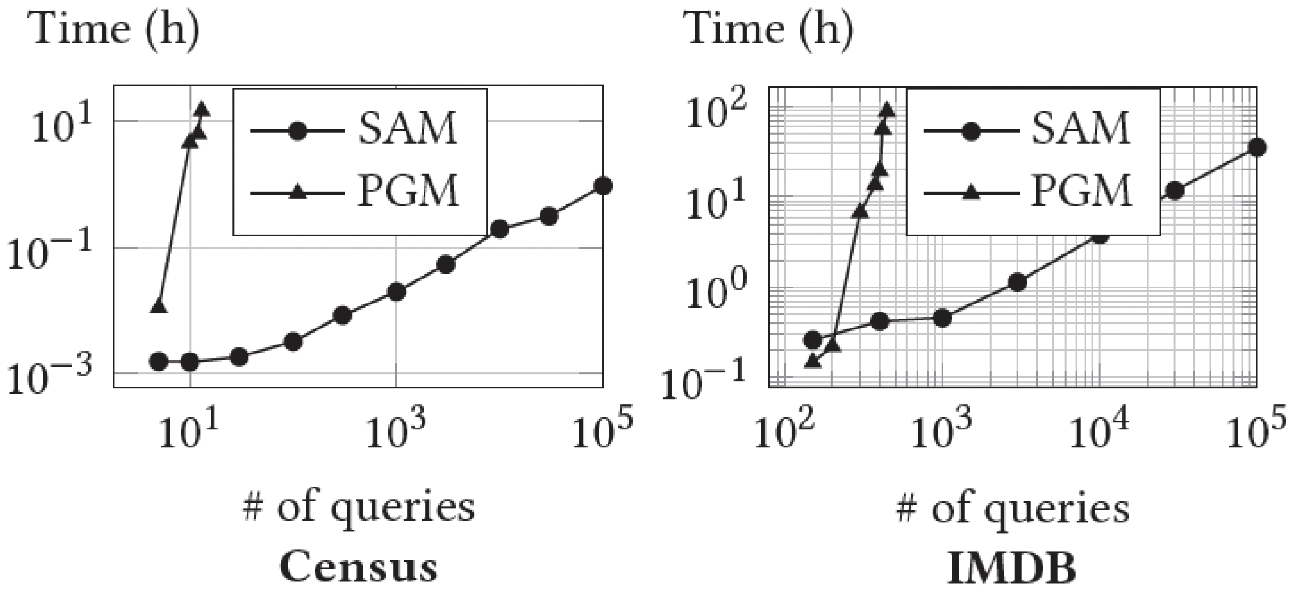


Figure 5: Processing time.

Overview of our Research on Machine Learning for Database (ML4DB)

Cardinality Estimation (SIGMOD’21, EBDT’21, VLDB’22, SIGMOD’24)

Multi-Query Optimization (SIGMOD’24)

Query Optimization

ML-enhanced Indexes, data partitioning (SIGMOD’23, VLDB’23, SIGMOD’24)

Index Selection (VLDB’23)

Data Access Methods

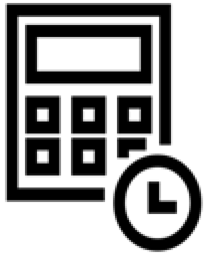
Query Representation Learning (VLDB’22)

ML4DB Foundation

Database Generation (SIGMOD’22)

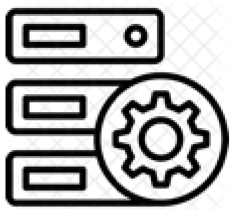
Database Testing & Admin

ML4DB Foundations



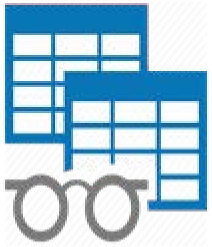
Cost/Cardinality Estimator

Join Order Selection



Learned Optimizer

Index Recommendation



View Advisor

…

ML4DB Tasks

Question： Can we have some **foundation** of **different ML4DB tasks?**

Query Plans are used as inputs in many ML4DB tasks

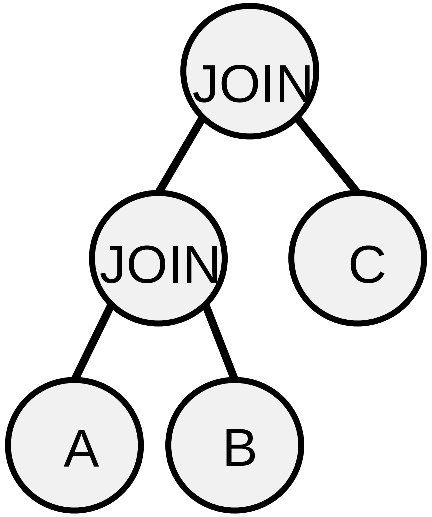
Query plan representation is a key operation

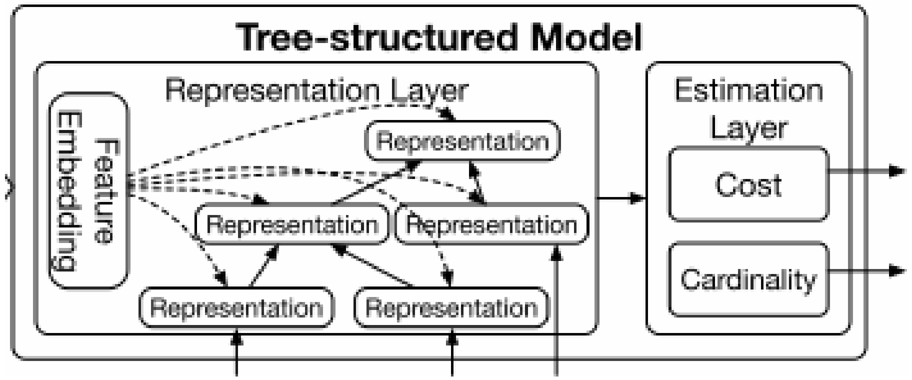
Example: Cost Estimation

- Cost and Cardinality Estimation [Sun., et al. VLDB 19]
  - Uses Tree-LSTM to extract feature representation from a *query plan*
  - Uses MLP to predict cost and cardinality

Model Input

Model Output



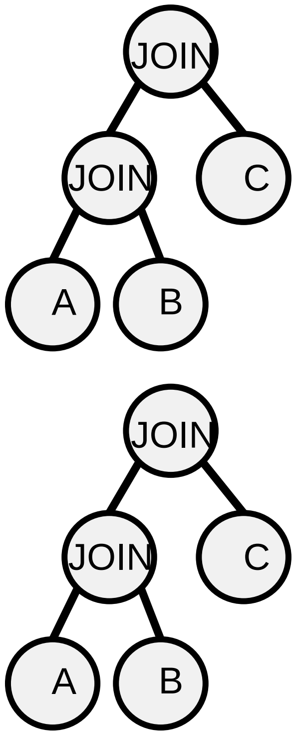


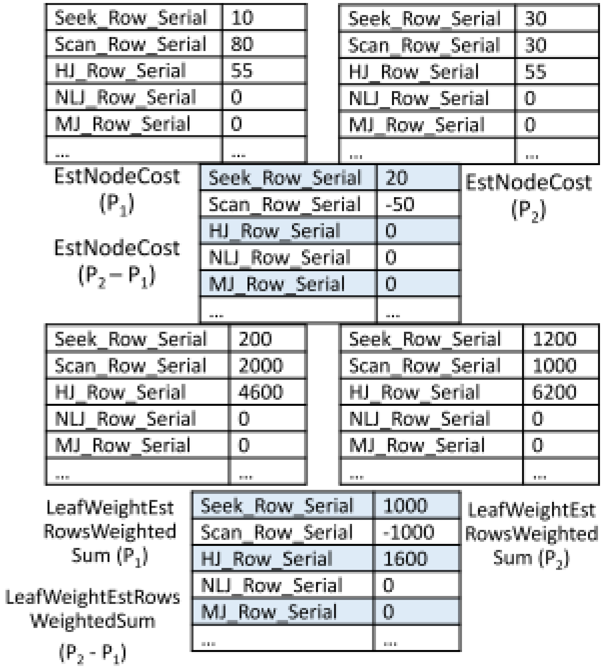
Est. Cost: 123 ms
Est. Cardinality: 12345

Example: Index Recommendation

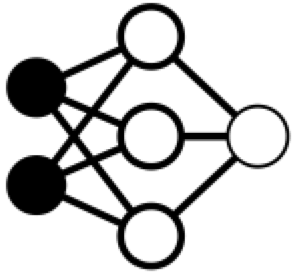
- Index Recommendation [Bailu, D., et al. SIGMOD 19]
  - Featurize a *query plan* by creating feature channels for each physical operator
  - Perform classification on *query plan* pairs

Model Input





Featuring Query Plan by Bailu, D. (2019).



Classifier

Model Output

First plan is better

⬇

First index is better

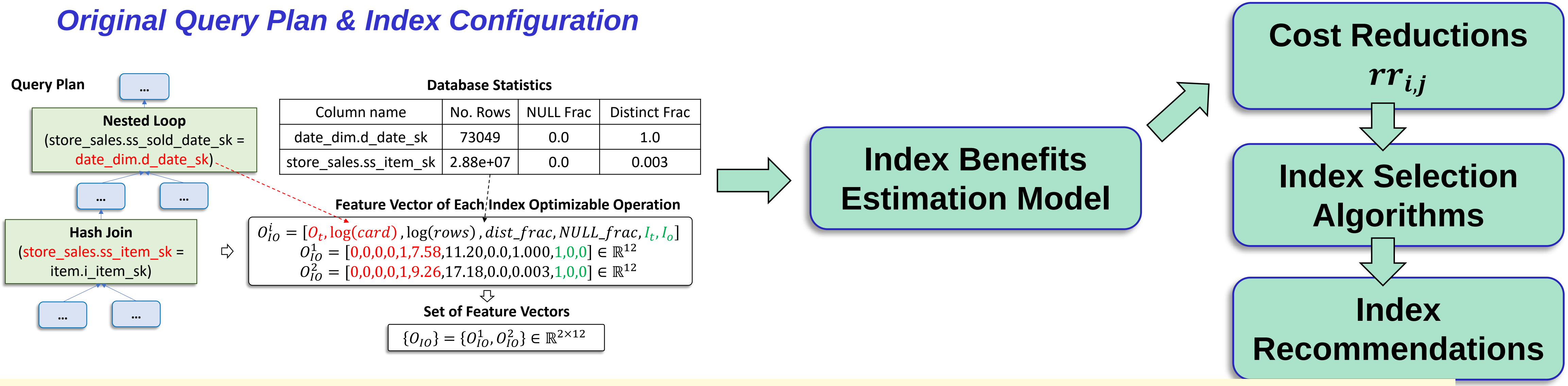
Example: Index Recommendation

- Index Recommendation [Shi, et al. VLDB 23]
  - Featurize a *query plan & an index configuration* as a set of *index optimizable operations*.
  - Adopting attention-based model for interrelations between operations and indexes.
  - Replacing "What-if" call to perform index cost reduction estimation.

Model Input

Model Output

Original Query Plan & Index Configuration

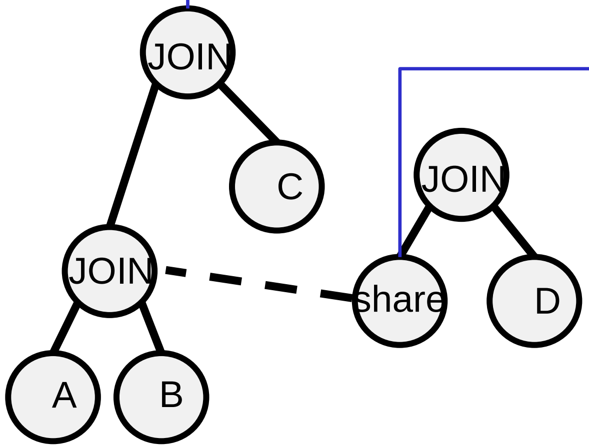


Jiachen Shi, Gao Cong, Xiaoli Li. Learned Index Benefits: Machine Learning Based **Index Performance Estimation**. VLDB 2023

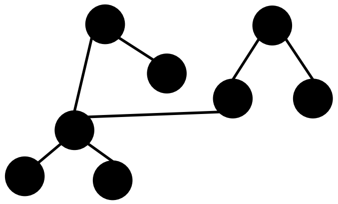
Example： Multiple query optimization

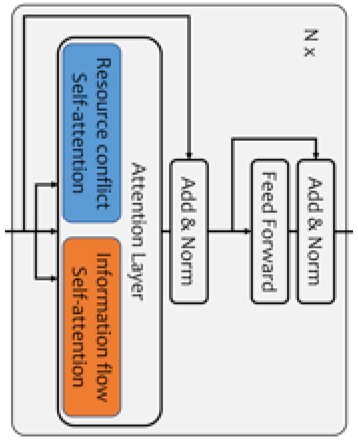
- Multiple query optimization [Mo, et al. SIGMOD 24]
  - Featurize *concurrent query plans* by creating feature channels for each node
  - Featurize *SQL query* by extracting join graph and predicate information
  - Predict the cost for plan generation

Model Input



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





Cost estimation model

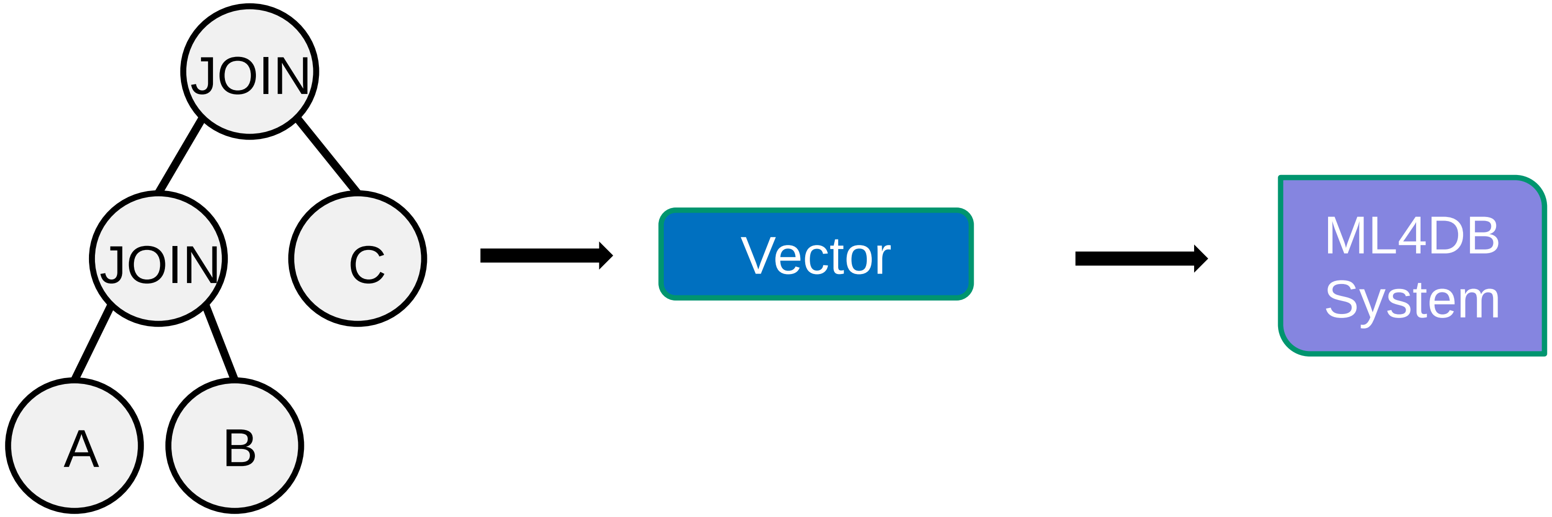
Model Output

cost

Song Song Mo, Yile Chen, Hao Wang, Gao Cong, Zhifeng Bao. Lemo: A Cache-Enhanced **Learned Optimizer** for Concurrent Queries. SIGMOD 2024

ML4DB Foundation Research Problem

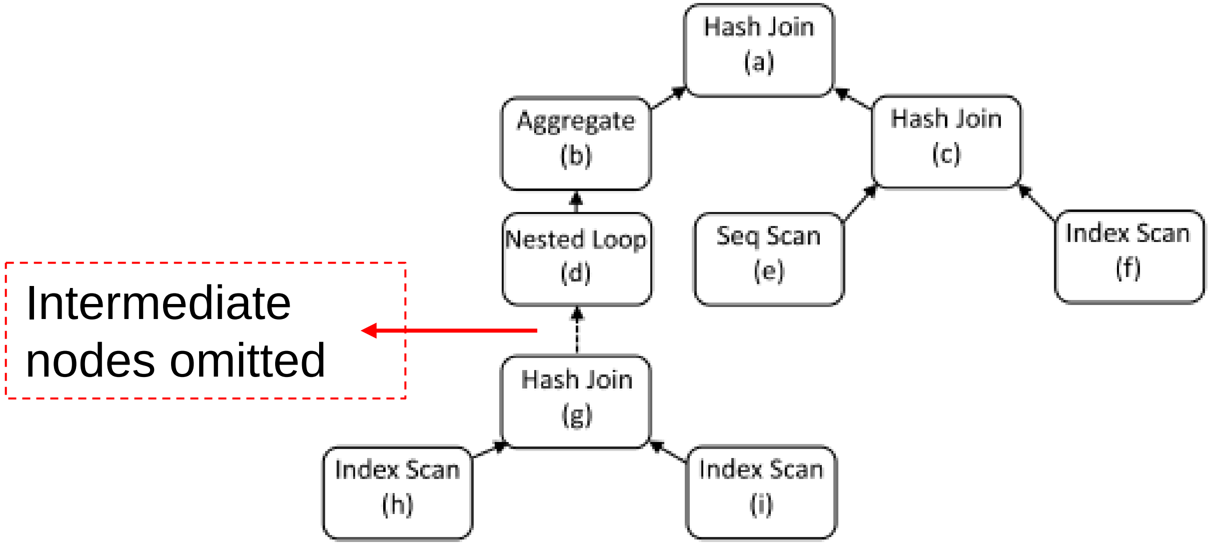
- Why is representation learning important?
  - Non-trivial to define features from a query plan
  - Difficult to deal with the tree structure of a query plan
  - Input encoding is a key factor to the performance of all these methods
- Research Problem: Given a *query plan*, learn a vector representation to be used as the input to a ML4DB system



Yue Zhao, Gao Cong, Jiachen Shi, Chunyan Miao. QueryFormer: a tree transformer model for **query plan representation**. VLDB 2022.

Challenges

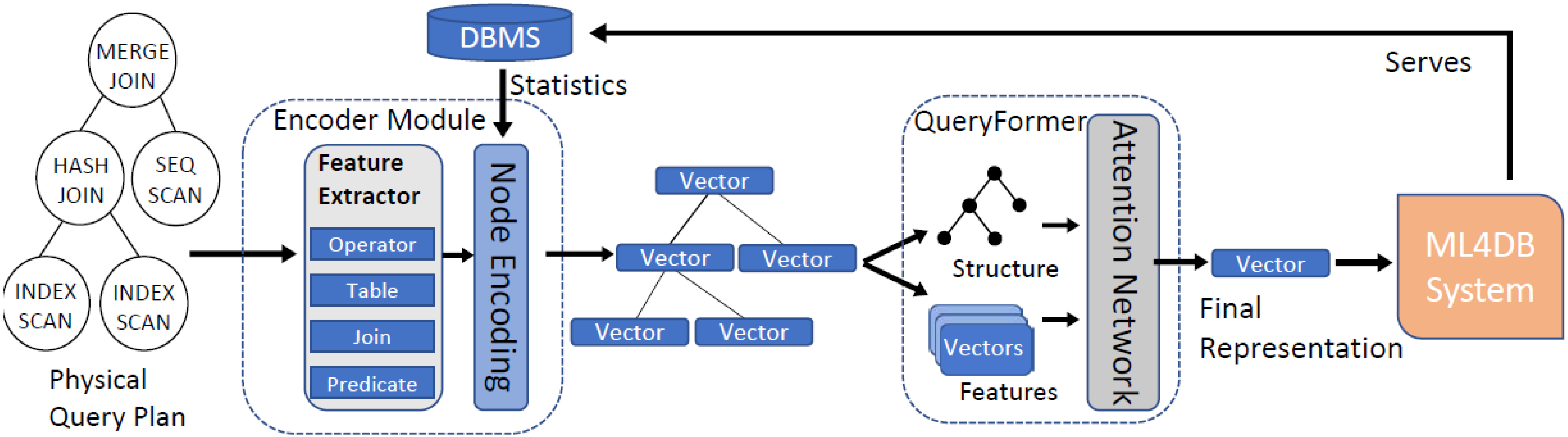
- Incorporate the statistics stored in a database
- Encode the tree structure of the input
  - Parent-children dependency
  - Long paths of information flow



Example Query Plan derived from TPC-DS query 18.

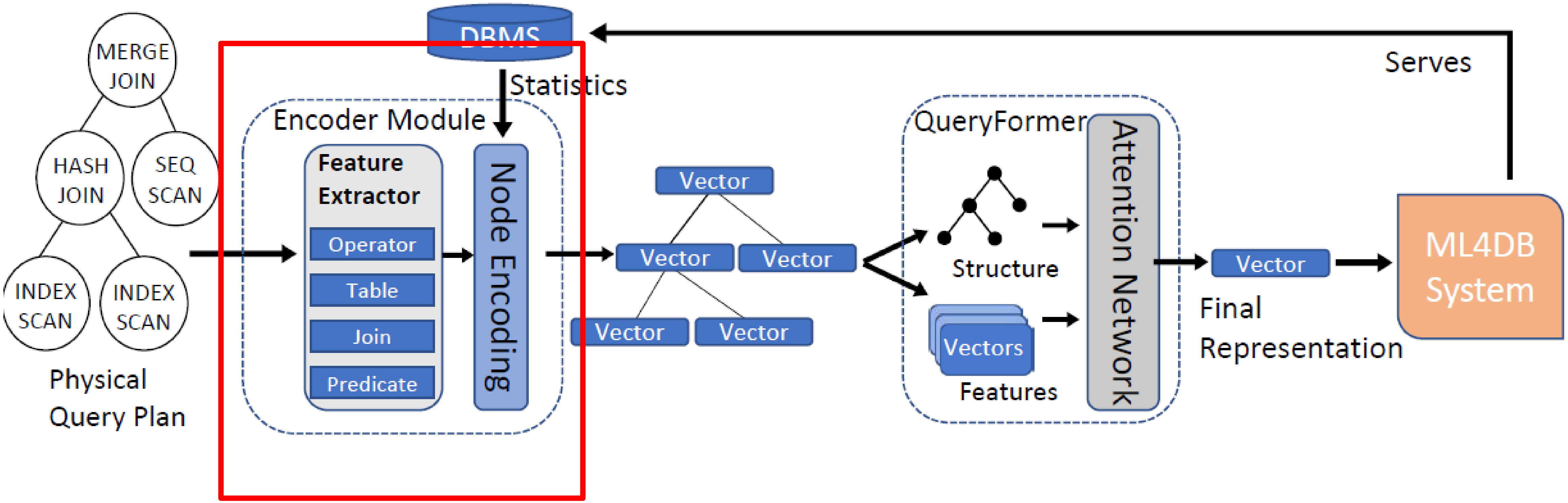
System Overview

- Plug and Play for existing ML4DB works



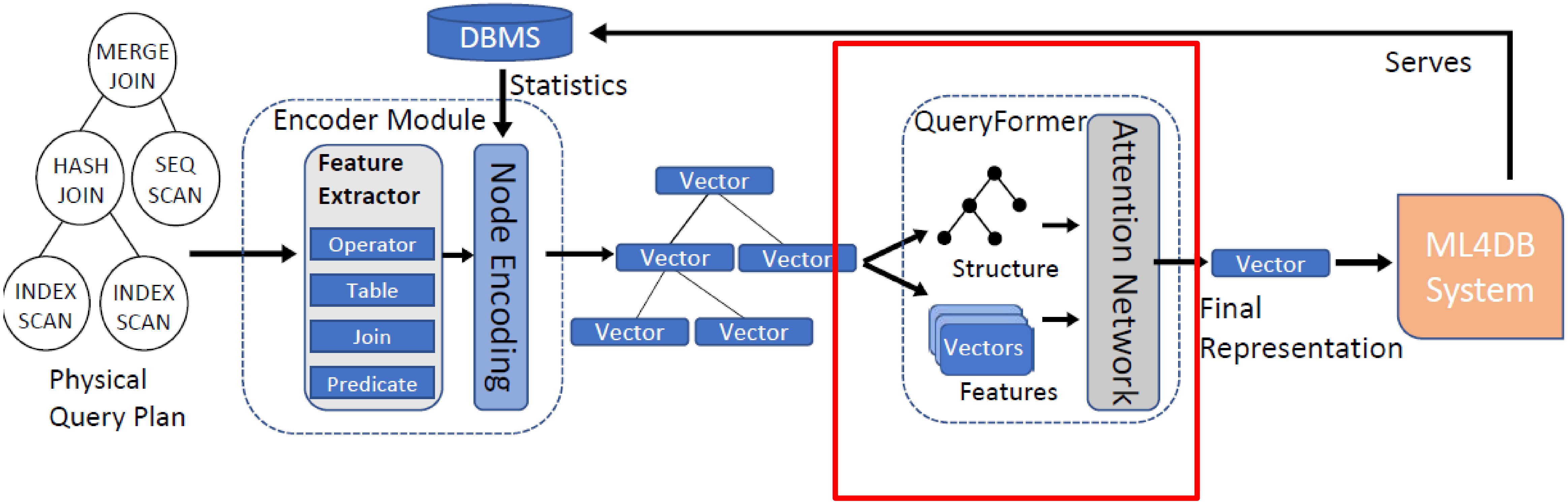
System Overview

- Plug and Play for existing ML4DB works



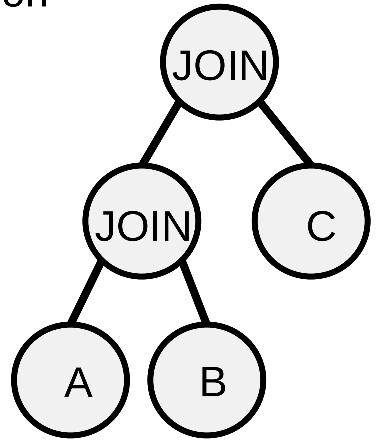
System Overview

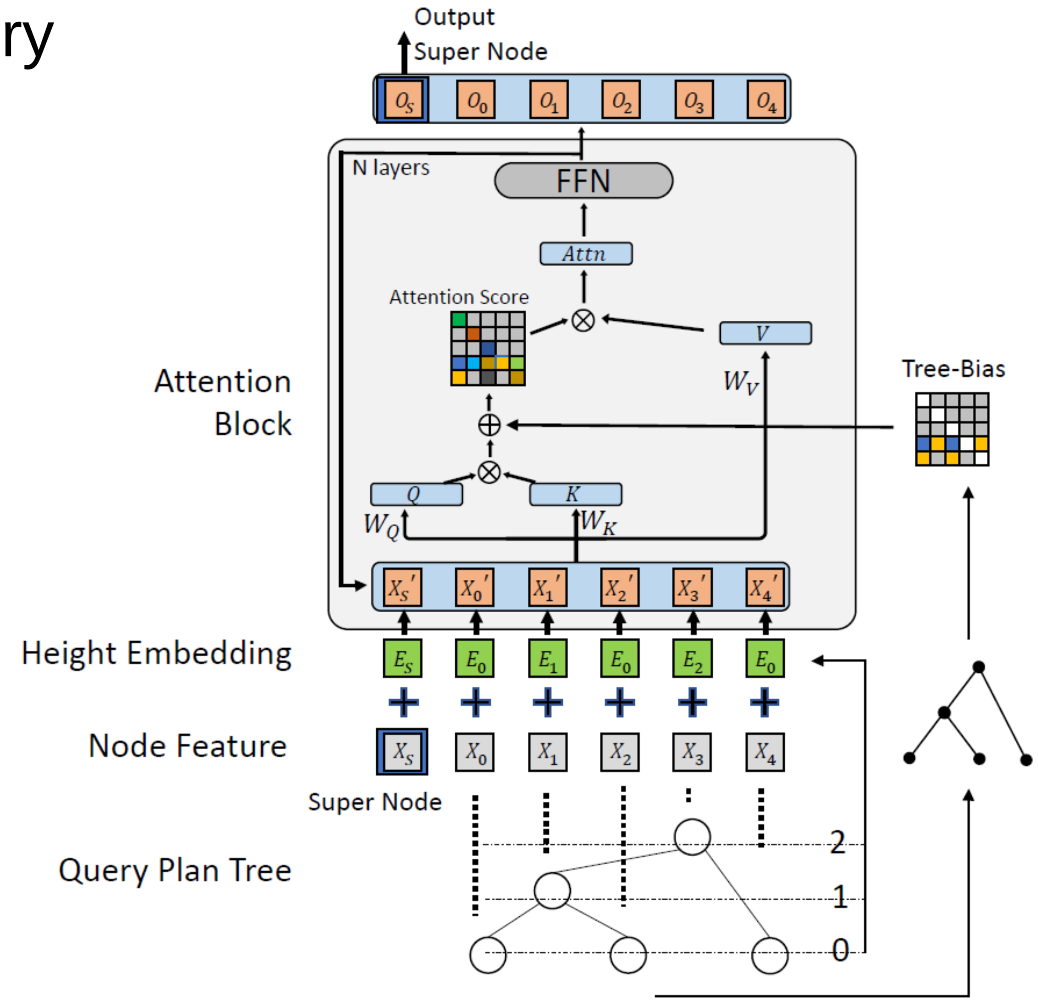
- Plug and Play for existing ML4DB works



QueryFormer Architecture

- Goal: encode the tree structure of query plan
  - Parent-children dependencies
  - Long information paths
- Incorporate the tree structure:
  - 3 new designs from vanilla Transformer
    1. Height Embedding
    2. Tree-Bias Attention
    3. Super Node





Experimental Settings

- Methodology:
  - Perform database tasks by replacing **query plan representation** of ML4DB work. Compare the performance with original ML4DB works
  - Tasks: cost estimation, cardinality estimation, index recommendation, learned optimizer

- Dataset: both synthetic and real workloads with different characteristics

Table 1: Query Plan Sizes in datasets.

Dataset	Max Nodes	Avg Nodes	Max Depth	Avg Depth
JOB-light	14	8.44	10	5.75
Synthetic	10	4.9	7	3.65
TPC-H	26	16.8	15	10.2
TPC-DS	143	44.4	20	15.2
JOB-extend	35	21.2	19	12.2

Experimental Results: Cost Estimation

- Adopt the exact setting of E2E-Cost [Ji, S., et al. VLDB 19]
- Evaluation Metrics:
  - Q-Error:

$$Q(c) = max\left(\frac{actual(c)}{predicted(c)}, \frac{predicted(c)}{actual(c)}\right),$$

  - Pearson Correlation of prediction and labels
- Results:
  - more than 40% improvement in Q-Error when comparing both:
    - **QF** vs **E2E-Cost**
    - **QF-Multi** vs **E2E-Multi**

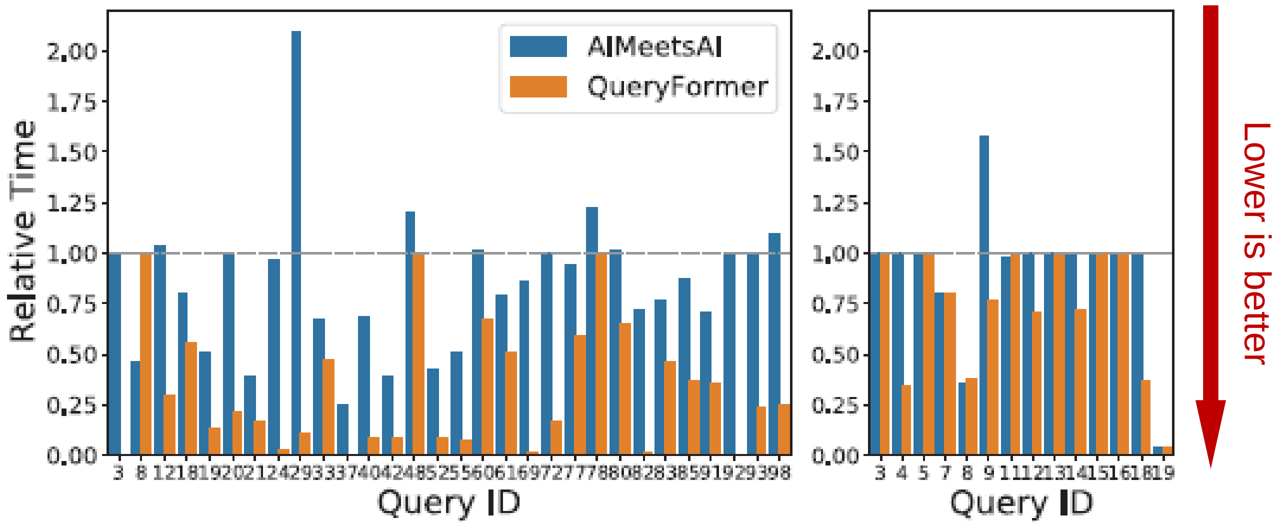
Table 3: Cost Estimation Results.

Synthetic	Q-Error			Corr
	Mean	Median	90%	
PostgreSQL	12.94	3.78	16.48	0.84
MSCN	1.65	1.17	3.67	0.94
E2E-Cost	4.96	1.81	6.13	0.93
E2E-Multi	2.40	1.55	4.24	0.95
QF (no-hist)	1.61	1.09	2.16	0.98
QF (simple)	2.16	1.21	3.40	0.97
QF	**1.48**	1.08	**1.92**	0.992
QF-Multi	1.49	**1.07**	1.94	**0.994**

JOB-light	Q-Error			Corr
	Mean	Median	90%	
PostgreSQL	25.57	2.74	20.90	0.86
MSCN	25.94	3.43	25.53	0.84
E2E-Cost	45.37	3.39	21.80	0.86
E2E-Multi	21.53	4.84	28.21	0.88
QF (no-hist)	17.86	1.52	28.48	0.86
QF (simple)	15.12	2.47	18.40	0.88
QF	**10.43**	**1.50**	**15.46**	**0.91**
QF-Multi	11.41	1.74	17.77	0.90

Experimental Results: Index Recommendation

- Adopt the exact setting of AIMeetsAI [Bailu, D., et al. SIGMOD 19]
- Goal: to select indexes that accelerate query execution
- Relative time:
  - $\frac{\text{Exec. time with indexes}}{\text{Exec. time without any index}}$
- Results:
  - Better indexes are selected → 20% les execution time on average



(a) TPC-DS (b) TPC-H

Fig 3. Relative Execution time of index recommended.

Experimental Results: Optimizer

- Adopt the exact setting of BAO [Ryan, M., et al. SIGMOD 21]
- Goal:
  - To execute a workload (2240 queries) as fast as possible
- Results:
  - 16% less execution time

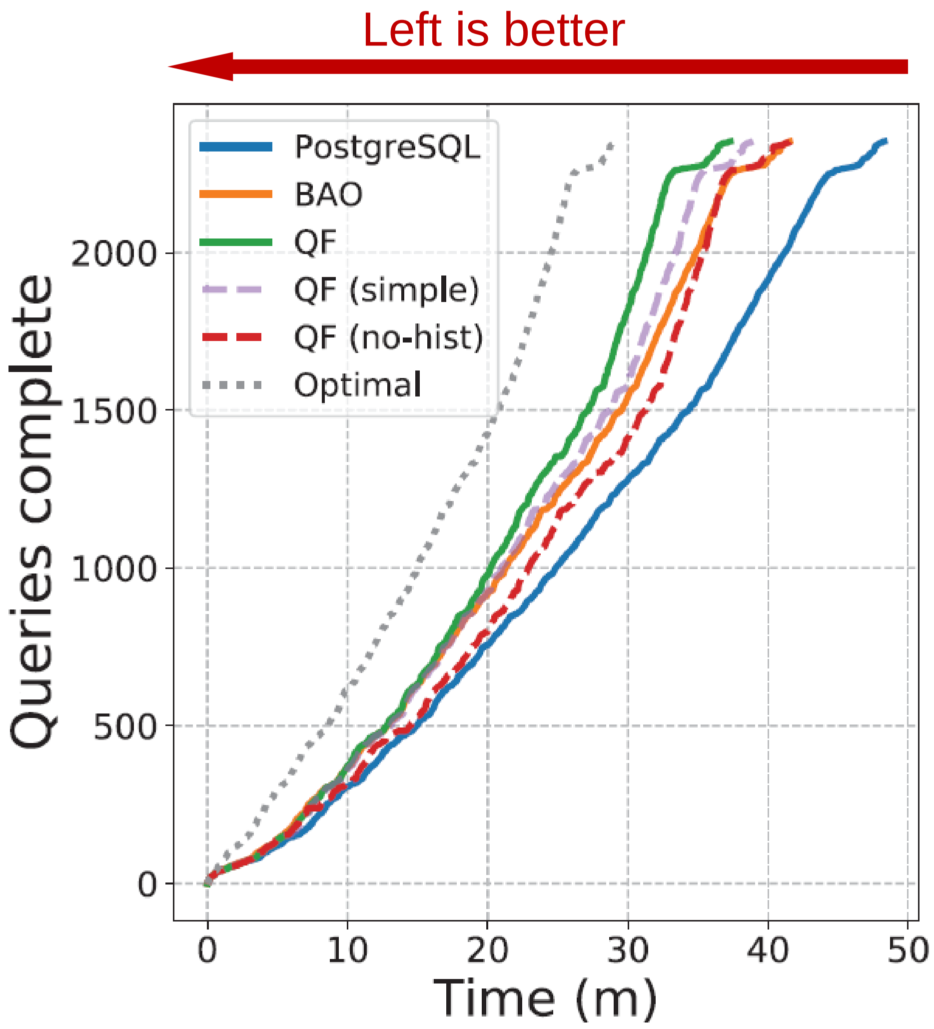


Fig 3. Queries completed over time.

Overview of our Research on Machine Learning for Database (ML4DB)

Query Optimization

- Cardinality Estimation (SIGMOD'21, EBDT'21, VLDB'22, SIGMOD'24)
- Multi-Query Optimization (SIGMOD'24)

Data Access Methods

- ML-enhanced Indexes, data partitioning (SIGMOD'23, VLDB'23, SIGMOD'24)
- Index Selection (VLDB'23)

ML4DB Foundation

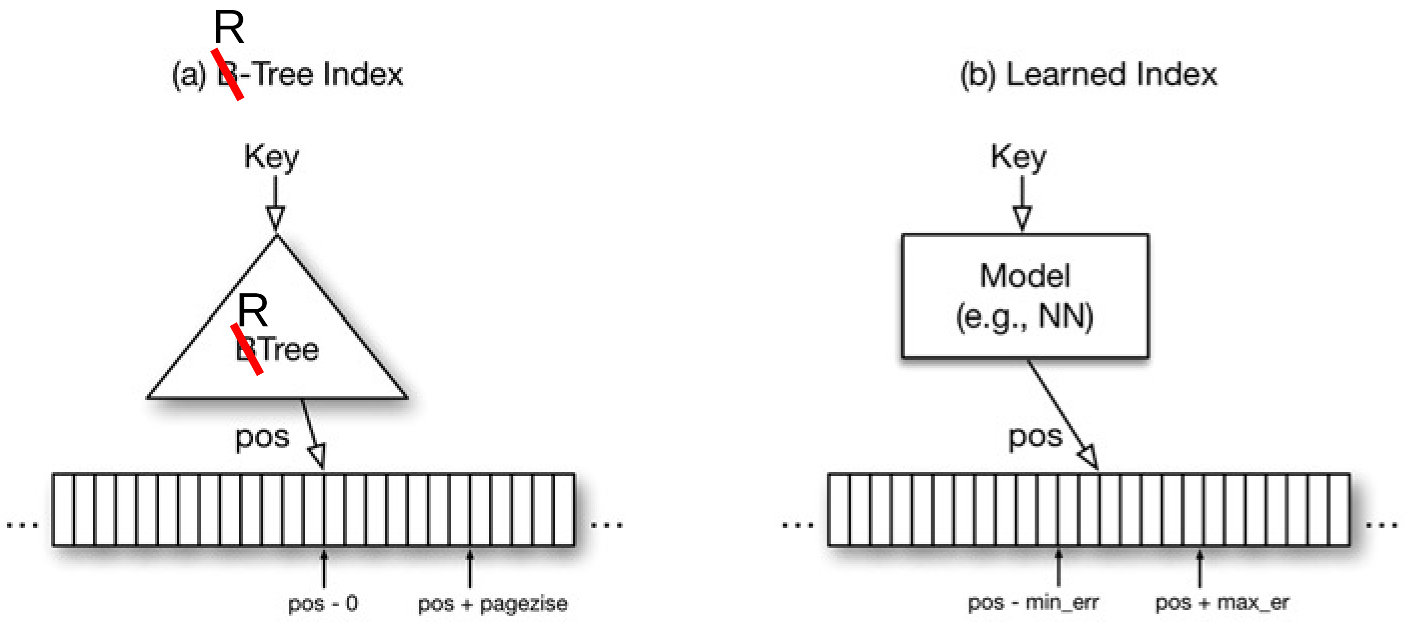
- Query Representation Learning (VLDB'22)

Database Testing & Admin

- Database Generation (SIGMOD'22)

Learned (Spatial) Index

- Learned (spatial) indexes use machine learning models to map real values (spatial coordinates) to storage locations, e.g., RMI, PGM, ZM, RSMI, LISA, etc



Learned Indexes vs. ML Enhanced Indexes

Learned Indexes

- They need to **replace both the index structures and query processing algorithms currently used by the database systems.** Such a radical departure -- >difficult to be deployed in database systems.

- Technical Limitations: Type of **data, Type of queries, Updates, etc.**

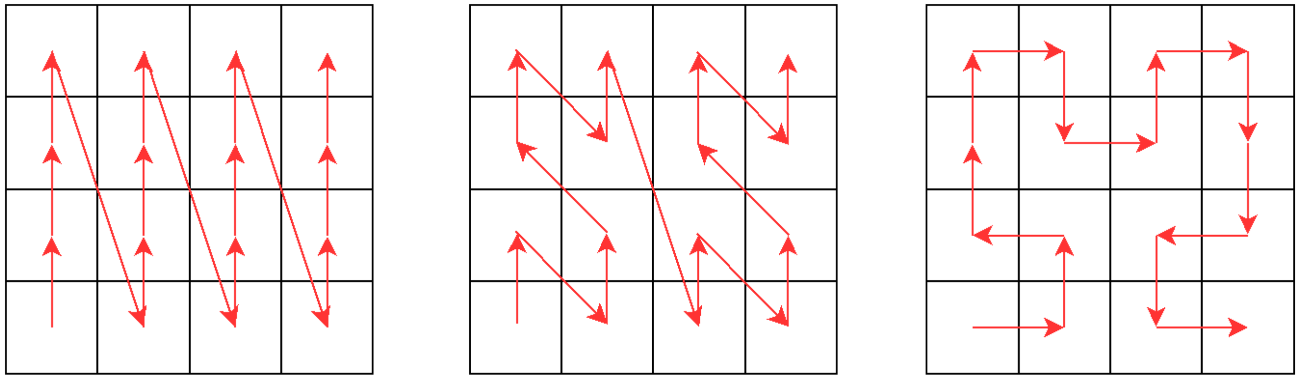
ML Enhanced Indexes: Use ML to enhance (NOT replace) existing indexes

- **RLR-Tree** (SIGMOD'23)**:** a better R-Tree for dynamic data
- **Packing R-tree** (SIGMOD'24): a better R-tree for bulk loading
- **Learned Space-filling Curves**(VLDB'23): for multiple dimensional data indexing or partitioning.

Do not replace the index structures or the query processing algorithms

Space-Filling Curve (SFC)

- A SFC is used to map a multi-dimensional data point to a value
- Then a one-dimensional index can be used to index the mapped values
  - B+tree index, supported by many DBMS, such as PostgreSQL, DynamoDB, HBase
  - Learned indexes

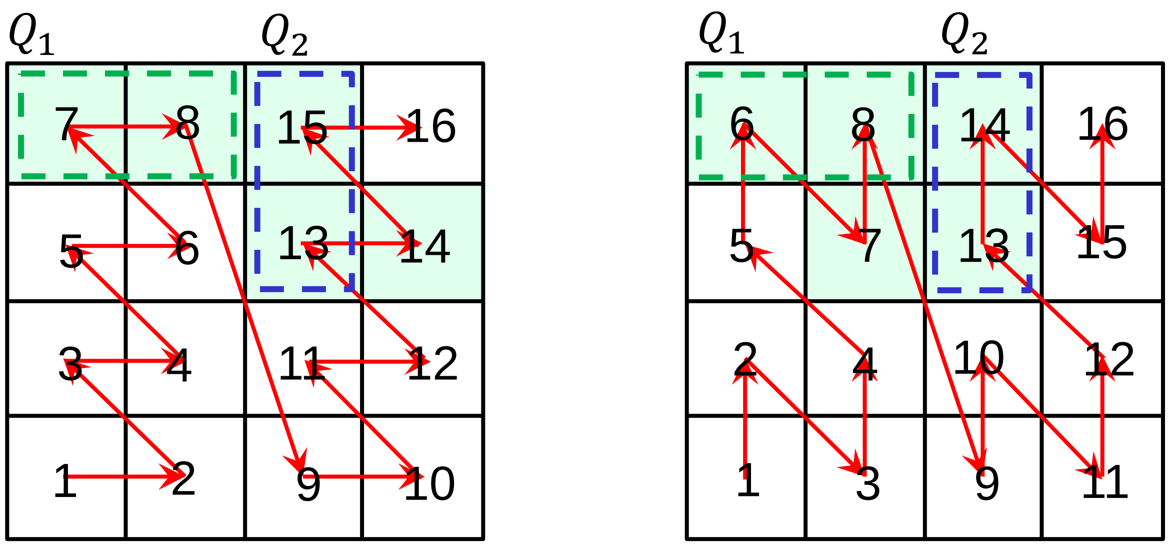


(a) C-curve (b) Z-curve (c) Hilbert curve

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

Design instance-optimized SFCs

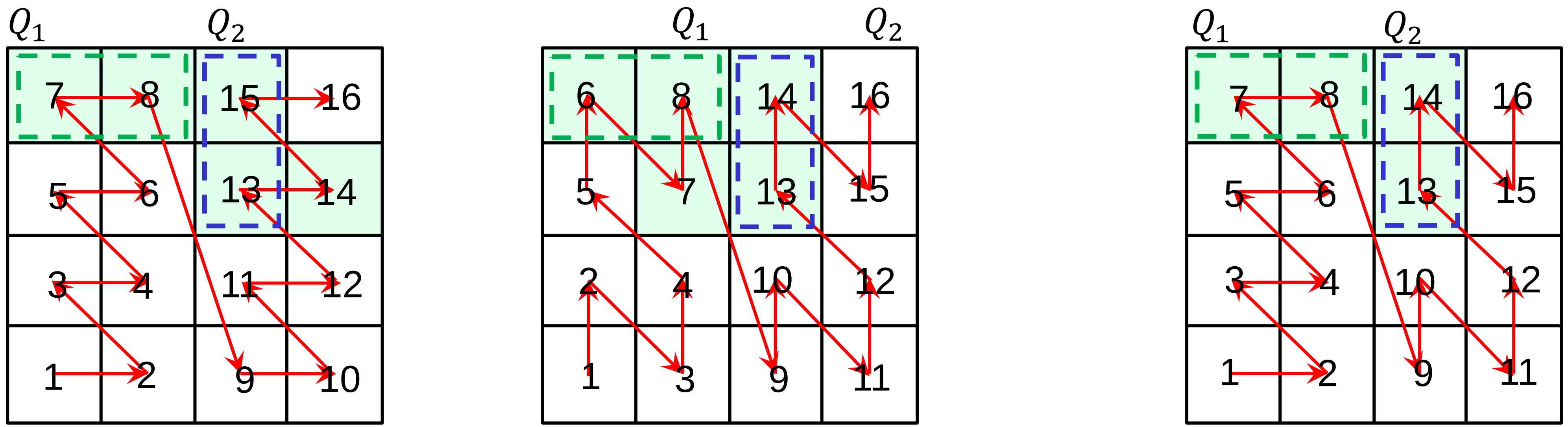
- No single SFC can dominate the performance on all datasets and query workloads



(a) SFC-1 works best for Q_1.(b) SFC-2 works best for Q_2.

Our Idea

- 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, Zheng Wang, Gao Cong, Cheng Long, Han Mao Kiah, Bin Cui. **Towards Designing and Learning Piecewise Space-Filling Curves.** VLDB23

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.

Desired Properties

- Two preferred properties for an SFC mapping $T: \mathbf{x} \rightarrow v$
  - Injection property:

$$\forall \mathbf{x}_1 \neq \mathbf{x}_2, T(\mathbf{x}_1) \neq T(\mathbf{x}_2)$$

  - Monotonicity property:

$$\mathbf{x}' = \{b_1', \dots, b_n'\}$$
$$\mathbf{x}'' = \{b_1'', \dots, b_n''\}$$

If $d_i' \geq d_i''$ is satisfied for $\forall i \in [1, n]$:

$$T(\mathbf{x}') \geq T(\mathbf{x}'')$$

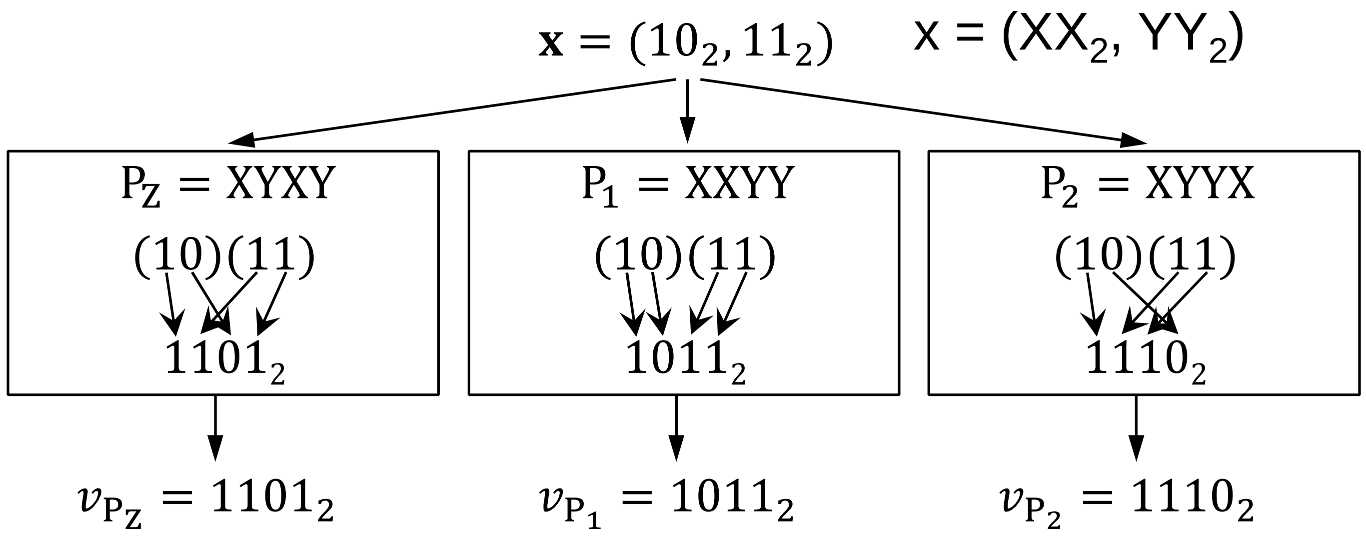
Monotonicity is desirable for designing window query algorithms:
It guarantees that the SFC values of data points in a query rectangle fall in the range of **the SFC values formed by two boundary points of the query rectangle**

Design Challenges

1. How to **partition the space and design an effective BMP** for each subspace?
2. How to design **piecewise SFCs** such that two properties hold?
3. How to design an **instance-optimized** piecewise SFC, given a database and query workload?

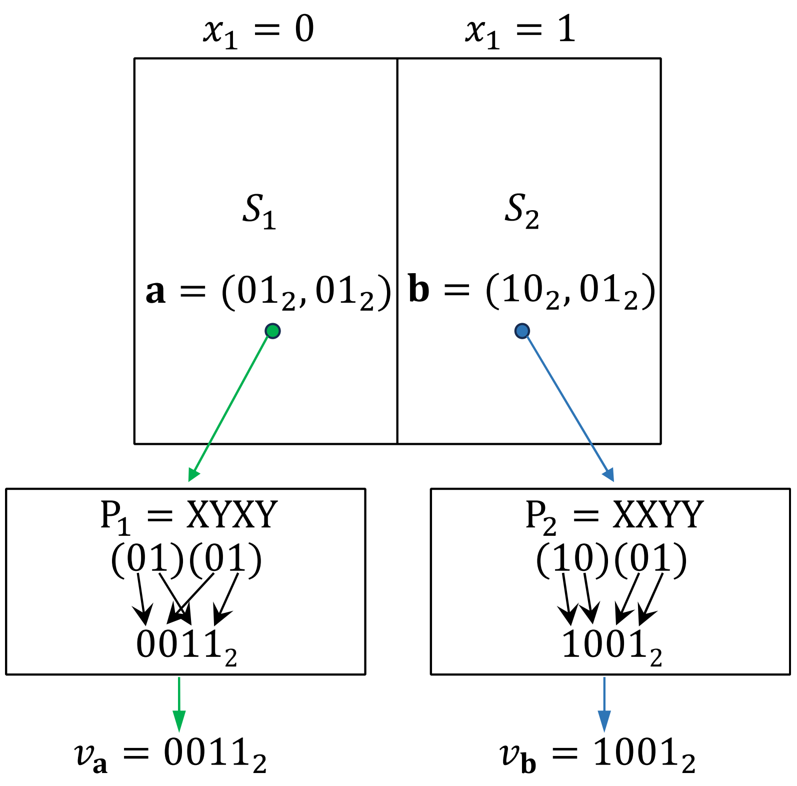
Bit Merging Pattern (BMP)

- 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)



Piecewise SFC Design

- We propose a way of seamlessly integrating the subspace partitioning and BMP generation.



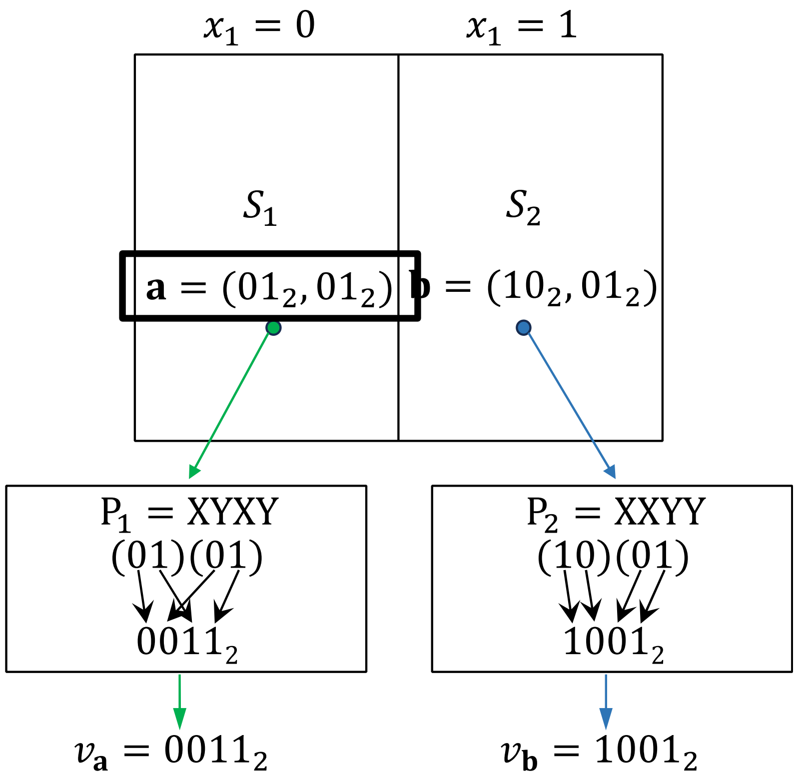
(a) Example of Piecewise SFC Design.

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).
- …

Piecewise SFC Design

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



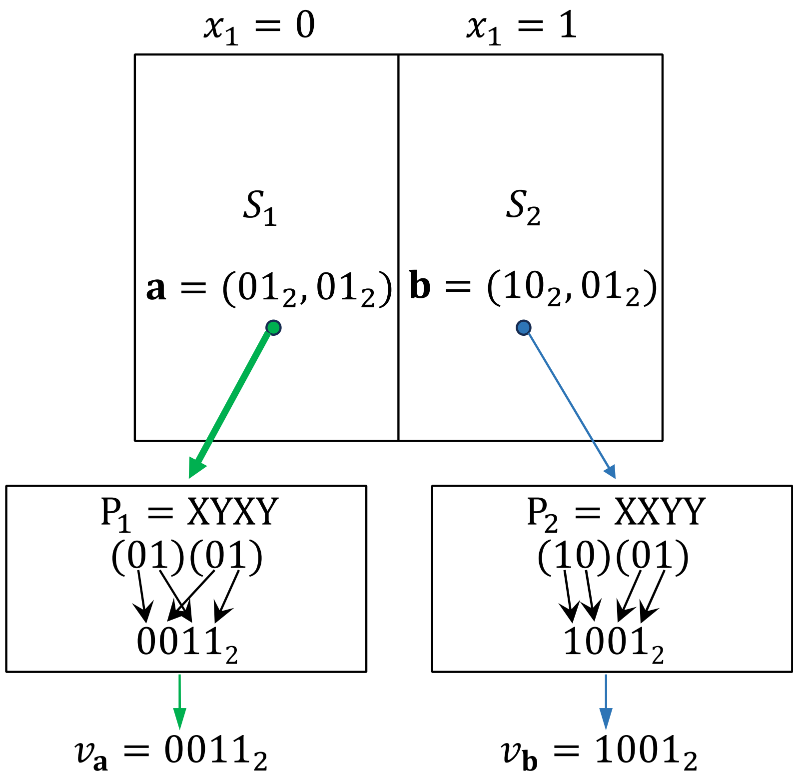
(a) Example of Piecewise SFC Design.

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).
- …

Piecewise SFC Design

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



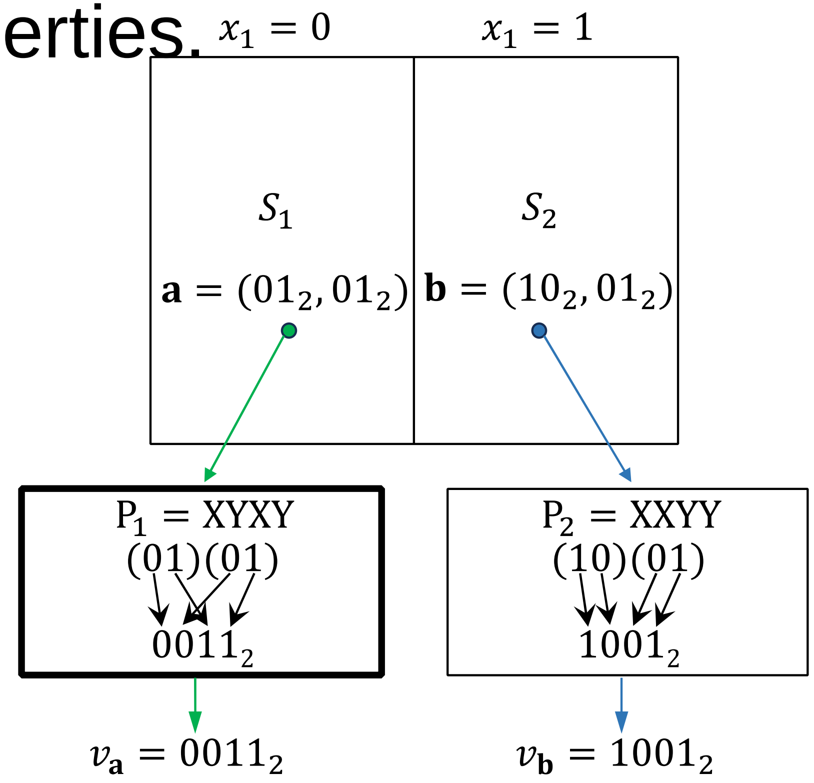
(a) Example of Piecewise SFC Design.

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).
- …

Piecewise SFC Design

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



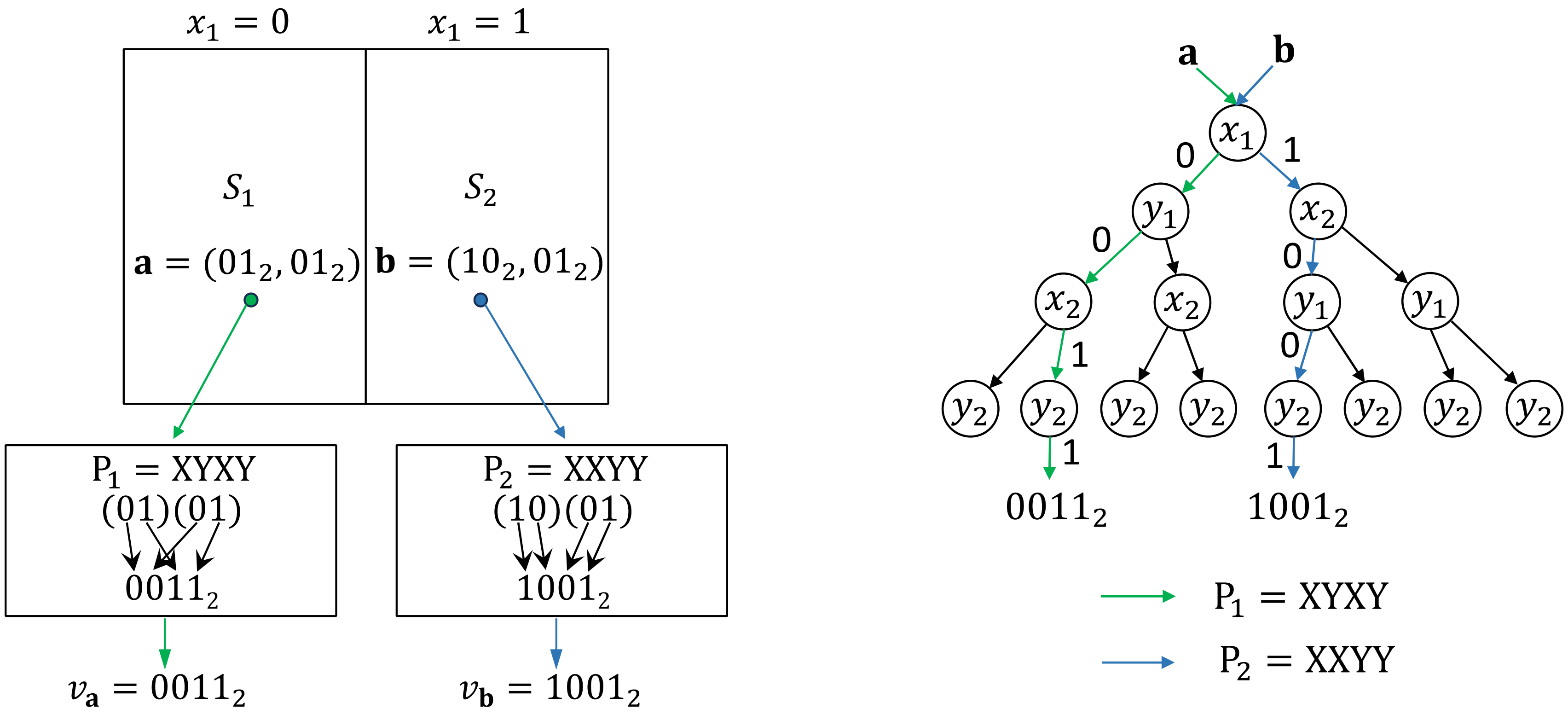
(a) Example of Piecewise SFC Design.

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).
- …

Bit Merging Tree (BMTree)

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

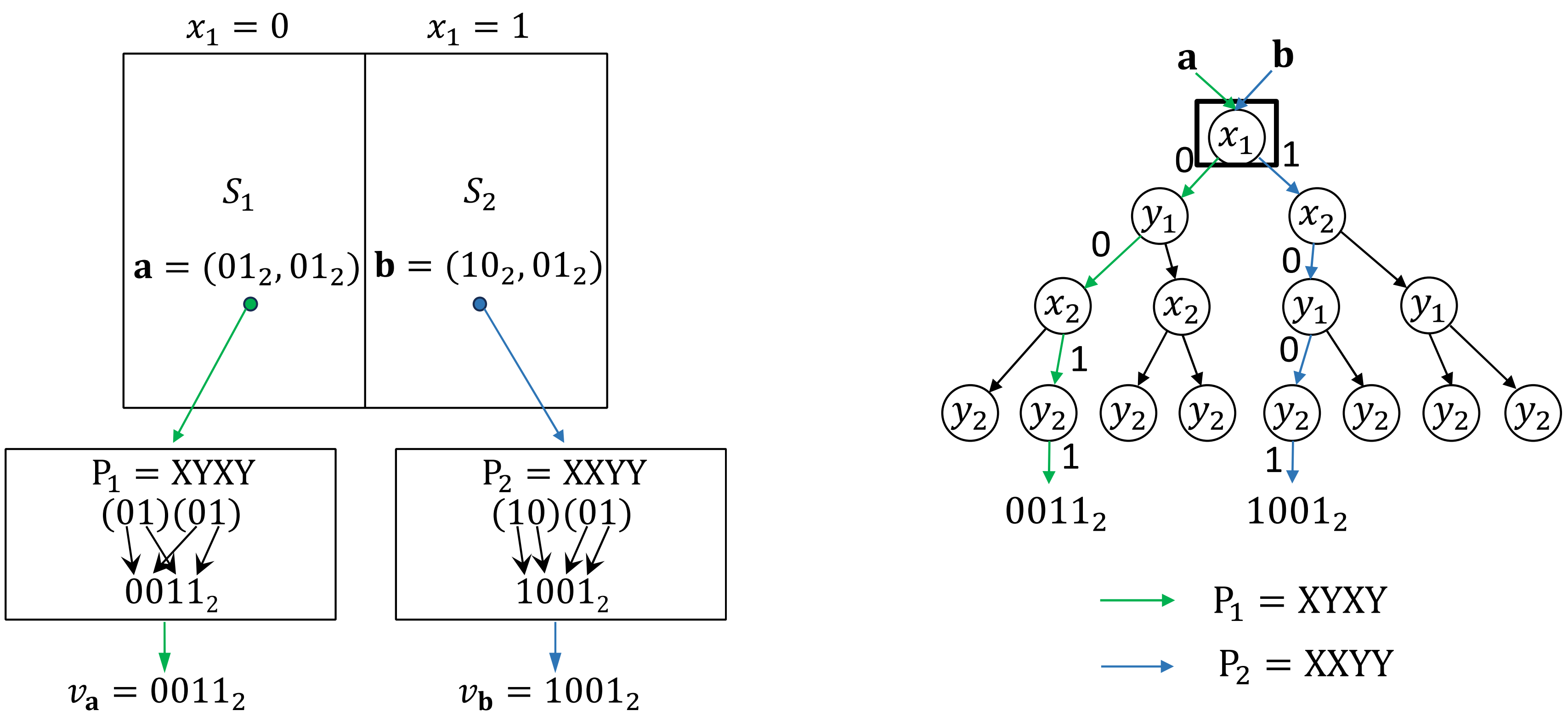


(a) Example of Piecewise SFC Design.

(b) Example of BMTree Structure.

Bit Merging Tree (BMTree)

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

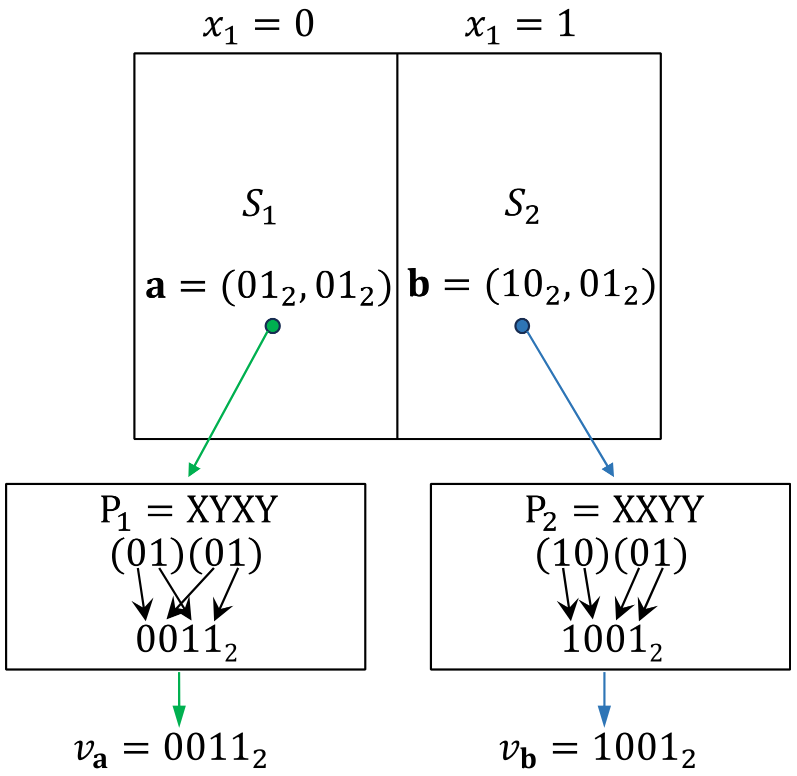


(a) Example of Piecewise SFC Design.

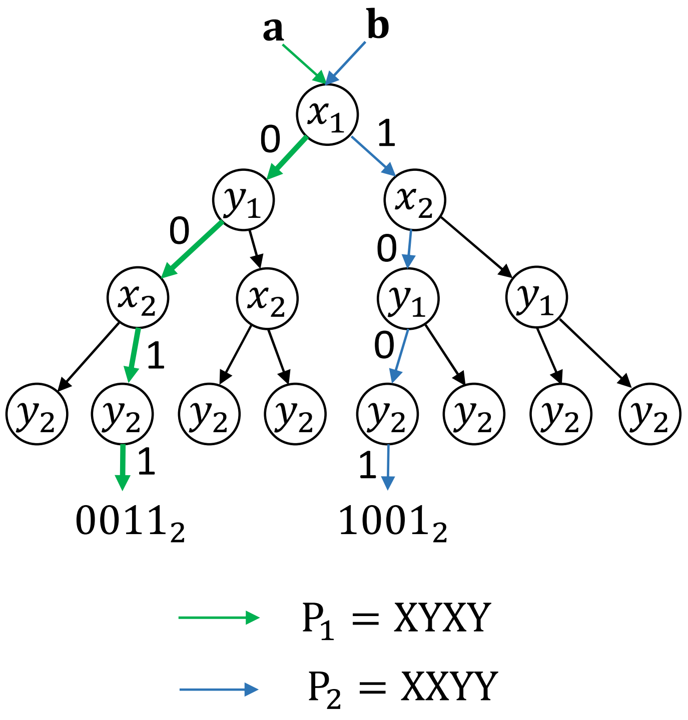
(b) Example of BMTree Structure.

Bit Merging Tree (BMTree)

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



(a) Example of Piecewise SFC Design.



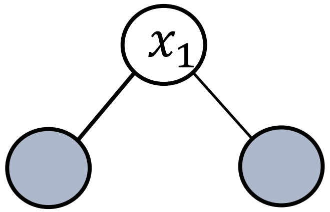
(b) Example of BMTree Structure.

BMTree Construction

- We model the SFC design procedure as the BMTree construction procedure.
  - Each time we fill one level of BMTree with the selected bits---partition more subspaces and generate the next level of nodes.

(1) BMTree whose root node is filled with x_1

x = $(x_1\ x_2\ ,\ y_1\ y_2)$



(2) Possible bit choices to fill the two leaf nodes

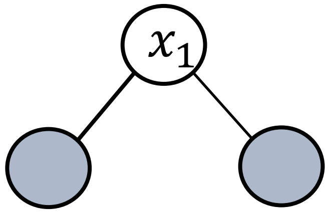
1. Left: x_2, Right: x_2
2. Left: x_2, Right: y_1
3. Left: y_1, Right x_2
4. Left: y_1, Right y_1

BMTree Construction

- We model the piecewise SFC design procedure as the BMTree construction procedure
  - Each time we fill one level of BMTree with the selected bits---partition more subspaces and generate the next level of nodes.

(1) BMTree whose root node is filled with x_1

x = $(x_1\ x_2\ ,\ y_1\ y_2)$



(2) Possible bit choices to fill the two leaf nodes

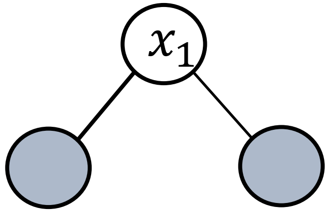
1. Left: x_2, Right: x_2
2. Left: x_2, Right: y_1
3. Left: y_1, Right x_2
4. Left: y_1, Right y_1

BMTree Construction

- We model the piecewise SFC design procedure as the BMTree construction procedure.
  - Each time we fill one level of BMTree with the selected bits---partition more subspaces and generate the next level of nodes.

(1) BMTree whose root node is filled with x_1

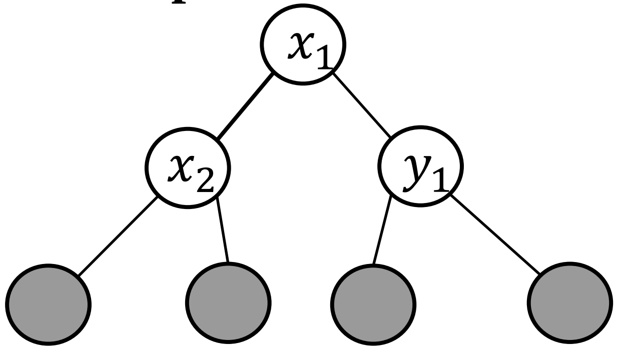
x = $(x_1\ x_2\ ,\ y_1\ y_2)$



(2) Possible bit choices to fill the two leaf nodes

1. Left: x_2, Right: x_2
2. Left: x_2, Right: y_1
3. Left: y_1, Right x_2
4. Left: y_1, Right y_1

(3) BMTree constructed one level deeper

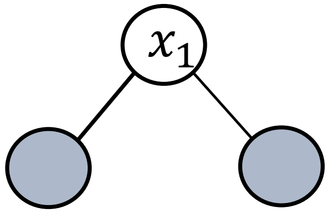


BMTree Construction

- We model the piecewise SFC design procedure as the BMTree construction procedure
  - Each time we fill one level of BMTree with the selected bits---partition more subspaces and generate the next level of nodes.

(1) BMTree whose root node is filled with x_1

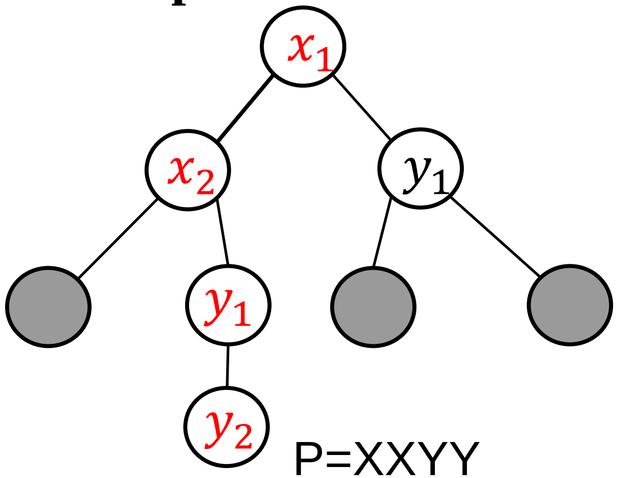
x = $(x_1\ x_2\ ,\ y_1\ y_2)$



(2) Possible bit choices to fill the two leaf nodes

1. Left: x_2, Right: x_2
2. Left: x_2, Right: y_1
3. Left: y_1, Right x_2
4. Left: y_1, Right y_1

(3) BMTree constructed one level deeper



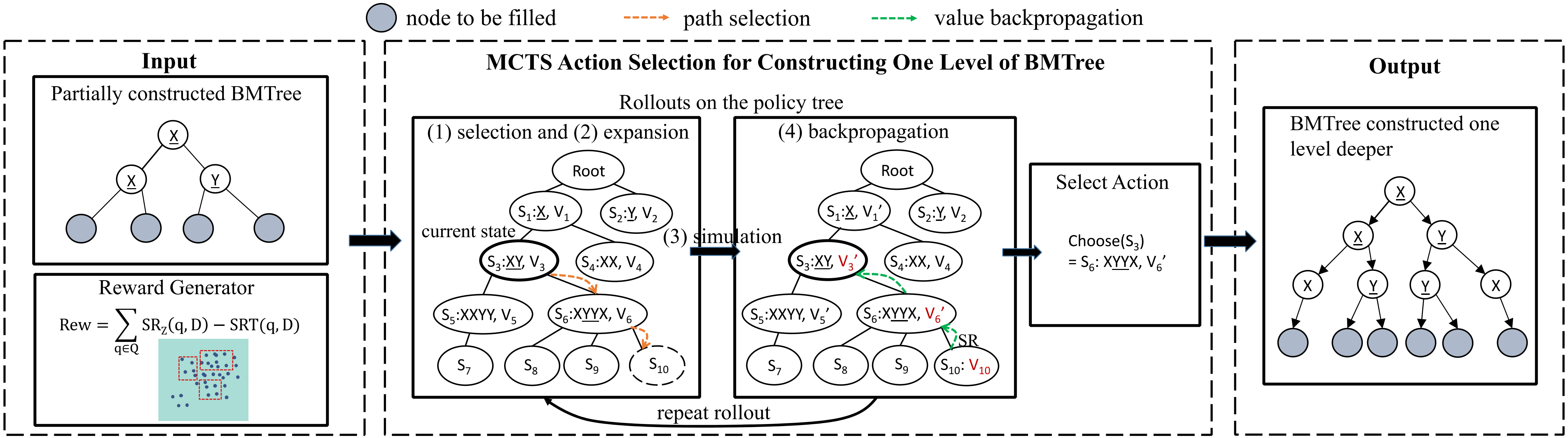
Use Reinforcement Learning to construct BMTree

Why reinforcement learning:

- Heuristic methods are difficult to be designed.
- Modeled as a sequence of actions to select bits for tree nodes
- Utilizing reinforcement learning can directly optimize the BMTree based on the reward.

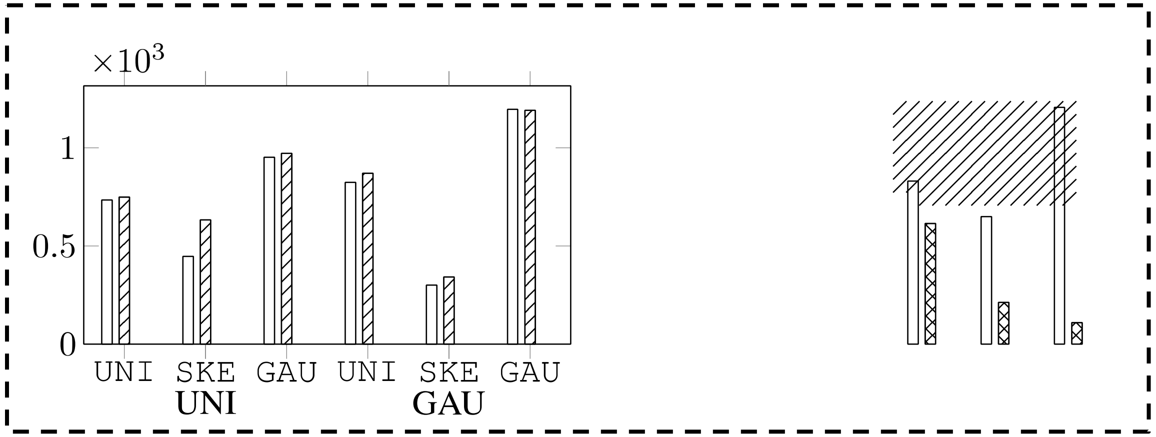
MCTS based BMTree Construction

- We leverage Monte Carlo Tree Search method to help constructing BMTree.

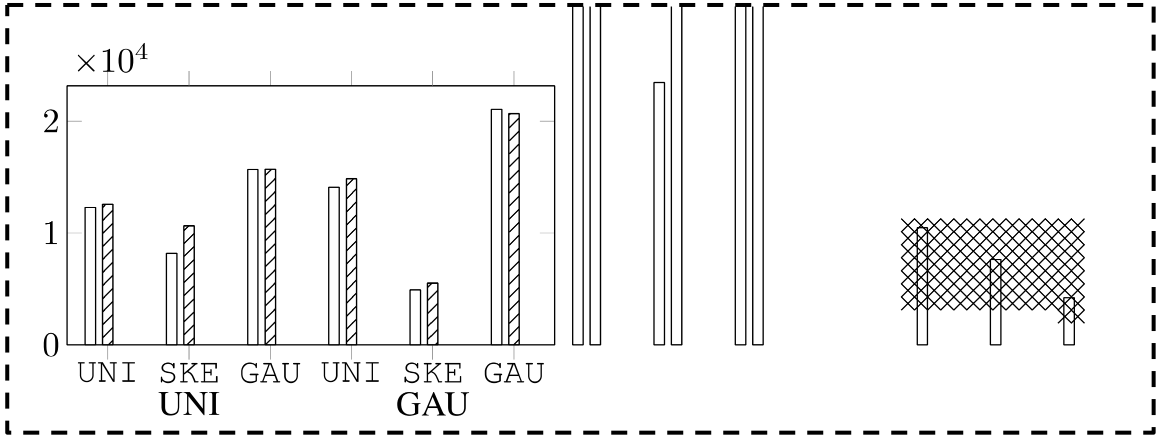


Comparing between SFCs

- Experiment on PostgreSQL.



(a) I/O Cost



(b) Query Latency

QUILTS, SIGMOD'17

Motivation: RLR-tree

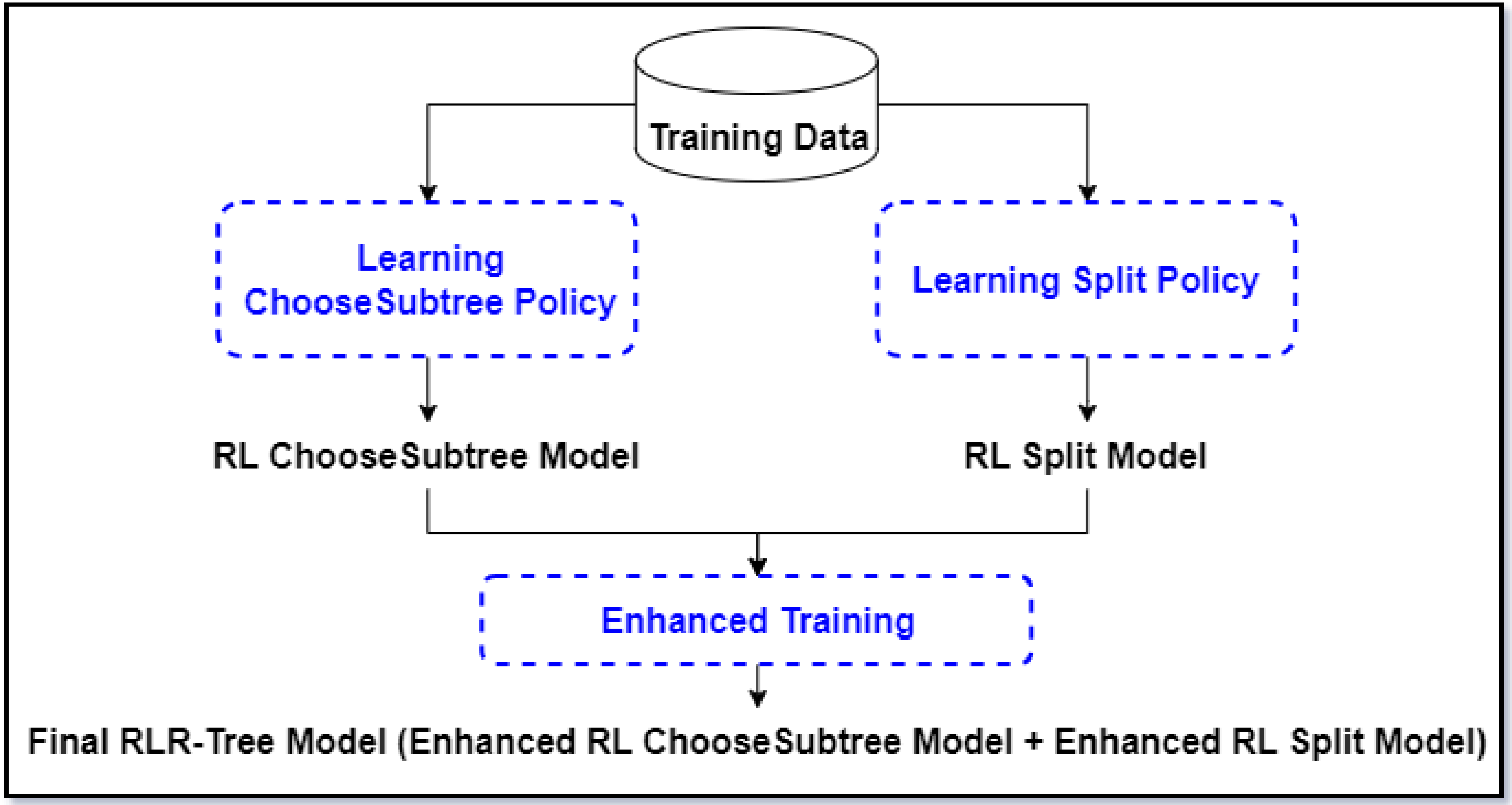
- Two key operations of R-Tree, i.e., **ChooseSubtree** and **Split**.
  - **ChooseSubtree**: starting from the tree root, recursively choose which child node to insert the new data object, until a leaf node is reached.
  - **Split**: If the number of entries in a node exceeds the capacity, the Split 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:** use **machine learning (ML) to construct a better R-Tree** for better query efficiency in a dynamic environment.
  - We **do NOT learn the data distribution (CDF)**.
  - We model ChooseSubtree and Split as two Markov Decision Processes (MDPs) and train **reinforcement learning (RL) models** to learn optimal policies.

Jiangneng Li, Zheng Wang, Gao Cong, Cheng Long, Han Mao Kiah, Bin Cui. **Towards Designing and Learning Piecewise Space-Filling Curves.** VLDB23

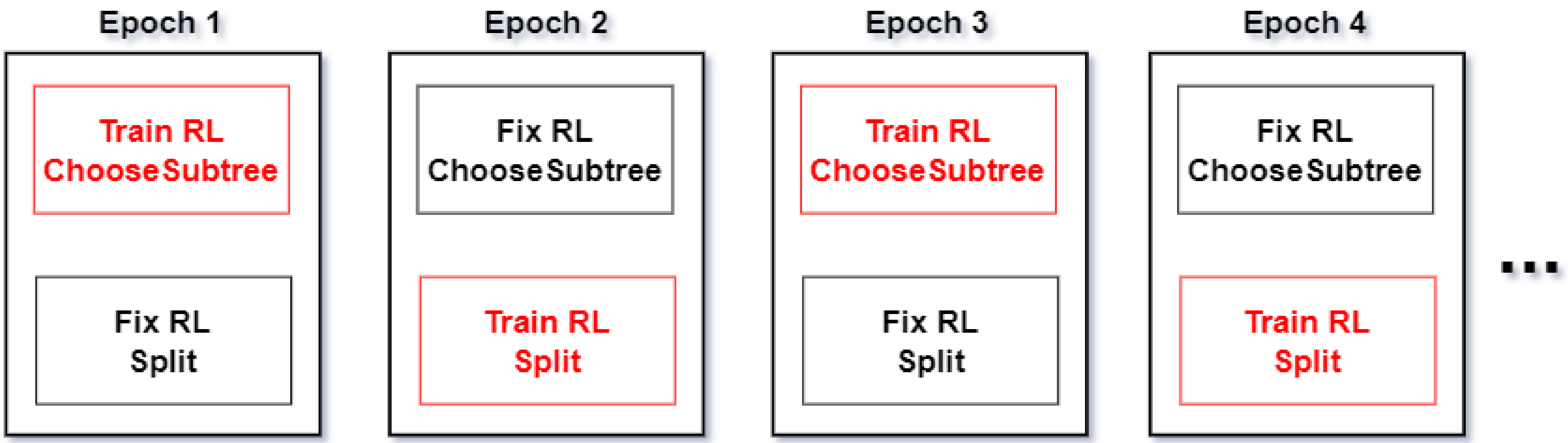
RLR-Tree Overview

Overview (Offline Training):



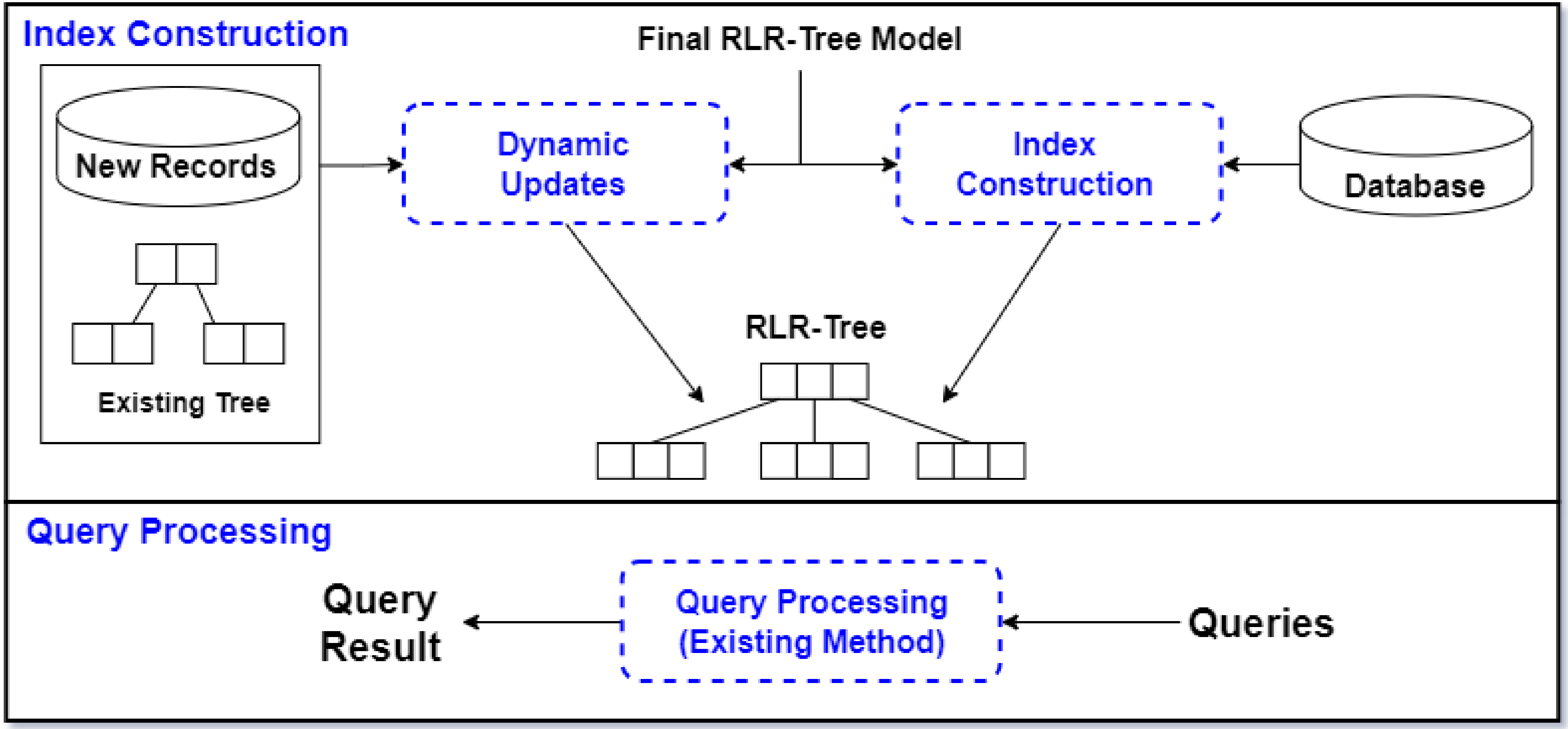
RLR-Tree Enhanced Training

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



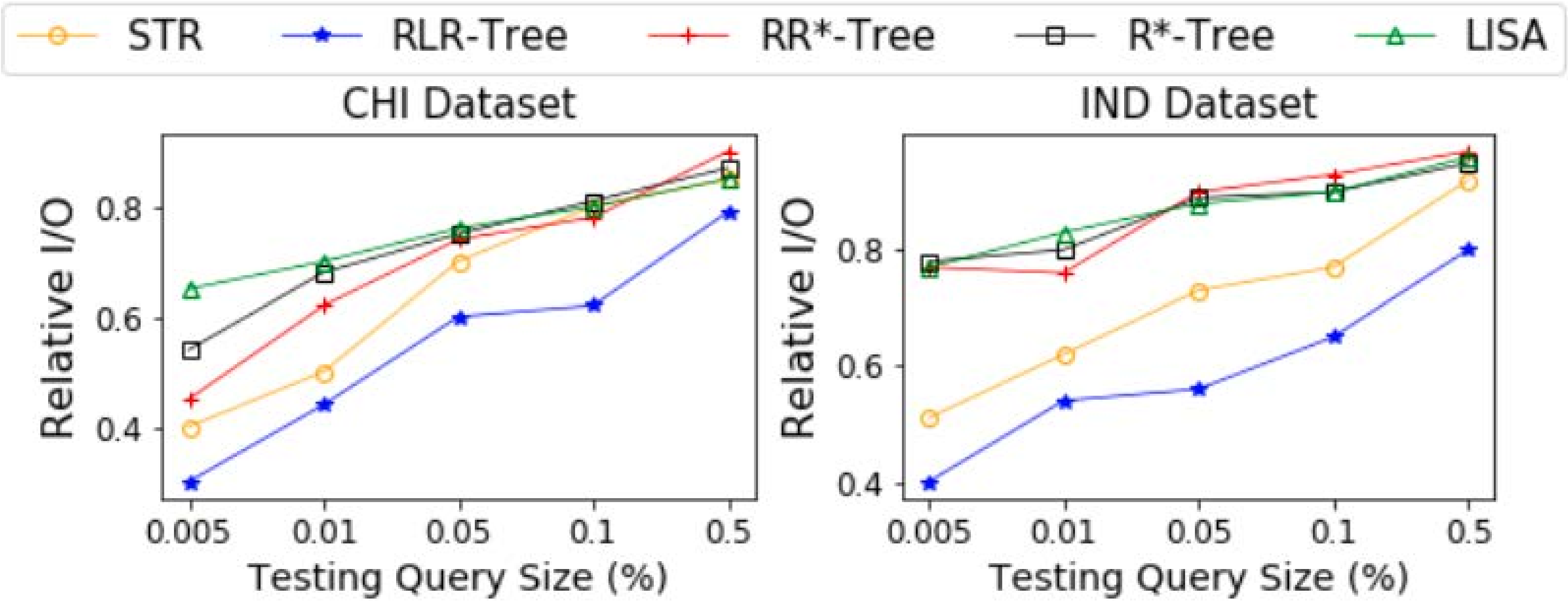
RLR-Tree Overview

Overview (Index Construction & Query Processing):



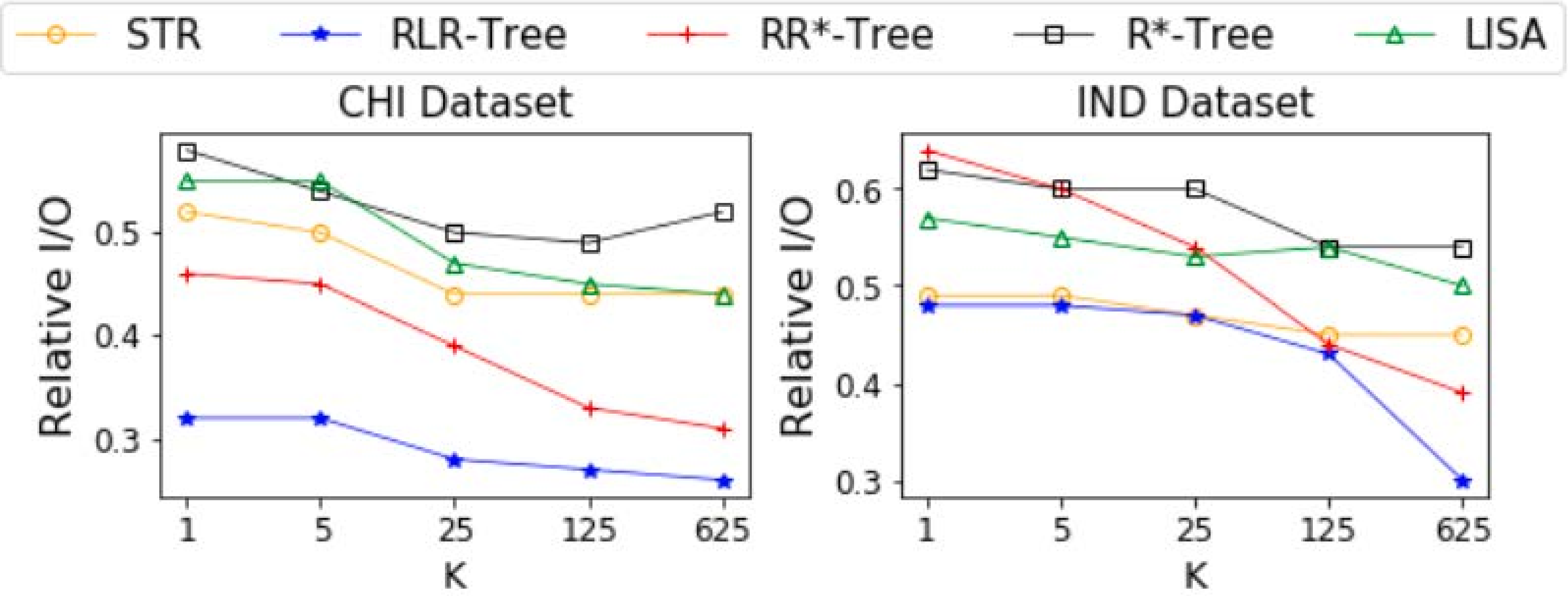
Experimental Results

RLR-Tree performance on range queries



Experimental Results

RLR-Tree performance on KNN queries



Summary

Query Optimization

- Cardinality Estimation (SIGMOD’21, EBDT’21, VLDB’22, SIGMOD’24)
- Multi-Query Optimization (SIGMOD’24)

Data Access Methods

- ML-enhanced Indexes, data partitioning (SIGMOD’23, VLDB’23, SIGMOD’24)
- Index Selection (VLDB’23)

ML4DB Foundation

- Query Representation Learning (VLDB’22)

Database Testing & Admin

- Database Generation (SIGMOD’22)

Open problems

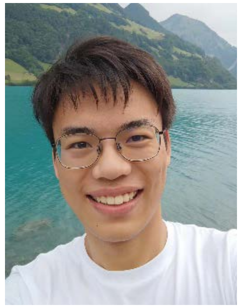
- Foundations for ML4DB tasks
- Foundation models for ML4DB tasks
  - Self-supervised
  - Capability to generalize to different data
  - Capability to generate across tasks
- How to handle data shift and workload shift
  - Fine-tune models
  - Transfer learning
- How to generate training data of high quality and of low cost
- What are important open problems in data systems?

References and Acknowledgement

- Jingyi Yang, Peizhi Wu, Gao Cong, Tieying Zhang, Xiao He. SAM: **Database Generation** from Query Workloads with Supervised Autoregressive Models. SIGMOD 2022.
- Yue Zhao, Gao Cong, Jiachen Shi, Chunyan Miao. QueryFormer: a tree transformer model for **query plan representation**. VLDB 2022.
- Song Song Mo, Yile Chen, Hao Wang, Gao Cong, Zhifeng Bao. Lemo: A Cache-Enhanced **Learned Optimizer** for Concurrent Queries. SIGMOD 2024.
- Jiachen Shi, Gao Cong, Xiaoli Li. Learned Index Benefits: Machine Learning Based **Index Performance Estimation**. VLDB 2023.
- Jiangneng Li, Zheng Wang, Gao Cong, Cheng Long, Han Mao Kiah, Bin Cui. Towards Designing and **Learning Piecewise Space-Filling Curves**. VLDB 2023.
- Tu Gu, Kaiyu Feng, Gao Cong, Cheng Long, Zheng Wang, Sheng Wang. The RLR-Tree: **A Reinforcement Learning Based R-Tree** for Spatial Data. SIGMOD 2023.
- Jingyi Yang, Gao Cong. PLATON: Top-down **R-tree Packing** with Learned Partition Policy. SIGMOD 2024.

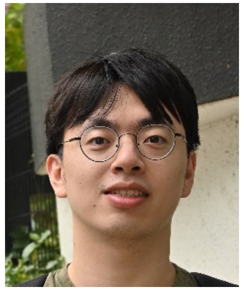
Jingyi Yang

Songsong Mo



Jiangneng Li

Peizhi Wu



Yue Zhao

Jiachen Shi

Tu Gu