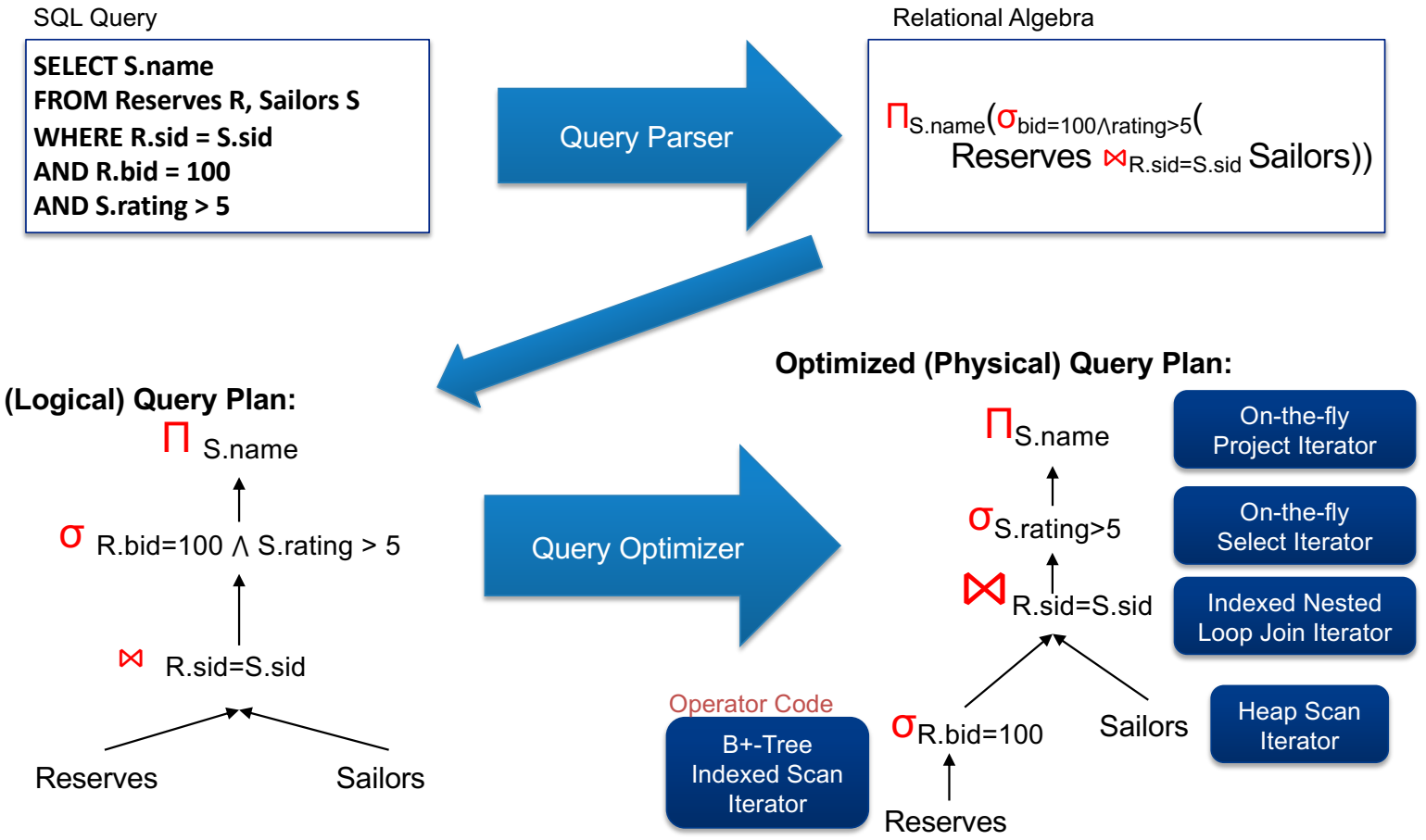
Learning-based Cardinality/Cost Estimation

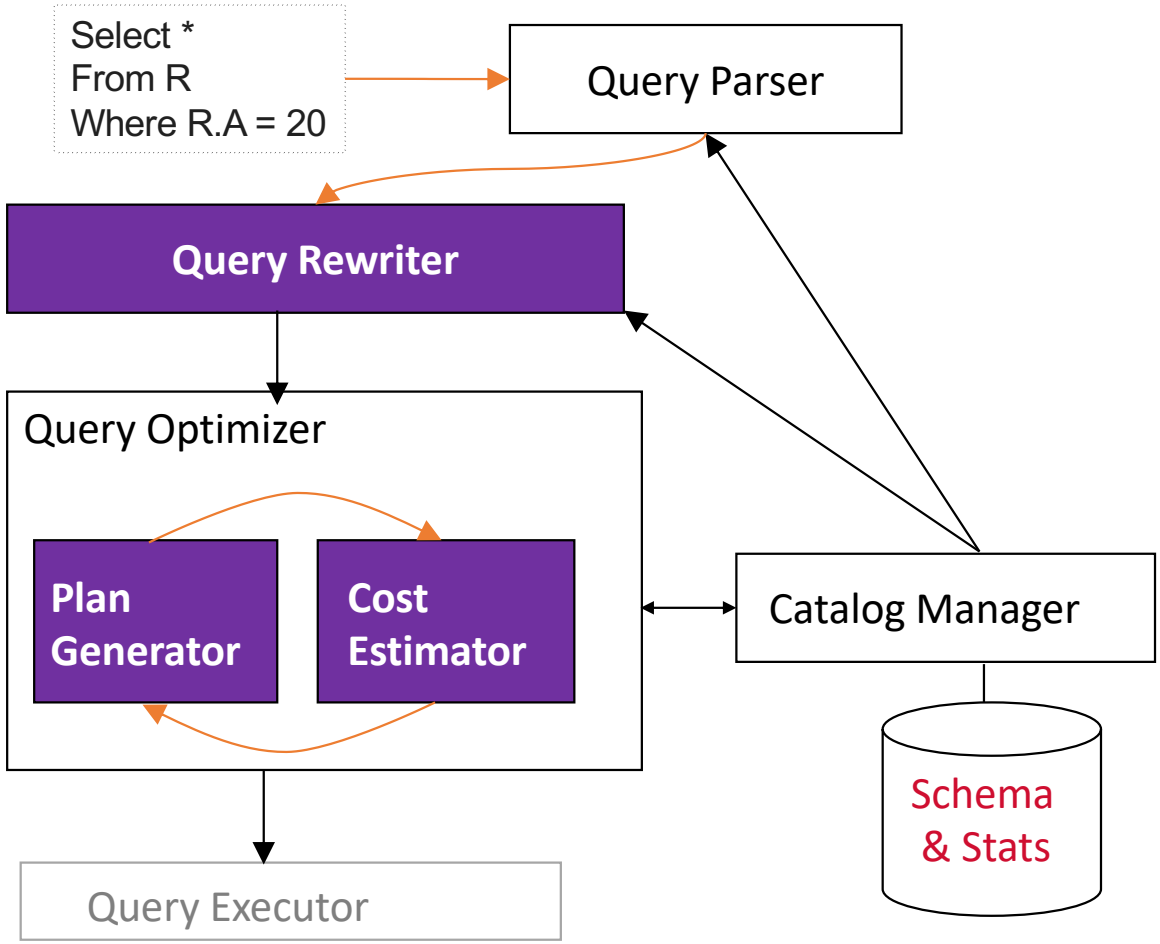
Guoliang Li

Tsinghua University

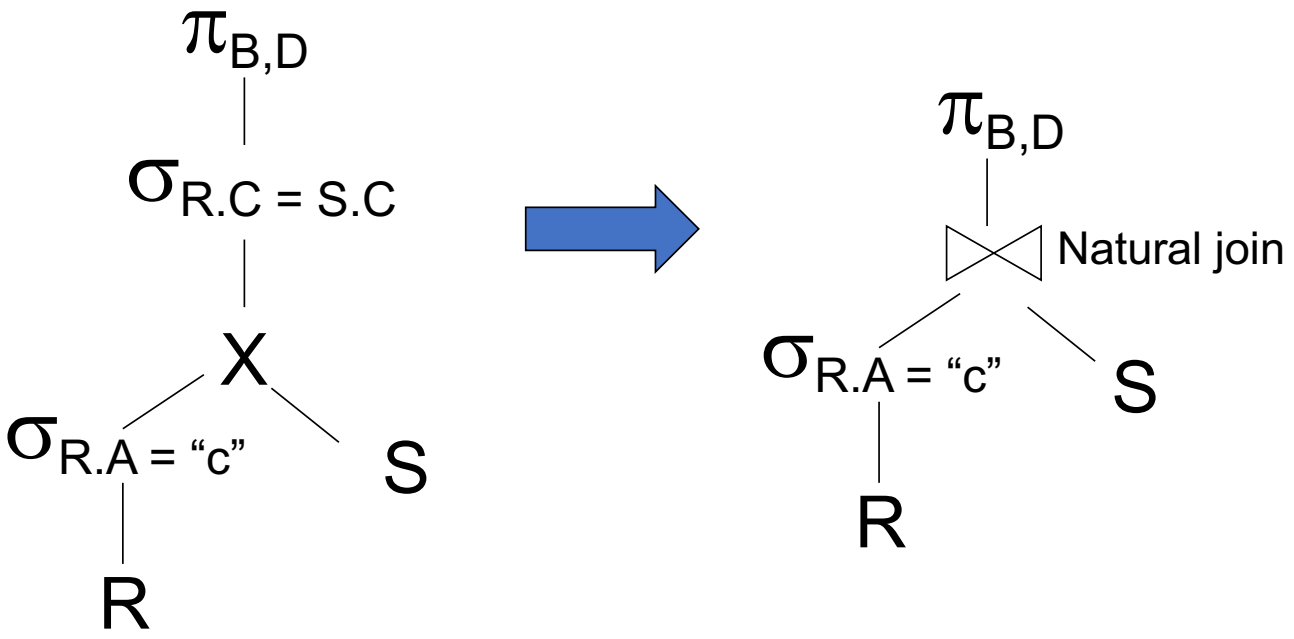
Query Optimizer



Query Optimizer

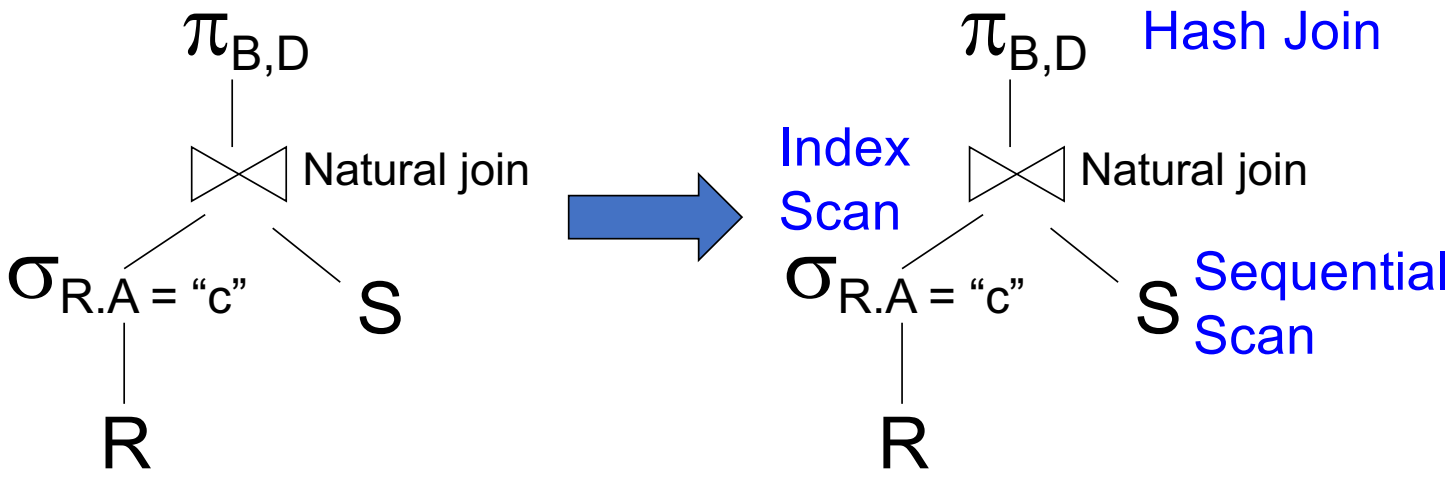


Logical Optimization – Query Rewrite



$$\pi_{B,D}[[\sigma_{R.A=\text{“c”}}(R)] \bowtie S]$$

Physical Optimization



$$\pi_{B,D}[[\sigma_{R.A=\text{“c”}}(R)] \bowtie S]$$

Cardinality Estimation: Selection

Selectivity Factor (SF) = Cardinality / #tuples

Assumptions:

- Uniformity
  - $\sigma_{A=c}(R)$ → SF = 1/V(R,A)
  - $\sigma_{A<c}(R)$→ SF = (c - Low(R, A))/(High(R,A) - Low(R,A))
- Independence
  - Cond1 and Cond2 → SF = SF(Cond1) * SF(Cond2)
  - Cond1 or Cond2 → SF = SF(Cond1)+SF(Cond2) -SF(Cond1)* SF(Cond2)
  - Not Cond1 → SF = 1- SF(Cond1)
- Containment of values
  - $R \bowtie_{A=B} S$ → SF = 1 / max(V(R,A),V(S,B))
- Preservation of values
  - $V(R \bowtie_{A=B} S, C) = V(R, C)$

Cardinality Estimation: Selection

Q = SELECT list
FROM R1, ..., Rn
WHERE $cond_1$ AND $cond_2$ AND . . . AND $cond_k$

- Estimate the number of results of Q: T(Q)
- Obtain number of tuples in each table: T(R1), T(R2), …, T(Rn)
- Also need the selectivity of each condition
  - Selectivity factor (SF) of selection and joins
  - SF(R1.A=v)=T(R1)/V(R1,A)
  - SF(R1.A=R2.B)=T(R1) T(R2)/ max(V(R1,A), V(R2,B))
  - e.g., selectivity(A=3) = 0.01
  - e.g., selective (R1.A=R2.B) = 0.001

T(Q) = T(R1) × ... × T(Rn) × SF(cond1) × ... × SF(condk)
Remark: T(Q) ≤ T(R1) × T(R2) × ... × T(Rn)

Cardinality Estimation: Predicate

- The **selectivity** (**sel**) of a predicate **P** is the fraction/probability of tuples that qualify.
- Formula depends on type of predicate:
  - Equality(=): sel(P(c=x))=count(c=x)/count(all)
  - Range(>=): sel(P(c>=x))=(max–x+1) / (max–min+1)
  - Negation (!=) : sel(P(c!=x))=1-sel(P(c=x))
  - Conjunction (and)
    - Independent assumption
      - sel(P1 & P2) = sel(P1) * sel(P2)
  - Disjunction (or)
      - sel(P1 or P2) = sel(P1) + sel(P2) - sel(P1) * sel(P2)

Cardinality Estimation: Join

$R \bowtie_{A=B} S$: Selectivity = 1 / max(V(R,A),V(S,A))

Q= SELECT * FROM R, S, T
WHERE R.B=S.B and S.C=T.C and R.A=40

T(R) = 30k, T(S) = 200k, T(T) = 10k
Selectivity of R.B = S.B is 1/3
Selectivity of S.C = T.C is 1/10
Selectivity of R.A = 40 is 1/200

R(A,B)
S(B,C)
T(C,D

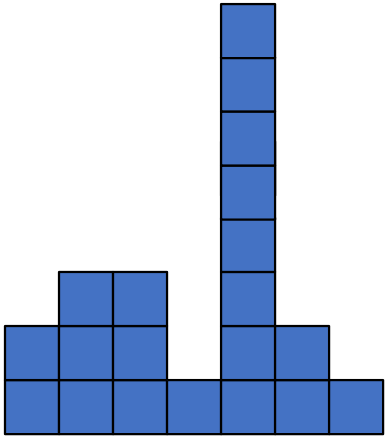
T(Q)=30k*200k*10k*1/3*1/10*1/200 =10^{10}

Histograms

- For better estimation, use a histogram

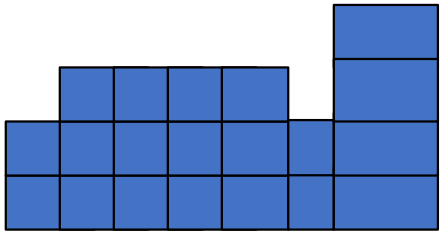
equiwidth

No. of Values	2	3	3	1	8	2	1
Value	0-.99	1-1.99	2-2.99	3-3.99	4-4.99	5-5.99	6-6.99



equidepth

No. of Values	2	3	3	3	3	2	4
Value	0-.99	1-1.99	2-2.99	3-4.05	4.06-4.67	4.68-4.99	5-6.99



Note: 10-bucket equidepth histogram divides the data into *deciles*
- akin to quantiles, median, etc.

Common trick: "end-biased" histogram
- very frequent values in their own buckets

Sketch

- How to count the number of values in a column?
  - E.g., Age = 20?
- Sketch: a cost-model can replace histograms with sketches to improve its selectivity estimate accuracy.
- Probabilistic data structures that generate approximate statistics about a data set.
- Most common examples:
  - Count-Min Sketch (1988): Approximate frequency count of elements in a set.
  - HyperLogLog (2007): Approximate the number of distinct elements in a set.

Sketch

- Given a column with a set of values, build a hash function H, and a sketch M with w entries.
  - M[i] is initialized as 0 for $0 \le i \le w-1$
  - For each value **v** in the set,
    - M[H[v]%w] ++;
- Given a value x, use M[H[x]%w] to estimate its account.

Overestimate! Because of collisions!

Count-min Sketch

- How about use *d* hash functions to reduce collisions?
- A matrix M with *d* rows and *w* columns, initialized with 0 for each cell value; *d* hash functions
- For each value v, each hash function h_i:
  - $M[i][h_i(v)]=M[i][h_i(v)]+1$; $h_i(v)$ in [0, w)
- Given x, the frequency f(x) can be estimated as
  - $f(x)=\min_{i \text{ in } [0,d-1]} M[i][h_i(v)]$

	0	1	2	3	4	…	w-1
$h_0(v)$	1	0	0	2	0	0	1
$h_1(v)$	1	0	0	1	0	0	0
…	0	0	0	0	2	0	0
$H_{d-1}(v)$	3	0	0	1	0	0	1

Count-min Sketch

- *d=4* hash functions, w=7 columns
- Given values {2, 3, 2, 4, 3, 2,5}
  - $h_0(v)=v \% w$; $h_1(v)=v^2 \% w$; $h_2(v)=(2v+1) \% w$; $h_2(v)=(3v^2+1) \% w$;

- Add 2,

	0	1	2	3	4	5	6
$h_0(v)$	0	0	1	0	0	0	0
$h_1(v)$	0	0	0	0	1	0	0
$h_2(v)$	0	0	0	0	0	1	0
$h_3(v)$	0	0	0	0	0	0	1

Count-min Sketch

- *d=4* hash functions, w=7 columns
- Given values {2, 3, 2, 4, 3, 2,5}
  - $h_0(v)=v \% w$; $h_1(v)=v^2 \% w$; $h_2(v)=(2v+1) \% w$; $h_2(v)=(3v^2+1) \% w$;

- Add 2, 3

	0	1	2	3	4	5	6
$h_0(v)$	0	0	1	1	0	0	0
$h_1(v)$	0	0	1	0	1	0	0
$h_2(v)$	1	0	0	0	0	1	0
$h_3(v)$	1	0	0	0	0	0	1

Count-min Sketch

- *d=4* hash functions, w=7 columns
- Given values {2, 3, 2, 4, 3, 2,5}
  - $h_0(v)=v \% w$; $h_1(v)=v^2 \% w$; $h_2(v)=(2v+1) \% w$; $h_2(v)=(3v^2+1) \% w$;

- Add 2, 3, 2

	0	1	2	3	4	5	6
$h_0(v)$	0	0	2	1	0	0	0
$h_1(v)$	0	0	1	0	2	0	0
$h_2(v)$	1	0	0	0	0	2	0
$h_3(v)$	1	0	0	0	0	0	2

Count-min Sketch

- *d=4* hash functions, w=7 columns
- Given values {2, 3, 2, 4, 3, 2,5}
  - $h_0(v)=v \% w$; $h_1(v)=v^2 \% w$; $h_2(v)=(2v+1) \% w$; $h_2(v)=(3v^2+1) \% w$;

- Add 2, 3, 2, 4

	0	1	2	3	4	5	6
$h_0(v)$	0	0	2	1	1	0	0
$h_1(v)$	0	0	1+1	0	2	0	0
$h_2(v)$	1	0	1	0	0	2	0
$h_3(v)$	1+1	0	0	0	0	0	2

Count-min Sketch

- *d=4* hash functions, w=7 columns
- Given values {2, 3, 2, 4, 5}
  - $h_0(v)=v \% w$; $h_1(v)=v^2 \% w$; $h_2(v)=(2v+1) \% w$; $h_2(v)=(3v^2+1) \% w$;
- Add 2, 3, 2, 4, 5

	0	1	2	3	4	5	6
$h_0(v)$	0	0	2	1	1	1	0
$h_1(v)$	0	0	1+1	0	2+1	0	0
$h_2(v)$	1	0	1	1	0	2	0
$h_3(v)$	1+1	0	0	0	0	0	2+1

count(2)=2; count(3)=1; count(4)=1; count(5)=1

Loglog

Crucial insight: suppose we have a perfect hash function h taking an integer from [1,r] and reporting an integer [0, n)

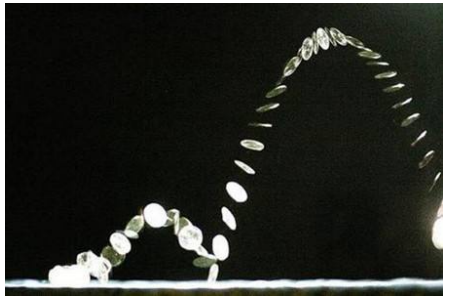
h(x) in [0, n) for x in [1,r]

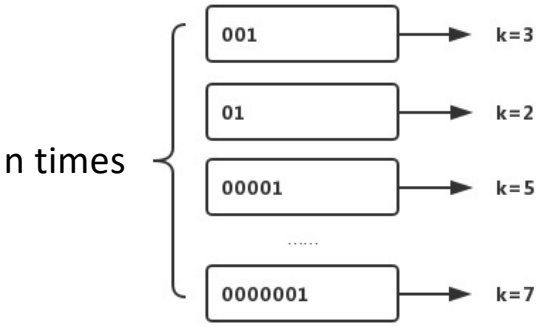
The probability that the hash contains:

0 leading zeroes: 1/2
1 leading zero: 1/4
2 leading zeroes: 1/8
3 leading zeroes: 1/16
…
w<log n leading zeroes $1/2^w$

LogLog

- Estimate the number of distinct values in a column
- Linear Counting: inefficient
- Hash: large space
- Log Counting: $n=2^{\max(\text{leading 0s})}$
- dv=0: initial distinct number
- For each element v in the column
  - Hash(v) to 0/1 values
  - C0=count leading 0s
  - $dv=\max(dv, 2^{C0})$





Loglog

- Crucial insight: maintaining the largest number of leading zeroes across all hashes allows us to get a (very) rough estimate of the number of distinct elements.
- take multiple independent hashes for each element, average them out?

Estimate: 2^3 = **8**
Actual: 10

7 → 11101100
6 → 11010101
15 → 10110100
8 → 11100110
15 → 10110100
2 → 00100010
2 → 00100010
9 → 01100100
11 → 00011000
8 → 11100110
14 → 10110111
16 → 01001101
12 → 00110110
6 → 11010101
2 → 00100010

Loglog

$\sum 2^w$

Estimate: $2^4 + 2^2 + 2^0 + 2^1 = \mathbf{23}$

Actual: 10

00 stream

2 → 00100010
2 → 00100010
11 → 00000010
12 → 00110110
2 → 00100010

4

01 stream

9 → 01100100
16 → 01001101

2

10 stream

15 → 10110100
15 → 10110100
14 → 10110111

0

11 stream

7 → 11101100
6 → 11010101
8 → 11100110
8 → 11100110
6 → 11010101

1

Loglog

$$m \times 2^{(\sum w)/m}$$

Estimate: $4 \cdot 2^{(4+2+0+1)/4} \approx \mathbf{13.45}$

Actual: 10

00 stream

2 → 00100010
2 → 00100010
11 → 00000010
12 → 00110110
2 → 00100010

4

01 stream

9 → 01100100
16 → 01001101

2

10 stream

15 → 10110100
15 → 10110100
14 → 10110111

0

11 stream

7 → 11101100
6 → 11010101
8 → 11100110
8 → 11100110
6 → 11010101

1

HyperLoglog

LogLog uses the arithmetic mean

$$\boldsymbol{\alpha}_m \times m \times 2^{(\sum w)/m}$$

HyperLogLog uses an alternative, the harmonic mean

$$\boldsymbol{\alpha}_m \times m \times m/(\sum 2^{-w})$$

where $\boldsymbol{\alpha}_m$ is a constant.

$$\alpha_m = \begin{cases} 0.673, & m = 16 \\ 0.697, & m = 32 \\ 0.709, & m = 64 \\ \dfrac{0.7213}{1 + \dfrac{1.079}{m}}, & m \geq 128 \end{cases}$$

Loglog

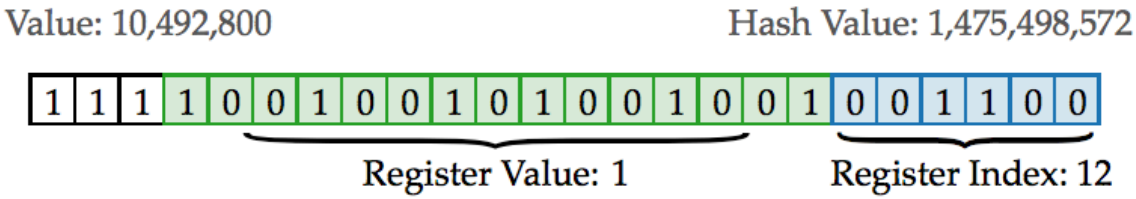
Thus the LogLog algorithm can be succinctly described. We need a hash function h from [1,r] to [0, n)

1. Initialize an array **w** with size $m=2^b$. Let $w_i = 0$ for all i
2. For each element x, compute its hash **h(x)**
   - Split **h(x)** to its first **b** bits and remaining **m-b** bits.
   - Let **c** be the number represented by the first **b** bits and w_0 be the number of leading zeroes in the **m-b** bits.
   - Set $\mathbf{w_c = max\ (w_c, w_0)}$
3. Output

$$\boldsymbol{\alpha}_m \times m \times 2^{(\sum w_c)/m}$$

HyperLogLog

- Estimate the number of distinct values in a column
- m buckets, α_m : a constant; M_j: # of leading 0s
- LogLog: $\alpha_m \cdot m \cdot 2^{\sum_{j=1}^{m} M_j/m}$
- HyperLoglog: $\alpha_m \cdot m^2 \cdot \left(\sum_{j=1}^{m} 2^{-M[j]}\right)^{-1}$

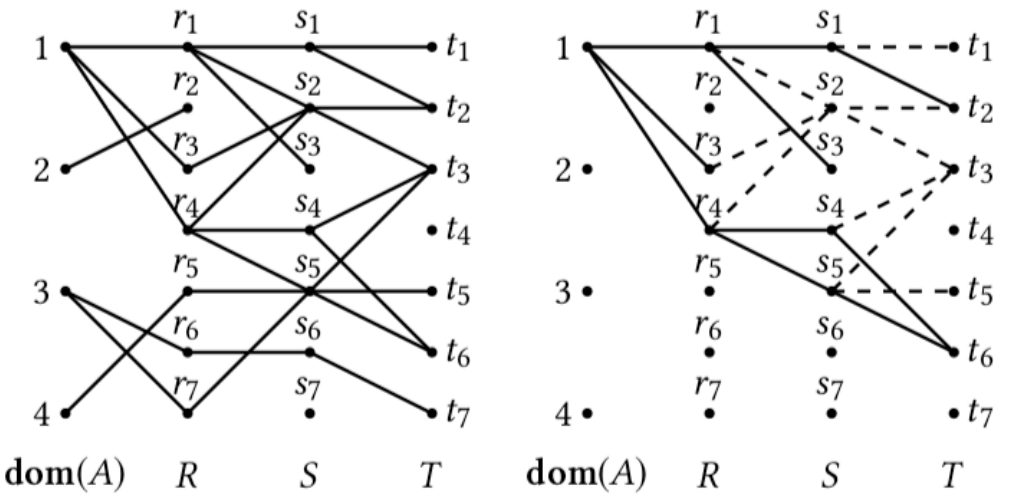


Register Values: $m = 64$

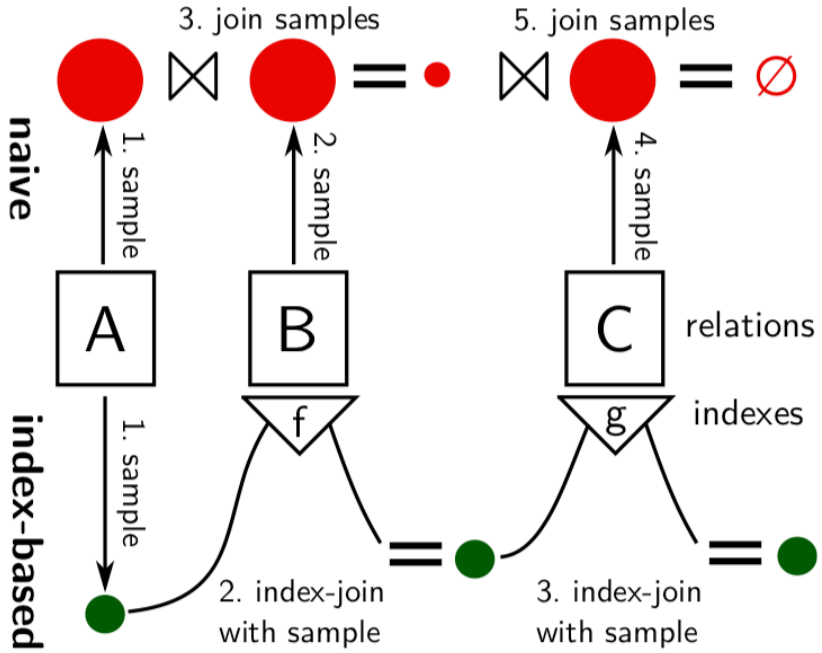
0	0	0	0	0	0	1	0	0	0	1	0	1	0	0	0
2	0	0	0	0	0	0	2	0	0	3	0	2	0	0	0
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0

Sampling

- Modern DBMSs also collect samples from tables to estimate selectivity.
- Update samples when the underlying tables changes significantly.



(a) Adding conceptual column. (b) Distinct sampling from join.



Traditional Cost Model Summary

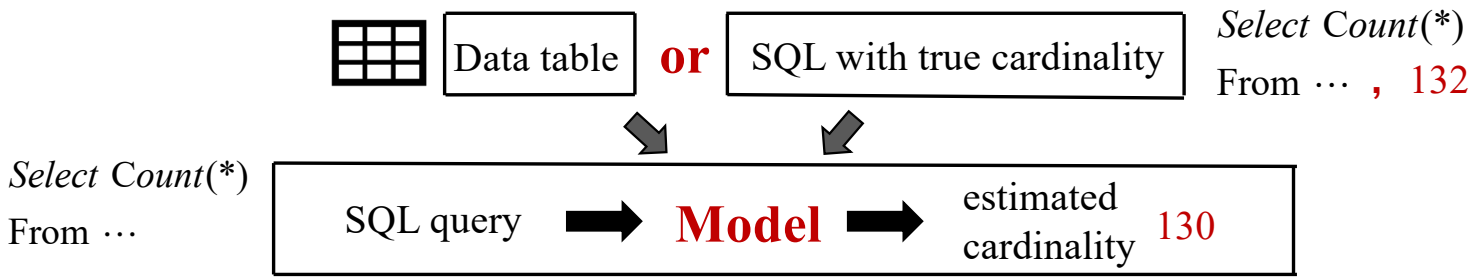
- Sampling-based
- Histogram
- Sketch

Algorithm	**Ref.**	**Year**	**Observables** [14]	**Intuition** [27]	**Core method** (Sec. 3)
FM	[15]	1985	Bit-pattern	Logarithmic hashing	Count trailing 1s
PCSA	[15]	1985	Bit-pattern	Logarithmic hashing	Count trailing 1s
AMS	[3]	1996	Bit-pattern	Logarithmic hashing	Count leading 0s
BJKST	[4]	2002	Order statistics	Bucket-based	Count leading 0s
LogLog	[11]	2003	Bit-pattern	Logarithmic hashing	Count leading 0s
SuperLogLog	[11]	2003	Bit-pattern	Logarithmic hashing	Count leading 0s
HyperLogLog	[14]	2008	Bit-pattern (order statistics)	Logarithmic hashing	Count leading 0s
HyperLogLog++	[24]	2013	Bit-pattern	Logarithmic hashing	Count leading 0s
MinCount	[21]	2005	Order statistics	Interval-based	k-th minimum value
AKMV	[7]	2007	Order statistics	Interval-based	k-th minimum value
LC	[32]	1990	No observable	Bucket-based	Linear synopses
BF	[28]	2010	No observable	Bucket-based	Linear synopses

Cardinality Estimation

Problem Formulation:

- **Cardinality:** The result size of a query.
- **Input:** A SQL Query.
- **Output:** An Estimated Cardinality.



Cardinality Estimation Example

Traditional Cardinality Estimation

- **Sampling**
  - **Core idea:** Estimating selectivity of target query by sampling.
  - **Limitation:** Inference is slow and inaccurate when the amount of data is large.
- **Histogram**
  - **Core idea:** Store the value distribution of each attribute, and calculate the selectivity according to the independence assumption.
  - **Limitation:** Strong independence assumption makes it inaccurate when the data distribution is complex.
- **Sketch:** Estimate the number of distinct elements of a set.

Regression Problems

Database estimation problems can be modeled as regression problems, which fit the high-dimension input variables into target features (e.g., cost, utility) and estimate the value of another variable.

- **Cardinality/Cost Estimation** aims to estimate the cardinality of a query and a regression model (e.g., deep learning model) can be used.

- **Index/View Benefit Estimation** aims to estimate the benefit of creating an index (or a view), and a regression model can be used to estimate the benefit.

- **Query Latency Estimation** aims to estimate the execution time of a query and a regression model can be used to estimate the performance based on query and concurrency features.

Learned Cardinality Estimation

- **Motivation**
  - Due to the attribute value independence assumption as well as the assumption of uniform distribution, traditional cardinality estimation methods tend to fail when the data distribution is complex.
- **Challenge**
  - Correlation between data columns and columns.
  - Multi-table join increases data volume and query types.
- **Optimization Goal**
  - Accuracy, Inference Latency, Model Size, Training Cost

Automatic Cardinality/Cost Estimation

Motivation:

One of the most challenging problems in databases

- Achilles Heel of modern query optimizers

Traditional methods for cardinality estimation

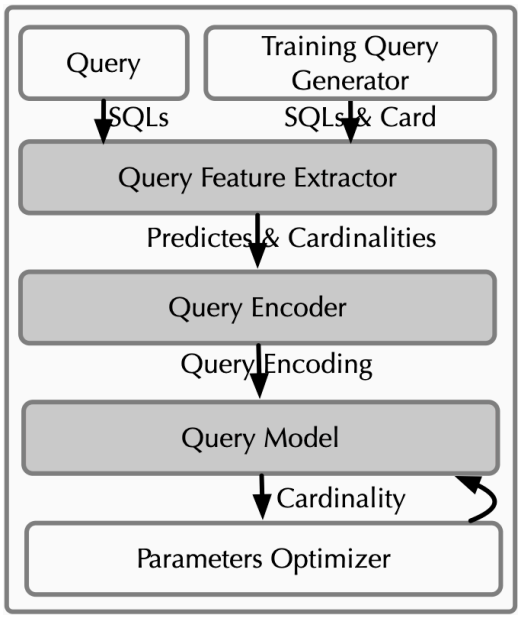
- Sampling (on base tables or joins)
- Kernel-based Methods (Gaussian Model on Samples)
- Histogram (on single column or multiple columns)

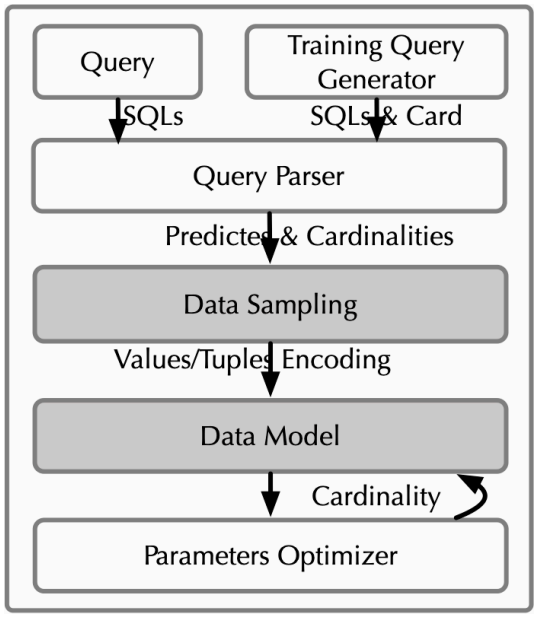
Traditional cost models

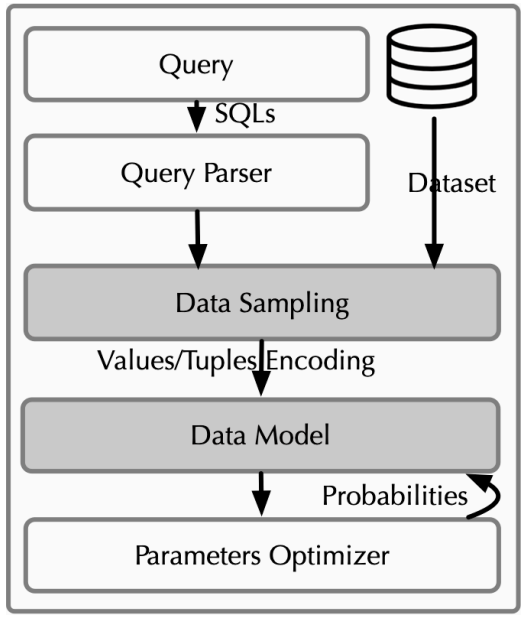
- Data sketching/data histogram based methods
- Sampling based methods

Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. How good are query optimizers, really? In VLDB, 2015.

Categories of Learned Cardinality Estimation







(1) Supervised Query Methods

- Neural Network
- XGBoost
- Multi-set Convolutional network

(2) Supervised Data Methods

- Kernel-density model
- Uniform mixture model
- Pre-training summarization model

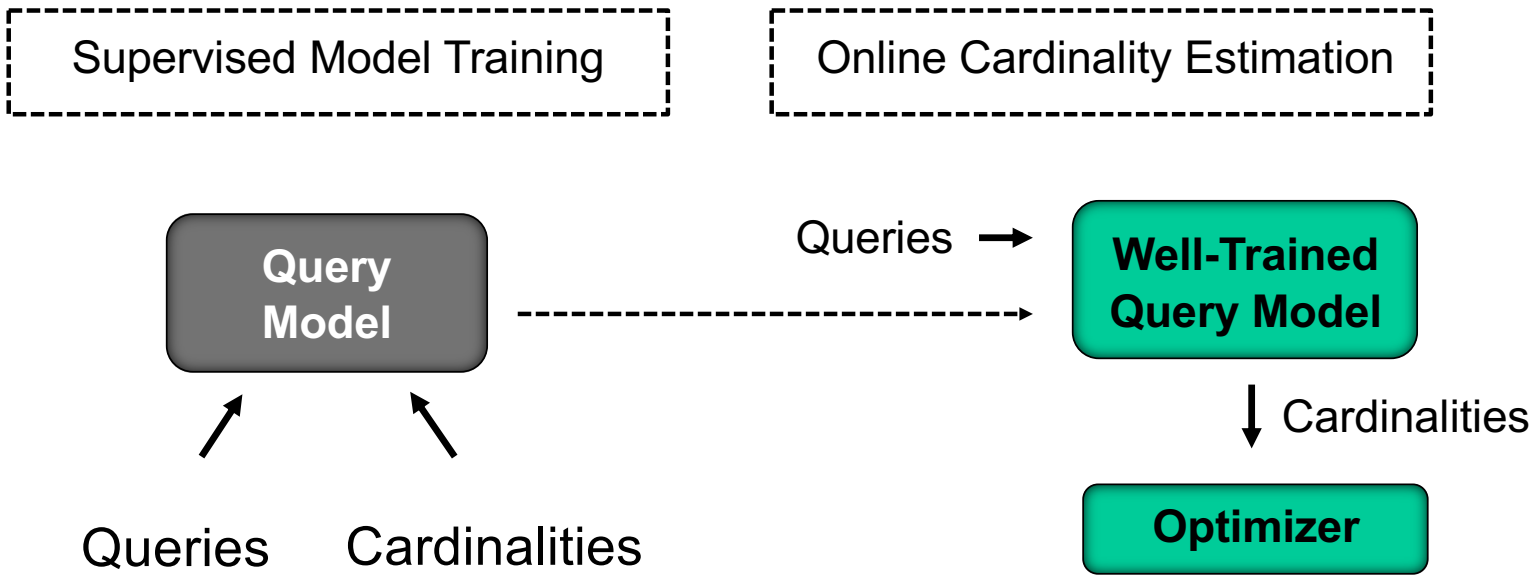
(3) Unsupervised Data Methods

- Sum product network
- Autoregressive (AR) model
- Normalizing Flow (NF) model

1 Supervised Query Methods for Cardinality Estimation

☐ Problem Definition

A regression problem: learn the mapping function between query Q and its actual cardinality



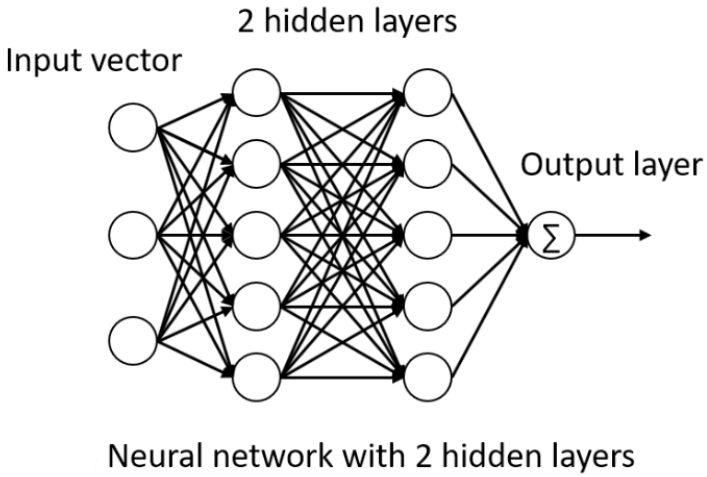
1.1 Query-Driven: Neural Network on Single Tables

- **Motivation:** Traditional estimation methods assume column independence.

- **Solution:**

$$(c_1 \leq lb_1 < c_2) \wedge (c_3 < ub_1 \leq c_4) \wedge (c_5 \leq ub_2 \leq c_6)$$

  - **Training**
    - Represent a query $(lb_1 < A_1 < ub_1, \ldots, lb\_d < A_d < ub\_d)$ on a table T with d attributes $A_1, \ldots A\_d$ as $< lb_1, ub_1, \ldots, lb_d, ub_d >$.
    - A neural network with two hidden layers is used to fit the mapping between the representation of the query and its cardinality.
  - **Inference**
    - Answer a query by the trained network.

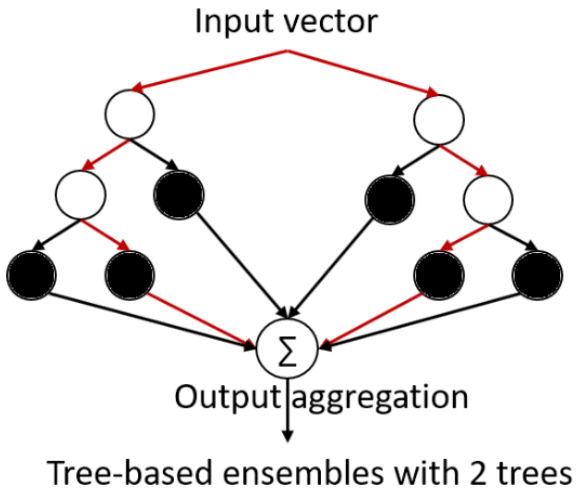


Neural network with 2 hidden layers

A. Dutt, C. Wang, A. Nazi, S. Kandula, V. R. Narasayya, and S. Chaudhuri. Selectivity estimation for range predicates using lightweight models. PVLDB, 2019.

1.2 Query-Driven: XGBoost for Cardinality Estimation

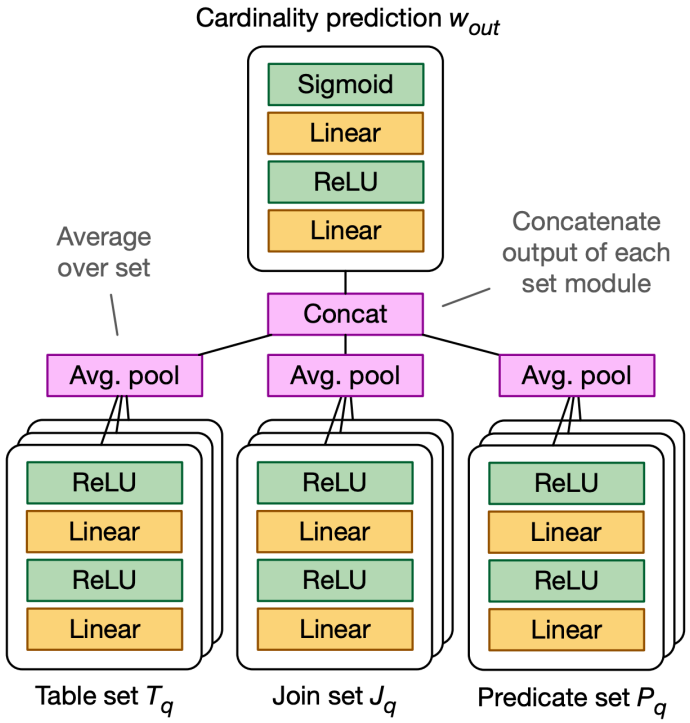
- **Solution:**
  - The query is represented in the same way as the neural network based approach.
  - Use XGBoost, a decision tree-based ensemble model to fit a mapping between a query's representation and its cardinality.
- **Comparison with Neural Network:**
  - Neural Network-based method are better when training data is sufficient.
  - XGBoost is better when the training data is insufficient.



A. Dutt, C. Wang, A. Nazi, S. Kandula, V. R. Narasayya, and S. Chaudhuri. Selectivity estimation for range predicates using lightweight models. PVLDB, 2019.

1.3 Query-Driven: Deep Learning for Multi-tables

- **Motivation:** It's difficult for traditional methods to capture join-crossing correlations.
- **Solution:**
  - For table set and join set in the input, encode each table, join with one-hot encoding.
  - For predicates of the form (col, op, val), encode columns col and operators op with one-hot encoding, and represent val as a normalized value in [0, 1].
  - Use some samples to address 0-tuple problem



A. Kipf, T. Kipf, B. Radke, V. Leis, P. A. Boncz, and A. Kemper. Learned cardinalities: Estimating correlated joins with deep learning. In CIDR, 2019.

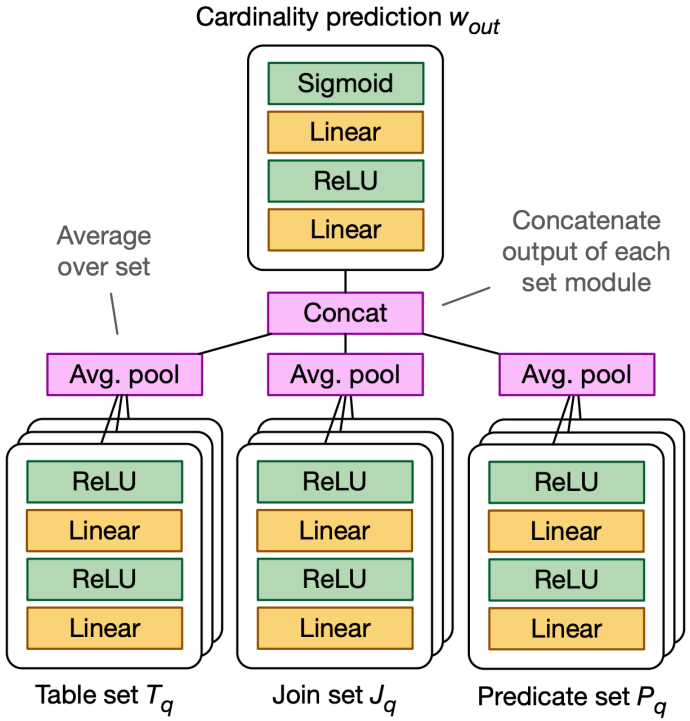
1.3 Query-Driven: Deep Learning for Multi-tables

Training:

- The three parts of the input are spliced together after going through the linear layer, activation layer, and the dimensionality reduction layer.
- Then go through the linear and activation layer again to get the estimated cardinality.
- Tuning model parameters by backpropagating gradients.

Inference

- Answer a query by the trained network.

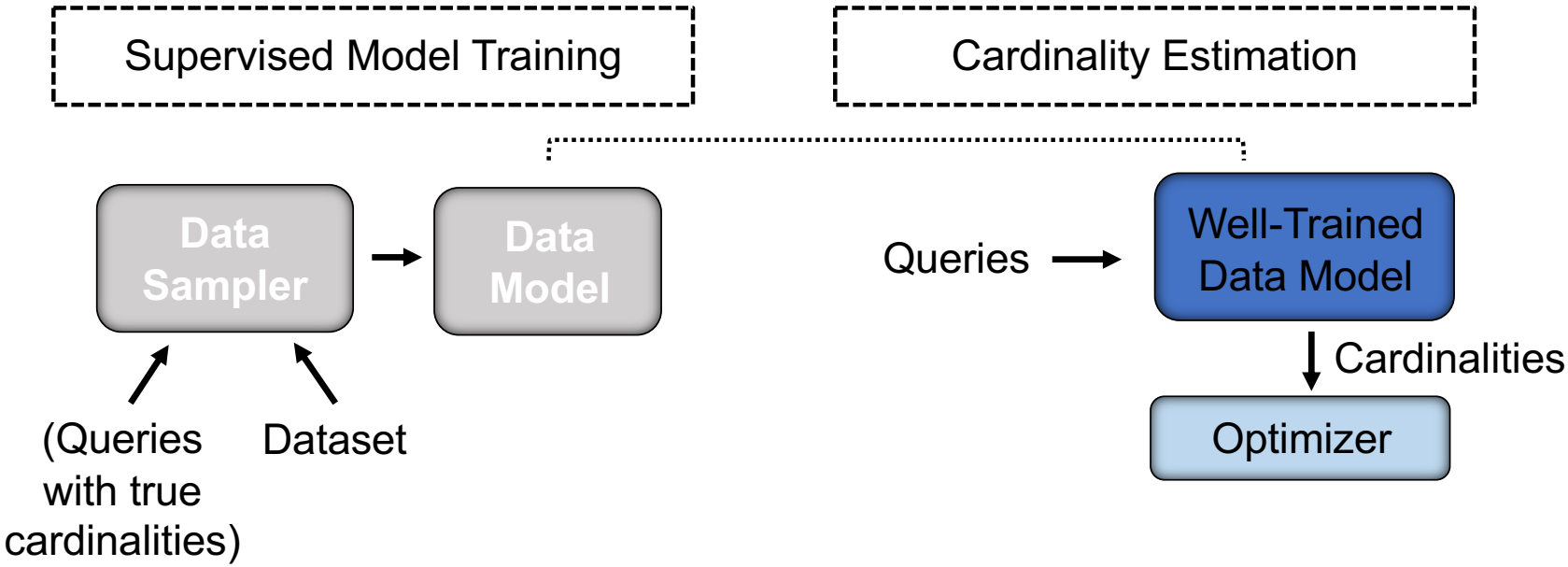


A. Kipf, T. Kipf, B. Radke, V. Leis, P. A. Boncz, and A. Kemper. Learned cardinalities: Estimating correlated joins with deep learning. In CIDR, 2019.

2 Supervised Data Methods for Cardinality Estimation

☐ Problem Definition

A density estimation problem: learn a joint data distribution of each data point. (except for the pre-training summarization model)



2.1 Supervised Data-Driven: Kernel-Density Model on Single Table

- **Motivation:** Multi-dimensional histograms are complex to construct and hard to maintain.
- **Key-idea:** Fit the probability density distribution of a data table by kernel density model.

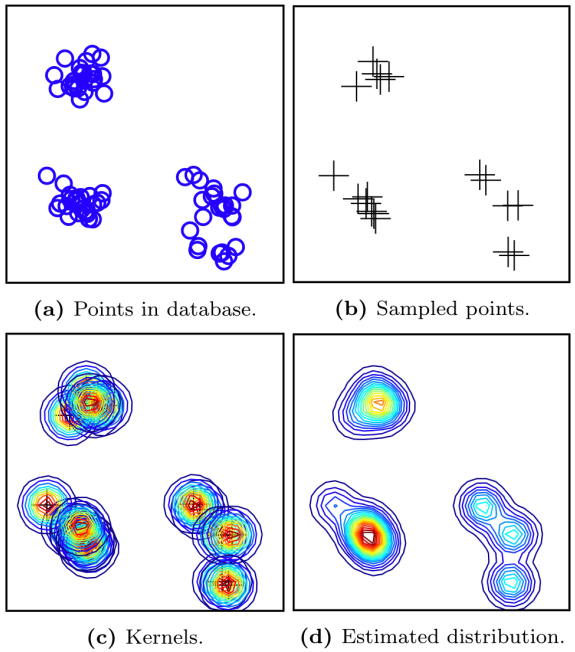
➢ Kernel-Density Model

Model: $\hat{p}_H(\vec{x}) = \frac{1}{s \cdot |H|} \sum_{i=1}^{s} K\left(H^{-1}\left[\vec{t}^{(i)} - \vec{x}\right]\right)$

- s is sample size; K is Gaussian function;
- $\vec{t}$ is sampled point; H is a parameter that needs to be learned.,

➢ Inference: $\hat{p}_H(\Omega) = \int_{\Omega} \hat{p}(\vec{x})\, d\vec{x}$

- Ω is the space represented by a query.



(a) Points in database. (b) Sampled points.

(c) Kernels. (d) Estimated distribution.

M. Heimel, M. Kiefer, and V. Markl. Self-tuning, gpu-accelerated kernel density models for multidimensional selectivity estimation. SIGMOD, 2015.

2.1 Supervised Data-Driven: Kernel-Density Model on Single Table

Kernel-Density Estimation

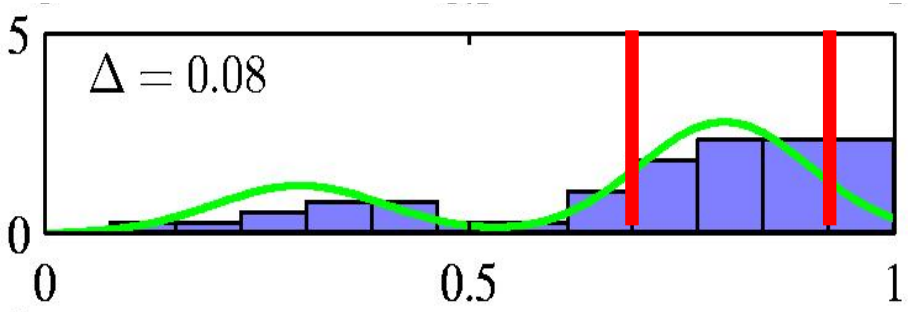
$$p_i = \frac{n_i}{N\,\Delta_i}$$

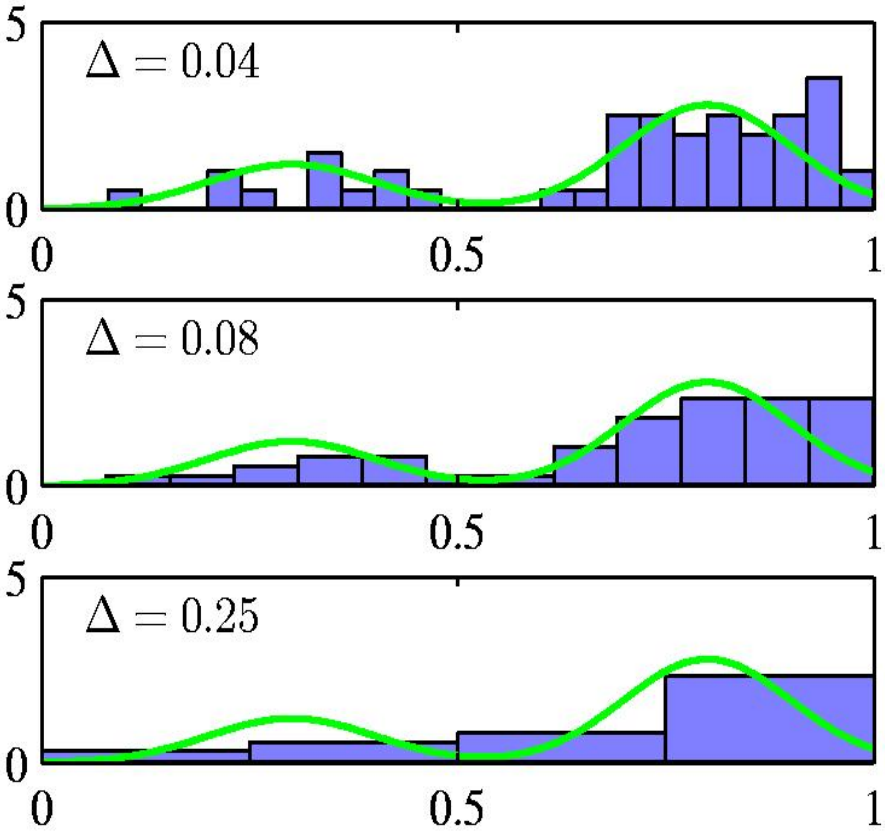
p_i: probability

n_i: number of points

N: total number of points

Δ: bandwidth





M. Heimel, M. Kiefer, and V. Markl. Self-tuning, GPU-accelerated kernel density models for multidimensional selectivity estimation. SIGMOD, 2015.

2.1 Supervised Data-Driven: Kernel-Density Model on Single Table

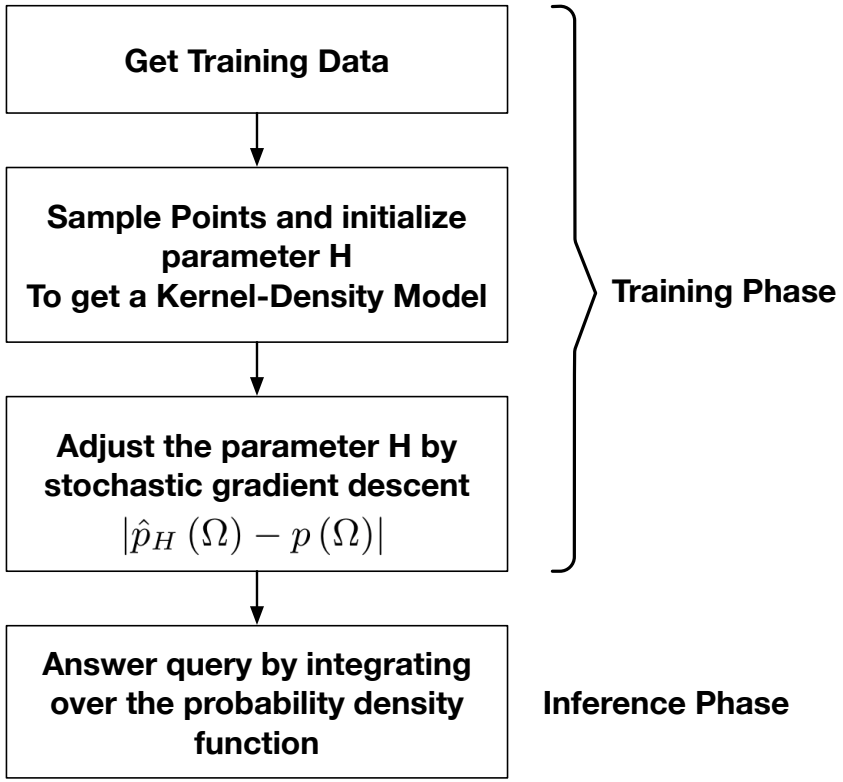
Solution

Training

- Get a lot of queries with true cardinalities.
- Sample points (rows) from the table and initialize the parameter H.
- Adjust the parameter H by stochastic gradient descent according to estimated cardinalities by Kernel-Density Model and the true cardinalities.

Inference

- Answer queries by accumulating kernel density based on the kernel-density model.



M. Heimel, M. Kiefer, and V. Markl. Self-tuning, gpu-accelerated kernel density models for multidimensional selectivity estimation. SIGMOD, 2015.

2.2 Supervised Data-Driven: Uniform Mixture Model on Single Table

- **Motivation:** Traditional methods need to be populated in advance by performing costly table scans.
- **Key-idea:** Fit the probability density distribution of a data table by uniform mixture model.
- **Uniform Mixture Model:**

Model: $f(x) = \sum_{z=1}^{m} h(z)\, g_z(x) = \sum_{z=1}^{m} w_z\, g_z(x)$

- w_z is the weight for a subpopulation z
- $g_z(x) = 1/|\mathrm{G_z}|$ is uniform distribution function
- $|G_z|$ is area of subpopulation z

Inference: $\int_{B_i} f(x)\, dx = \int_{B_i} \sum_{z=1}^{m} w_z\, g_z(x)\, dx$

- B_i is the space represented by a query.

Yongjoo Park, Shucheng Zhong, and Barzan Mozafari. Quicksel: Quick selectivity learning with mixture models. SIGMOD 2020

2.2 Supervised Data-Driven: Uniform Mixture Model on Single Table

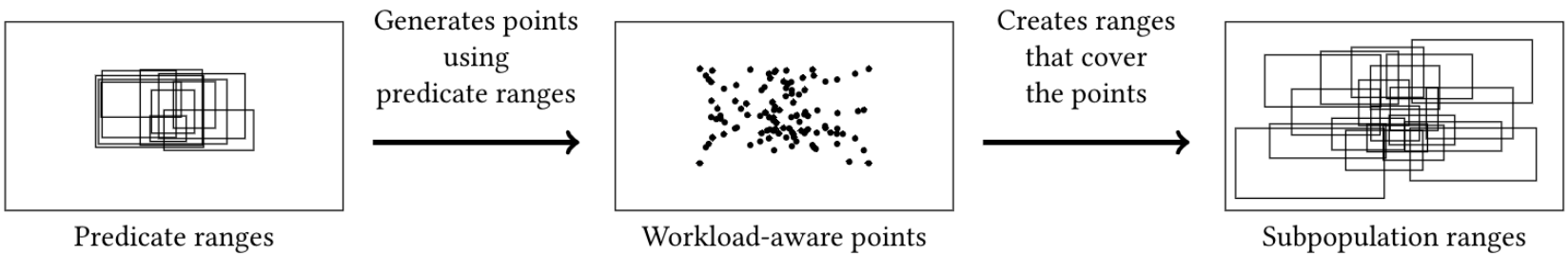
Solution:

Training

- Sample some points within queries with true cardinalities.
- Generating subgroups for the points.
- Learn the weights w_z of the uniformity mixture model.

Inference

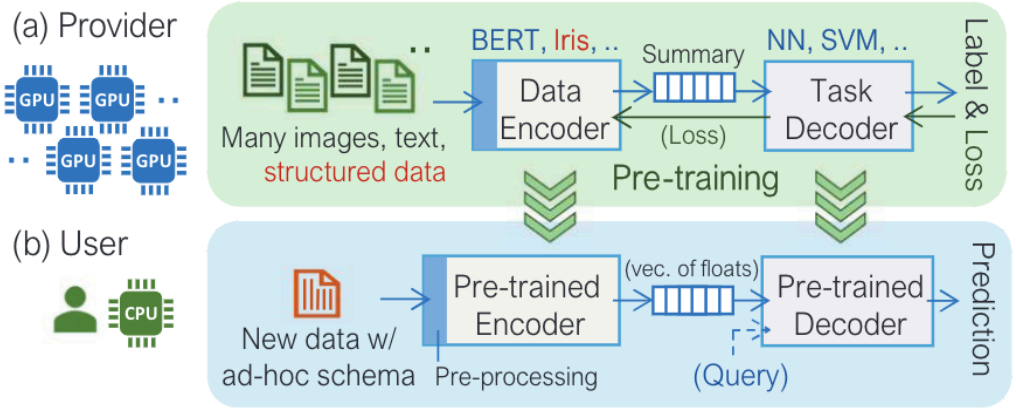
- Answer a query by Calculate the cumulative probability density (i.e., selectivity) according to the mixture density function.
- overlap area between query rectangle and data rectangle



Yongjoo Park, Shucheng Zhong, and Barzan Mozafari. Quicksel: Quick selectivity learning with mixture models. SIGMOD 2020

2.3 Supervised Data-Driven: Pre-training Summarization Model on Single Table

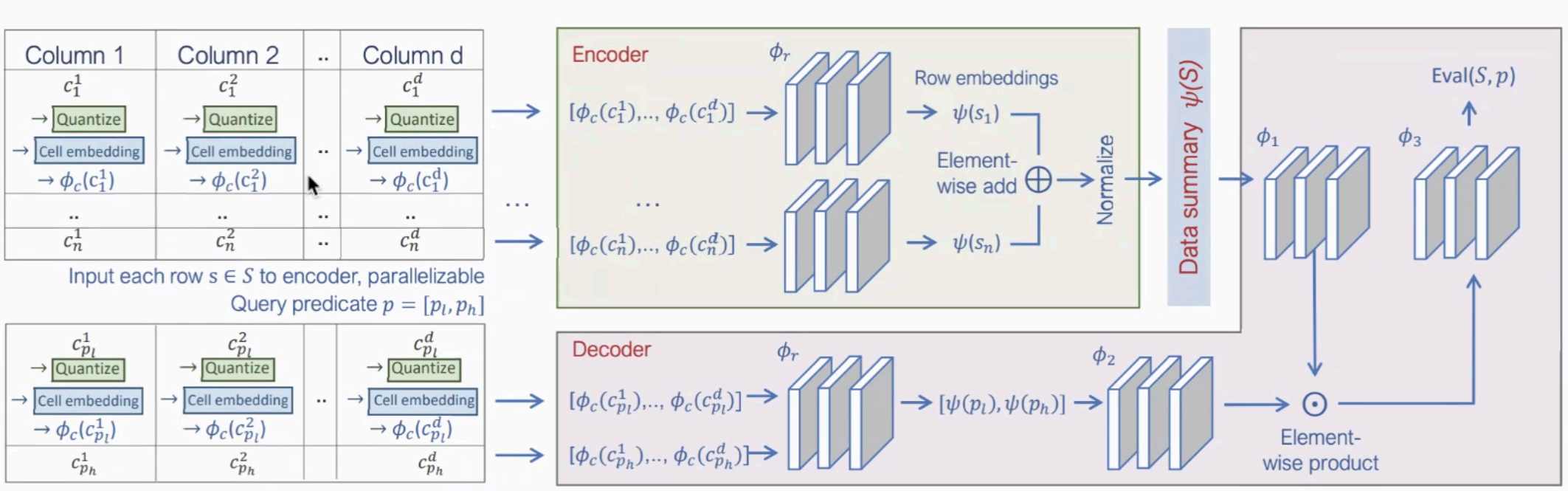
- **Motivation:** Pre-training models avoid per-dataset training.
- **Solution:**
  - Pre-train encoder and decoder with large data tables via gradient descent (Loss function is $\left|log\ \frac{true\ card}{est.card}\right|$).
  - Encode a table with the pretrained encoder.
  - Input a query and the encoded result of the table into the pre-trained decoder to get the cardinality.



Lu Y, Kandula S, König A C, et al. Pre-training summarization models of structured datasets for cardinality estimation. PVLDB, 15(3): 414-426, 2021

2.3 Supervised Data-Driven: Pre-training Summarization Model on Single Table

- Data Encoder
- Query Decoder

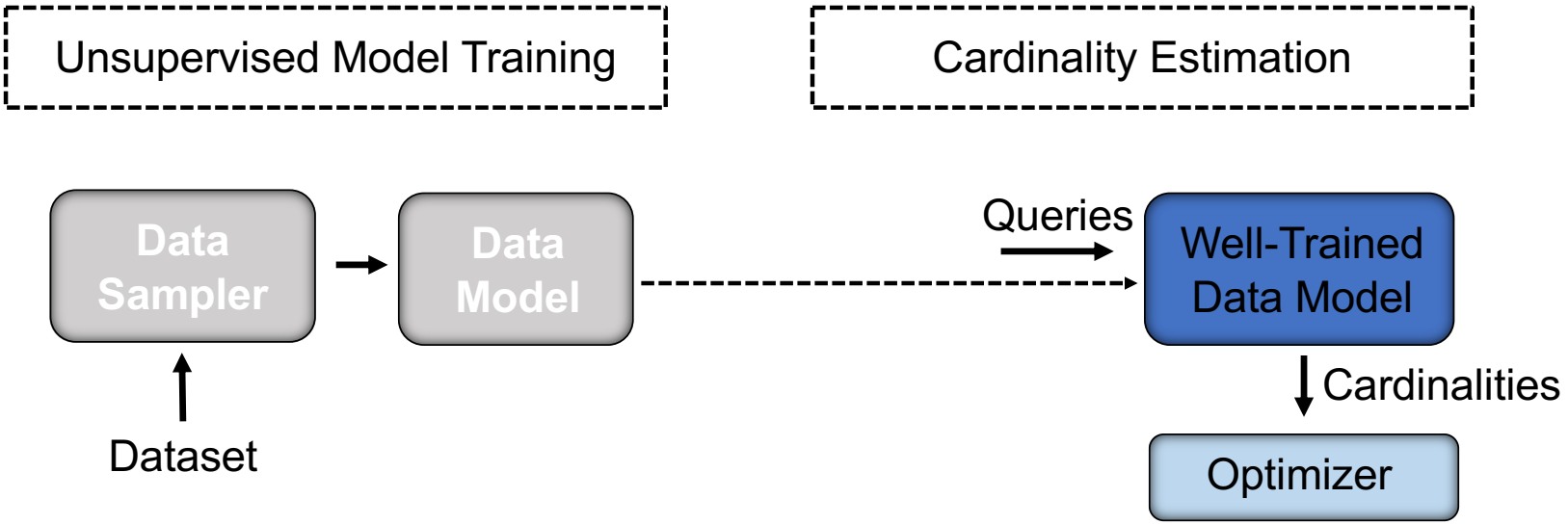


Lu Y, Kandula S, König A C, et al. Pre-training summarization models of structured datasets for cardinality estimation. PVLDB, 15(3): 414-426, 2021

3 Unsupervised Data Methods for Cardinality Estimation

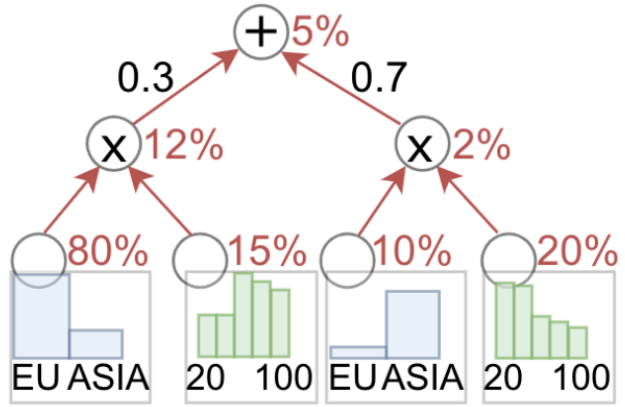
☐ Problem Definition

A regression problem: learn a probability function for each data point.



3.1 Unsupervised Data-Driven: Sum-Product Network for Multi-tables

- **Motivation:** Most of the existing estimators require SQL queries.
- **Base-idea:** Learn the joint probability distribution by Sum-Product Network .
- Relational Sum Product Network (RSPN)
  - RSPN consists of three types of node:
    - product node: split the columns of a table.
    - sum node: split the rows of a table.
    - leaf node: represent probability distributions for individual variables.



(d) Probability of European Customers younger than 30

Benjamin Hilprecht, Andreas Schmidt, et al . DeepDB: Learn from Data, not from Queries! PVLDB 13, 13(7): 992-1005, 2020

3.1 Unsupervised Data-Driven: Sum-Product Network for Multi-tables

Solution:

Training

- Generate some queries and their cardinalities.
- Build a RSPN by recursively partitioning
  - Row: K-means clustering
  - Column: randomized dependency coefficient.

Inference

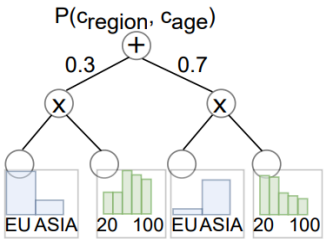
- Estimate cardinality in bottom-up.
- Supports multi-table queries via join sampling, and supports queries on sub-schemas based on fanout scaling.

c_id	c_age	c_region
1	80	EU
2	70	EU
3	60	ASIA
4	20	EU
...	...	...
998	20	ASIA
998	25	EU
999	30	ASIA
1000	70	ASIA

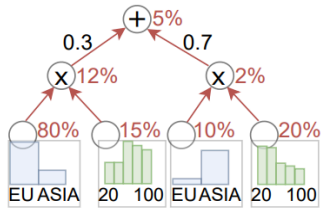
(a) Example Table

c_age	c_region
80	EU
70	EU
60	ASIA
20	EU
...	...
...	...
20	ASIA
25	EU
30	ASIA
70	ASIA

(b) Learning with Row/Column Clustering



(c) Resulting SPN



(d) Probability of European Customers younger than 30

Benjamin Hilprecht, Andreas Schmidt, et al . DeepDB: Learn from Data, not from Queries! PVLDB 13, 13(7): 992-1005, 2020

Fanout scaling for multi-tables

Fanout scaling is to support sub-schema queries on a full outer join table. (full outer join table contains duplicate tuples)

- **Solution:** for each $foreign\ key \rightarrow primary\ key$ relationship add a column denoting how many corresponding join partners a tuple has.

- Example:

```
SELECT COUNT(*)
  FROM CUSTOMER C
 WHERE c_region='EUROPE';
```

- True answer is 2, but there are 3 tuples in the outer join table.

- Fanout scaling result: $|\mathtt{C} \bowtie \mathtt{O}| \cdot \mathbb{E}(1/\mathcal{F}'_{C \leftarrow O} \cdot \mathbf{1}_{\mathtt{c\_region='EU'}} \cdot \mathcal{N}_C)$

$$= 5 \cdot \frac{1/2+1/2+1}{5} = 2$$

Customer

c_id	c_age	c_region
1	20	EUROPE
2	50	EUROPE
3	80	ASIA

Order

o_id	c_id	o_channel
1	1	ONLINE
2	1	STORE
3	3	ONLINE
4	3	STORE

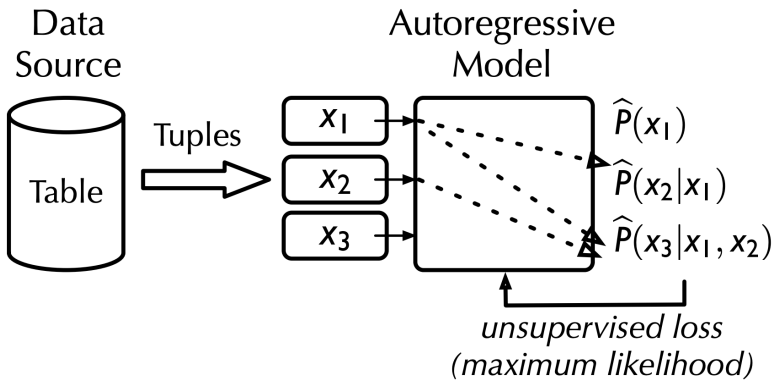
Customer ⟗ Order

$\mathcal{N}_C$	c_id	c_age	c_region	$\mathcal{F}'_{C \leftarrow O}$	$\mathcal{N}_O$	o_id	o_channel
1	1	20	EUROPE	2	1	1	ONLINE
1	1	20	EUROPE	2	1	2	STORE
1	2	50	EUROPE	1	0	NULL	NULL
1	3	80	ASIA	2	1	3	ONLINE
1	3	80	ASIA	2	1	4	STORE

3.2 Unsupervised Data-Driven: Autoregressive Model on Single Table

- **Motivation:** Existing estimators struggle to capture the rich multivariate distributions of relational tables.
- **Solution:**
  - **Training:** Use Autoregressive (AR) Model to fit the joint probability of different columns.
  - **Inference:** Estimate the cardinality of the equivalent query based on the AR model.
    - Monte Carlo sampling
  - Range queries are supported by progressive sampling (sampling by learned probability distribution).

$$\begin{aligned}\widehat{P}(\boldsymbol{x}) &= \widehat{P}(x_1, x_2, \cdots, x_n) \\ &= \widehat{P}(x_1)\widehat{P}(x_2|x_1)\cdots\widehat{P}(x_n|x_1, \ldots, x_{n-1})\end{aligned}$$

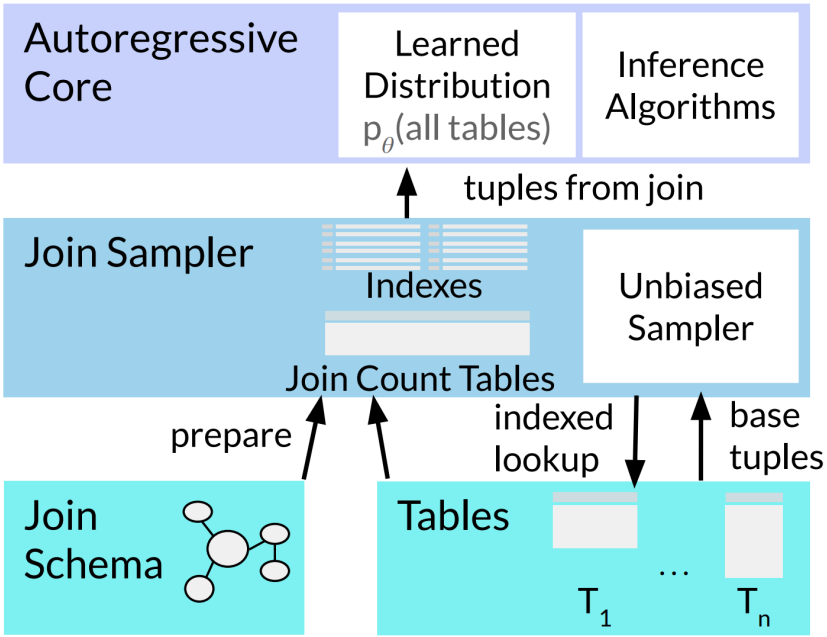


Z. Yang, E. Liang, A. Kamsetty, C. Wu, Y. Duan, X. Chen, P. Abbeel, J. M. Hellerstein, S. Krishnan, and I. Stoica. Deep Unsupervised Cardinality Estimation. PVLDB, 13(3): 279-292, 2019.

3.2 Unsupervised Data-Driven: Autoregressive Model on Single Table

- **Motivation:** Previous AR models do not support multi-table queries.

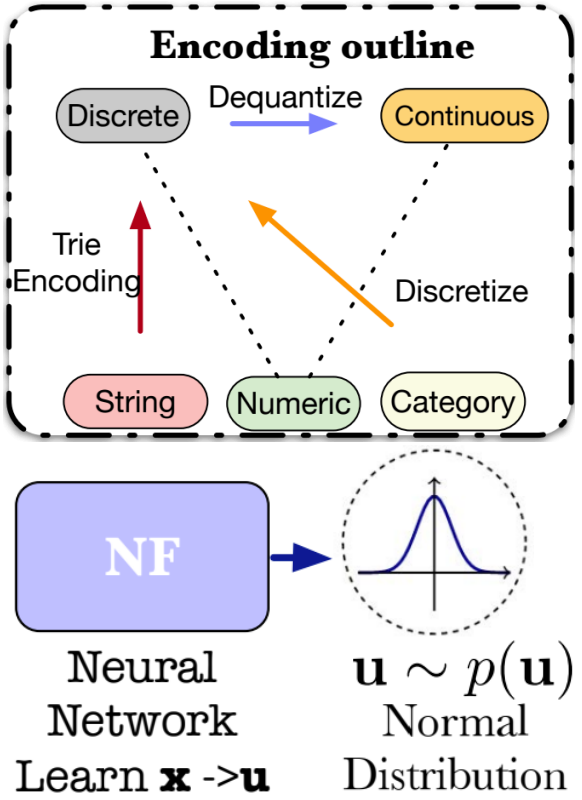
- **Solution:**
  - Learn an autoregressive model for the outer join of all tables.
  - Supports multi-table queries via join sampling, and supports queries on sub-schemas through fanout scaling.



Zongheng Yang, Amog Kamsetty, et al. NeuroCard: One Cardinality Estimator for All Tables. PVLDB, 14(1): 61-73, 2021

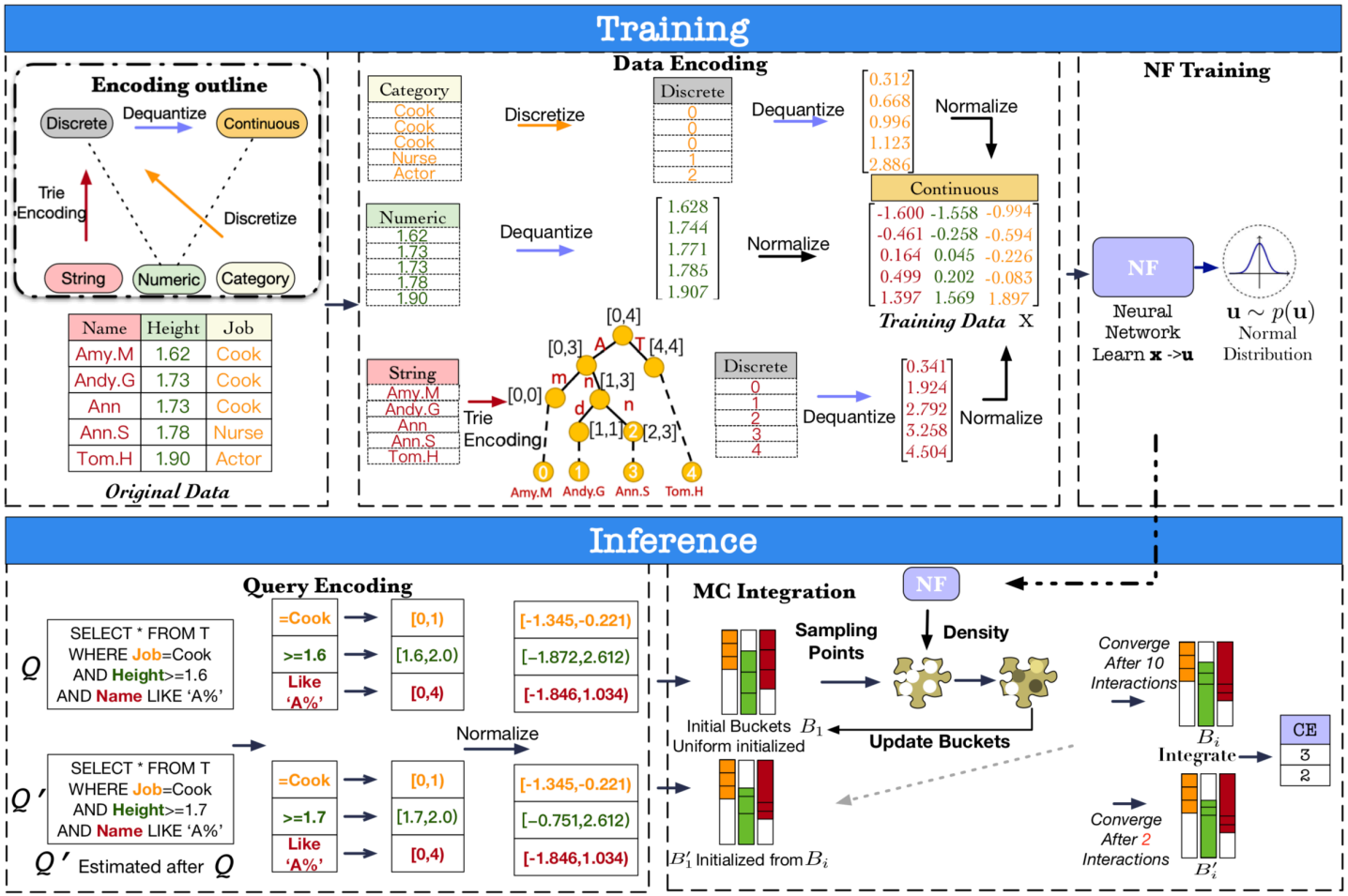
3.3 Unsupervised Data-Driven: Normalizing Flow (NF) model for Multi-tables

- **Motivation:** Previous data-driven approaches do not handle tables with large domain sizes well.
- **Solution:**
  - Dequantize and normalize discrete variables to continuous variables.
  - Use Normalizing Flow (NF) model to learn the joint probability distribution of data points.
  - Accumulating continuous normalized flow distribution function by adaptive importance sampling to answer a query.
  - Support multi-table query through fanout scaling.



Wang J, Chai C, Liu J, et al. FACE: a normalizing flow based cardinality estimator[J]. Proceedings of the VLDB Endowment, 15(1): 72-84, 2021

3.3 Unsupervised Data-Driven: Normalizing Flow (NF) model for Multi-tables



Wang J, Chai C, Liu J, et al. FACE: a normalizing flow based cardinality estimator[J]. Proceedings of the VLDB Endowment, 15(1): 72-84, 2021

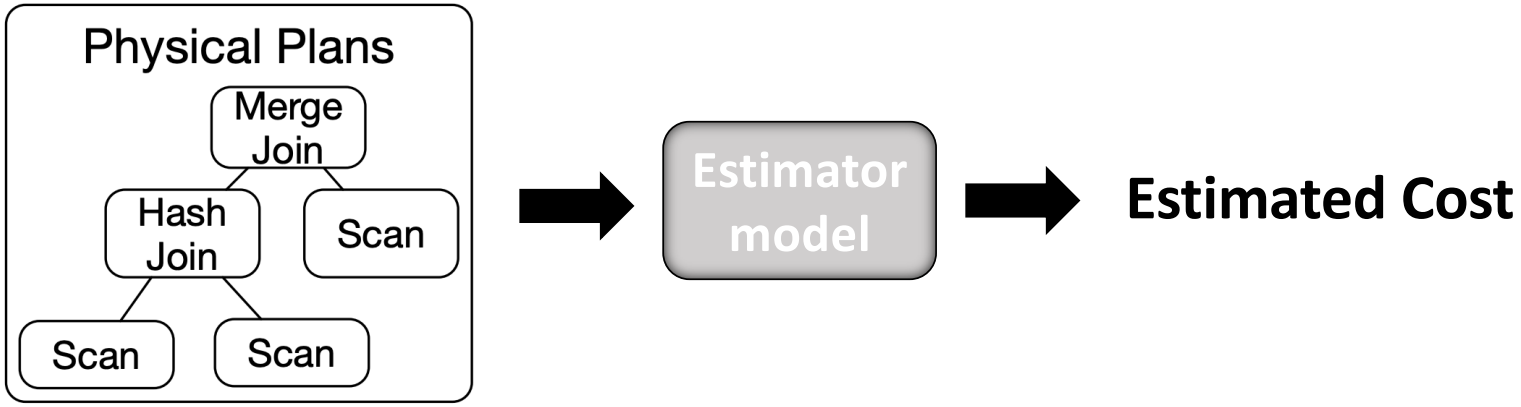
Summarization of Learned Cardinality Estimation

Method	Quality	Training overhead	Training Data	Adaptive	Model Size	Inference Latency
Lightweight Neural Network & XGBoost	✓	low	many queries	✓	small	fast
Convolutional Neural Network	✓✓	low	many queries	✓✓	small	fast
Kernel-Density Model	✓	medium	Data samples	✓	small	medium
Uniform Mixture Model	✓	medium	Data samples	✓	small	medium
Autoregressive (AR) Model	✓✓	high	Data samples	✓✓✓	high	slow
Sum-Product Network	✓✓	high	Data samples	✓✓✓	medium	medium
Normalizing Flow (NF) model	✓✓	high	Data samples	✓✓✓	medium	medium
Pre-training summarization model	✓	high	Lots of tables	✓✓	very small	fast

Cost Estimation

Problem Formulation:

- **Cost:** Execution cost of a query plan.
- **Input:** A SQL Query Plan.
- **Output:** An Estimated Cost.

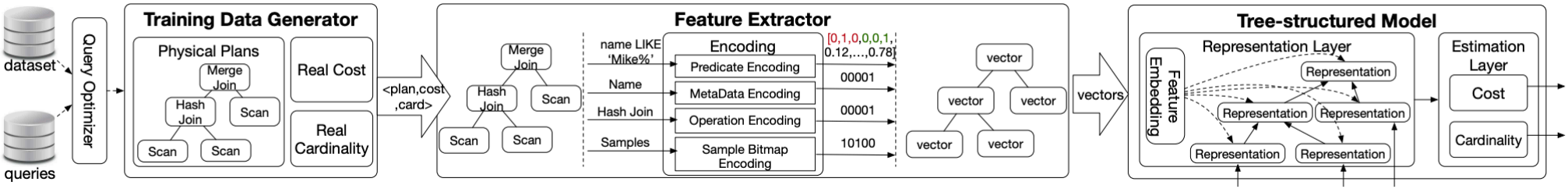


Relations of Cardinality/Cost Estimation

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

Tree-LSTM for Cost Estimation

- **Motivation:** Traditional cost estimation is inaccurate without learned plan representation.
- **Solution:**
  - Generate many query plans and true costs as training data.
  - Encode the query plan via one-hot encoding.
  - Representation layer learns an embedding of each query plan by Tree-LSTM.
  - Estimation layer outputs estimated cost based on the representation layer's output.

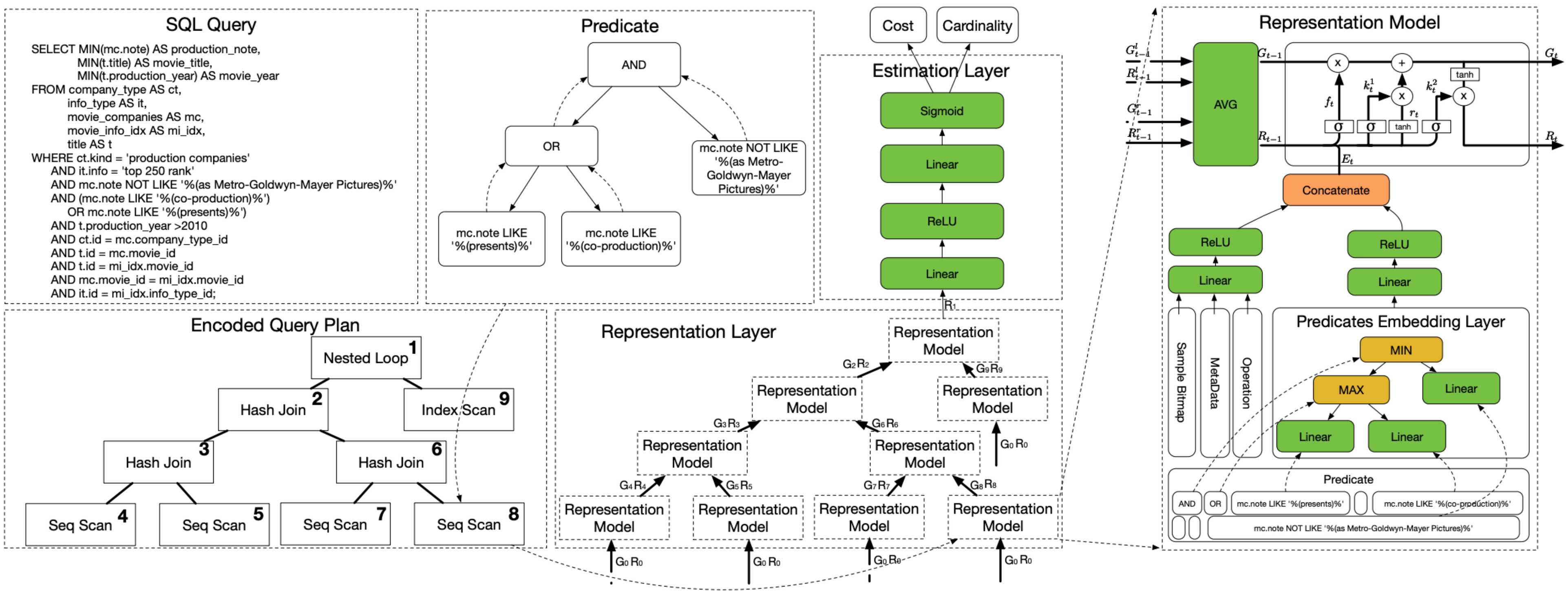


J. Sun and G. Li. An end-to-end learning-based cost estimator. PVLDB, 13(3):307–319, 2019.

Tree-LSTM for Cost Estimation

☐ Model Construction

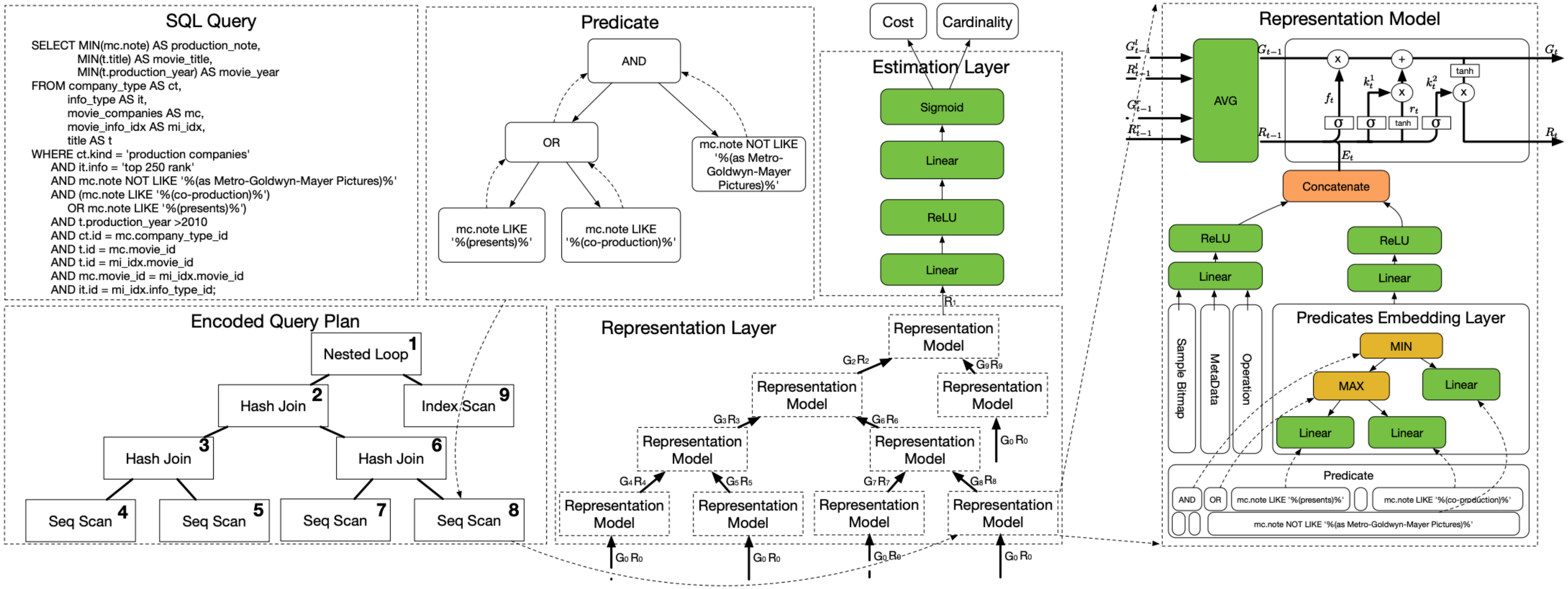
- **Traditional cost estimation uses estimated card, which is inaccurate without predicate encoding →**



J. Sun and G. Li. An end-to-end learning-based cost estimator. PVLDB, 13(3):307–319, 2019.

Tree-LSTM for Cost Estimation

- **The representation layer** learns an embedding of each subquery (global vector denotes the subquery, local vector denotes the root operator)
- **The estimation layer** outputs cardinality & cost simultaneously



J. Sun and G. Li. An end-to-end learning-based cost estimator. PVLDB, 13(3):307–319, 2019.

Take-aways of Cardinality Estimation

- Cardinality Estimation
  - Data-driven methods are more effective for single tables.
  - Query-driven methods are more efficient than Data-driven methods.
  - Data-driven methods are more robust than Query-driven methods.
  - Samples are crucial to most Data-driven methods.
- Cost Estimation
  - Accurate cost estimation requires better plan embedding.
- Open Problems
  - High Accuracy with small model size and inference latency
  - Adaptivity

Deep Learning for Query Latency Estimation

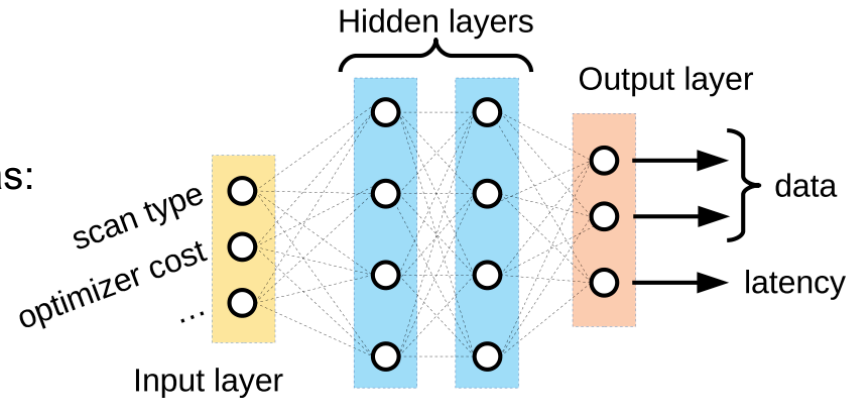
Motivation

- Statistical methods fail to estimate based on query structures, and cause great errors;
- Compared with cost estimation, latency estimation is more complex because (1) it relies on system resources and (2) Cost is one important factor of latency estimation.

Core Idea: Utilize deep learning to capture the relations between input tables, operators, and the final performance

Solution

- **Represent each operator q_i with a neural unit**
  - E.g., for a scan operator, the neural unit is designed as:



R. Marcus and O. Papaemmanouil. Plan-structured deep neural network models for query performance prediction. VLDB, 2019.

Deep Learning for Query Latency Estimation

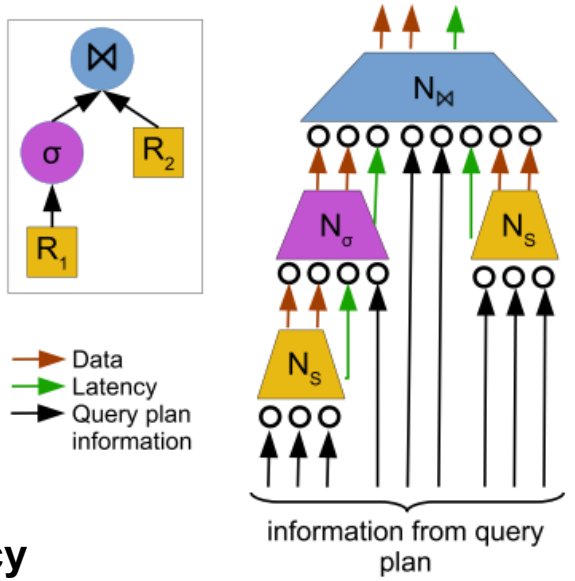
Motivation

- Statistical methods fail to estimate based on query structures, and cause great errors;
- Compared with cost estimation, latency estimation is more complex because (1) it relies on system resources and (2) Cost is one important factor of latency estimation.

Core Idea: Utilize deep learning to capture the relations between input tables, operators, and the final performance

Solution

- **Represent each operator q_i with a neural unit**
- **Concatenate neural units by following the query structures**
  - **Example Query *Q* (**2 Scans, 1 Join**)**
  - **Tree-structured Network for *Q*:**
    - The outputs of the scan units (Nσ + Ns)
    - Input of the join operator ($N_{\bowtie}$)
    - The final predicted query latency.
  - **Matches the query structure to predict the query latency**

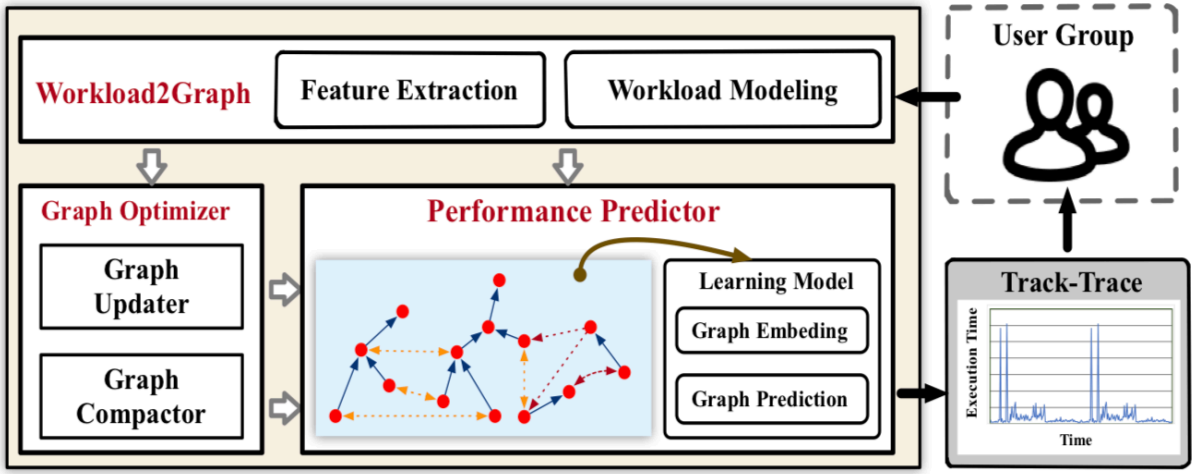


R. Marcus and O. Papaemmanouil. Plan-structured deep neural network models for query performance prediction. VLDB, 2019.

Graph Embedding for Query Latency Estimation

Latency Estimation for Concurrent Queries

- data-sharing
- data-conflict
- resource-competition
- parent-child relationship

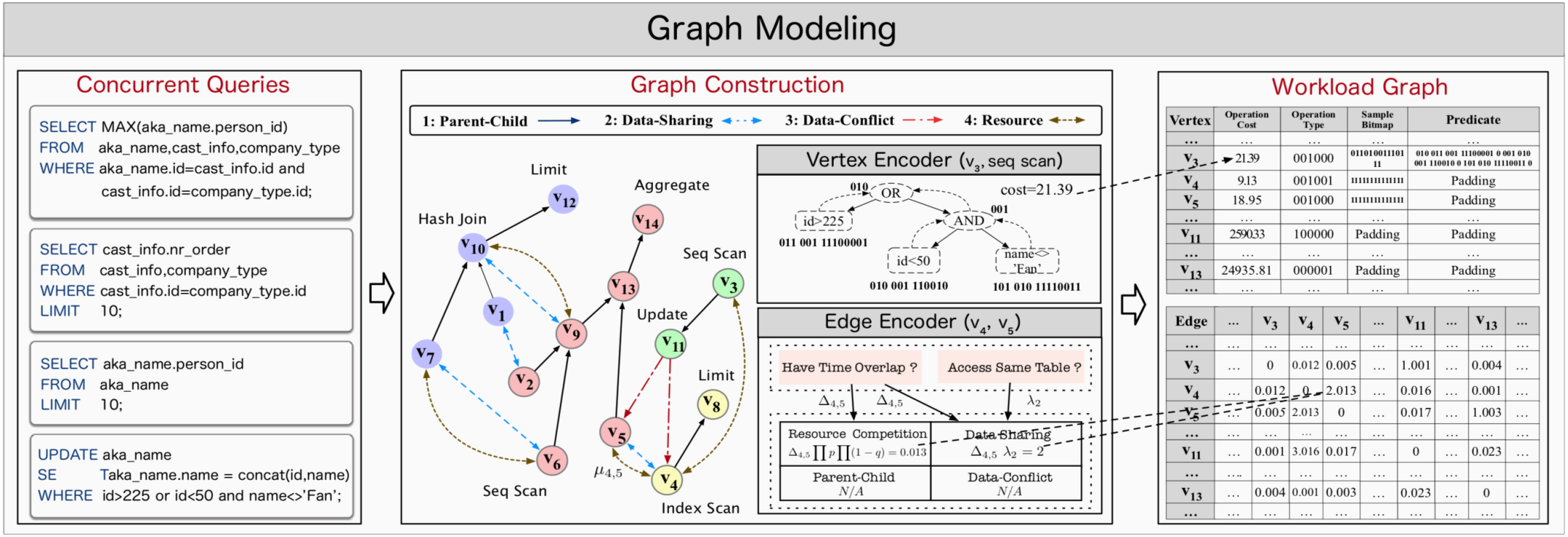


Graph-based method

- Workload2Graph: graph modeling
- Graph prediction: GNN to predict the latency
- Graph update: on-the-fly update the model

X. Zhou, J. Sun, G. Li, et al. Query Performance Prediction for Concurrent Queries using Graph Embedding. VLDB, 2020.

Graph Embedding for Query Latency Estimation

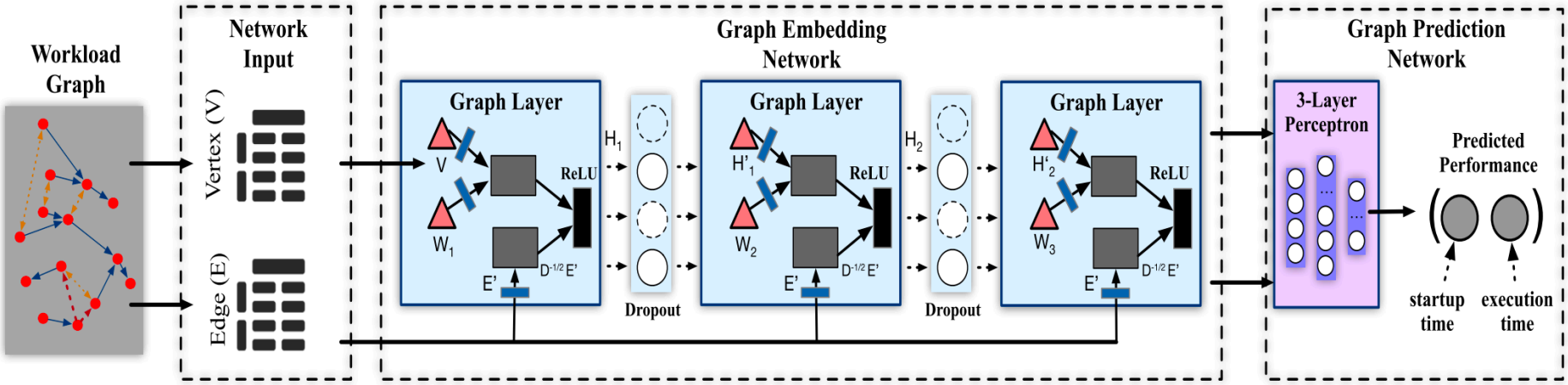


X. Zhou, J. Sun, G. Li, et al. Query Performance Prediction for Concurrent Queries using Graph Embedding. VLDB, 2020.

Graph Embedding for Query Latency Estimation

☐ Model Construction

- **Performance prediction of concurrent queries**
  - ➢ Represent concurrent queries with a graph model
  - ➢ Embed the graph with graph convolution network and predict the latency of all the operators with a simple dense network



X. Zhou, J. Sun, G. Li, et al. Query Performance Prediction for Concurrent Queries using Graph Embedding. VLDB, 2020.

Deep Learning for Index Benefit Estimation

Challenge

The index/view benefit is hard to evaluate

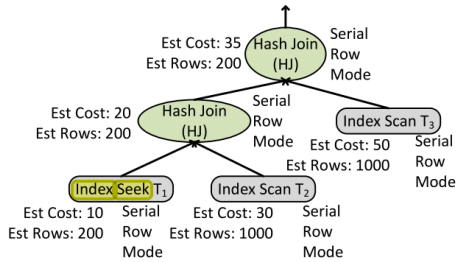
- Multiple evaluation metrics (e.g., index benefit, space cost)
- Cost estimation by the optimizer is inaccurate

Interactions between existing data structures

- Multiple column access, Data refresh
- Conflicts between MVs

Deep Learning for Index Benefit Estimation

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



(a) Example query plan.

EstNodeCost

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

LeafWeightEstRows WeightedSum

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

(b) Feature channels for the plan.

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

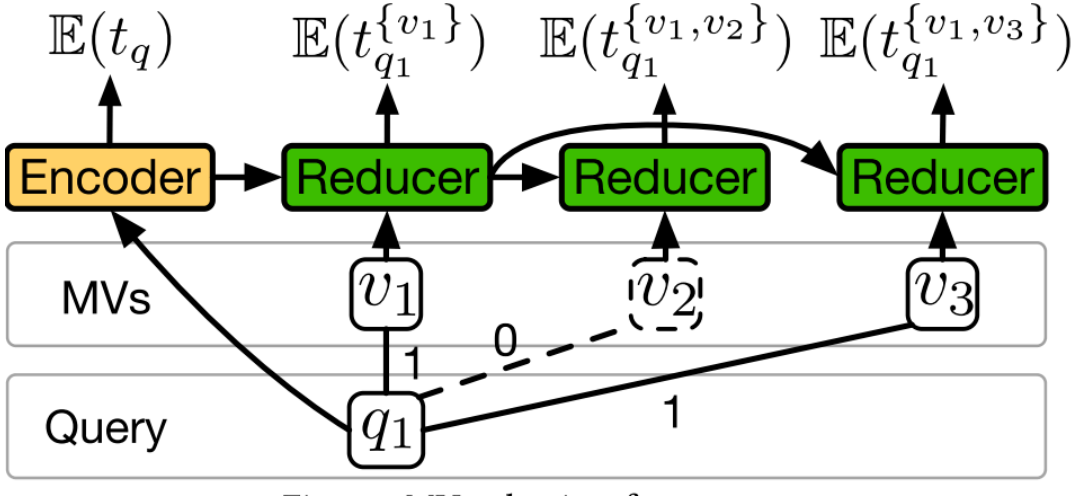
Encoder-Decoder for View Benefit Estimation

Feature Extraction

- Previous work take candidate views as fixed length →
- Encode various number and length of queries and views with an *encoder-reducer model*, which captures correlations with *attention*

Model Construction

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



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

Take-aways of Benefit Estimation

- **Learned utility estimation is more accurate than traditional empirical methods**
- **Learned utility estimation is also accurate for multiple-MV optimization**
- **Query encoding models need to be trained periodically when data update**
- **Open problems:**
  - Benefit prediction for future workload
  - Cost of initialization and future updates

Thanks