DATABASE WORKLOAD CHARACTERIZATION WITH QUERY PLAN ENCODERS

DEBJYOTI PAUL, JIE CAO, FEIFEI LI, VIVEK SRIKUMAR
THE UNIVERSITY OF UTAH

INTIAL DLAB THE UNIVERSITY OF UTAH VLDB 2022

Debjyoti Paul currently working as a Senior Research Scientist at Meta

OUTLINE

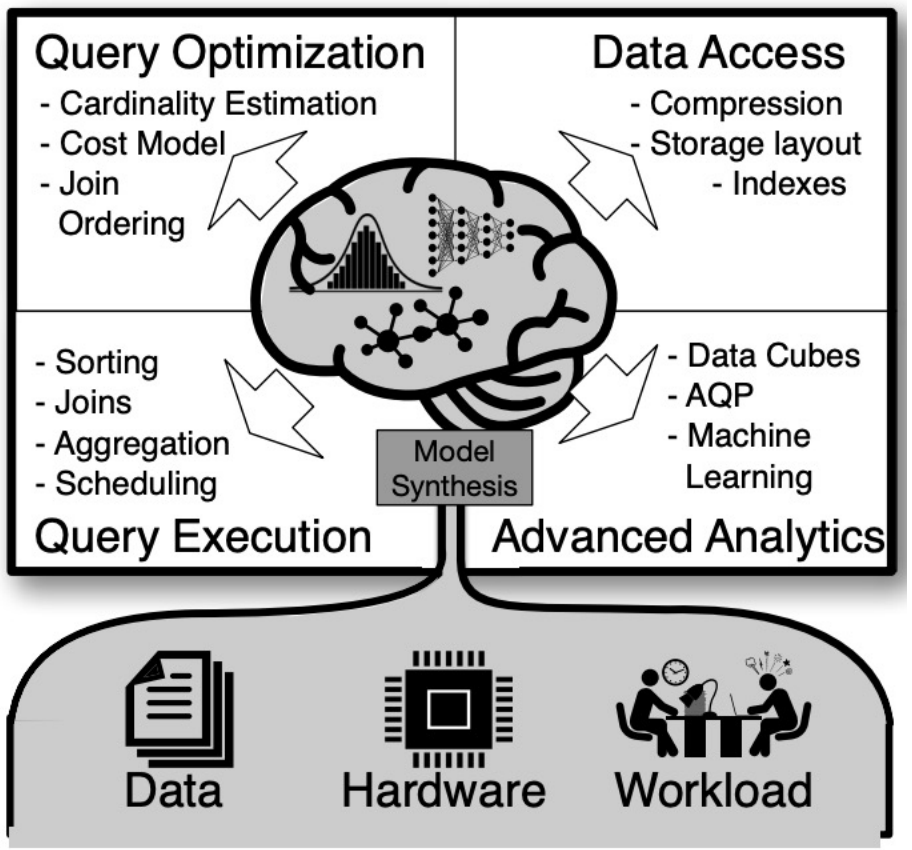
Motivation and Background

Problem Statement

Query Plan Encoders

Experiments

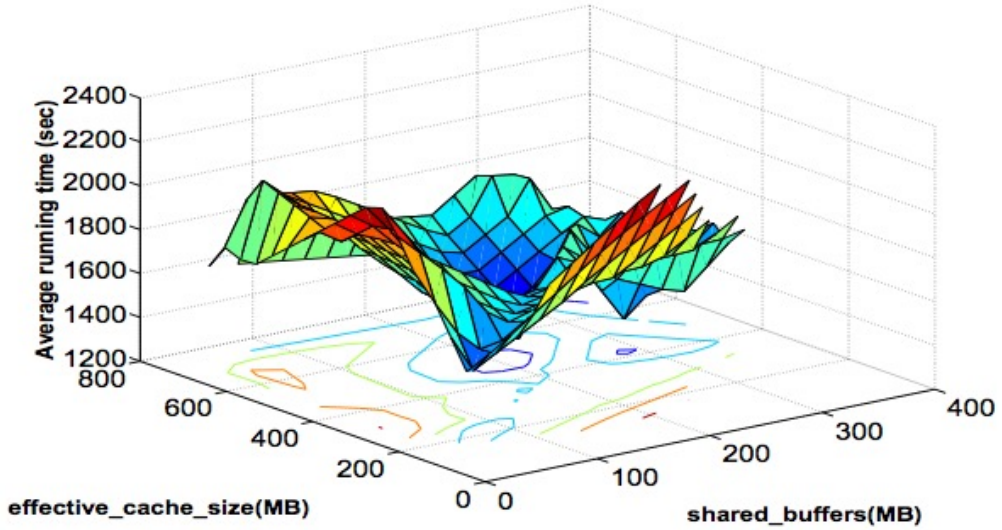
MOTIVATION



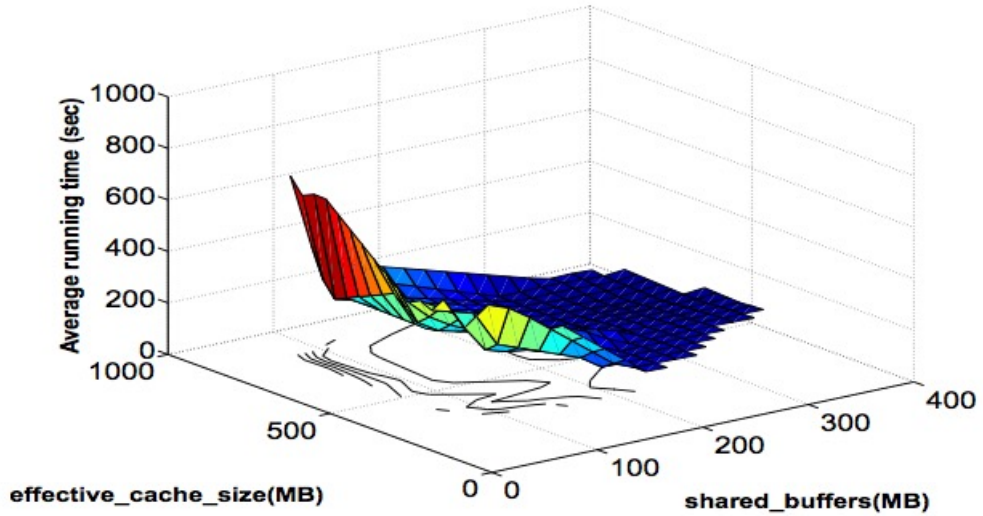
Architecture of a learned database system [1]

[1] Kraska, Tim, et al. "Sagedb: A learned database system." (2021).

MOTIVATION



TPC-H Q7



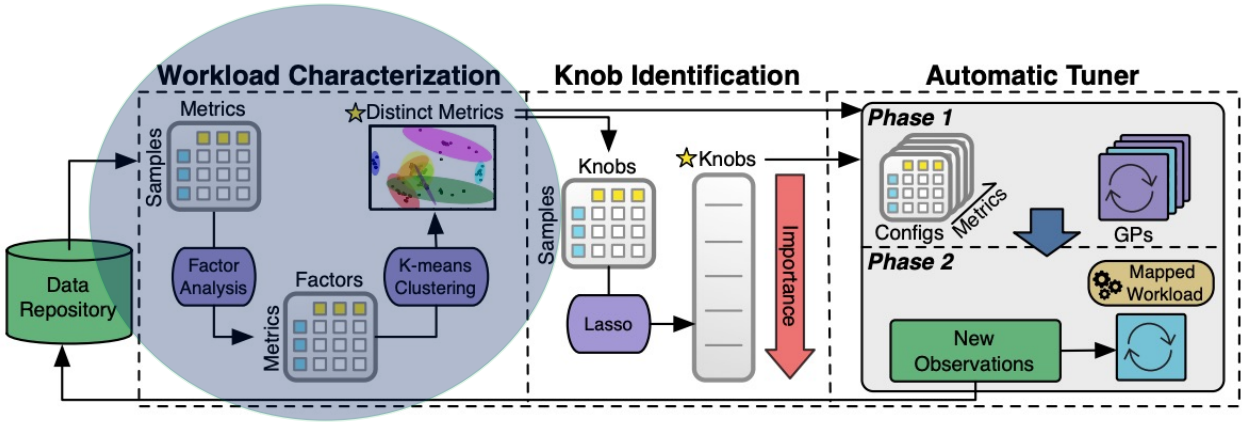
TPC-H Q18

Partial visualization of average running time of TPC-H Query 7 and 18 respectively with respect to two database configuration settings. [2]

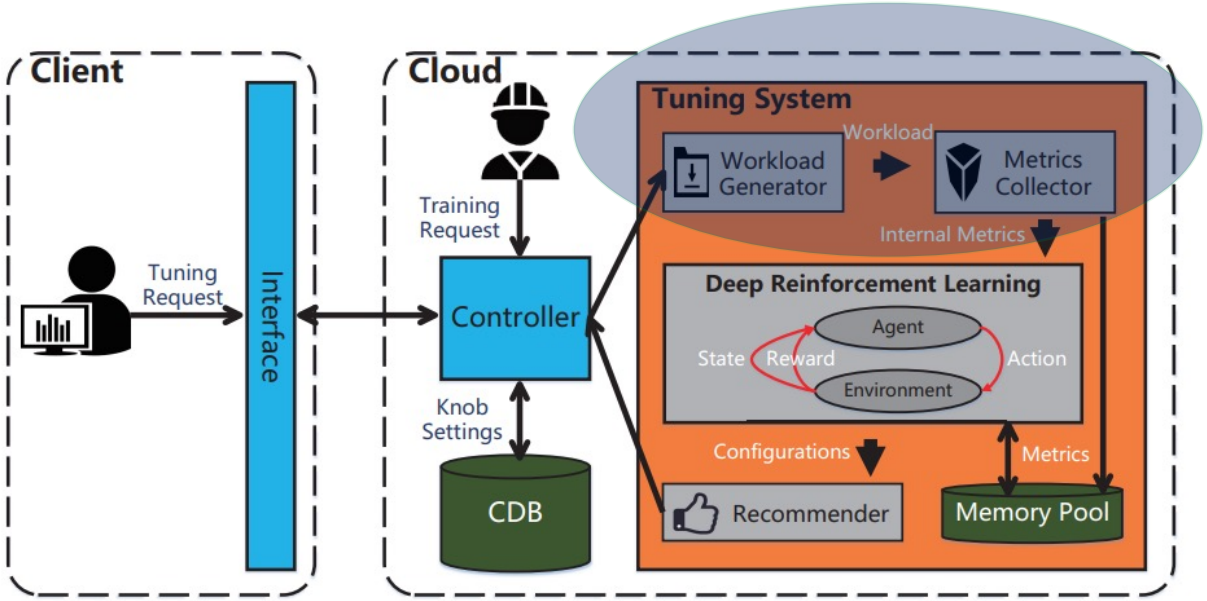
[2] From presentation slides of Vamsidhar Thummala and Shivnath Babu. 2010. iTuned: a tool for configuring and visualizing database parameters. In Proceedings of the 2010 ACM SIGMOD International Conference on Management of data. ACM.

BACKGROUND

- Understanding workload properties is necessary.
- OtterTune performs Factor Analysis on metrics on pre-executed workload on that specific DB.
- CDBTune learns by examples with DRL agent. Also, learns workload from scratch for that specific DB.



OtterTune[3] Database optimization pipeline



Database tuning system from CDBTune[4]

[3] Van Aken, Dana, et al. "Automatic database management system tuning through large-scale machine learning." Proceedings of the 2017 ACM international conference on management of data. 2017.
[4] Zhang, Ji, et al. "An end-to-end automatic cloud database tuning system using deep reinforcement learning." Proceedings of the 2019 International Conference on Management of Data. 2019.

BACKGROUND

```
Limit  (cost=4846863.88..4846864.07 rows=75 width=49) (actual time=2158.613..2158.624 rows=75 loops=1)
  Buffers: shared hit=494749, temp read=1738 written=1732
  ->  Sort  (cost=4846863.88..4846867.54 rows=1465 width=49) (actual time=2158.612..2158.618 rows=75 loops=1)
        Sort Key: businesspa3_.name
        Sort Method: top-N heapsort  Memory: 32kB
        Buffers: shared hit=494749, temp read=1738 written=1732
        ->  HashAggregate  (cost=4846796.28..4846810.93 rows=1465 width=49) (actual time=2158.432..2158.460 rows=204 loops=1)
              Group Key: businesspa3_.name, businesspa3_.c_bpartner_id
              Buffers: shared hit=494749, temp read=1738 written=1732
              ->  Nested Loop  (cost=76345.62..4846788.96 rows=1465 width=49) (actual time=564.139..2158.252 rows=222 loops=1)
                    Buffers: shared hit=494749, temp read=1738 written=1732
                    ->  Nested Loop Anti Join  (cost=76345.06..4836399.98 rows=1465 width=33) (actual time=564.125..2155.696 rows=222 loops=1)
                          Buffers: shared hit=493637, temp read=1738 written=1732
                          ->  Nested Loop  (cost=76344.64..4826479.91 rows=1466 width=66) (actual time=564.111..2154.366 rows=241 loops=1)
                                Buffers: shared hit=492893, temp read=1738 written=1732
                                ->  Hash Join  (cost=76344.08..3994752.83 rows=112973 width=66) (actual time=530.833..2145.510 rows=946 loops=1)
                                      Hash Cond: ((orderline0_.m_product_id)::text = (product2_.m_product_id)::text)
                                      Buffers: shared hit=488149, temp read=1738 written=1732
                                      ->  Bitmap Heap Scan on c_orderline orderline0_  (cost=56622.97..3918541.26 rows=1539661 width=99) (actual time=193.664..1798.113 rows=81707 loops=1)
                                            Recheck Cond: ((ad_org_id)::text = '392FC5F103F442A597F2F4C07698A9A9'::text)
                                            Rows Removed by Index Recheck: 3835710
                                            Filter: (qtydelivered <> qtyordered)
                                            Rows Removed by Filter: 1313143
                                            Heap Blocks: exact=17242 lossy=388911
                                            Buffers: shared hit=416181
                                            ->  Bitmap Index Scan on em_oborpre_ordline_org  (cost=0.00..56238.05 rows=1547398 width=0) (actual time=187.271..187.271 rows=1401574 loops=1)
                                                  Index Cond: ((ad_org_id)::text = '392FC5F103F442A597F2F4C07698A9A9'::text)
                                                  Buffers: shared hit=10028
                                      ->  Hash  (cost=15897.14..15897.14 rows=188238 width=33) (actual time=307.022..307.022 rows=188956 loops=1)
                                            Buckets: 131072  Batches: 4  Memory Usage: 4034kB
                                            Buffers: shared hit=71968, temp written=915
                                ->  Index Scan using c_order_key on c_order order1_  (cost=0.56..7.35 rows=1 width=66) (actual time=0.009..0.009 rows=0 loops=946)
                                      Index Cond: ((c_order_id)::text = (orderline0_.c_order_id)::text)
                                      Rows Removed by Filter: 1
                                      Buffers: shared hit=4744
                          ->  Index Only Scan using obre_reservation_cordline_idx on obre_reservation obre_reser5_  (cost=0.42..6.77 rows=1 width=33) (actual time=0.005..0.005 rows=0 loops=241)
                                Index Cond: (c_orderline_id = (orderline0_.c_orderline_id)::text)
                                Heap Fetches: 19
                                Buffers: shared hit=744
                    ->  Index Scan using c_bpartner_key on c_bpartner businesspa3_  (cost=0.56..7.08 rows=1 width=49) (actual time=0.011..0.011 rows=1 loops=222)
                          Index Cond: ((c_bpartner_id)::text = (order1_.c_bpartner_id)::text)
                          Buffers: shared hit=1112
Planning time: 2.080 ms
Execution time: 2158.777 ms
(60 rows)
```

A query plan example. It is challenging to comprehend relevant features from query plans for optimal execution. Only an expert DBA can optimize DB configuration for queries, but it is a tedious and manual effort.

BACKGROUND

30. Tuning PostgreSQL for Spatial

PostgreSQL is a very versatile database system, capable of running efficiently in very low-resource environments and environments shared with a variety of other applications. In order to ensure it will run properly for many different environments, the default configuration is very conservative and not terribly appropriate for a high-performance production database. Add the fact that geospatial databases have different usage patterns, and the data tend to consist of fewer, much larger records than non-geospatial databases, and you can see that the default configuration will not be totally appropriate for our purposes.

shared_buffers

- Default: ~128MB in PostgreSQL 9.6
- Set to about 25% to 40% of available RAM. On windows you may not be able to set as high.

bump up `max_worker_processes` to at least as high as this number.

While there is no magic bullet for this value, the default is
ıg the `SET random_page_cost TO 2.0` command.

work_mem (the memory used for sort operations and complex queries)

- Default: 1-4MB
- Adjust up for large dbs, complex queries, lots of RAM
- Adjust down for many concurrent users or low RAM.

```
SET maintenance_work_mem TO '1GB';
```

```
SET maintenance_work_mem TO '128MB';
VACUUM ANALYZE;
SET maintenance_work_mem TO '16MB';
```

PRELIMINARIES

Database workload defined as,

$$W = \{(q_1, \theta_1), (q_2, \theta_2), \dots\dots, (q_n, \theta_n)\}$$

where θ_i is normalized weight of query q_i in workload W such that $\sum_{i=1}^{n} \theta_i = 1$.

A query q_i can generate different query plans say $\{p_k, p_l\}$ on different DB instances, based on database configuration and underlying data.

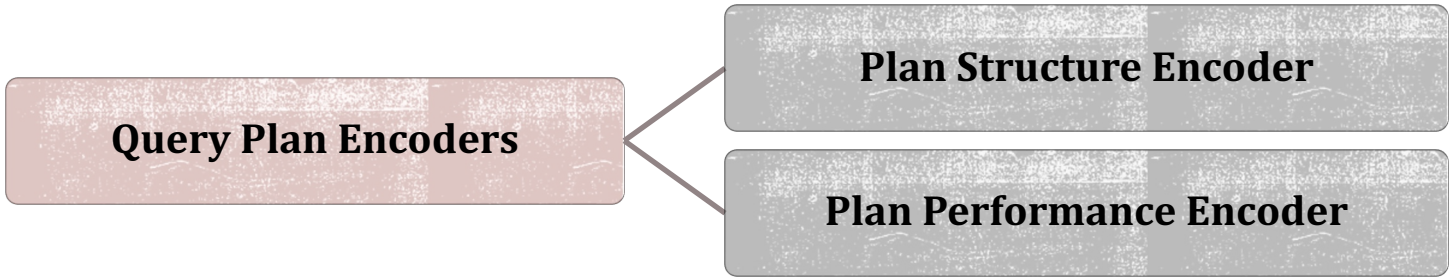
It will be wise to rewrite W as,

$$W = \{(p_1, \theta_1), (p_2, \theta_2), \dots\dots, (p_m, \theta_m)\}$$

where θ_i is normalized weight of query plan p_i in workload W such that $\sum_{i=1}^{m} \theta_i = 1$.

PROBLEM STATEMENT

Represent a query plan p_i with a distributed representation model i.e., a **query plan encoder** that captures the inherent characteristics such as **structure, computational performance**, and **database feature manifests** embedded within a query plan structure.



Query plan representations can help aggregate similar queries. With more insights on queries from encodings, DBAs (or downstream AI-based tasks) will optimize database performance efficiently.

ASSUMPTIONS AND CHALLENGES

- Independent Query Plan Features
- Diverse Plan Structure
- Modeling Computational Performance
- Environment Dependencies
- Domain Adaptation

STRUCTURE ENCODER

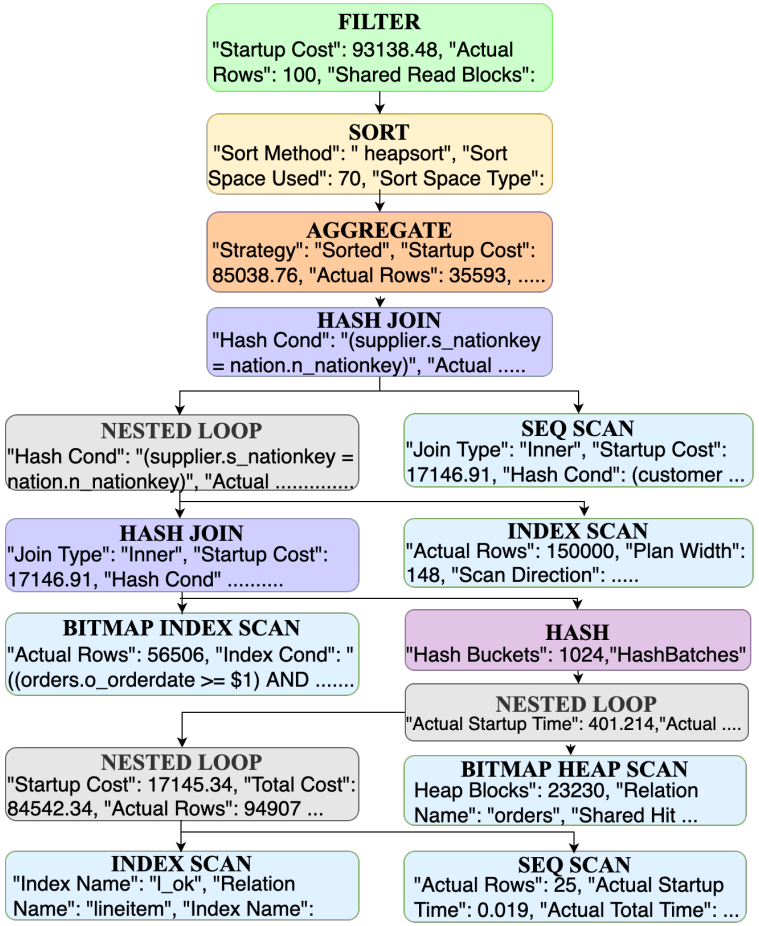


Figure 1: Query Plan Example: TPC-H Query Template 5

Level	Operator Subtypes
Level 1	Aggregate, Append, Count, Delete, Enum, Gather, Aggregate (Group, GroupAggregate), Insert, Intersect, Join (Nested Loop), Limit, LockRows, Loop, ModifyTable, Network, Result, Scan, Sequence, Set(SetOp), Sort, Union, Unique, Update, Window, WindowAgg, Materialize
Level 2	And, CTE, Except, Exists, Foreign, Hash, Heap, Index, IndexOnly, LoopHash, Merge, Or, Query, Quick, Seq, SetOp, Subquery, Table, WorkTable
Level 3	Anti, Bitmap, Full, Left, Parallel, Partial, Partition, Right, Semi, XN

Table 1: The taxonomy of operator types for every node

STRUCTURE ENCODER

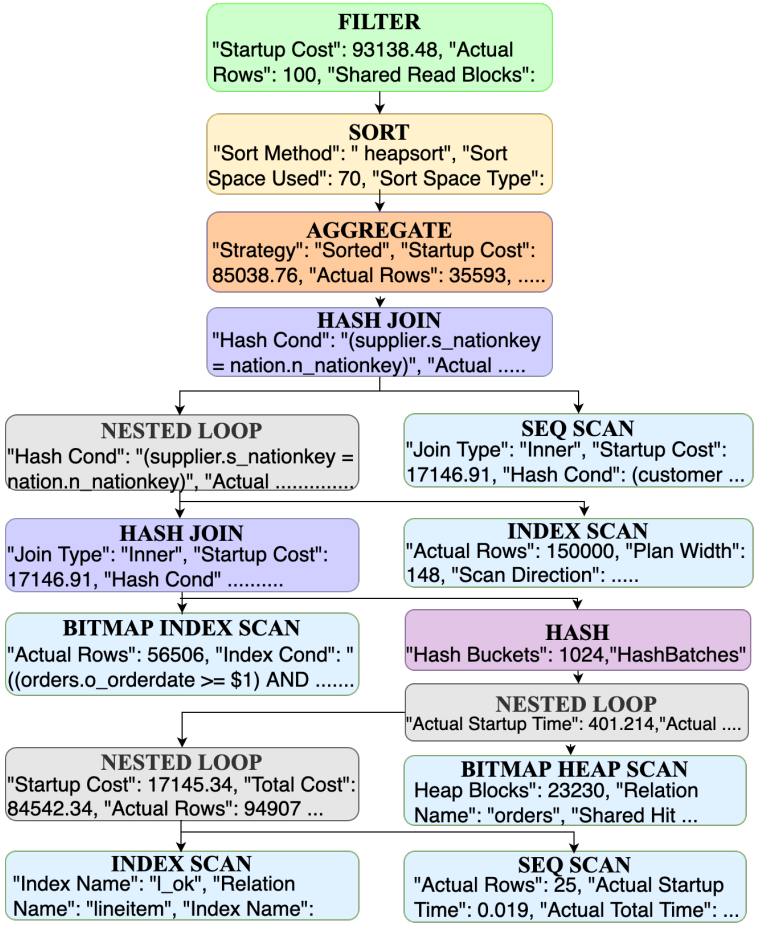


Figure 1: Query Plan Example: TPC-H Query Template 5

Strategy	Node Sequence
DFS	Filter–, Sort–, Aggregate–, Join-Hash-, Join–, Join-Hash-, Scan-Heap-Bitmap, Hash–, Join–, Join–, Scan-Index-, Scan-Seq-, Scan-Heap-Bitmap, Scan-Index-, Scan-Seq
BFS	Filter–, Sort–, Aggregate–, Join-Hash-, Join–, Scan-Seq-, Join-Hash-, Scan-Index-, Scan- Heap-Bitmap, Hash–, Join–, Join–, Scan-Heap-Bitmap, Scan-Index-, Scan-Seq-
DFS Bracket	(Filter–, (Sort–, (Aggregate–, (Join-Hash-, (Loop–Nested, (Join-Hash-, (Hash–, (Loop–Nested, (Loop–Nested, ScanIndex-, Scan-Seq-) Scan-Heap-Bitmap)) Scan-IndexBitmap) Scan-Index-) Scan-Seq-))))

Table 2: Three Tree Traversal Strategies for Node Sequencing on Figure 1 Plan

STRUCTURE ENCODER: POSITIONAL ENCODING

DFS Bracket Strategy

a. (((→ 1,1,1 → [0,0,1,0,0,1,0,0,1,0,0,0],

b. (( ) ((→ 1,2,1 → [0,0,1,0,1,0,0,0,1,0,0,0],

c. (((( )) ((→ 1,1,2,2 → [0,0,1,0,0,1,0,1,0,0,1,0]

STRUCTURE ENCODER

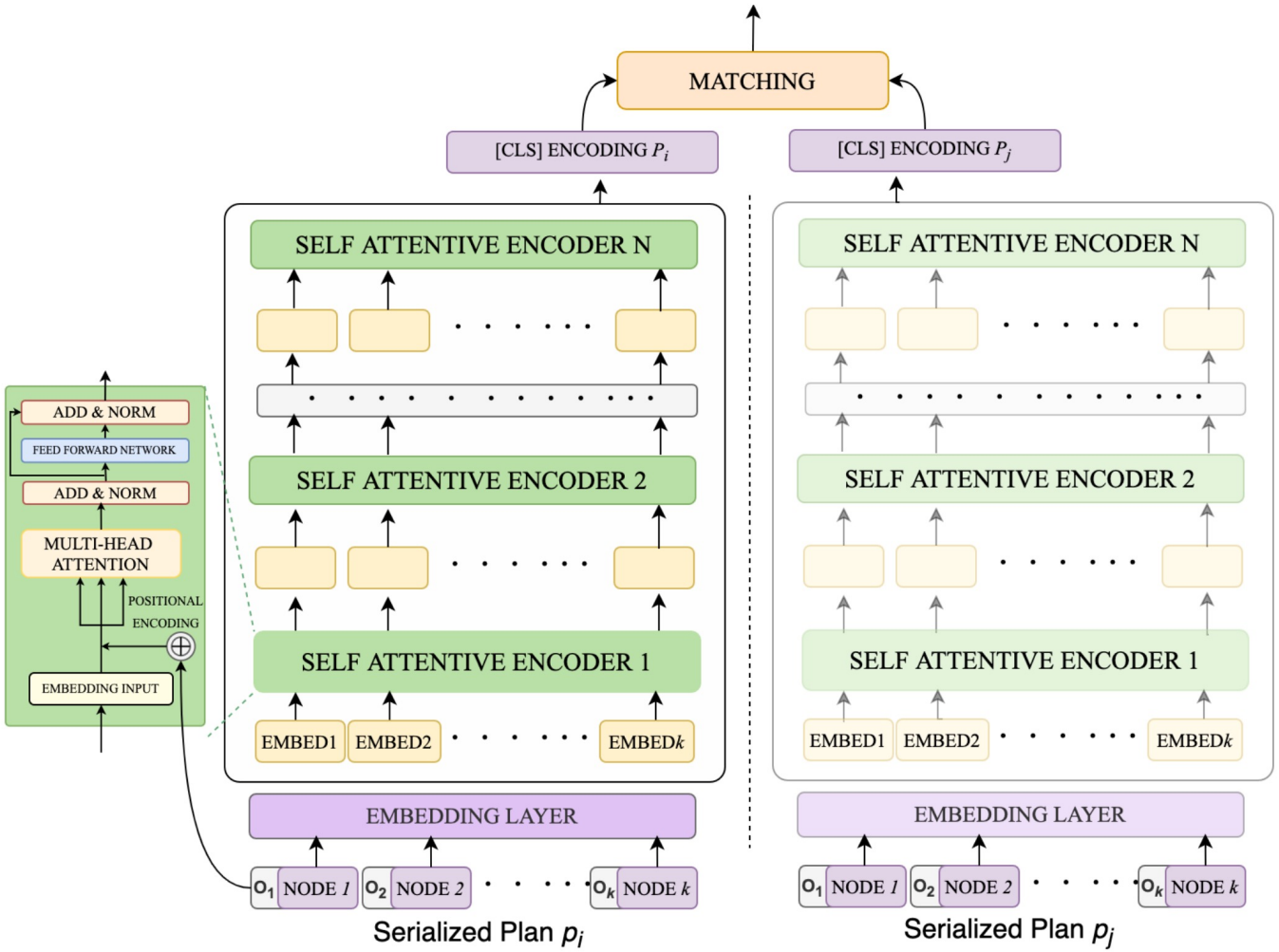
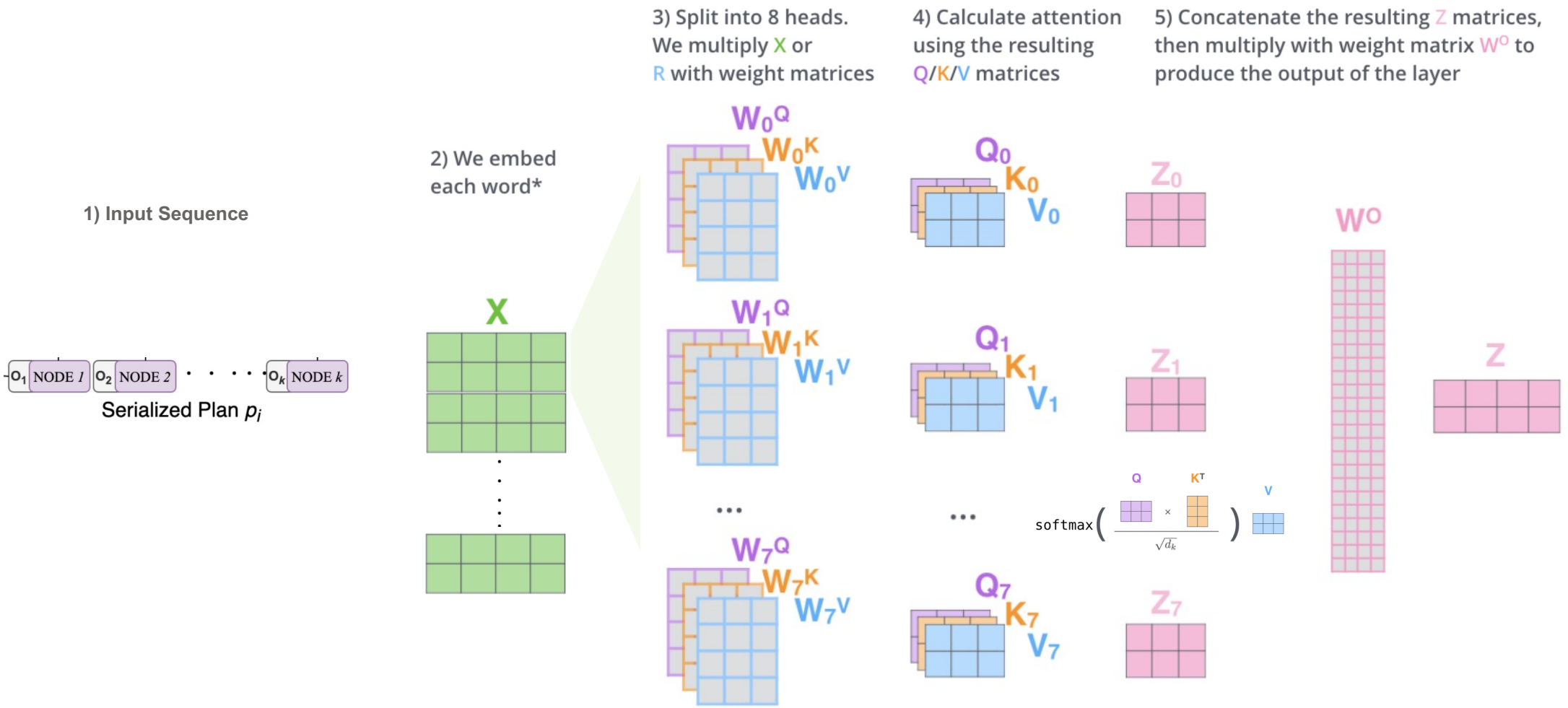


Figure 2: Structure Encoder modeling with a pair of serialized plans p_i and p_j.

SELF ATTENTION ENCODER



PERFORMANCE ENCODER

Operator	Plan Node Features
All	Actual Loops , Actual Rows , Local Dirtied Blocks , Local Hit Blocks , Local Read Blocks , Local Written Blocks , Plan Rows , Plan Width , Shared Dirtied Blocks , Shared Hit Blocks , Shared Read Blocks , Shared Written Blocks , Temp Read Blocks , Temp Written Blocks , Parent Relationship , Plan Buffers
Scan	Relation Name, Scan Direction, Index Name, Index Condition, Scan Condition, Filter, Rows Removed, Heap Blocks, Parallel, Recheck Condition
Join	Join Type, Inner Unique, Merge Condition, Hash Condition, Rows Removed by Join Filter, Parent Relationship, Hash Algorithm, Hash Algo, Hash Buckets, Hash Batches, Peak Memory
Sort	Sort Type, Sort Method, Sort Space, Sort Key, Sort Space Type, Sort Space Used, Peak Memory
Aggregate	Strategy, Hash Algo, Hash Buckets, Hash Batches, Parallel Aware, Partial Mode, Peak Memory

Table 3: The properties from query execution plan that are common to all the operators and a few specific to major operators like Scan, Join, Sort and Aggregate.

PERFORMANCE ENCODER

Strategy	Node Sequence
Meta Features	rel_name, att_name, rel_tuples, rel_pages, rel_file_node, rel_access_method, n_distinct, distinct_values, selectivity, avg_width, correlation
DB Settings	bgwriter_delay, shared_buffers, bgwriter_lru_maxpages, wal_buffers, random_page_cost, bgwriter_lru_multiplier, checkpoint_completion_target, checkpoint_timeout, cpu_tuple_cost, max_stack_depth, deadlock_timeout, default_statistics_target, work_mem effective_cache_size, effective_io_concurrency, join_collapse_limit, from_collapse_limit, maintenance_work_mem

Table 4: Meta Features and DB Settings used as input features to Computational Performance Encoder

An encoder for each major functional operators

(i) Scan (ii) Join (iii) Sort (iv) Aggregate (v) Others

Regression Task Labels:

(i) Execution Time (Latency) (ii) Cost (iii) Startup Time

PERFORMANCE ENCODER

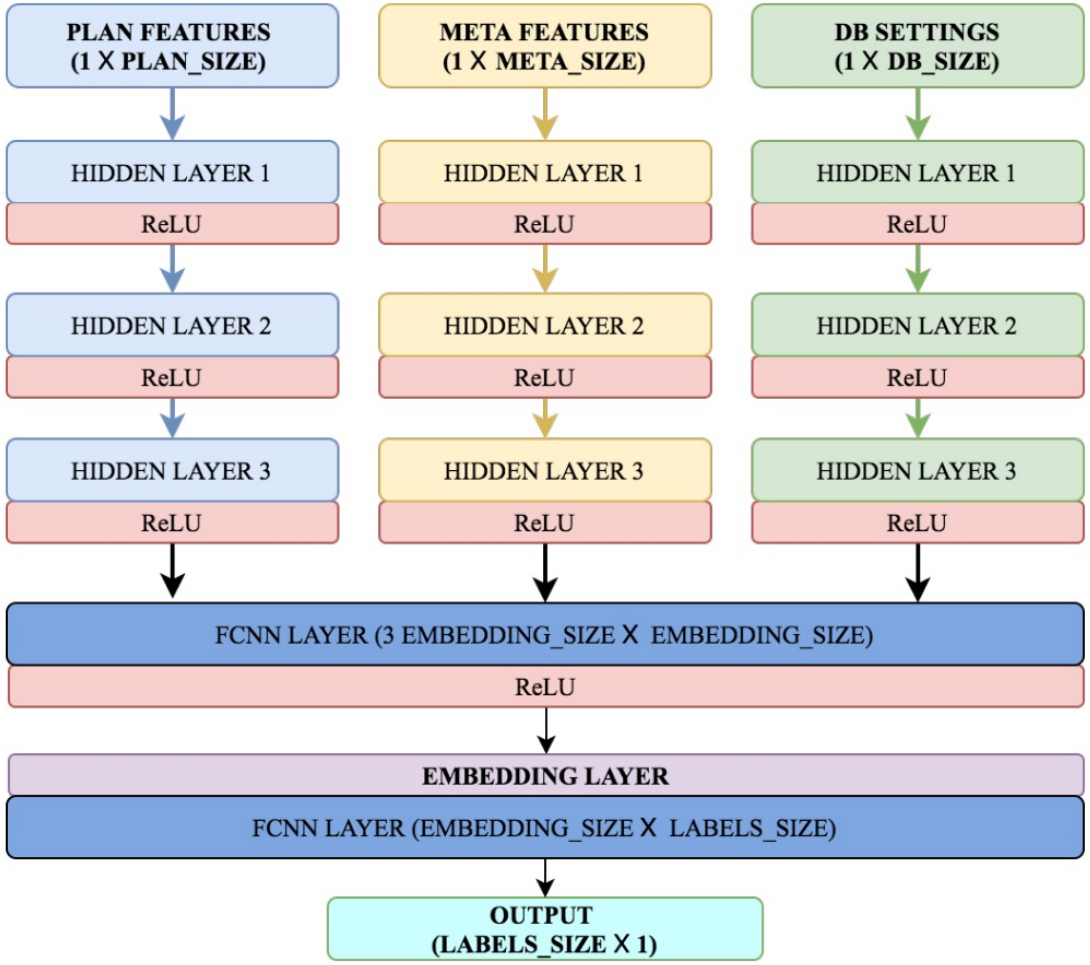


Figure 3: The multi-column deep neural network(DNN) for our computational performance encoder.

DOWNSTREAM TASK

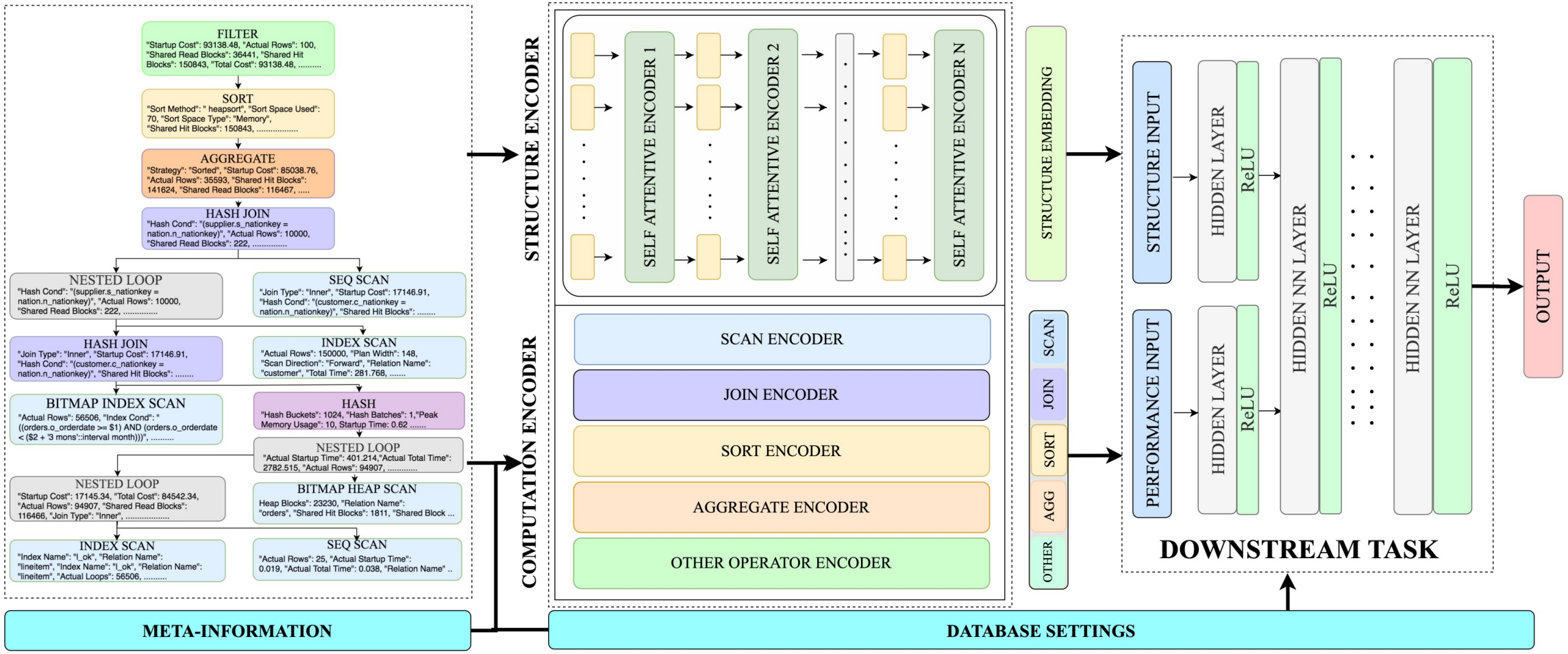


Figure 4: A bird-view diagram, showing the role of plan encoders for a downstream task.

Latency Prediction Task: *Given a query plan, meta-features of the database, and a new database configuration settings, the model predicts the latency of the query on the given database knob configuration.*

DATASETS AND RESOURCES

- **Crowdsourced Plan Dataset [2] :** For Structure Encoder training. 131,521 structurally diverse plans. 14,016 similar plan.
- **Industry Standard Benchmarks TPC-H and TPC-DS:** For Performance Encoder training, evaluation and ablation studies.
- **Spatial Benchmarks:** For Latency Prediction task.
  - **Jackpine [3]:** Revised Jackpine benchmark supporting current PostGIS and new shapefiles. Contains diverse queries on spatial join with multipolygons, lines, points and combination of them.
  - **Open Street Map (OSM):** This workload contains queries for spatial overlap, distance and pair point routing. Used OSM map of Los Angeles County and New York City.
- **Join-Order Benchmarks:** For Query Classification task.
- **Cloud resources:** https://cloudlab.us

  We used 50+ cloud machine instances for running our experiment and benchmarks.
- **Tools:**

  (a) GITHUB: WORKLOAD_SCRIPTS (b) GITHUB: JACKPINE (c) GITHUB: OSM_BENCHMARK

[2] PostgreSQL's explain analyze made readable: https://explain.depesz.com/
[3] Suprio Ray, Bogdan Simion, and Angela Demke Brown. 2011. Jackpine: A benchmark to evaluate spatial database performance. In 2011 IEEE 27th International Conference on Data Engineering.

EXPERIMENTS: LATENCY PREDICTION TASK

- **Pretrained Structure and Performance Encoders:** Trained with Crowdsourced Query, TPC-H and TPC-DS benchmarks.
- **Finetuning In-Domain Data:** Jackpine and OSM workload instances with 120 LHS-generated database settings.
- **Test Data:** Jackpine and OSM workload instances with 50 LHS-generated database settings.

Database Settings	Unit	Median	95th Percentile	5th Percentile
bgwriter_delay	ms	4860.00	9,421.05	456.00
bgwriter_lru_maxpages	integer	515.00	958.05	55.00
checkpoint_timeout	ms	300.00	540.00	60.00
deadlock_timeout	ms	300,000.00	540,000.00	26,000.00
default_statistics_target	integer	4827.50	9,563.00	454.85
effective_cache_size	bytes	1,048,576.00	1,966,080.00	131,072.00
effective_io_concurrency	integer	52.00	96.00	6.00
maintenance_work_mem	bytes	7,340,032.00	15,728,640.00	876,953.60
max_stack_depth	integer	3,072.00	5,120.00	417.95
random_page_cost	number	5,028.60	9,507.39	560.40
shared_buffers	bytes	2,0977,152.00	3,932,160.00	131,072.00
wal_buffers	bytes	130,624.00	131,072.00	12,416.00
work_mem	bytes	15,728,640.00	31,457,280.00	1,048,576.00

EXPERIMENTS: LATENCY PREDICTION TASK

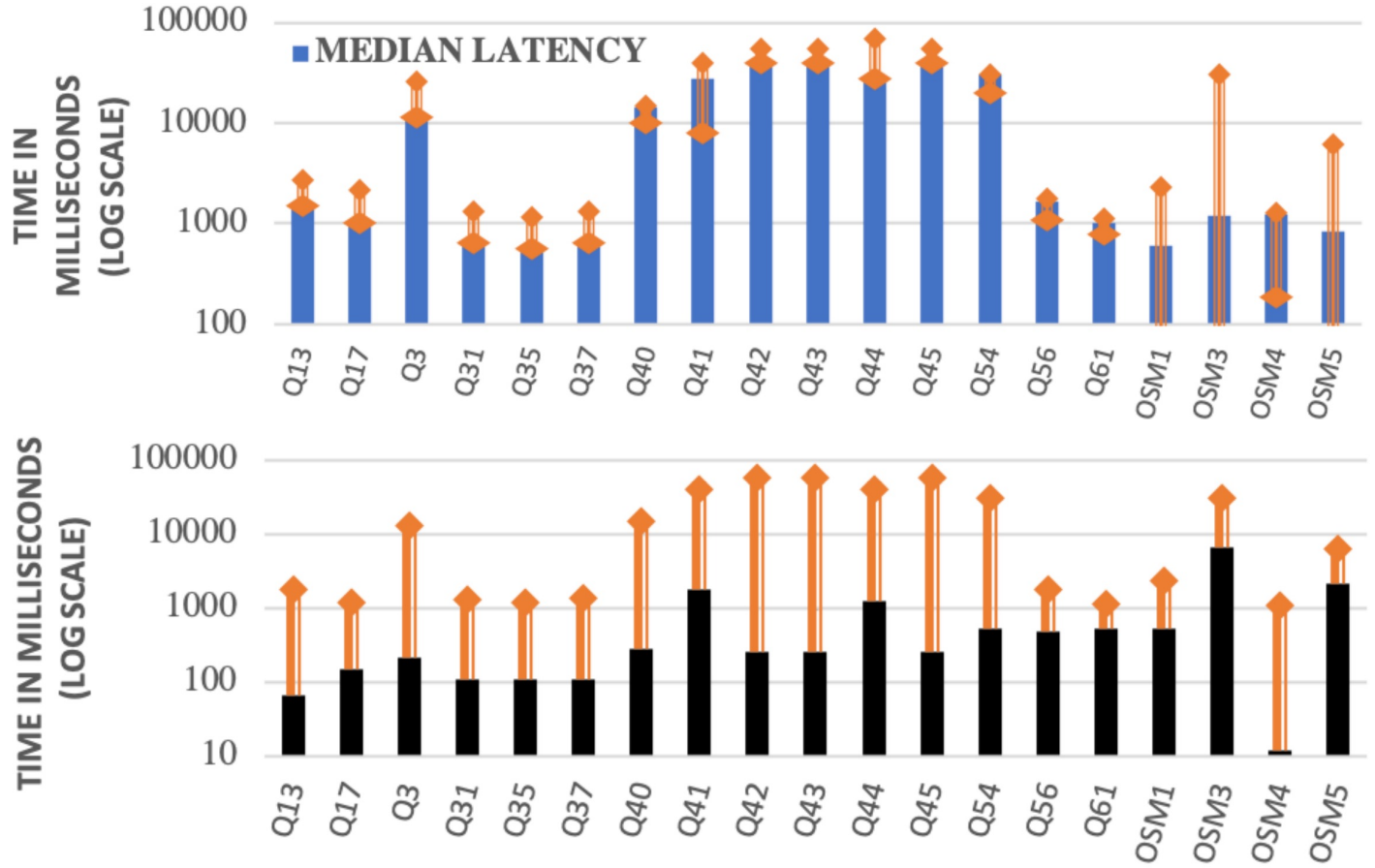


Figure 5: Latency of spatial queries (> 500 ms) from Jackpine[3] and OSM benchmark with Error Analysis. Blue bar is median of latency, Orange line ranges between 5th and 95th percentile of query latency. Black bar is mean absolute error (MAE), a low black bar on a high orange line bar means better results.

[3] Suprio Ray, Bogdan Simion, and Angela Demke Brown. 2011. Jackpine: A benchmark to evaluate spatial database performance. In 2011 IEEE 27th International Conference on Data Engineering.

EXPERIMENTS: LATENCY PREDICTION TASK

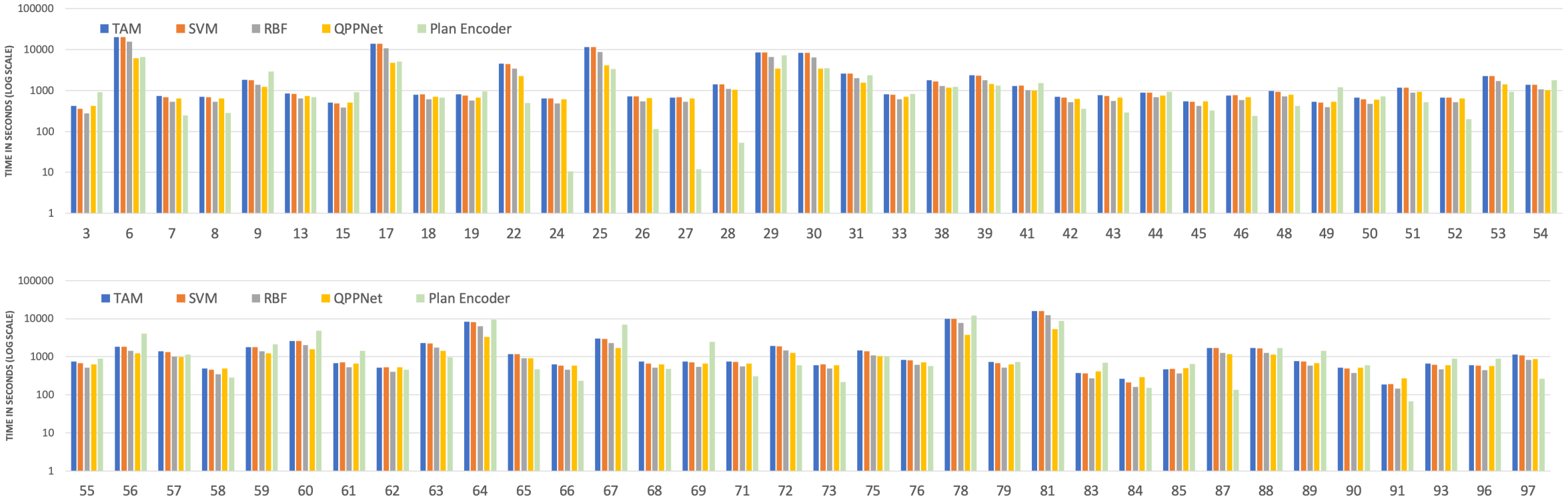


Figure 6: Ablation study of mean absolute error (MAE) (y-axis in logarithmic scale) for the all the TPC-DS query templates (x-axis) with scale factor 100.

EXPERIMENTS: LATENCY PREDICTION TASK

Models	$R \leq 1.5$	$1.5 < R \leq 2.0$	$R > 2.0$
TAM [4]	51%	22%	27%
SVF [5]	68%	15%	17%
RBF [6]	85%	6%	9%
QPPNet [7]	89%	**7%**	4%
Plan Encoder	**91%**	**7%**	**2%**

Table 5: Queries from TPC-DS SF-100 test set binned based on *R* factor for all the models.

$$R(q) = \max\left(\frac{predicted(q)}{original(q)}, \frac{original(q)}{predicted(q)}\right)$$

[4] M. Akdere et al. Learning-based query performance modeling and prediction. In ICDE '12.
[5] J. Li et al. Robust estimation of resource consumption for SQL queries using statistical techniques. VLDB '12
[6] H. Hacigumus et al. Predicting Query Execution Time: Are Optimizer Cost Models Really Unusable? In ICDE '13.
[7] Marcus, Ryan, and Olga Papaemmanouil. "Plan-Structured Deep Neural Networks for Query Performance Prediction." The VLDB journal (2019).

EXPERIMENTS: QUERY CLASSIFICATION TASK

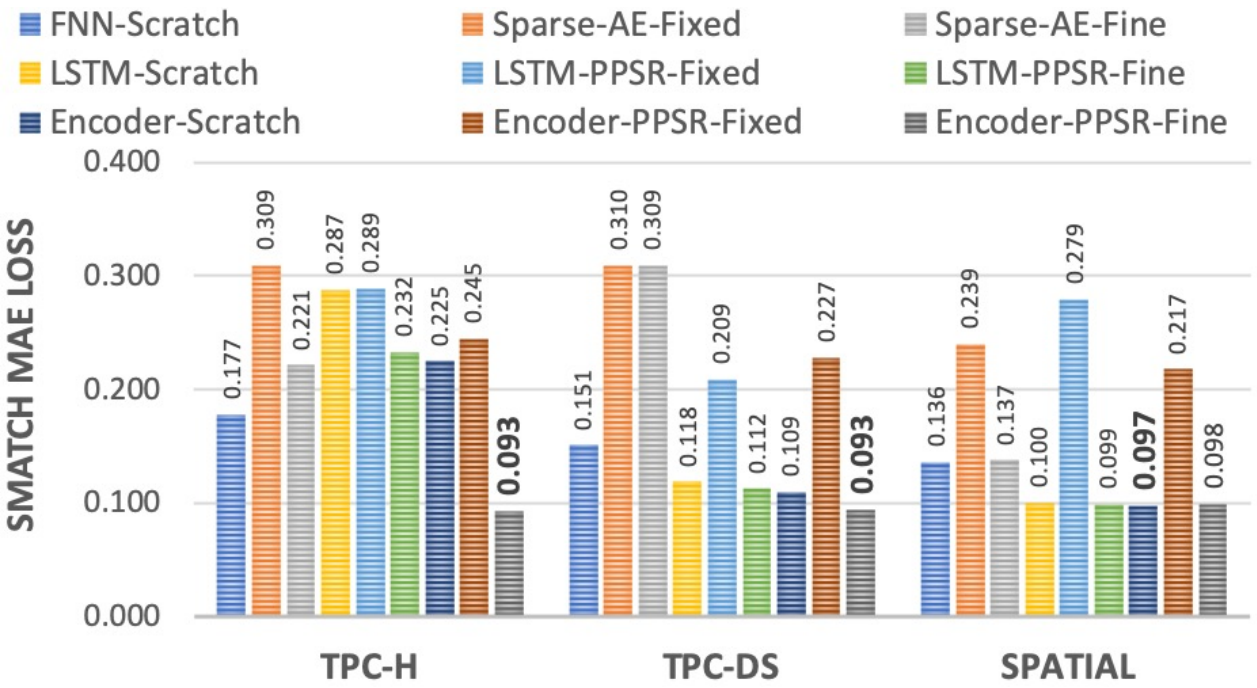
Models	Validation		Test	
	template	cluster	template	cluster
Structure only	0.2452	0.4670	0.1946	0.3847
Performance only	0.1645	0.2973	0.0977	0.1769
Both encoders	**0.2783**	**0.5573**	**0.2518 (+29%)**	**0.4647 (+21%)**
Both encoders 10% data	0.2000	0.4927	0.151	0.334
Both encoders 30% data	0.2555	0.5228	0.1843	0.3855

Table 6: F1-scores of models for template and cluster query classification task on validation and test set.

Join Order Dataset

- 113 query templates
- 33 clusters
- Total of 16,229 different plans on different DB configurations.
  - Train: 13,505 plans
  - Validation: 1,362
  - Test: 1,362

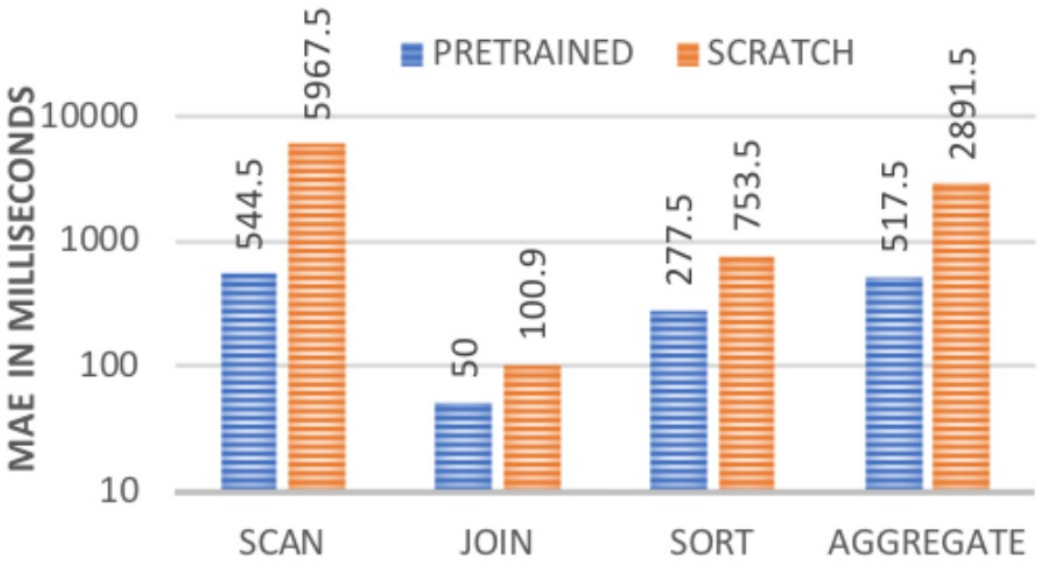
EXPERIMENTS: DOMAIN ADAPTATION



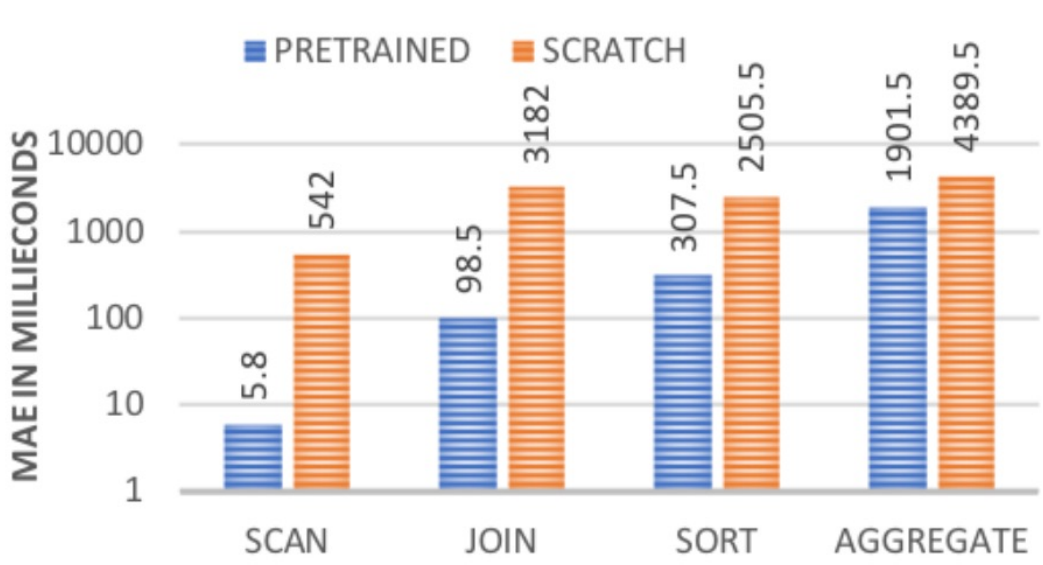
Taxonomies	Descriptions
Scratch	Untrained Encoder weights initialized.
Fixed	Pretrained Encoder no finetuning i.e., weights freeze.
Fine	Pretrained Encoder with finetuning.
FNN	Fully connected Neural Network
Sparse AE	Sparse Auto-Encoder
LSTM-PPSR	Long Short Memory Neural Network for plan pair similarity task
Encoder PPSR	Structure Encoder for Plan Pair Similarity Regression Task

Figure 7: Results of finetuning structure encoder on TPC-H, TPC-DS, and SPATIAL.
Note: Structure Encoder is pretrained on crowdsourced plan dataset.

EXPERIMENTS: DOMAIN ADAPTATION



(a) TPC-DS SF-8 benchmark.



(b) Spatial benchmark.

Figure 12: Comparison of MAEs for pretrained vs scratch models with 0.3 fraction of finetuning data.

EXPERIMENTS: DATABASE TUNING FOR SPATIAL WORKLOAD

Near Optimal Configurations

bgwriter_delay	6919
bgwriter_lru_maxpages	957
bgwriter_lru_multiplier	2.91304
checkpoint_completion_target	0.993648
checkpoint_timeout	480
cpu_tuple_cost	8.14646
deadlock_timeout	180000
default_statistics_target	2090
effective_cache_size	97024
effective_io_concurrency	80
maintenance_work_mem	5242880
max_stack_depth	1024
random_page_cost	8363.61
shared_buffers	1703936
wal_buffers	73856
work_mem	24117248

Jackpine workload

bgwriter_delay	5777
bgwriter_lru_maxpages	665
bgwriter_lru_multiplier	6.34007
checkpoint_completion_target	0.326022
checkpoint_timeout	240
cpu_tuple_cost	5.30396
deadlock_timeout	240000
default_statistics_target	6658
effective_cache_size	1703936
effective_io_concurrency	99
maintenance_work_mem	7340032
max_stack_depth	4096
random_page_cost	41.2078
shared_buffers	1310720
wal_buffers	131072
work_mem	9437184

OSM workload Los Angeles

bgwriter_delay	6856
bgwriter_lru_maxpages	518
bgwriter_lru_multiplier	7.26019
checkpoint_completion_target	0.69164
checkpoint_timeout	480
cpu_tuple_cost	4.10426
deadlock_timeout	300000
default_statistics_target	2837
effective_cache_size	1966080
effective_io_concurrency	58
maintenance_work_mem	14680064
max_stack_depth	4096
random_page_cost	8395.83
shared_buffers	2490368
wal_buffers	131072
work_mem	3145728

OSM workload New York City

CONCLUSIONS

- Representation learning with AI encoders for database queries works.
- May be using structure and performance encoder independently is not a bad idea.
- Large scale pretraining with small finetuning is better.
- Plan encoders as a core DB tool for workload characterization and gauging resource requirements.
- AI powered pretrained encoders prepackage with database management systems
- Can be integrated with Reinforcement Learning approaches to achieve instance optimality.

CHANGES IF I DO THIS WORK AGAIN

- Apply Transformer with ResNet architecture for performance encoders.
- Obtain large scale training dataset with many different configurations, I utilized cloudlab.us for running databases on cloud instances, and it takes a while to prepare data. Need for an open-sourced query, execution-plan dataset from cloud databases.
- Add experiment to integrate plan encoder with Reinforcement Learning based database tuning applications.

THANKS

RESOURCES