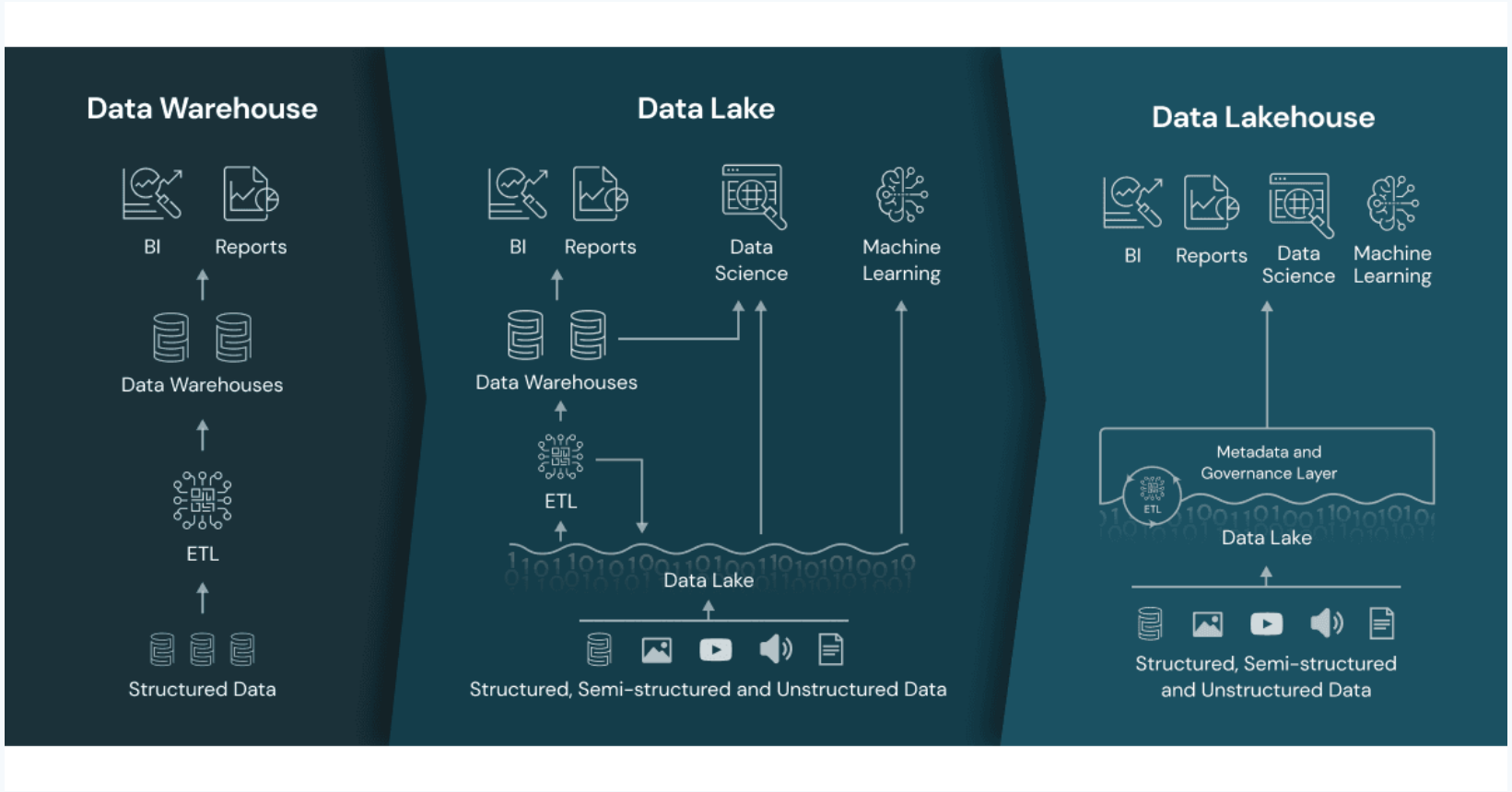
Beyond Indexes for Lakehouse Systems:

Bridging the Semantic Gap in RAG via Schema-Graphs and Optimized Physical Layouts

Vamsi Meduri

IBM Research – Silicon Valley

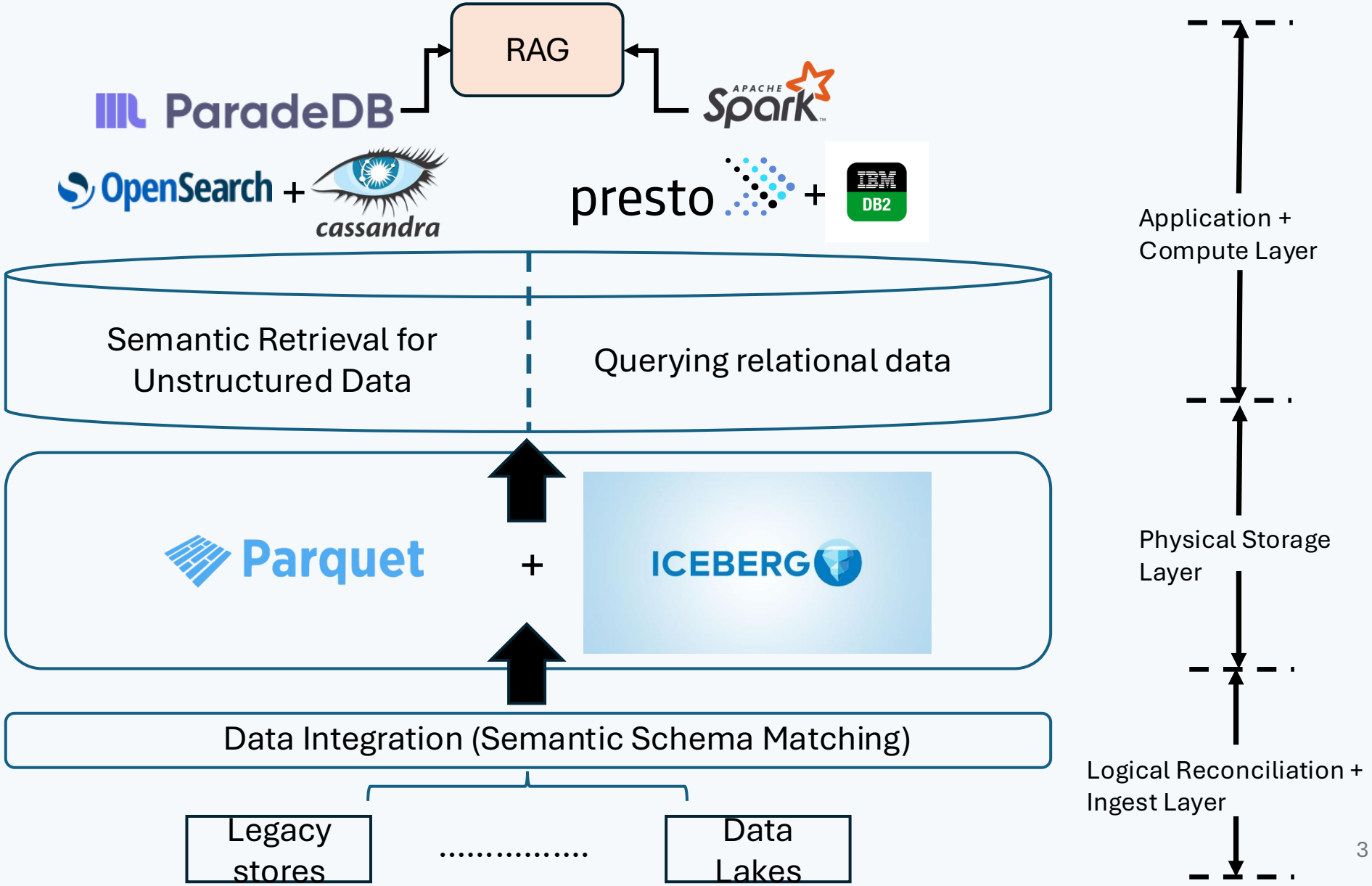
Lakehouses: Warehouses for Data Lakes



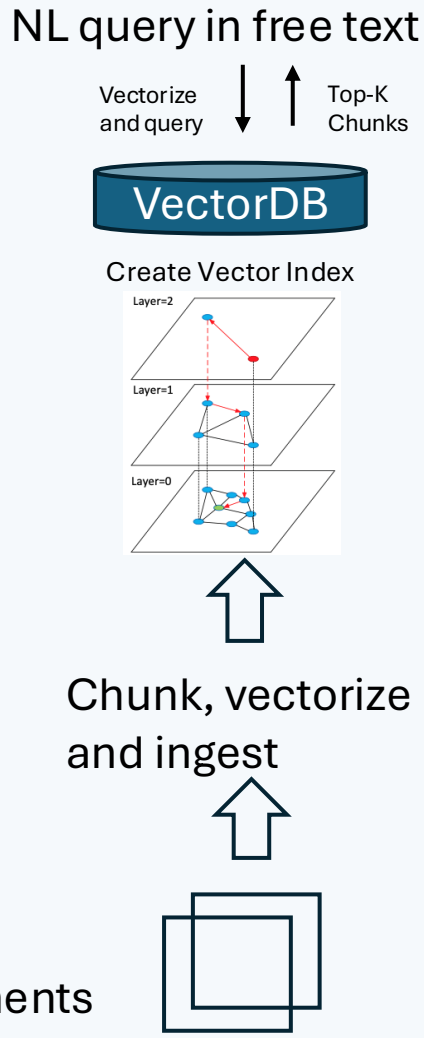
- Avoiding excessive ETLs by connecting compute to where data resides
- Decoupled data and compute
- Emergence of open data formats like Apache Iceberg, Hudi, Delta Tables

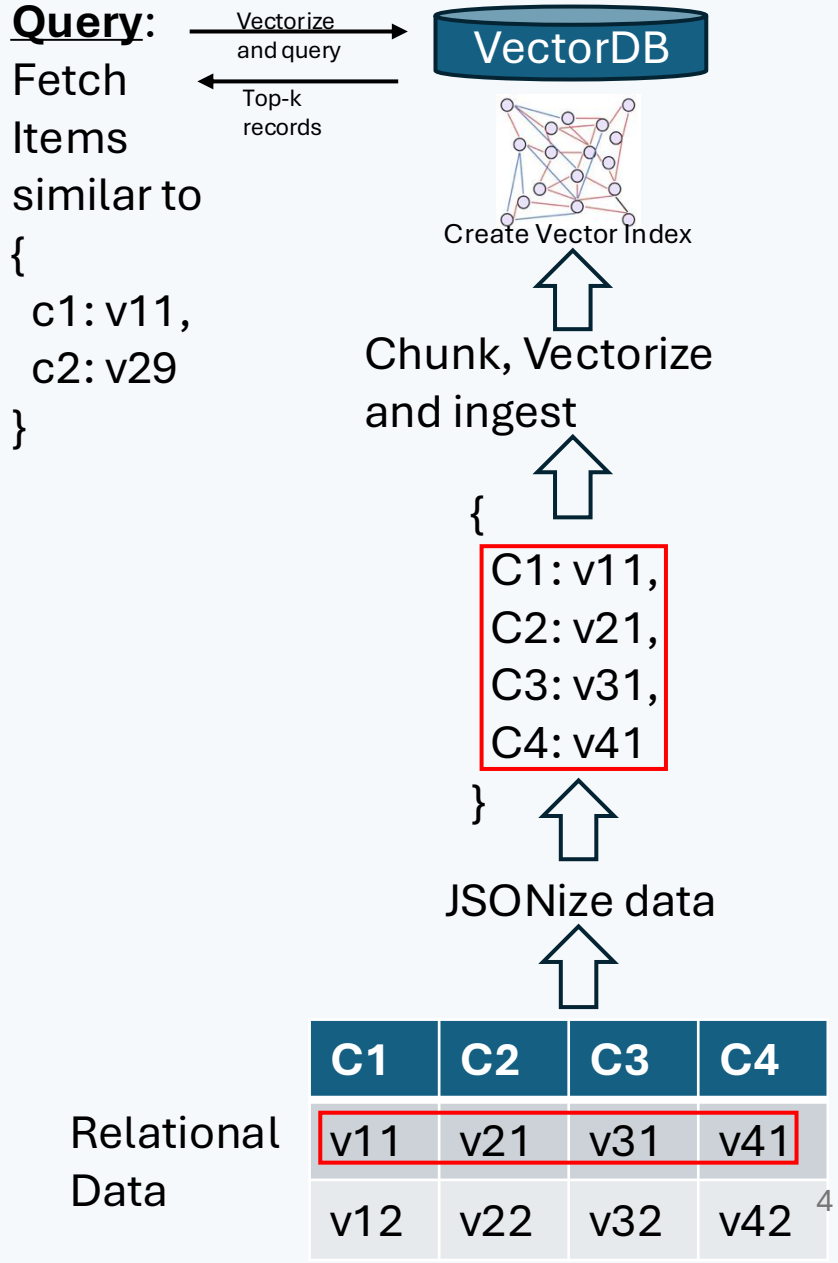
Image source: https://www.databricks.com/glossary/data-lakehouse

IBM Watsonx.data – Diverse query engines and retrieval mechanisms



Retrieval on Unstructured Data vs. Structured Data





- Declarative querying to perform retrieval

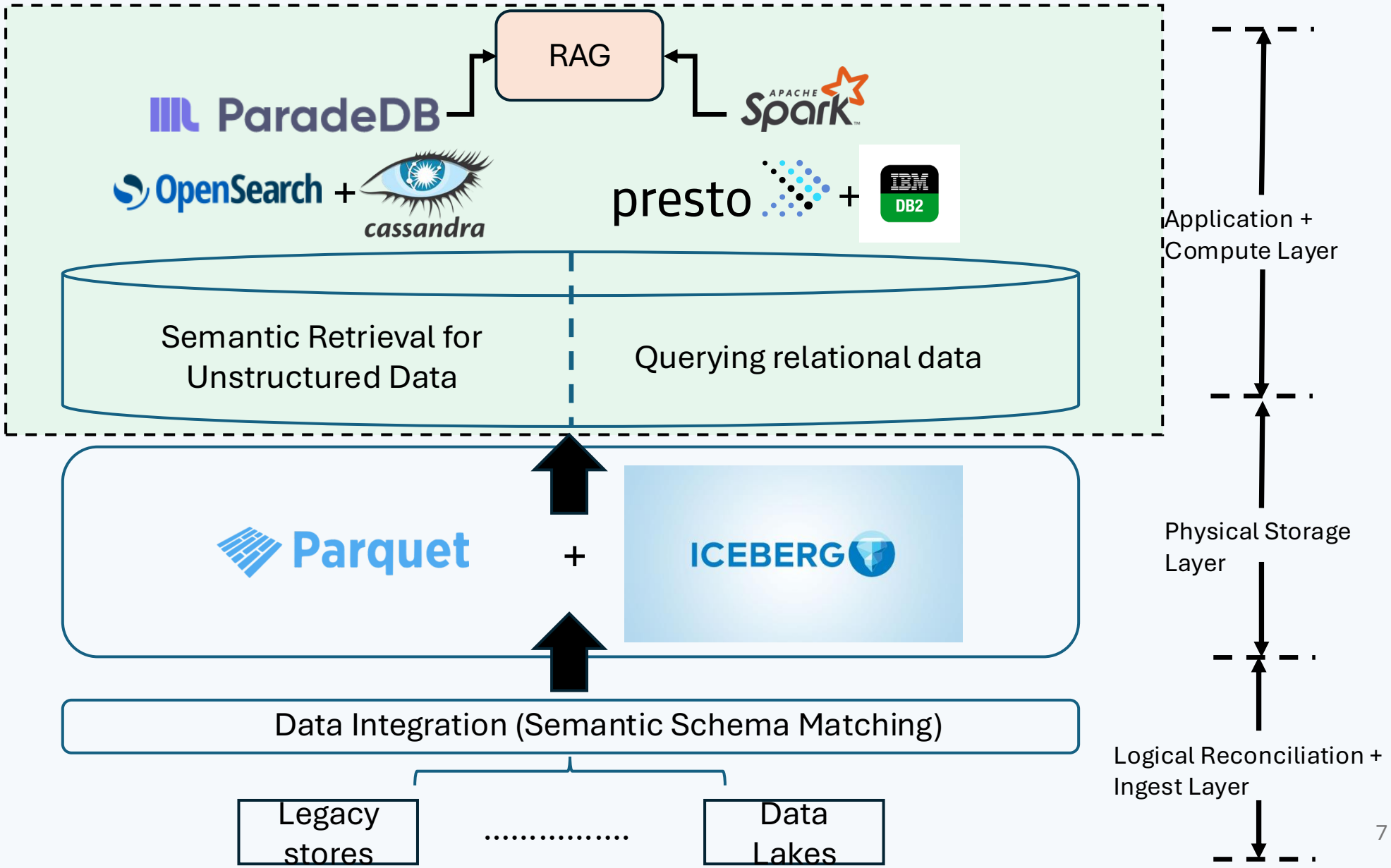
Challenges: Need to think beyond Indexes in Lakehouses

- **C1**: Low question answering accuracy of Vector Search Indexes
  - On both unstructured and structured data
  - even with high recall against exhaustive search on vectors
- **C2:** Complex queries containing both similarity search and relational predicates
  - Knowledge of schema is required to evaluate join predicates.
  - Join paths may be unknown in advance or may not be supported (Cassandra)
- **C3:** How do we accelerate evaluation of Relational Predicates?
  - Relational Indexes are cumbersome to maintain for PBs of data in lakehouses.
  - Optimal Physical design is key!
- **C4**: Data integration challenges are beyond the scope of indexes.
  - Blocking is the closest to indexing in an abstract sense.

Outline

- Semantic Search Evaluation
  - Unstructured data (BEIR)
  - Relational Data (TPC-DS + UNSPSC)
- Schema-GraphRAG (ICDE'26 Demo)
- Physical Layout Optimization (SIGMOD'26)
- Semantic Schema Matching (VLDB-J 2024)

C1: Retrieval on Unstructured and Structured Data



BEIR Datasets for Unstructured Data Retrieval

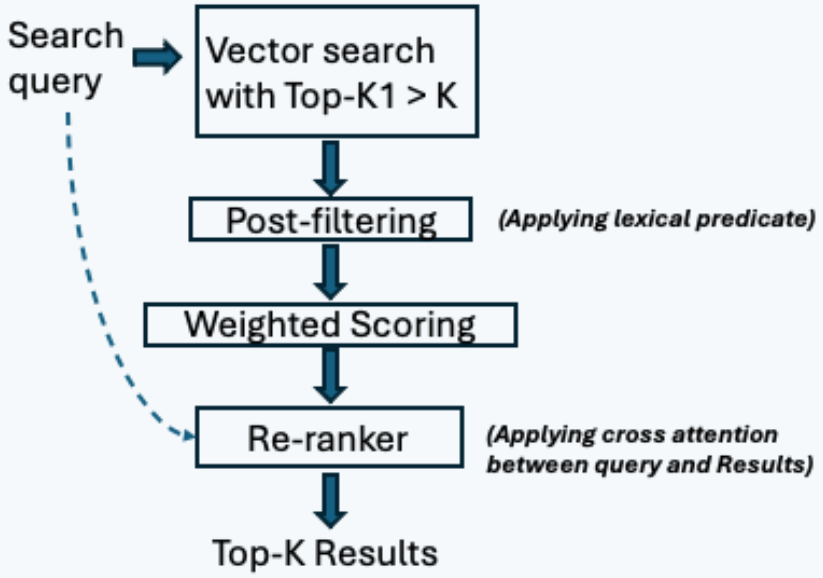
- https://github.com/beir-cellar/beir

- Text documents
- Chunking & Recall@100 are
- consistent with the BEIR NeurIPS 2021 paper https://arxiv.org/pdf/2104.08663

BEIR Dataset statistics

Name	#Docs = #Chunks	#Queries	Min #Chunks /Query in Ground Truth	Max #Chunks /Query in Ground Truth	Avg #Chunks /Query in Ground Truth
MSMarco	8,841,823	6,980	1	4	1.07
Climate-Fever	5,416,593	1,535	1	5	3.05
Fever	5,416,568	6,666	1	15	1.19
HotpotQA	5,233,329	7,405	2	2	2
NQ	2,681,468	3,452	1	4	1.22
TREC-COVID	171,332	50	631	1,941	1,326.72
FIQA	57,600	648	1	15	2.63

Postfiltered Search: DiskANN in ParadeDB + NVIDIA



- llama-32-nv-embedqa-1b-v2 model (2048 dims)
- llama-32-nv-rerankqa-1b-v2 as the optional re-ranker
- Cosine similarity-based score computation
- We move from hybrid search to
  - Postfiltered search
  - Without standing up two indexes
- Declarative querying

Postgres table with a DiskANN Index

```
app=# \d local_collection_fiqa;
                                    Table "public.local_collection_fiqa"
      Column       |       Type        | Collation | Nullable |                      Default
-------------------+-------------------+-----------+----------+-----------------------------------------------------
 pk                | bigint            |           | not null | nextval('local_collection_fiqa_pk_seq'::regclass)
 fileid            | character varying |           |          |
 text              | character varying |           |          |
 vector            | vector(2048)      |           |          |
 source            | jsonb             |           |          |
 content_metadata  | jsonb             |           |          |
Indexes:
    "local_collection_fiqa_pkey" PRIMARY KEY, btree (pk)
    "local_collection_fiqa_file_idx" btree (fileid)
    "local_collection_fiqa_vector_idx" diskann (vector)
```

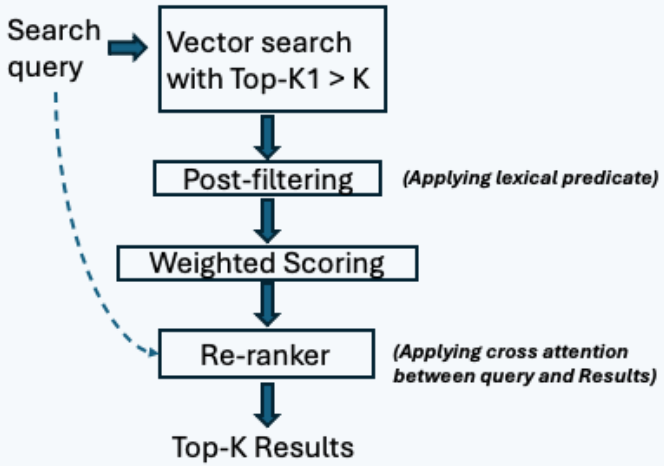
Semantic search as a SQL query

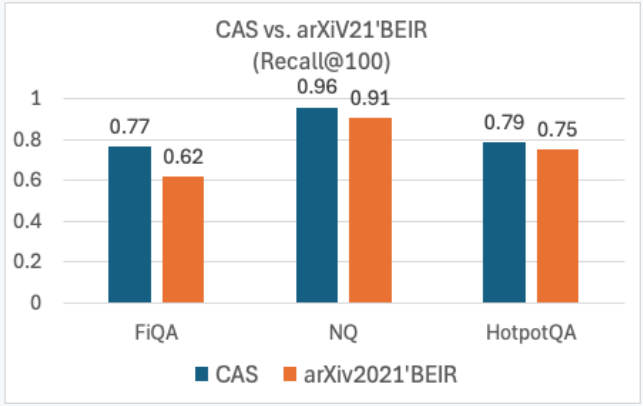
```
sql_semantic = sql.SQL("""
SELECT {}, vector<=> %s::vector as score
FROM {}
{}
ORDER BY score
LIMIT {}
""").format(
    sql.SQL(", ").join(map(sql.Identifier, columns[:-1])),
    sql.Identifier(collectionname),
    sql.SQL(""),  # can replace with output of build_predicate function when ready
    sql.Literal(top_n),
)

results = cur.execute(sql_semantic, [embedding,]).fetchall()
```

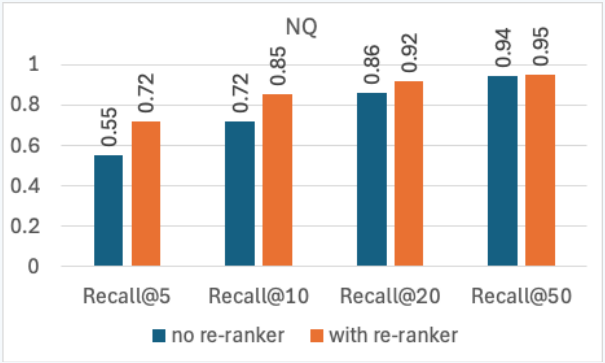
Query embedding

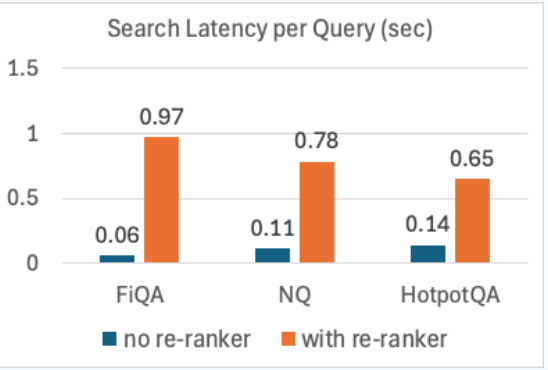
Subramanya et al: **DiskANN: Fast Accurate Billion-point Nearest Neighbor Search on a Single Node.** NeurIPS (2016)

Postfiltered Search: DiskANN in ParadeDB + NVIDIA









Summary

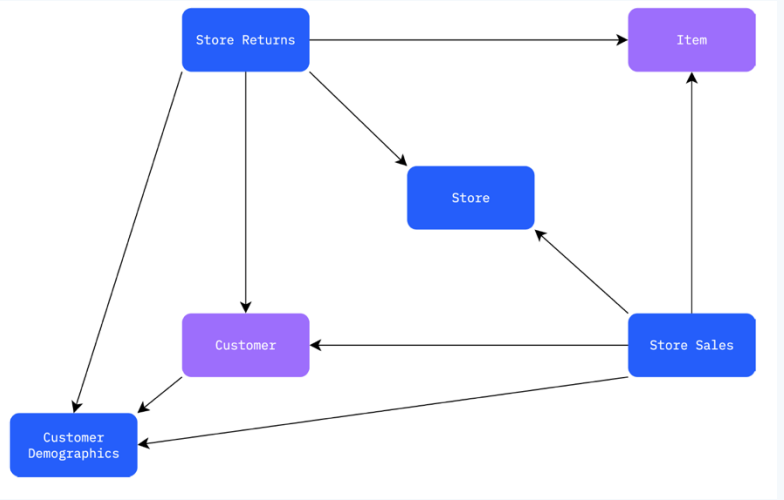
- Did not over-tune vector index to get 99% recall against exhaustive search.
  - separate Lexical Index was not stood up → simpler architecture
- Up to 24% question-answering (QA) recall improvement
  - compared to ColBERT and GenQ (transfer learning) BEIR baselines
- NVIDIA embeddings improved QA recall from
  - 68% to 77% (FIQA)
  - 93% to 96% (NQ)
  - 78% to 79% (HotpotQA)
- Re-ranker improves QA recall by up to 25% but is 16x slower, requires GPUs

Blog post: https://community.ibm.com/community/user/blogs/venkata-vamsikrishna-meduri/2026/02/08/evaluating-qa-accuracy-of-ibm-fusion-cas

TPC-DS + UNSPSC Dataset for Structured Data Retrieval

- A subgraph from the TPC-DS SF 10K schema
  - Augmented with knowledge from external ontology (UNSPSC)
- Relational ground truth creation
  - Exact matches
    - expected to appear in the top-k results
  - Sanity check
    - to verify if the embeddings capture the key:value notion
- Cassandra + DiskANN (Jvector)
  - from DataStax

Schema Graph



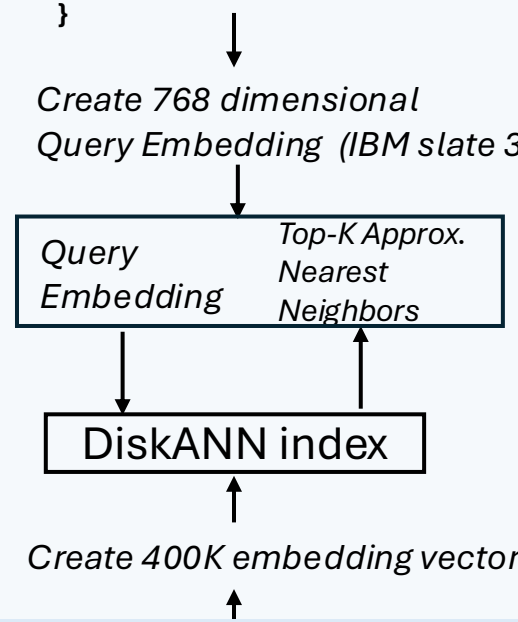
Sample Row in JSON


```
{
"ITEM":
      {
      "I_REC_START_DATE":"1999-10-28",
      "I_REC_END_DATE":"2001-10-26",
      "I_CURRENT_PRICE":0.43,
      "CURRENT_PRICE":"very_low",
      "I_WHOLESALE_COST":0.51,
      "WHOLESALE_COST":"low",
      "I_CLASS":"classical",
      "I_CATEGORY":"Music",
      "I_COLOR":"salmon",
      "I_UNITS":"Gram",
      "I_CONTAINER":"Unknown",
      "UNSPSC":{
            "COMMODITY_NAME":"Musicians services",
            "CLASS_NAME":"Performing arts professionals",
            "FAMILY_NAME":"Professional artists and performers",
            "SEGMENT_NAME":"Editorial and Design and Graphic and Fine Art Services"
            }
      }
}
```

Evaluation: Hybrid Search on Structured Data

Similarity search query

Show me all items which are similar to
{
"I_CLASS": "ATHLETIC SHOES",
"I_CATEGORY": "SPORTS"
}



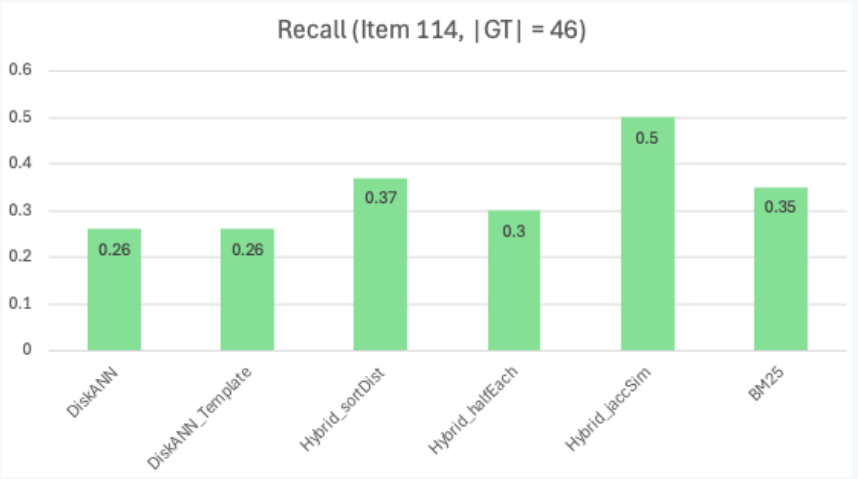
Source item JSONs are enriched with product information from United Nations Standard Products and Services Code (UNSPSC) ontology.


```
{"ITEM":
    {
        "I_ITEM_SK":85,
        "I_ITEM_ID":"AAAAAAAAFFAAAAA",
        "I_CURRENT_PRICE":5.45,
        "CURRENT_PRICE":"medium",
        "I_WHOLESALE_COST":4.74,
        "WHOLESALE_COST":"high",
        "I_BRAND_ID":10002001,
        "I_CLASS":"camcorders",
        "I_CATEGORY":"Electronics",
        "I_COLOR":"moccasin",
        "I_UNITS":"Pound",
        "UNSPSC":
        {
           "COMMODITY_NAME":"Digital camcorders or video cameras",
           "CLASS_NAME":"Cameras",
           "FAMILY_NAME":"Photographic or filming or video equipment",
           "SEGMENT_NAME":"Printing and Photographic and Audio and Visual Equipment and Supplies"
        }
    }
}
```


400K Source item entities in JSON format

QUERY

```
{
    "ITEM": {
        "I_CLASS": "athletic shoes",
        "I_CATEGORY": "Sports",
        "I_COLOR": "turquoise",
        "UNSPSC": {
            "COMMODITY_NAME": "Mens athletic footwear",
            "CLASS_NAME": "Athletic footwear",
        }
    }
}
```



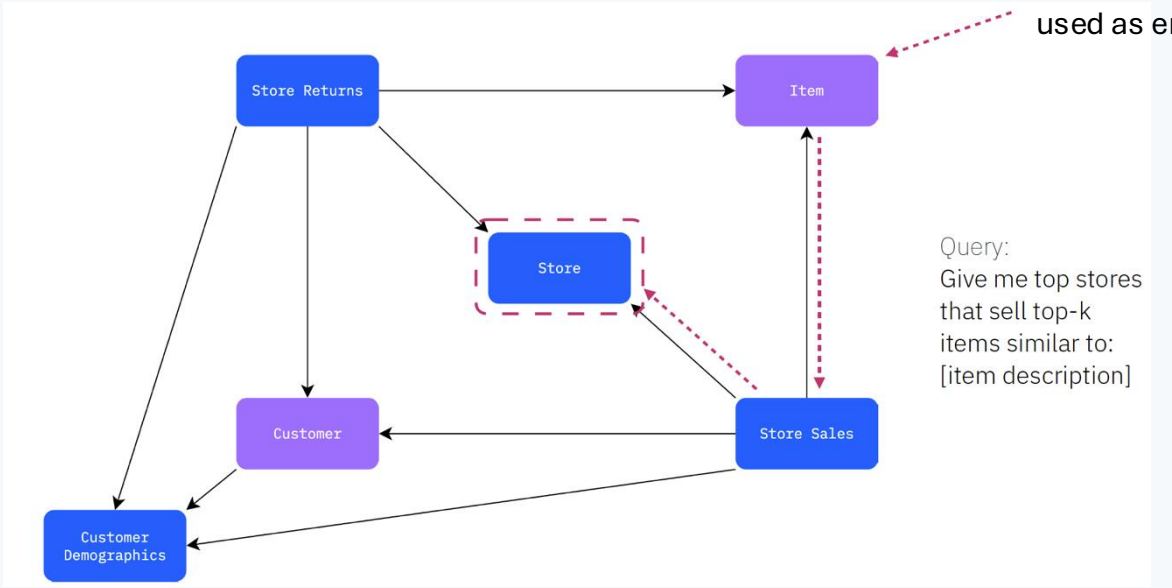
- Tried multiple fusion strategies.
- Core issue was not ranking – it was embedding semantics.

Summary

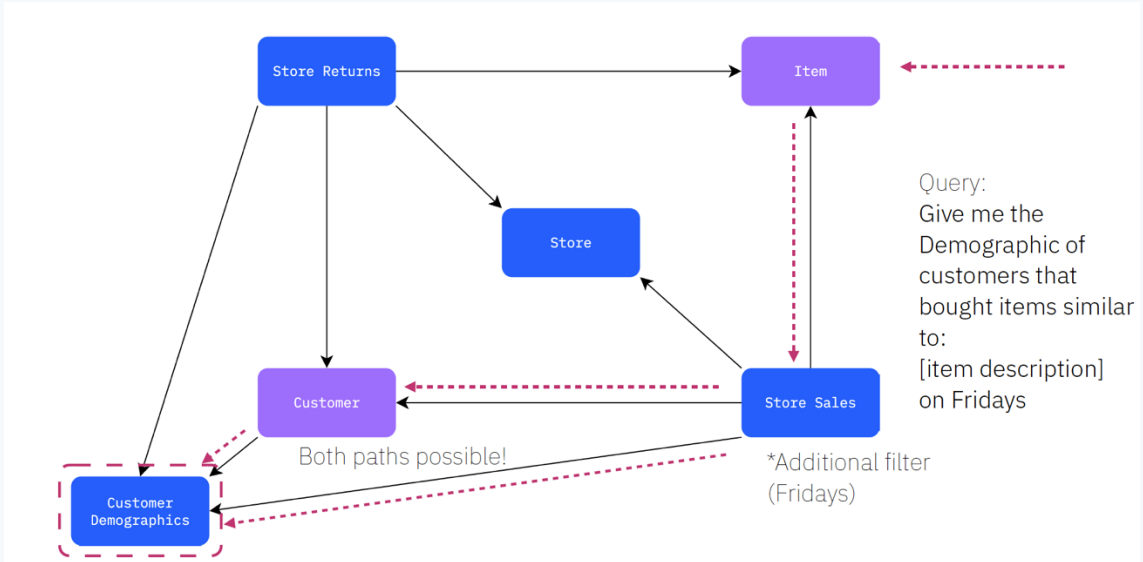
- Poor recall against relational ground truth establishes
  - the need for ***specialized*** relational data embeddings
  - RelGNN, RelGT, RelBench [Jure Leskovec], TURL (VLDB'21), Kv2Vec
- False Positives (FP) & False Negatives (FN) occur because
  - Embeddings being generated from slate model
    - treat JSON (semi-structured data) as NL sentences or documents.
    - Fields appearing early on in a source JSON carry more implicit weightage.
    - The distance computed varies depending on
      - where the queried field "I_CLASS: Shoes" appears in the JSON

C2: Complex queries containing relational predicates

Scenario 1



Scenario 2



- Similar items act as seeds but require schema graph traversal
- Join Paths may not be known in advance
- Native joins are not supported in Cassandra query language (CQL)
  - *Solution*: Use Gremlin for graph traversal

Schema-GraphRAG for Complex Retrieval [ICDE'26 Demo]

Demo Scenario 1:
Hybrid search + Deterministic Schema Graph Traversal

```
lexical_input = "Children economy royal"
similarity_input = """{
    "ITEM":{
        "I_CLASS":"toddlers",
        "I_CATEGORY":"Children",
        "I_COLOR":"royal"
    }
}"""

graph_query = """
              g.V().has('i_item_sk',{}).
              in('store_sales_to_item').
              out('store_sales_to_store')
              """

demo_two_results = schema_graphrag_demo(
    lexical_input=lexical_input,
    similarity_input=similarity_input,
    graph_query=graph_query
)

display(demo_two_results)
```

[5] ✓ 2.7s

	NodeType	Id
0	Store	8
1	Store	2
2	Store	4

Join Path is known

Demo Scenario 2:
Hybrid search + Flexible Schema-aware Traversal

```
lexical_input = "Children economy royal"
similarity_input = """{
    "ITEM":{
        "I_CLASS":"toddlers",
        "I_CATEGORY":"Children",
        "I_COLOR":"royal"
    }
}"""

graph_query = """
              g.V().has('i_item_sk',{}).
              repeat(both().simplePath()).
              until(hasLabel('store'))
              """

demo_three_results = schema_graphrag_demo(
    lexical_input=lexical_input,
    similarity_input=similarity_input,
    graph_query=graph_query
)

display(demo_three_results)
```

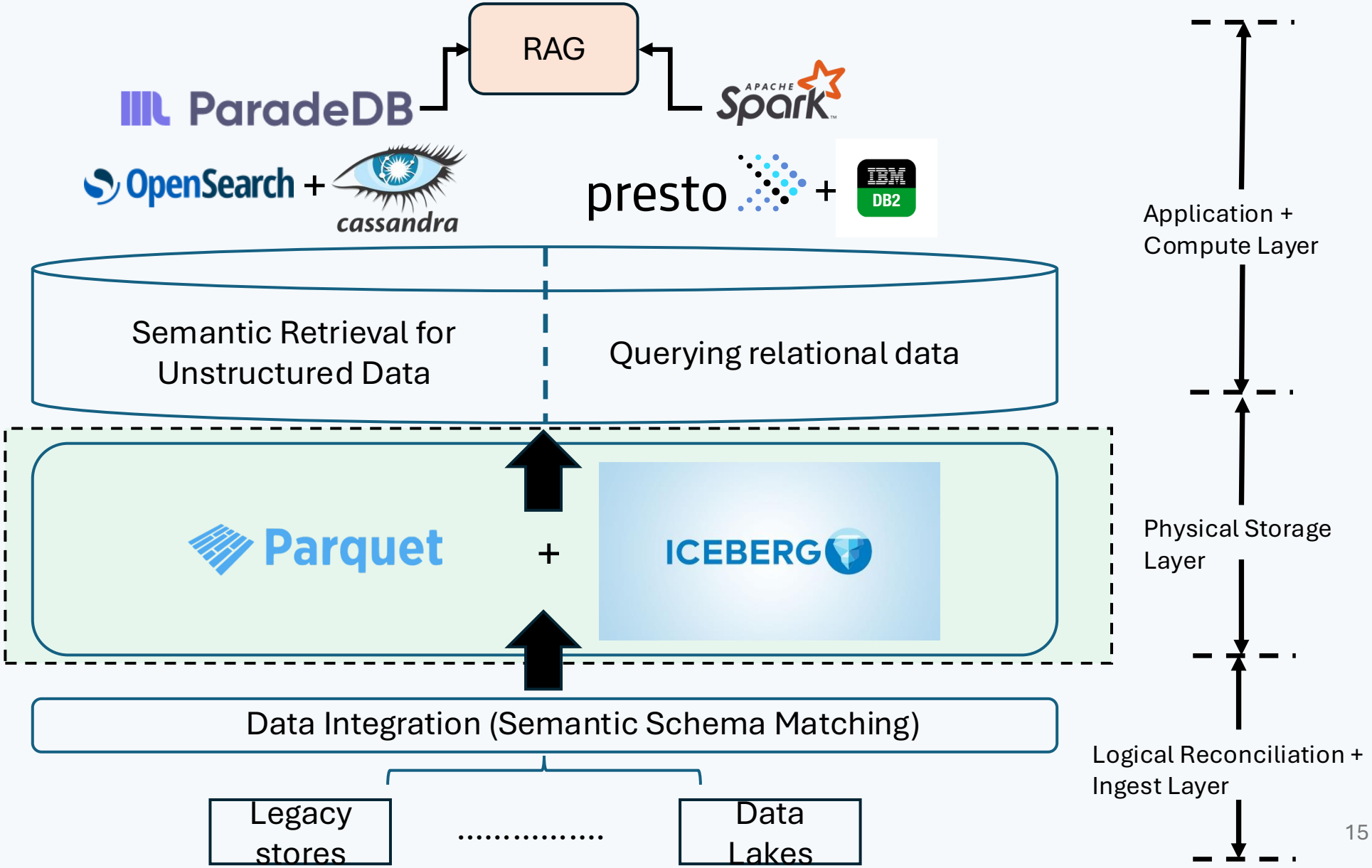
[6] ✓ 2.3s

	NodeType	Id
0	Store	8
1	Store	2
2	Store	4

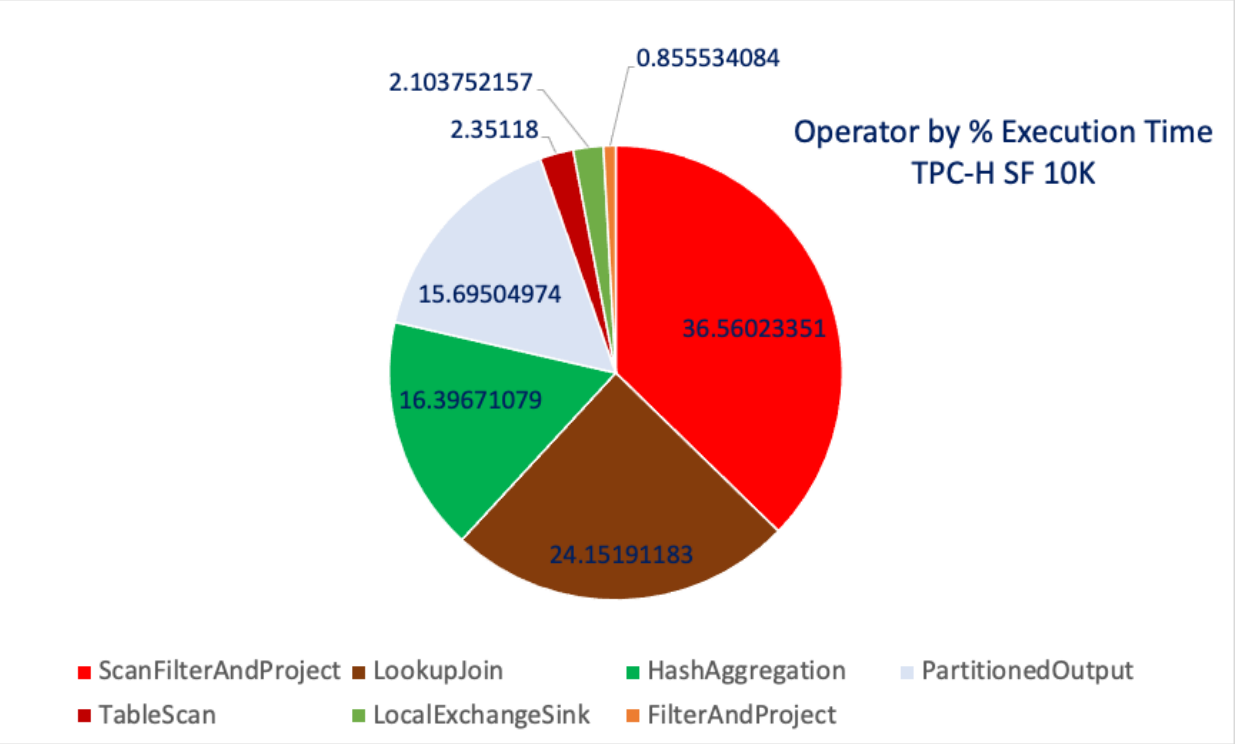
Join Path is unknown

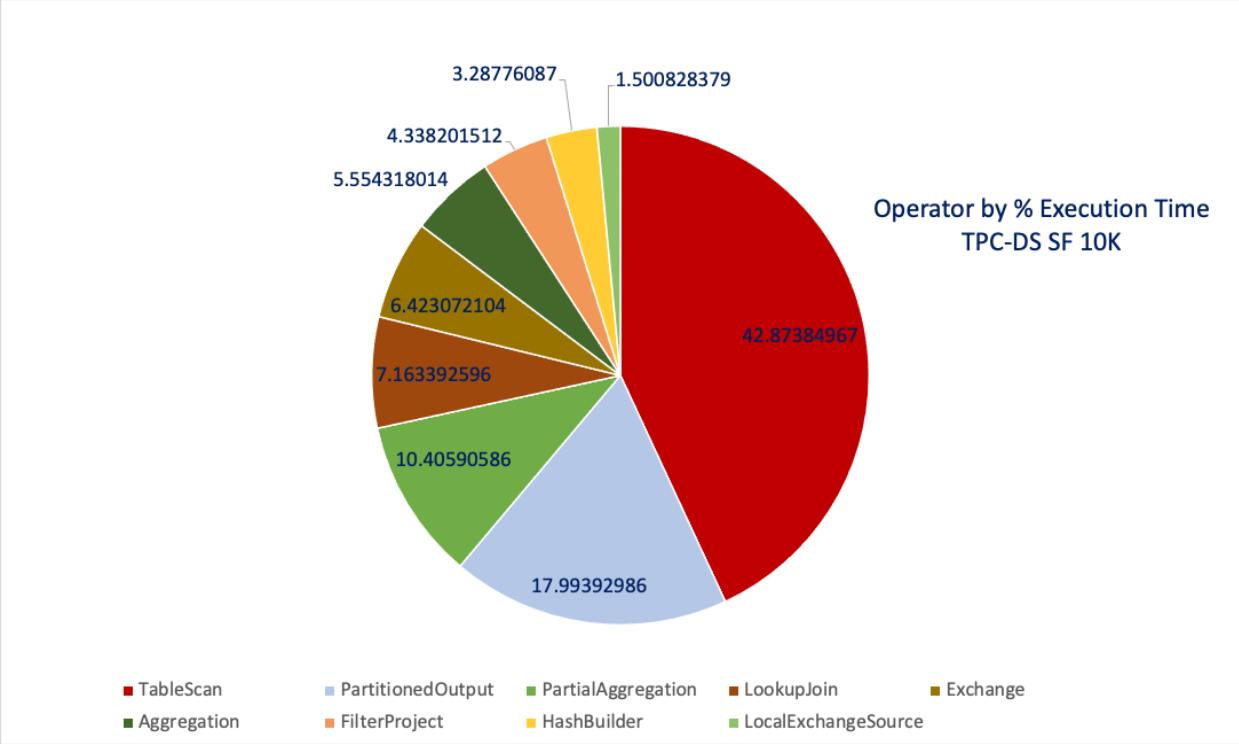
Bastian Lipka, Vamsi Meduri, Berthold Reinwald, Nasrullah Sheikh: **Schema-GraphRAG: Bridging Hybrid Search and Graph Traversal for Complex Retrieval Tasks.** ICDE Demo (2026)

C3: Fast Evaluation of Relational Predicates (without indexes)



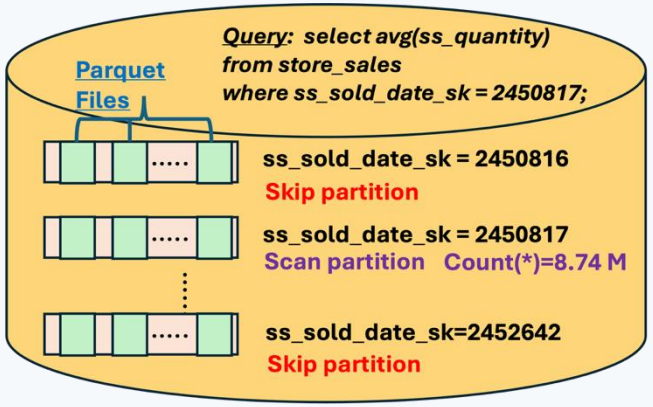
Scans are the most expensive operators!



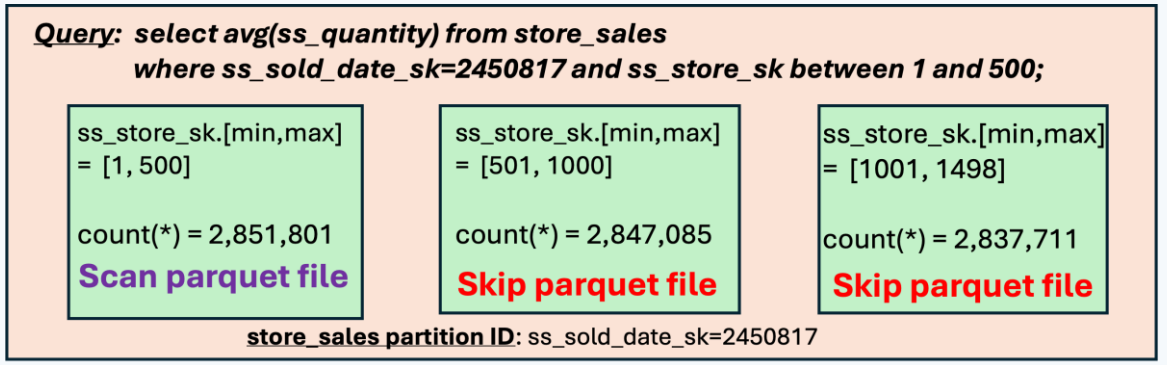


Impact of Iceberg Table Layout Optimization

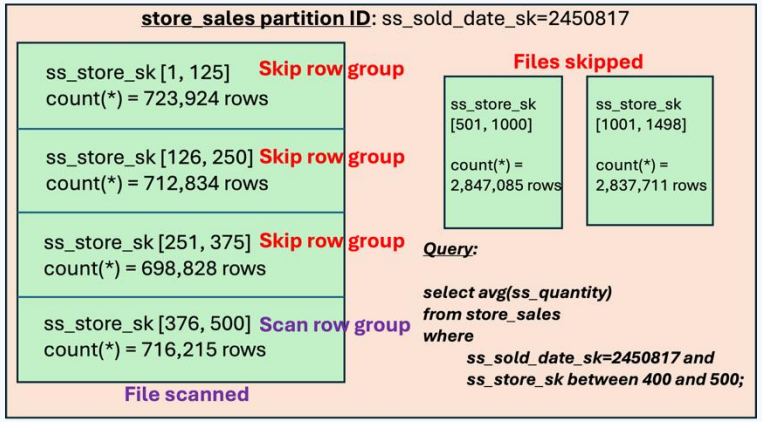
1. Partition Skipping



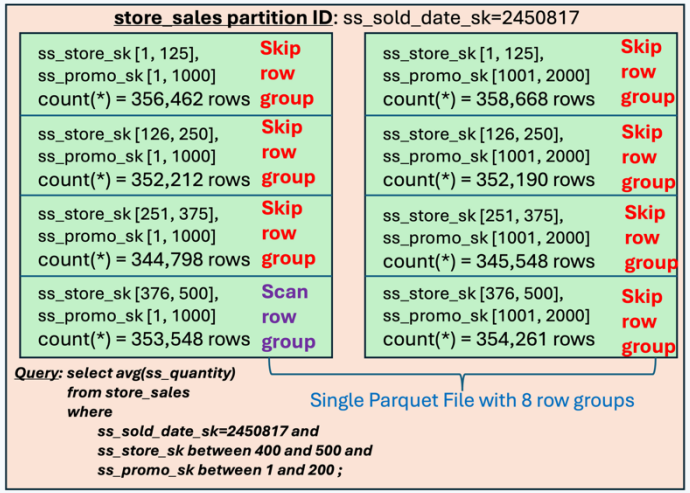
2. File Skipping



3. Row group Skipping



4. Sorting



Row group skipping on z-ordered store_sales

- Setting each of these **interdependent** parameters to their optimal values maximizes the skipping benefits

PTO: Replacing Relational Indexes with Predictive Layouts

Given

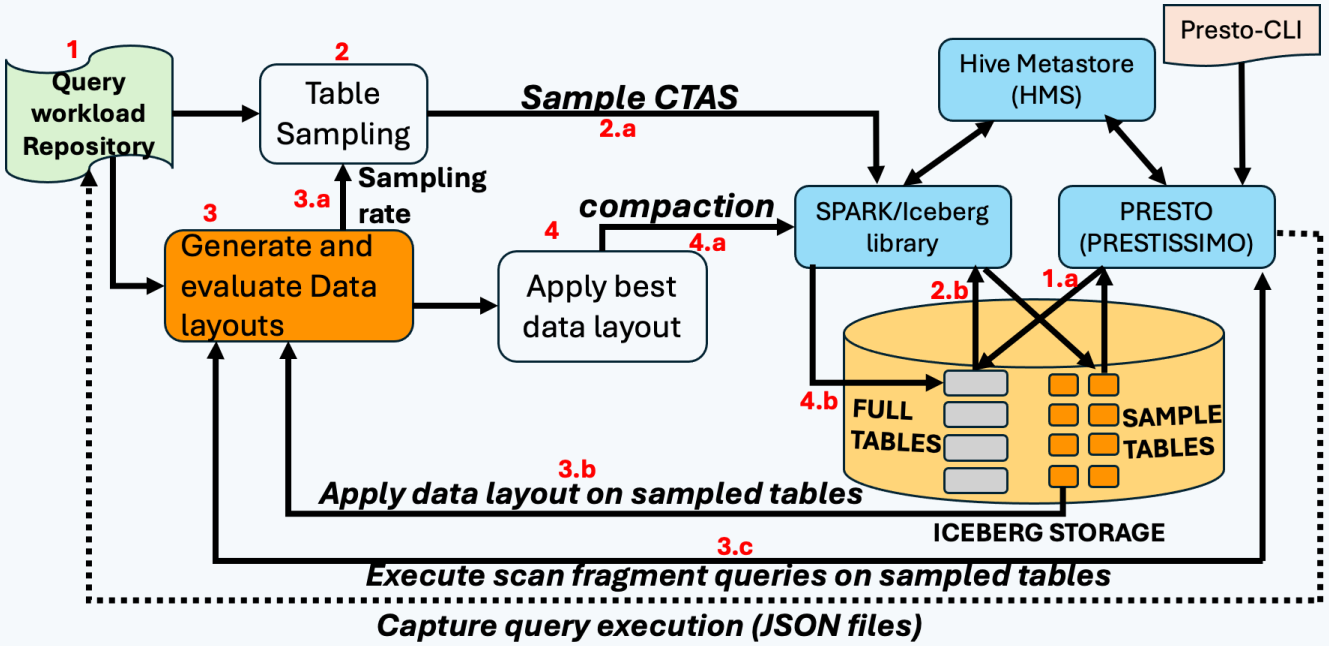
- a query workload,
- a set of tables to be optimized,
- a lakehouse engine,

our predictive table optimizer (PTO) determines the optimal layout parameters

- <partitioning column P(t),
- sort order S(t),
- target file size TFS(t),
- row group size RGS(t)>

such that the skipping benefits are maximized.

System Architecture of PTO [SIGMOD'26]

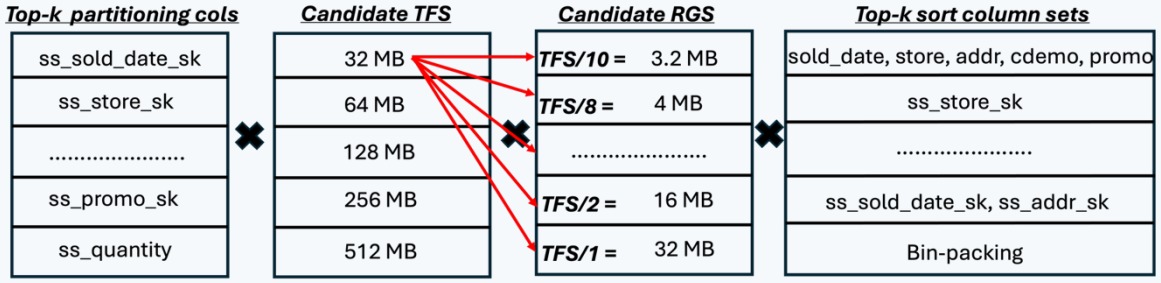


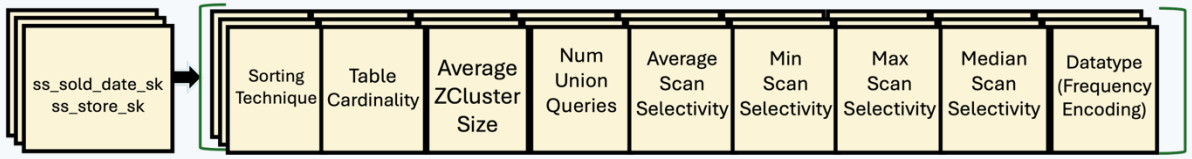
- Generation of top-k candidate layouts based on
  - filtering capability/selectivity
  - #queries benefitted
  - column cardinality
  - expected sort cluster sizes.
- Evaluation of candidate data layouts
  - Represents data layouts as numerical feature vectors.
  - Target labels are skipping benefits computed on sample tables.
  - Trains an ML model **gradient boosting trees (GBT)** on the labeled data layouts

Vamsi Meduri, David Kreismann, Ron Barber, Berthold Reinwald: **PTO: A Workload-driven Predictive Table Optimizer for Lakehouse Systems.** ACM SIGMOD (2026)

Labeling candidate data layouts

- Evaluating shortlisted layouts on sampled data still takes several days.
- We train a **predictive model (GBT)** on 3% of the candidate space.
- Generation of training data
  - Feature vector creation
  - Labeling the feature vectors with their skipping capability on sampled dataset as the target label.
    - We run scan queries on sampled data to measure the skipping benefits.
    - %Skipping benefit = $\left(1.0 - \frac{scanFragment.inputRows}{scanFragment.tableCardinality}\right) \times 100$
- We evaluate the ML model on the 96-97% of the candidate space
- Best layouts which are applied on the full table via compaction.





Feature vector for sort columns

Target Label to predict = % skipping capability

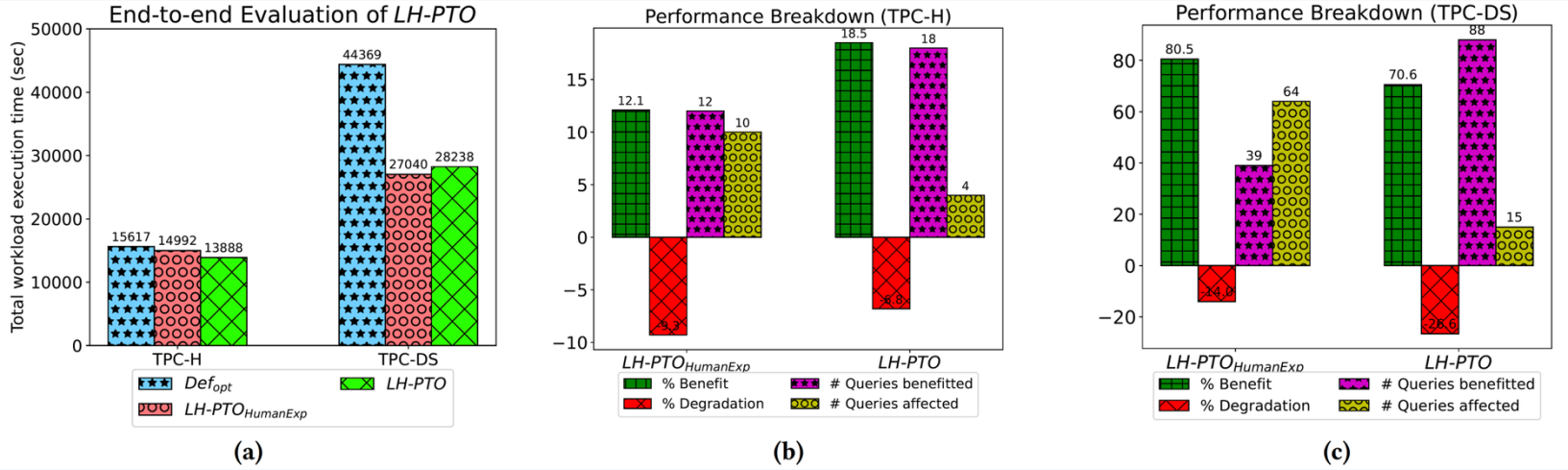
Results – Table layouts detected

Dataset	Table	Total Size	Def_{opt} Partitioning Column	Def_{opt} TFS (MB)	Def_{opt} RGS (MB)	Def_{opt} Sort Scheme	LH-PTO$_{HumanExp}$ Partitioning Column	LH-PTO$_{HumanExp}$ TFS (MB)	LH-PTO$_{HumanExp}$ RGS (MB)	LH-PTO$_{HumanExp}$ Sort Scheme	LH-PTO Partitioning Column	LH-PTO TFS (MB)	LH-PTO RGS (MB)	LH-PTO Sort Scheme
TPC-H (SF 10K)	customer	74.5 GB	c_mktsegment	150	75	binpack	c_nationkey	305	51	(c_mktsegment,c_acctbal)	c_nationkey	256	64	(c_mktsegment,c_acctbal)
	lineitem	1.6 TB	l_shipdate	343	114	binpack	l_commitdate	165	28	(l_shipdate,l_returnflag, l_shipmode,l_shipinstruct, l_discount)	l_shipdate	512	128	z-order(l_returnflag, l_shipmode,l_shipinstruct)
	nation	2.4 KB	-	32	8	binpack	-	32	8	n_name	-	32	8	n_name
	orders	412.4 GB	o_orderdate	14	14	binpack	o_orderdate	42	11	o_orderstatus	o_orderdate	512	128	o_orderstatus
	part	41 GB	p_brand	97	97	binpack	p_size	84	21	z-order(p_brand,p_type, p_container)	p_size	512	128	z-order(p_brand,p_type, p_container)
	partsupp	251 GB	-	29	29	binpack	-	512	128	binpack	-	512	128	binpack
	region	1.3 KB	-	32	8	binpack	-	32	8	r_name	-	32	8	r_name
	supplier	4.8 GB	-	43	43	binpack	s_nationkey	32	8	binpack	s_nationkey	512	128	binpack
TPC-DS (SF 10K)	catalog_returns	83.8 GB	-	97	97	binpack	cr_returned_date_sk	32	8	binpack	cr_returned_date_sk	128	32	(cr_item_sk,cr_return_amount)
	catalog_sales	763 GB	cs_sold_date_sk	128	128	binpack	cs_sold_date_sk	86	22	z-order(cs_bill_cdemo_sk, cs_bill_addr_sk, cs_sold_time_sk, cs_bill_customer_sk)	cs_sold_date_sk	512	128	z-order(cs_bill_cdemo_sk, cs_bill_addr_sk, cs_sold_time_sk, cs_bill_customer_sk)
	inventory	5.9 GB	-	128	128	binpack	inv_date_sk	32	4	binpack	inv_date_sk	512	128	inv_quantity_on_hand
	store_returns	111 GB	-	150	75	binpack	sr_returned_date_sk	32	8	(sr_reason_sk, sr_return_amt,sr_cdemo_sk)	sr_returned_date_sk	256	64	sr_reason_sk
	store_sales	893.8 GB	ss_sold_date_sk	323	108	binpack	ss_sold_date_sk	123	31	z-order(ss_store_sk, ss_addr_sk,ss_cdemo_sk, ss_promo_sk)	ss_sold_date_sk	512	128	ss_store_sk
	web_returns	39.2 GB	-	28	28	binpack	wr_returned_date_sk	32	8	wr_refunded_addr_sk	wr_returned_date_sk	512	128	wr_refunded_addr_sk
	web_sales	535.5 GB	ws_sold_date_sk	43	43	binpack	ws_sold_date_sk	37	9	z-order(ws_sold_time_sk, ws_bill_addr_sk, ws_ship_addr_sk, ws_ship_hdemo_sk)	ws_sold_date_sk	512	128	z-order(ws_sold_time_sk, ws_bill_addr_sk, ws_ship_addr_sk, ws_ship_hdemo_sk)

- Def_{opt}
  - Default (unoptimized) data layout
- $PTO_{HumanExp}$
  - Expert-driven gold standard ground truth
  - PTO can sometimes outperform $PTO_{HumanExp}$.

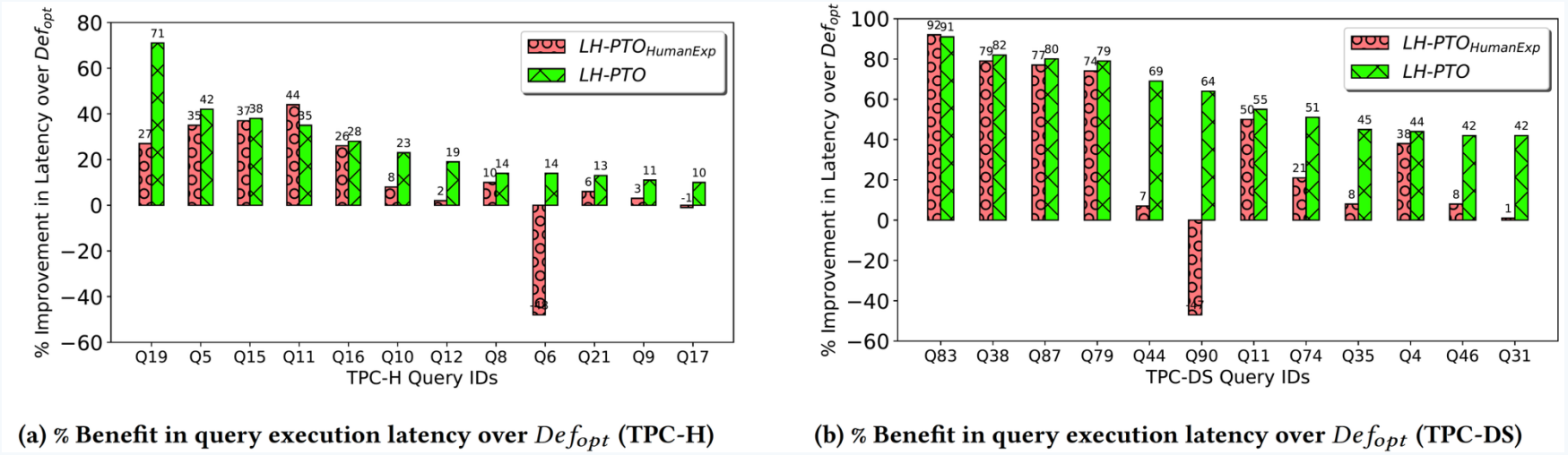
Results – Query workload execution latencies (TPC-H, TPC-DS SF 10K)

1. End-to-end evaluation



LH-PTO vs. LH-PTO$_{HumanExp}$ and Def_{opt} onTPC-H and TPC-DS workloads along with the performance breakdown.

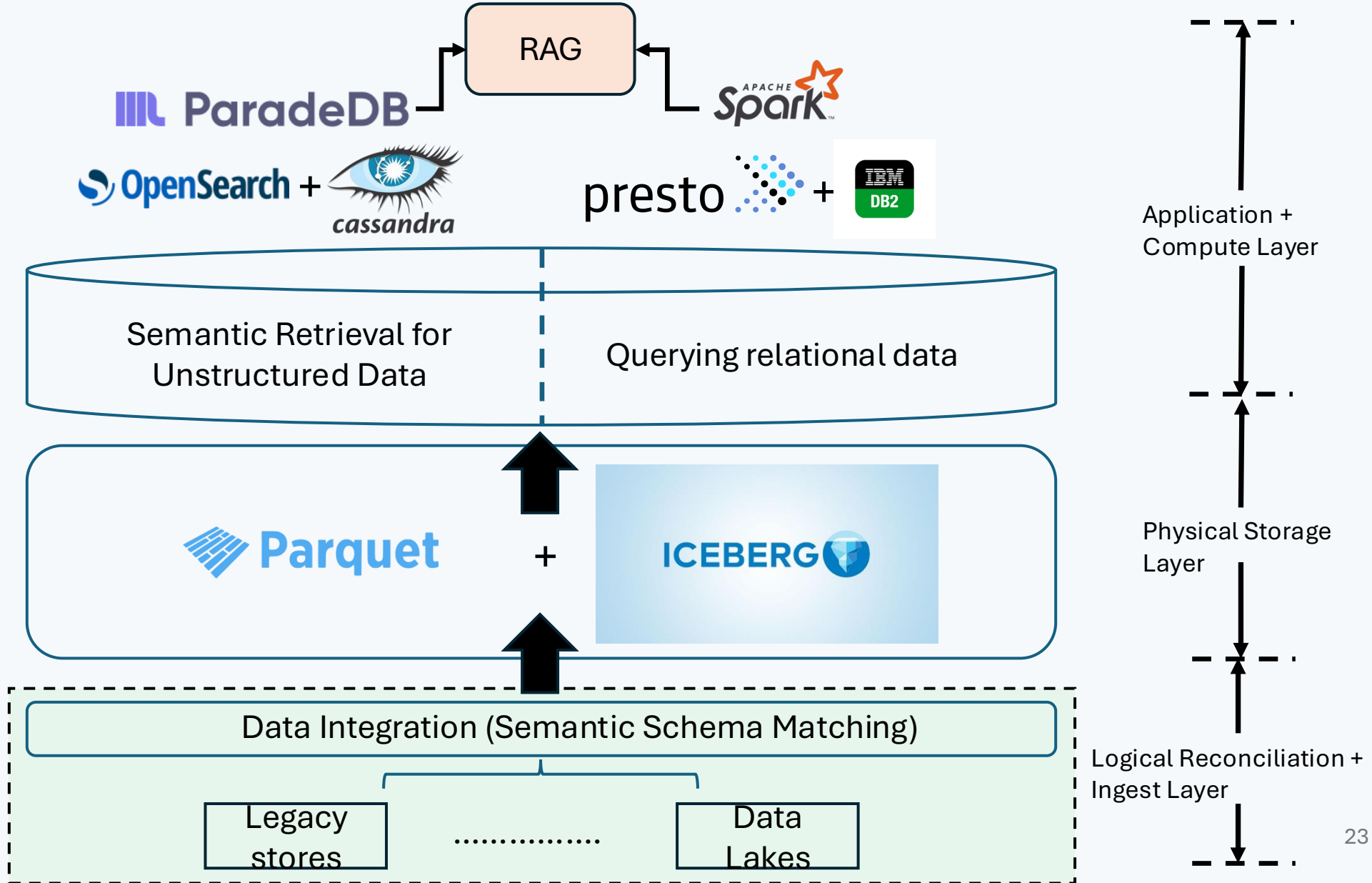
2. Evaluation on top-12 queries benefitted



LH-PTO vs. LH-PTO$_{HumanExp}$ on queries yielding highest % improvement in query execution latency over Def_{opt}.

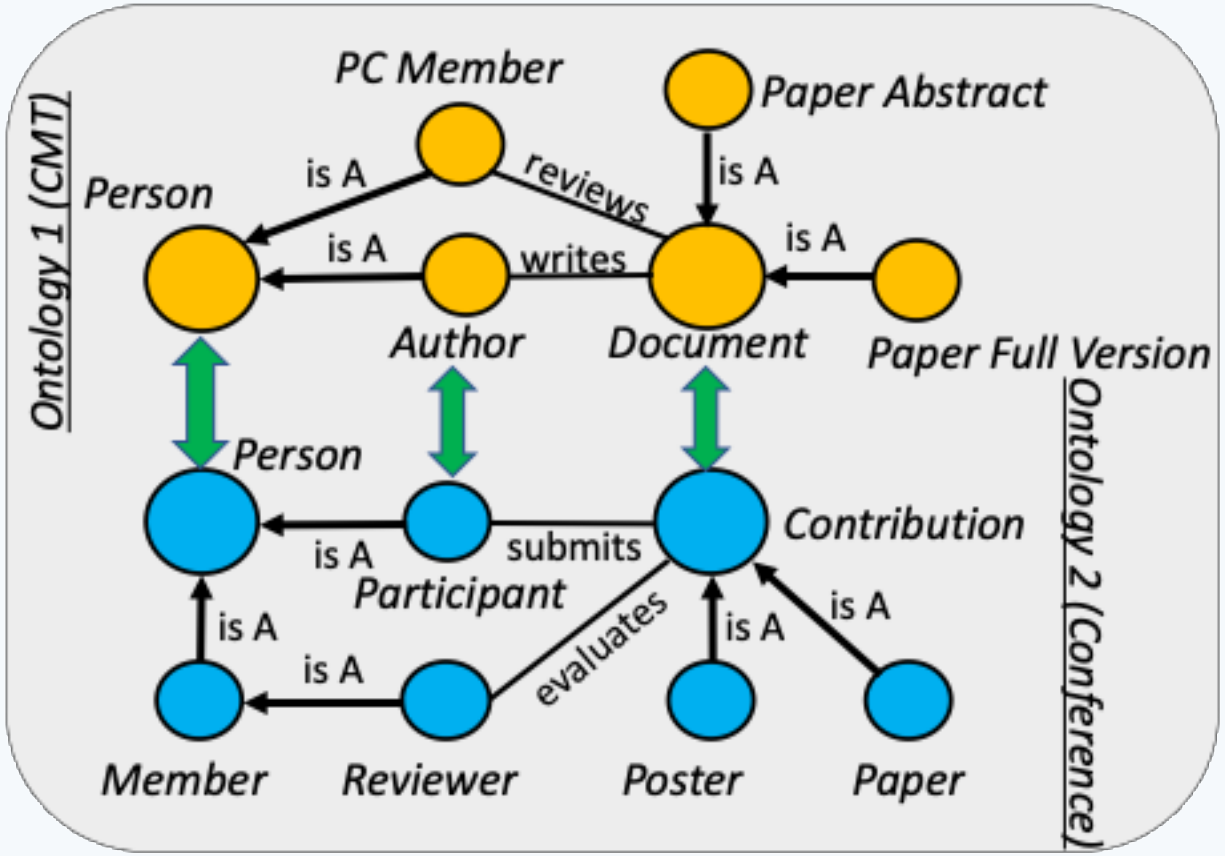
- PTO yields 11% average lower query execution latency on TPC-H and 36% on TPC-DS (SF 10K).
- Maximum benefit on TPC-H is 71% and on TPC-DS is 90% for Q19 and Q83 respectively.

C4: Enabling Consistent Semantics across Lakehouse Sources



Semantic Schema Matching – Enabling Consistent Semantics

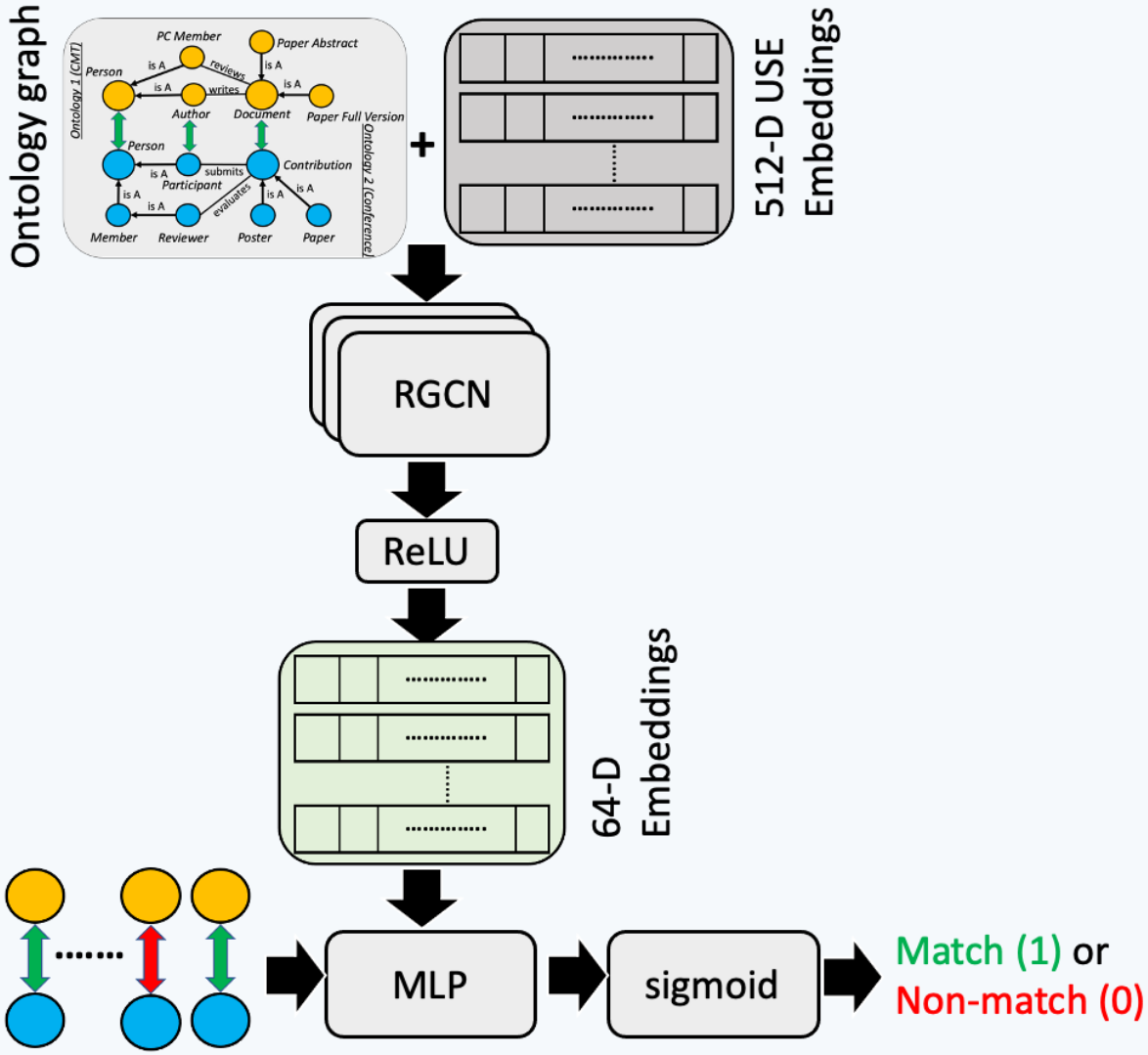
- Finds matching concept pairs
  - across schema graphs represented as ontologies
- Example matches
  - CMT.Person to Conf.Person
  - CMT.Author to Conf.Participant
  - CMT.Document to Conf.Contribution
- Critical step for downstream applications
  - Data Integration
  - Knowledge Graph Construction etc.



Joint Graph from an example ontology pair (CMT-Conf)

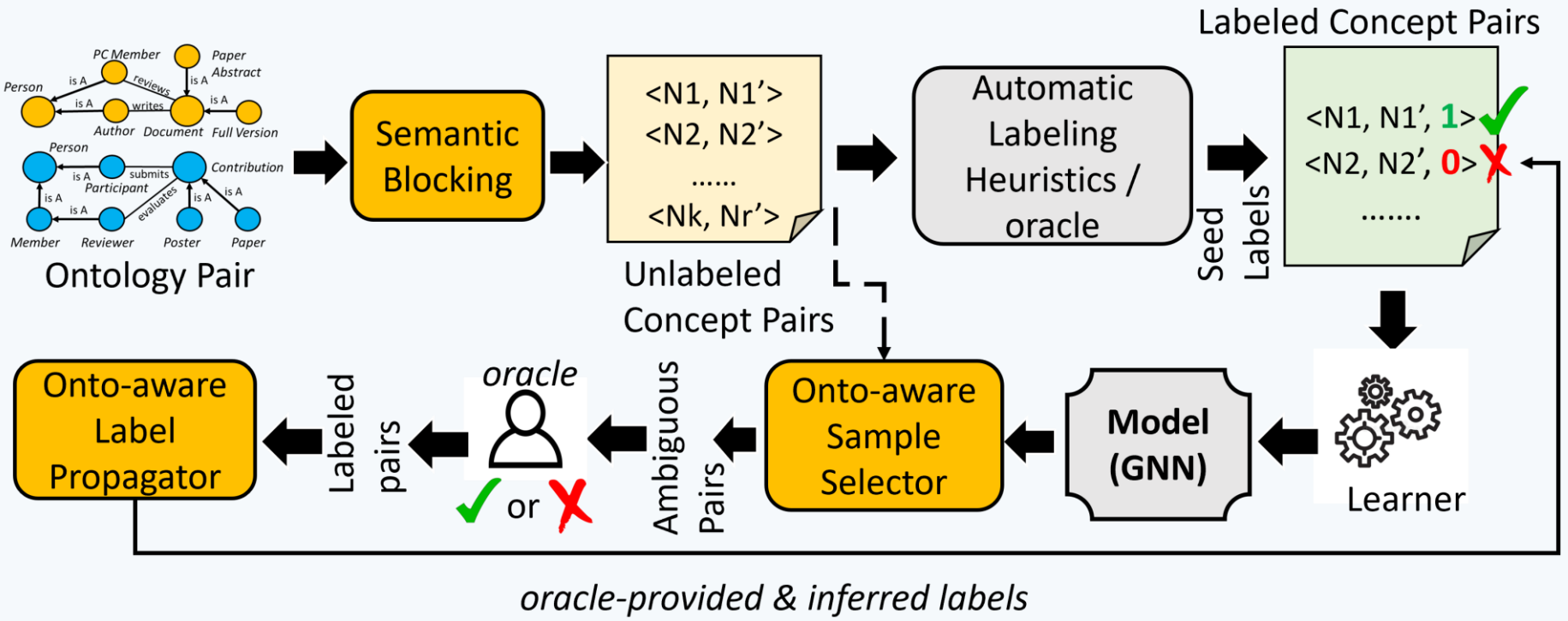
Need for Active Learning for Ontology Matching

- Currently done using Supervised Learning
  - RGCNs require several labels (OntoGNN*)
  - AL to alleviate the pre-labeling need
- Requires several pre-labeled concept pairs
  - Used to train ML models
- Pre-generation of training labels is expensive
  - Typically requires manual curation
    - Automatic labeling heuristics to pre-generate training labels may be weak
  - Need for diverse labeled patterns



[1] Hao et al. MEDTO: Medical Data to Ontology Matching using Hybrid Graph Neural Networks (KDD 2021)

System Architecture of ALFA [VLDB Journal '24]

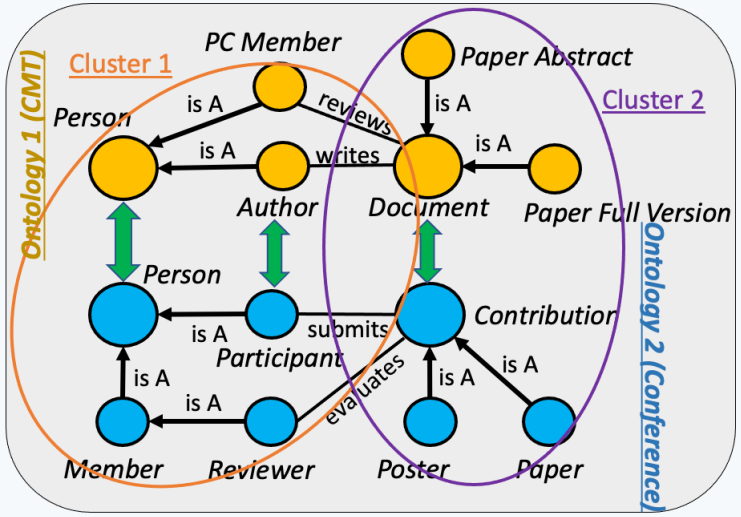


- Existing AL solutions are model-dependent but agnostic to ontology structure
- ALFA is both ontology-aware and model-aware
  - All the building blocks are based on ontology clustering
  - K-Means clustering is applied on model-generated concept embeddings

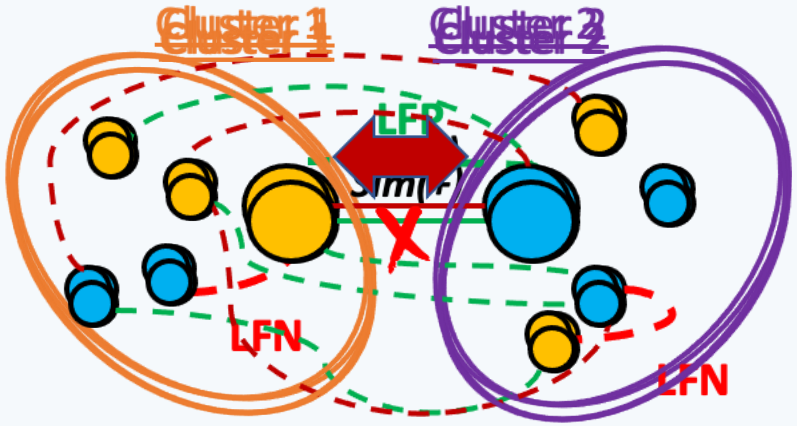
Vamsi Meduri, Abdul Quamar, Chuan Lei, Xiao Qin, Berthold Reinwald: **ALFA: active learning for GNN-based Semantic Schema Alignment.** VLDB Journal (2024)

Ontology-aware Components

- Sample selection
  - Positively classified pairs belonging to different clusters
  - Negatively classified pairs belonging to same cluster
- Label propagation
  - Cross-cluster pairs similar to labeled pairs
  - Pairs of Embedding similarity
    - >= sim(+) are labeled as matching (+)
    - <= sim(-) are labeled as non-matching (-)
- Blocking
  - Eliminate Cross-cluster pairs
    - labeled as mis-matching
  - Trade-off between False Negatives and skew
  - One-time pre-processing step
    - "#clusters" determined based on target # post-blocking pairs



Joint Graph from an example ontology pair (CMT-Conf)



LFP/LFN concepts borrowed from Kun Qian, Lucian Popa, Prithviraj Sen: **Active learning for Large-Scale Entity Resolution.** CIKM (2017)

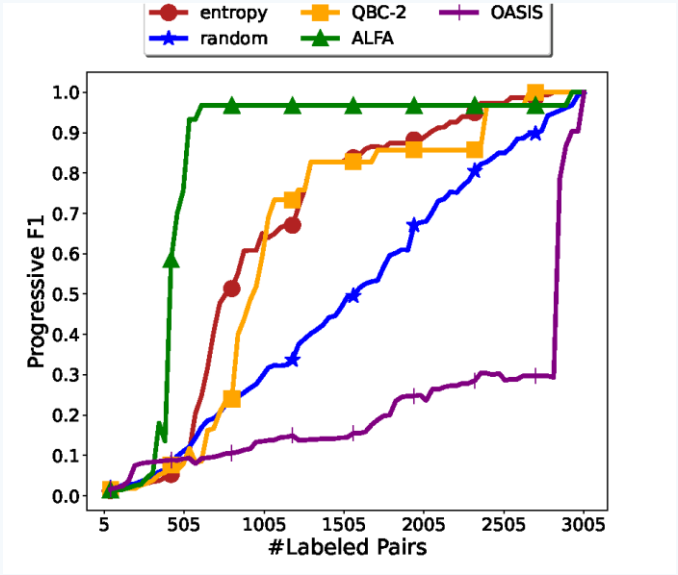
Datasets

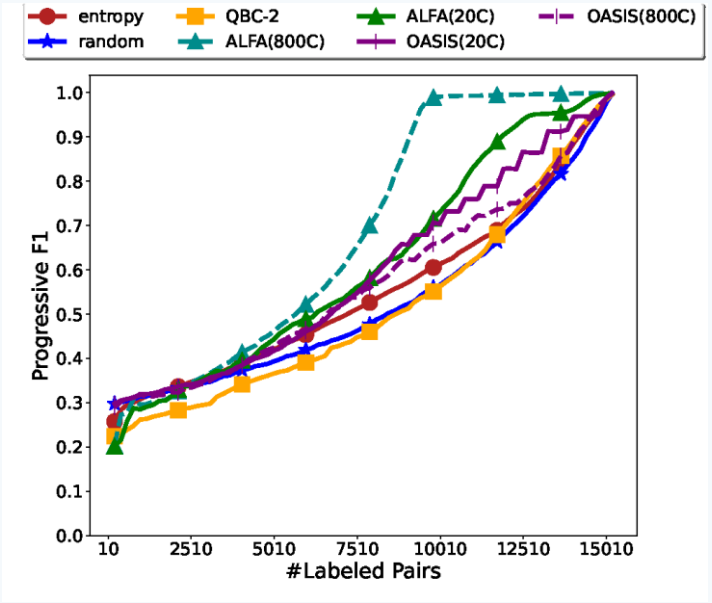
Dataset	$\lvert Nodes_{Left} \rvert$	$\lvert Nodes_{Right} \rvert$	$\lvert Pairs_{Matches} \rvert$	$\lvert Pairs_{Total} \rvert$	$Skew_{Label}$
CMT-CONF	39	77	15	3003	1:200
HUMAN-MOUSE	3298	2737	1516	9 Million	1:5937
BANK-KAFS	2148	7170	394	15.4 Million	1:39,086
FMA-NCI	3720	6488	2686	24.1 Million	1:8986

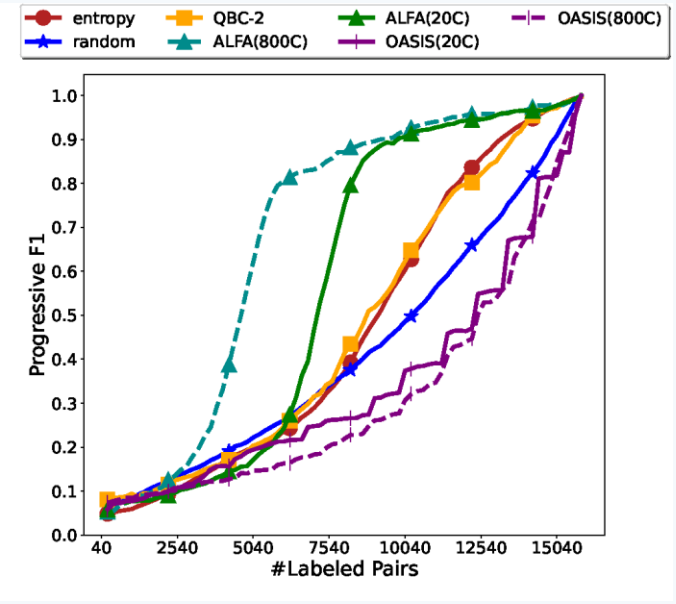
- CMT-Conference
  - http://oaei.ontologymatching.org/2021/conference/
- Human-Mouse anatomy
  - http://oaei.ontologymatching.org/2021/anatomy/
- ING-KAFS
  - IBM proprietary financial dataset
- FMA-NCI
  - https://www.cs.ox.ac.uk/isg/projects/SEALS/oaei/

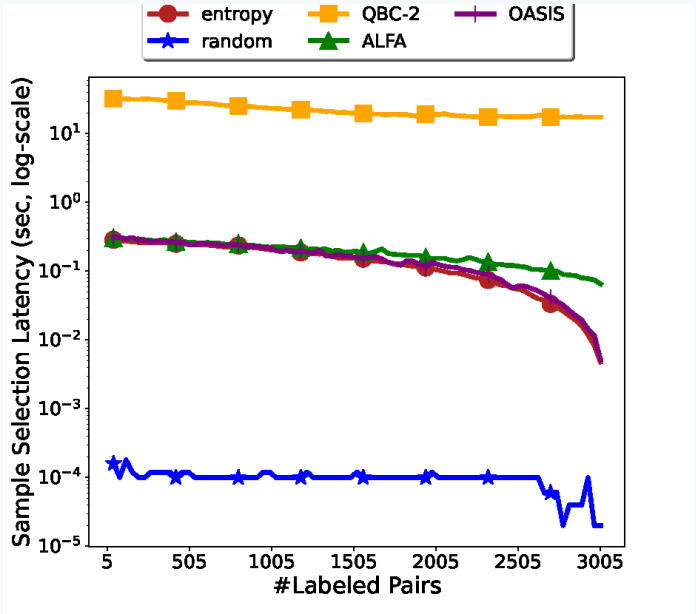
Evaluation of Sample Selector

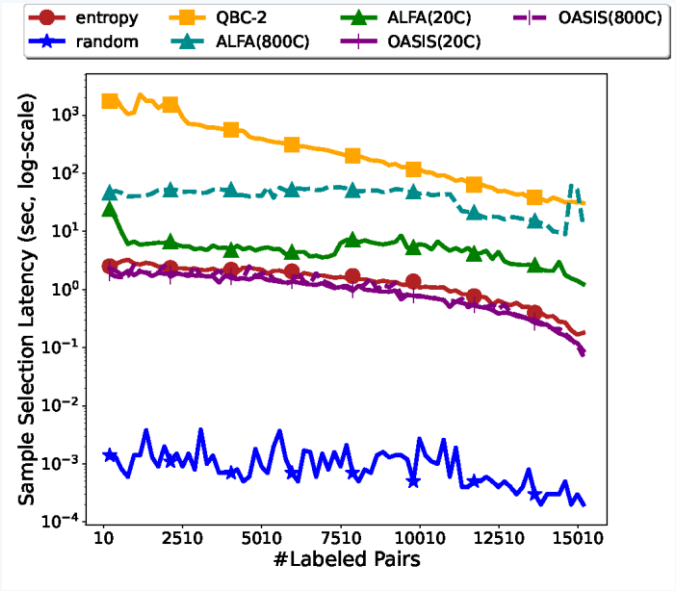
0.8 F1 at 20% (cmt), 30% (ING) and 60% (human-Mouse)
Example selection time 0.3 sec (cmt), 5 sec (ING) and 46 sec (human-Mouse)

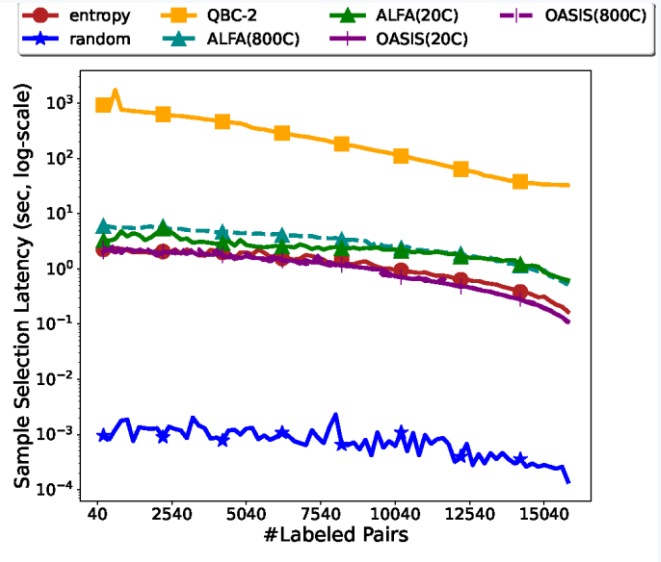






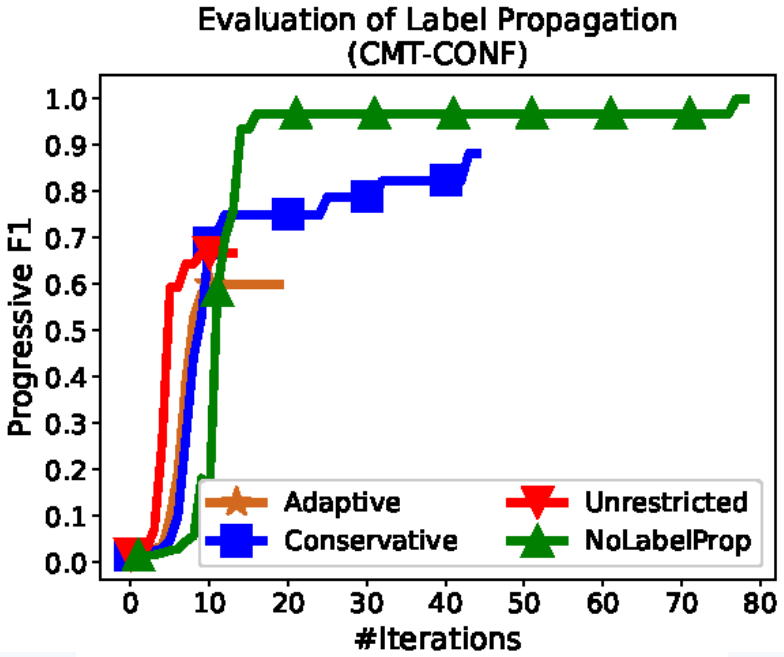


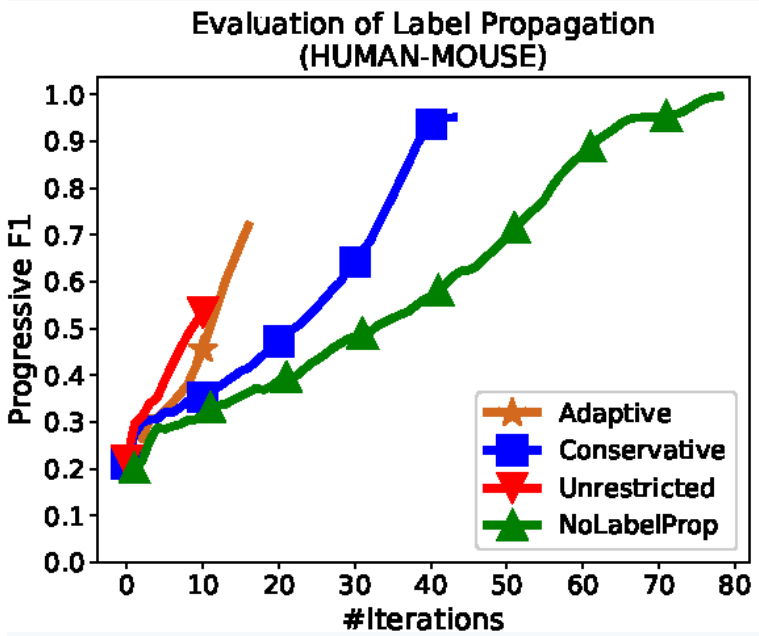


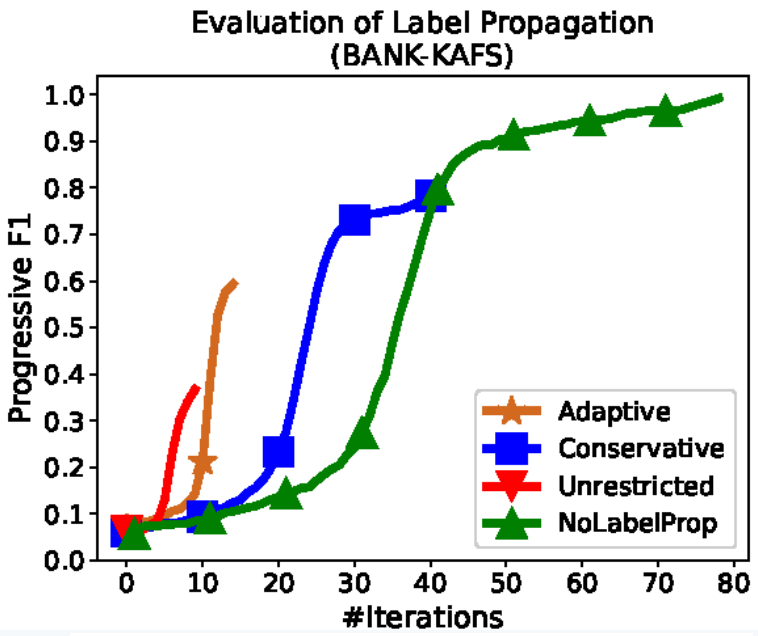


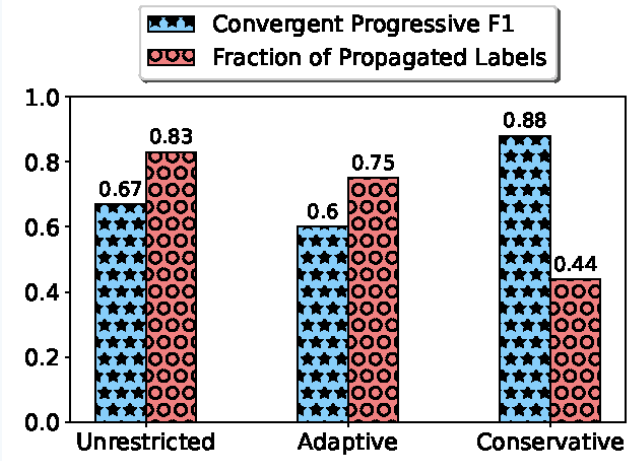
CMT-CONF **HUMAN-MOUSE** **BANK-KAFS**

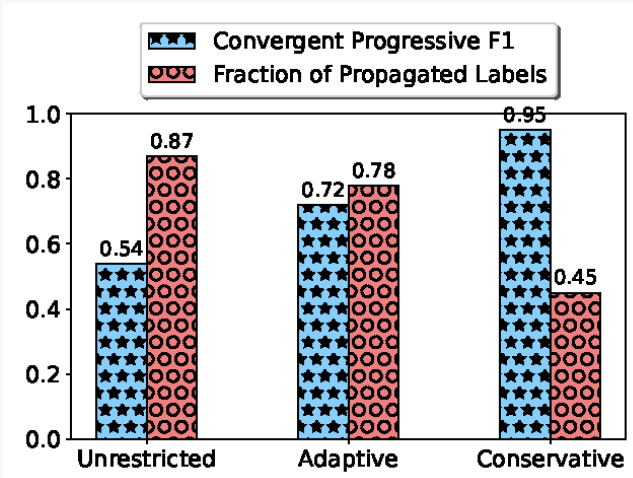
Evaluation of Label Propagation

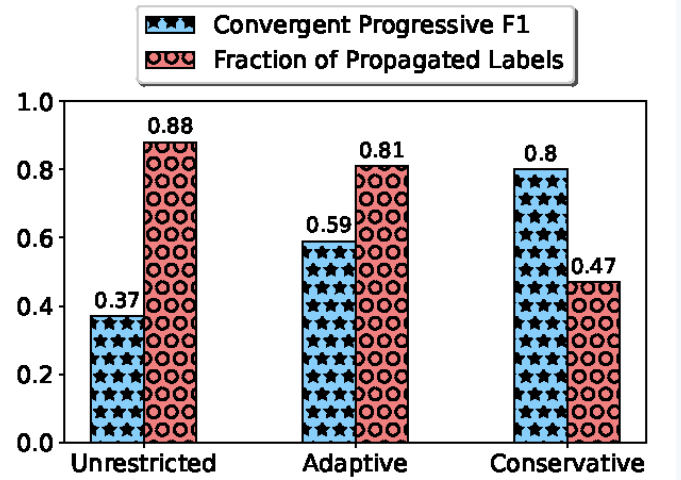












- Early termination is induced by unrestricted propagation
- Adaptive and conservative yield better quality than unrestricted propagation scheme

Overall Convergent F1-Score

Table 4 End-to-end evaluation breakdown of ALFA (default and ELBOW variants) vs. BERTMap

Dataset	#PB Pairs	%False negatives (blocking)		%Inferred labels (label propagation)			Overall convergent F1-score			
		ALFA (ELBOW)	BERTMap	ALFA	ALFA (ELBOW)	BERTMap	ALFA	ALFA (ELBOW)	BERTMap	BERTMap (FT)
D1	46	**26.7**	42.8	**2.3**	**18.6**	4.4	**0.85**	0.82	**0.2**	0.75
D2	3.2K	**23.9**	22.4	**40.4**	8.5	**3.9**	0.81	**0.85**	**0.49**	0.81
D3	6.9K	**39.3**	61.3	**44.6**	19.3	**0.3**	0.55	**0.68**	**0.01**	0.55
D4	6.4K	**53.8**	67.9	**42.8**	16.8	**0.5**	0.52	**0.58**	**0.01**	0.48

D1—CMT-CONF, D2—HUMAN-MOUSE, D3—BANK-KAFS, D4—FMA-NCI, #PB pairs—post-blocking pairs
The color green indicates best scores and color red indicates poor scores

- ELBOW with Silhouette coefficient used to automatically determine #clusters
- BERTMap (AAAI'22) uses
  - synonym-antonyms as labeling heuristics
  - hierarchical label propagation from parents to children
  - WordPiece Tokenizer-based inverted index to shortlist pairs containing overlapping words for blocking

Conclusion & Future Directions

- Vector index recall alone does not guarantee answer-level correctness
  - Question-answering (QA) recall for BEIR improves by 20% using NVIDIA embeddings
  - Hybrid search improves recall against relational ground truth by 2x
  - Schema-GraphRAG enables execution of complex semantic queries containing relational predicates

- In lakehouses, layout optimization replaces traditional indexing
  - Improves workload execution latency by 36% on TPC-DS and 11% on TPC-H (SF 10K)

- ALFA reduces labeling cost for semantic schema integration
  - on OAEI ontologies by 27% - 82% compared to supervised baselines (critical for multi-source RAG)

- Future Directions:

  - Relational Data Embeddings to avoid the need for separate lexical search
    - RelGNN, RelGT, RelBench [Jure Leskovec], TURL (VLDB'21), Kv2Vec

  - Enabling natural language query to CQL translation and automatic Gremlin query creation

  - Automatic data layout discovery with evolving data and workloads
    - Comparison against Databricks liquid clustering baselines

  - Enabling online data integration during query time in a ***pay-as-you-go*** fashion (SLMs distilled from LLMs for zero-shot integration)
    - Jayant Madhavan et al: Web-Scale Data Integration: You can afford to Pay as You Go. CIDR 2007: 342-350
    - Anish Das Sarma, Xin Dong, Alon Y. Halevy: Bootstrapping pay-as-you-go data integration systems. SIGMOD 2008: 861-874
    - Giovanni Simonini, Luca Zecchini, Sonia Bergamaschi, Felix Naumann. Entity Resolution On-Demand. PVLDB, 15(7): 1506 - 1518, 2022
    - Ihab Ilyas et al: Saga: A Platform for Continuous Construction and Serving of Knowledge at Scale. SIGMOD 2022: 2259-2272