join2vec: towards efficient and semantic-rich string similarity joins

Join models

Manos Chatzakis (emmanouil.chatzakis@epfl.ch)

Supervised by Viktor Sanca and Anastasia Ailamaki

Optional Semester Project Presentation

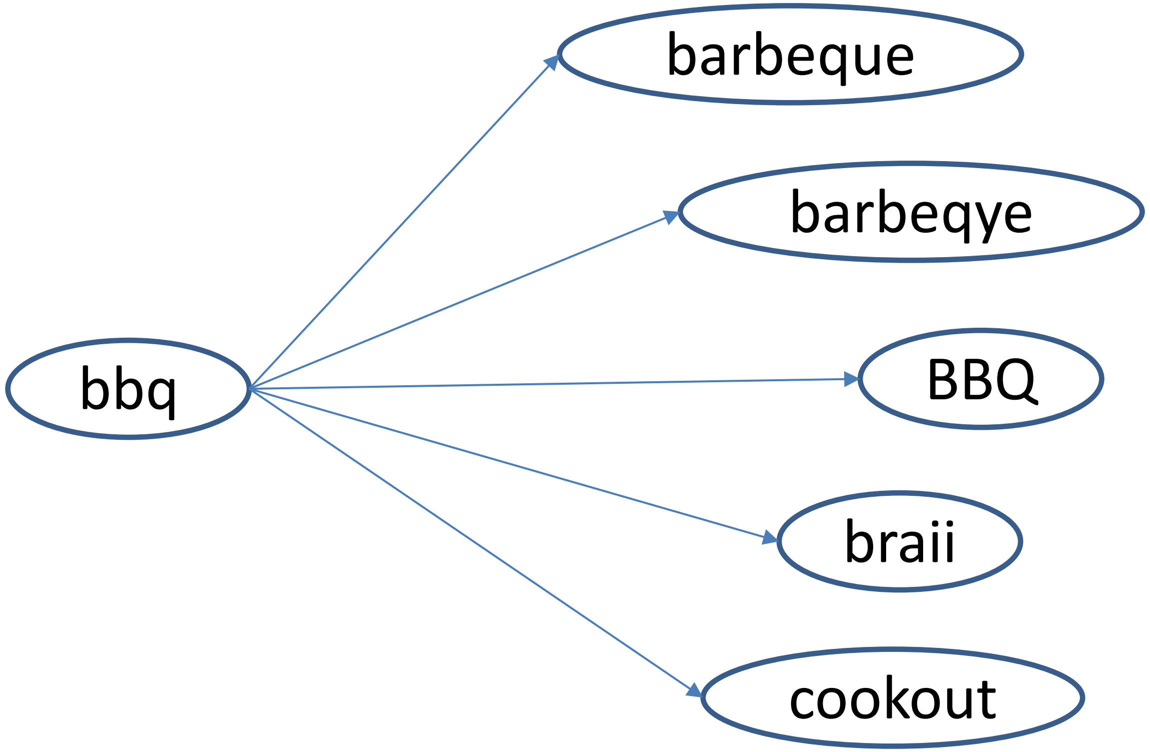
DATA-INTENSIVE APPLICATIONS AND SYSTEMS

EPFL

String Similarity Joins

...given a collection of strings, find the **most similar** pairs.

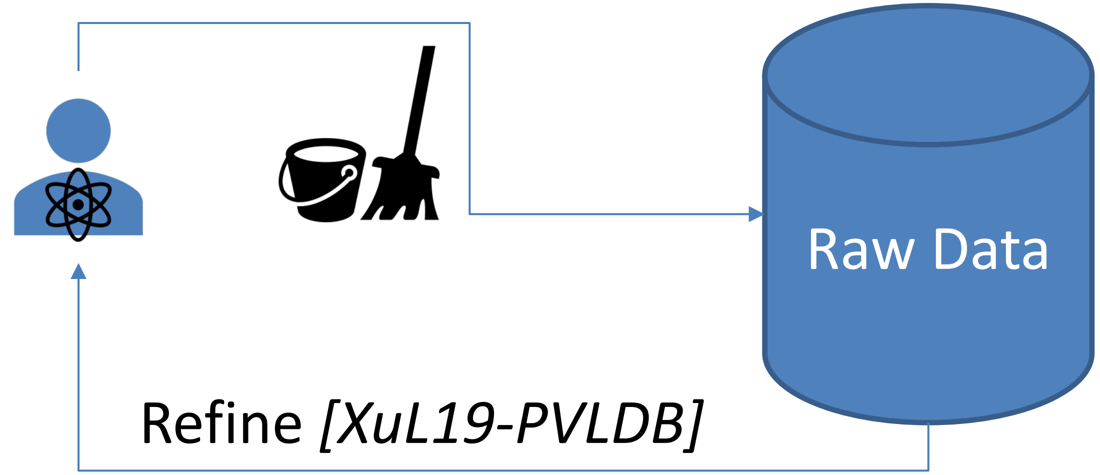
- Dataset Merging
- Duplicate Elimination
- Query Expansion
- Clustering



String similarity joins are indispensable in real-world analytics

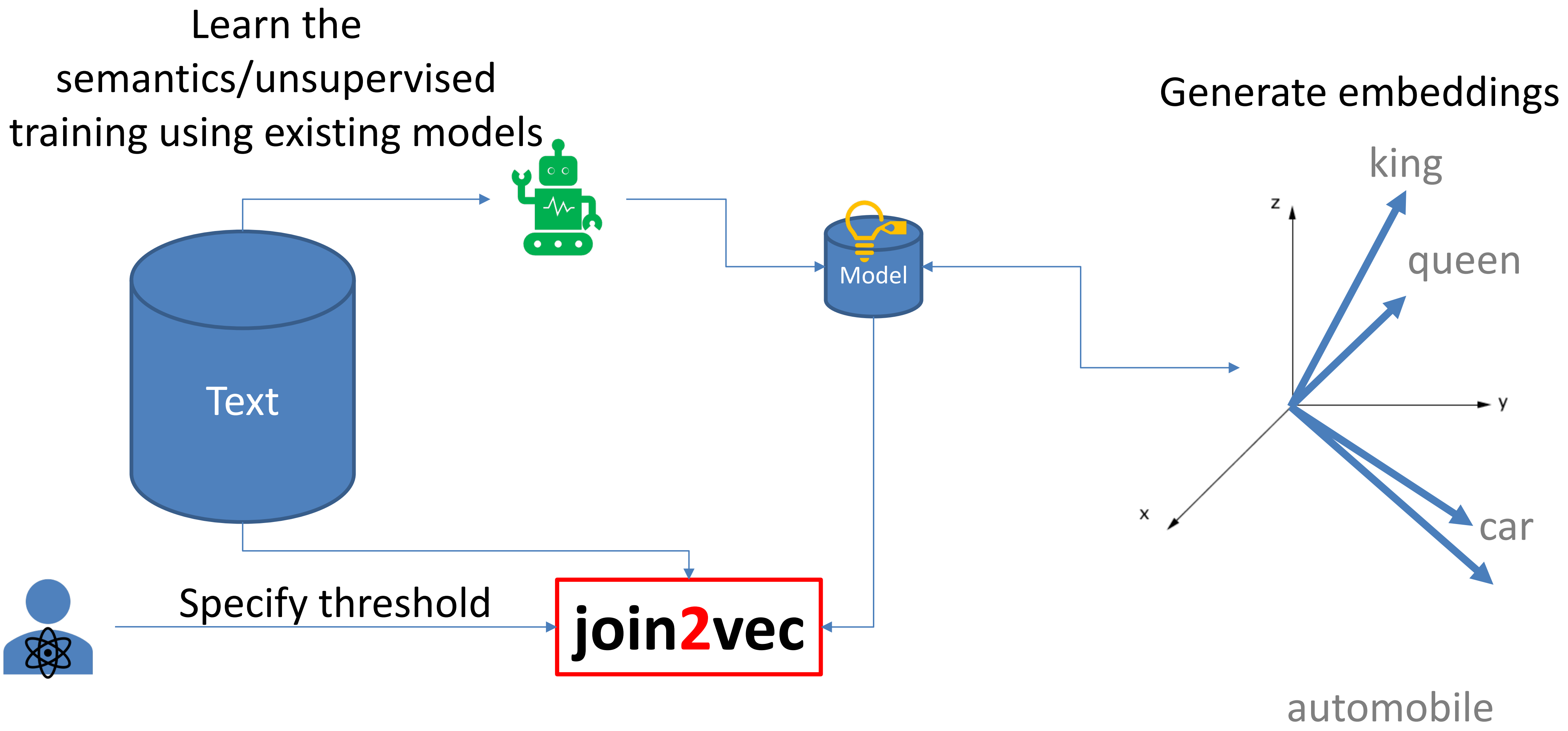
Similarity in Practice

Define similarity rules (Syntactic, Synonym, Taxonomy)



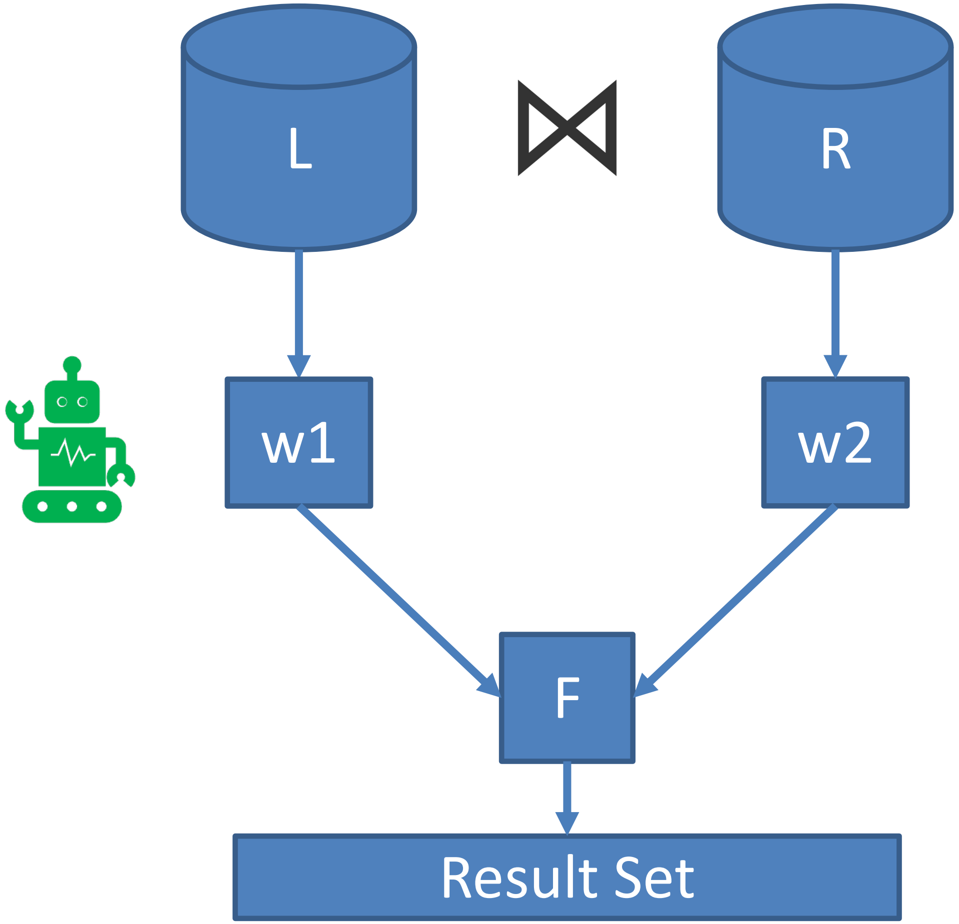
Defining similarity rules for strings is a difficult task

join2vec



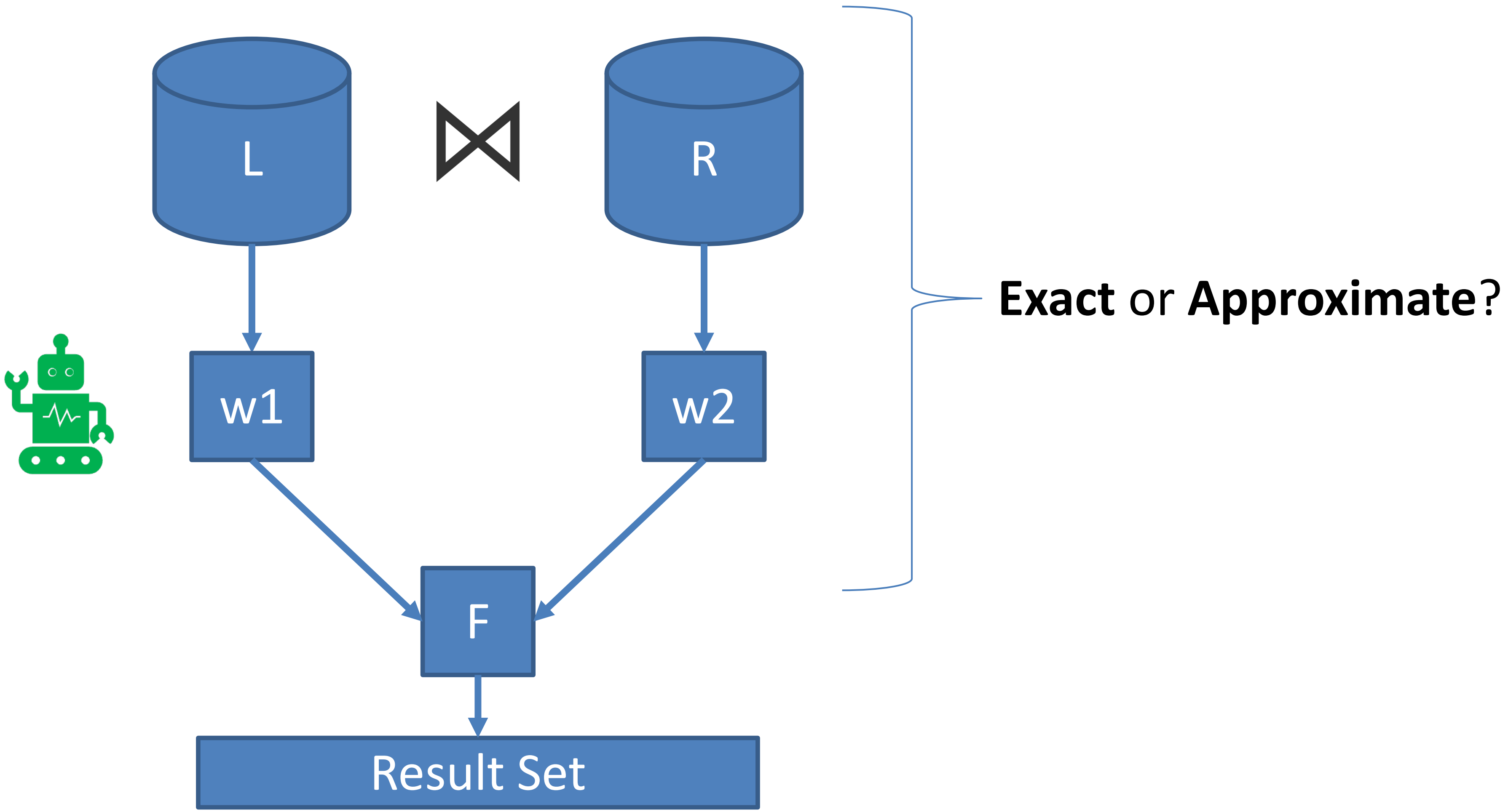
Model-Driven, automated rules that capture the semantic context

Join2vec Algorithm



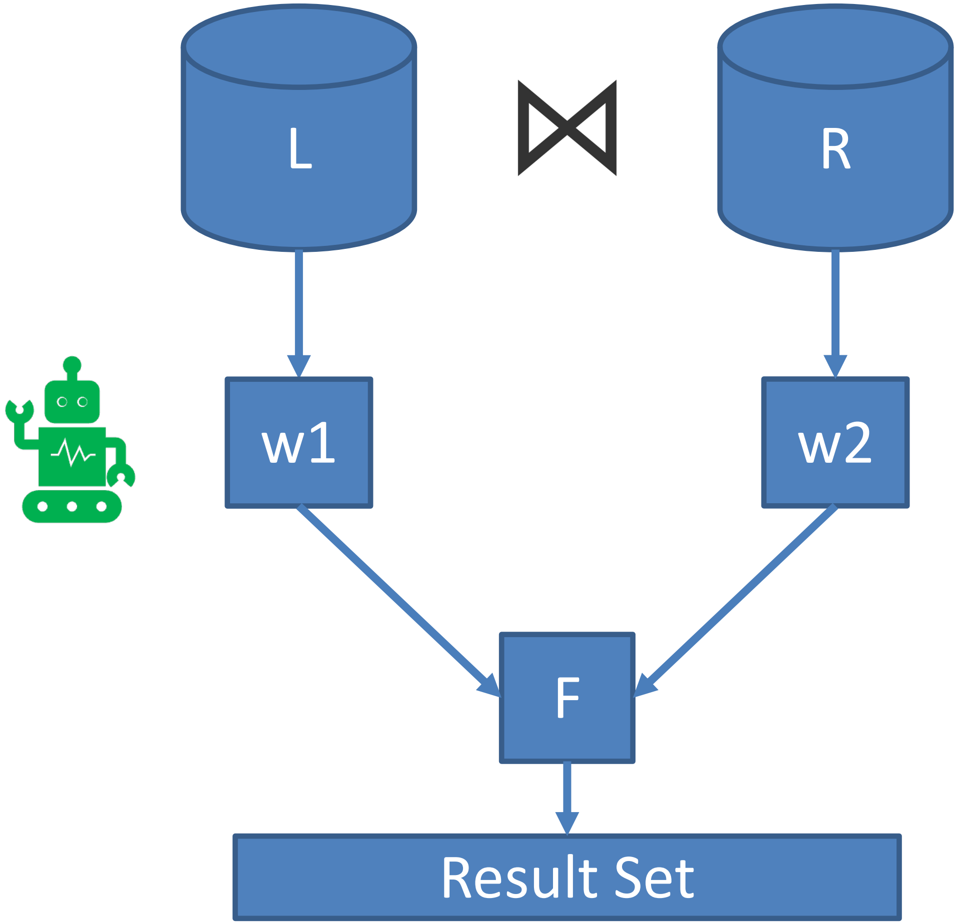
Tight join-model integration for efficient execution

Join2vec Algorithm



Tight join-model integration for efficient execution

Join2vec Algorithm



Nested Loop
or **Hash-Based?**

Tight join-model integration for efficient execution

Exact Joins

- Find **all similar string pairs**

- **Nested Loop**
  - Online iteration (validate **all pairs**)
  - No data structure
  - Parallelization, SIMD, Prefetching

- **Index-Based Join (FAISS)**
  - **Hash-Based** index
  - Big data collections
  - Fast lookup (**knn**) and filtering
  - Parallelization, SIMD

Exact joins can be supported by both nested loop and hash-based joins

Approximate Joins

- **Tradeoffs** between speed and retrieval quality

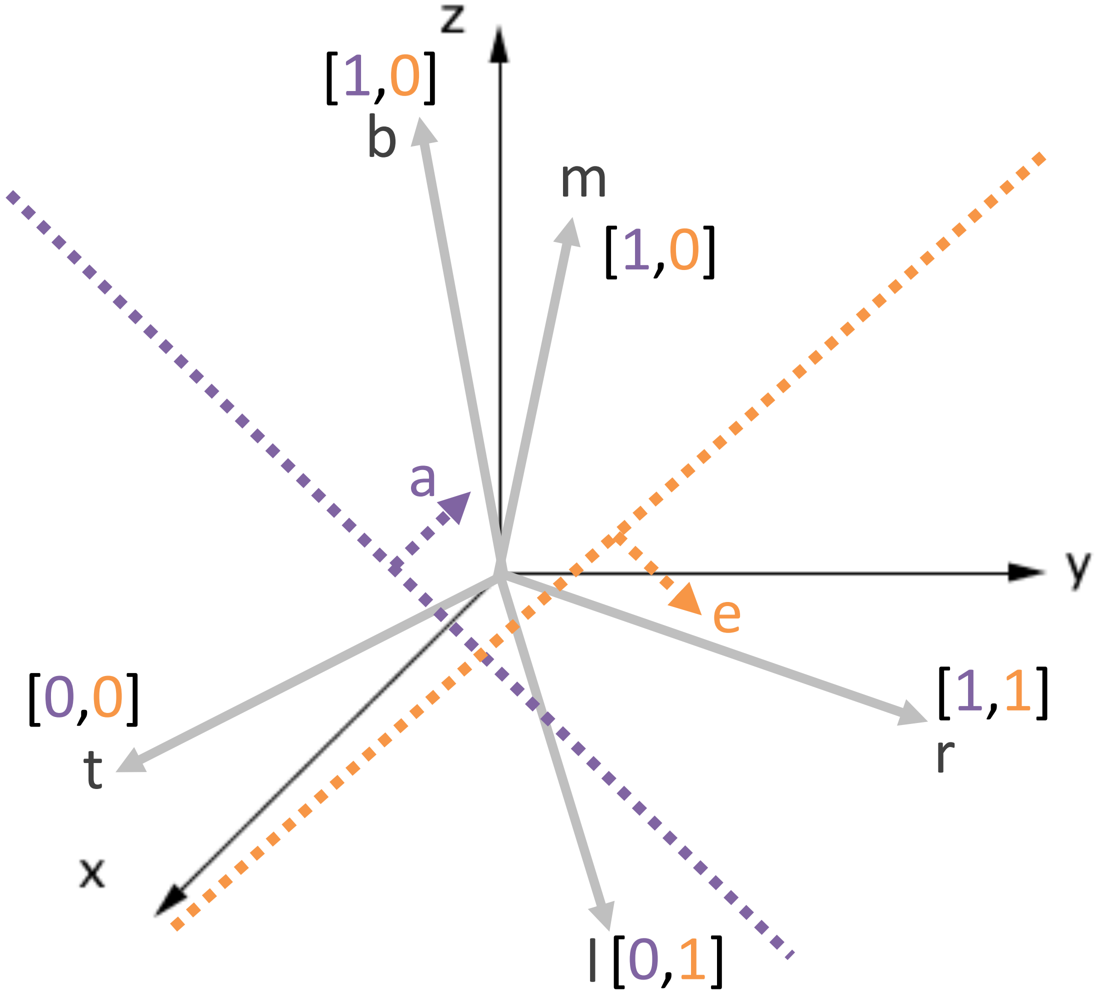
- **Approximate Hash-Based Index**
  - Index on one relation
  - Hash every query
  - Retrieve and validate entries

- Approaches
  - Bloom Filters
  - **Locality Sensitive Hashing**

Approximate joins are implemented with approximate hash-based indices

Locality Sensitive Hashing

- Regions using **hyperplanes**
- Dense to binary vectors
- **Quality** of hashing related to the number of **hyperplanes**



Region	Bucket
[0,0]	t
[0,1]	l
[1,0]	b, m
[1,1]	r

Retrieval quality strongly related to the number of hyperplanes

Cost Model Evaluation

- Specifications
  - R: Outer relation size
  - S: Inner relation size
  - M: Model access cost
  - P: Similarity calculation cost
  - Sel: Selectivity rate (%)

Cost Model Evaluation

- Nested Loop Joins
  - Simple (**NLJ**)
    - $costNLJ = R + (R \times S) + (sel \times R \times S \times P) + (M \times R \times S)$
  - Prefetching (**NLJ-P**)
    - $costNLJ{-}P = R + (R \times S) + (sel \times R \times S \times P) + (M \times (R + S))$

- Hash-Based Joins, **HASHJ**
    - $costHASHJ = R + S + (sel \times R \times S \times P) + M \times (R + S)$

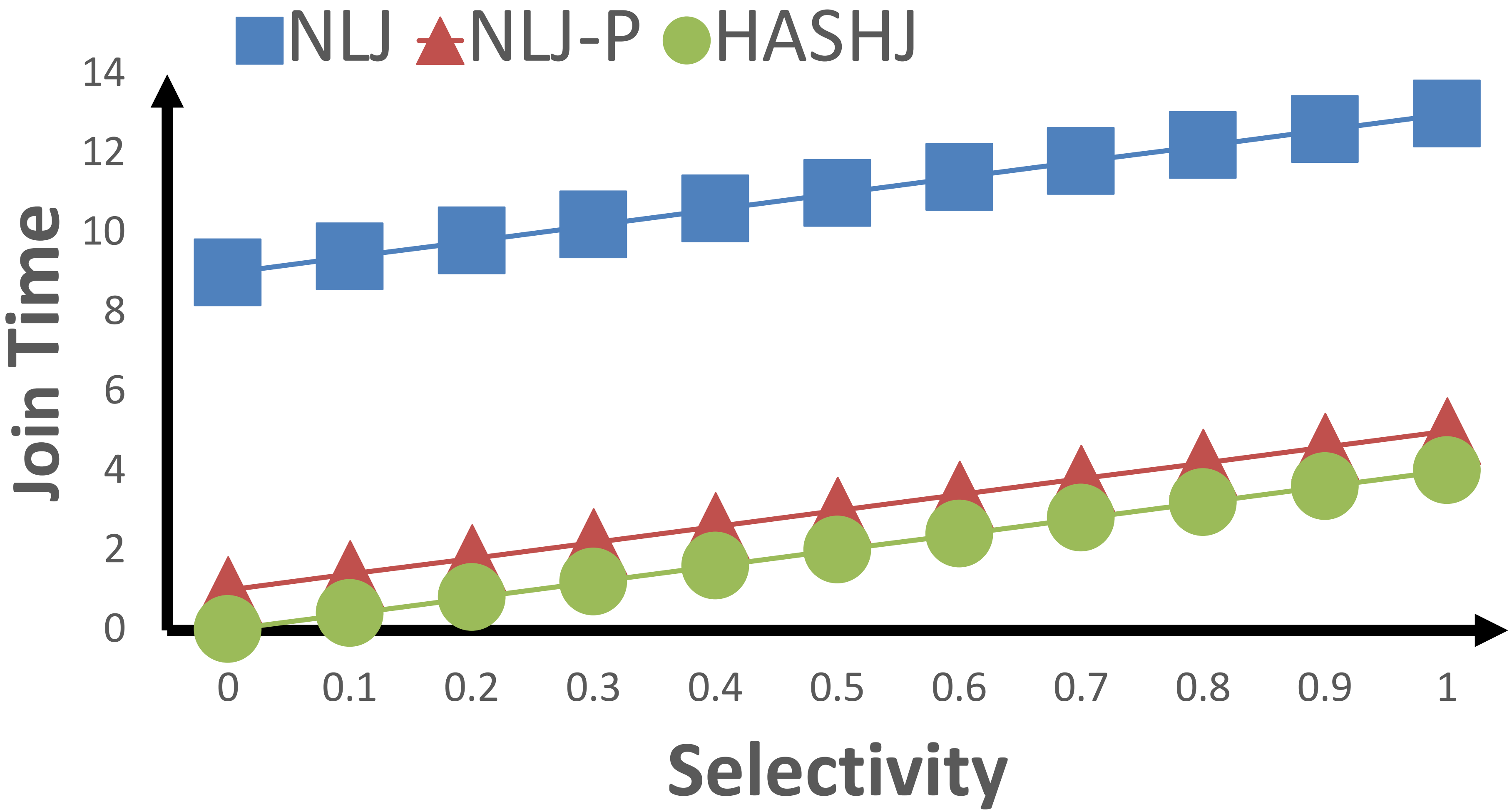
Model Accesses

Processing of pairs

Dataset Iterations

Cost differs in the way of accessing the model and iterating over the relations

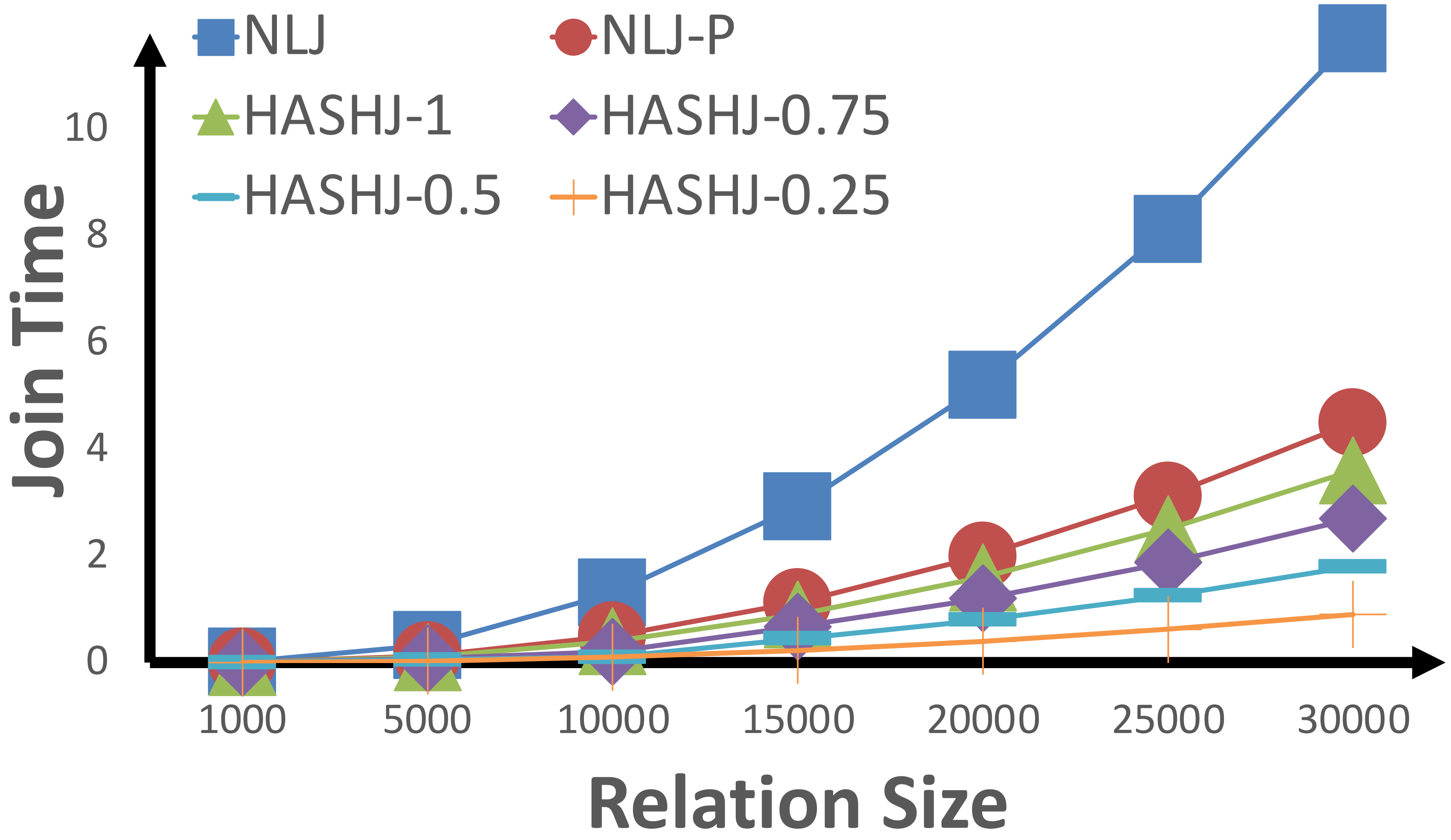
Selectivity Rates



- Assume filtering technique for nested loops (NLJ, NLJ-P)
- Selectivity can be tuned based on attributes and hyperplanes

Execution time grows linearly with selectivity, better performance with hashing

Dataset Size



- Selectivity rate can be tuned (e.g. hyperplanes)
- Selectivity of NLJ algorithms is 1.0

Strict selectivity and hash-based structures lead to better performance than NLJ

Concluding Remarks

- Different models applicable for exact and approximate string similarity joins
- Hash-Based joins lead to better performance for big vector collections
- Approximate LSH-based join has important tradeoffs between performance and quality of retrieval

Future Work

- Extensively evaluate all join methods
- Provide insight between the tradeoffs of approximate join quality
- Explore FAISS exact-knn indices and capabilities even further
- Explore FAISS approximate-knn indices
- Compare FAISS approximate results with LSH method

Thank you!