Embedding-Based SQL Query Similarity for Spatiotemporal Data Exploration

Evangelos Karageorgos
University of Pittsburgh
karageorgos@pitt.edu

Alexandros Labrinidis
University of Pittsburgh
labrinid@cs.pitt.edu

***Abstract*—Estimating the similarity of SQL queries is a fundamental building block for query recommendation systems, workload analysis, and interactive data exploration. We introduce TESS (Tree-Embedding SQL Similarity), a novel query similarity algorithm that constructs a weighted embedding vector from the label tree of a query, capturing both its structural and semantic content. We also present GESS (Global-Embedding SQL Similarity), a lightweight companion algorithm that embeds the full query text as a single vector. Our motivating application is query recommendation for spatiotemporal data exploration, where an analyst's queries must be compared by semantic intent rather than by syntactic structure alone. We introduce a rigorous evaluation framework and perform a comprehensive evaluation across five datasets and seven metrics. Our experiments show that TESS achieves superior performance across multiple metrics, making it well-suited for interactive recommendation at scale.**

I. Introduction

Relational database systems dominate our entire software infrastructure. The ability to estimate the similarity between SQL queries has applications across multiple domains. Query auto-completion, query recommendation systems, estimating query execution performance based on known performance of similar queries, grouping application interactions based on the queries they utilize, and identifying patterns in query logs all require a metric of query similarity. Being able to produce an estimate of query similarity, however, is a non-trivial task.

A major concern when it comes to SQL query similarity is that there is no single objective similarity criterion among queries. Different applications may compare queries in terms of their output, their structure, the data they probe, or other criteria. As such, the definition of similarity is subjective and depends on the application requirements. Even the simple case of query equivalence, whether two queries are identical in function, is an unsolved problem, proven to be NP complete for the general case [1].

In this work, we target *intent-based similarity*: two queries are considered similar if they probe related concepts at comparable levels of abstraction, regardless of syntactic formulation or query output. For example, a query retrieving ships that passed through a specific maritime region during morning hours should be considered similar to one retrieving ships in a neighboring region during evening hours. Both queries express the same spatiotemporal intent of filtering vessel trajectories by location and time window, but are dissimilar to a query counting distinct vessel types. This notion of similarity is more appropriate than output-based or purely structural similarity for recommendation applications, where the goal is to surface queries that reflect related user interests, not queries that happen to return the same rows or share the same syntactic skeleton.

Our motivating application is a query recommendation system for spatiotemporal data exploration. Consider two different scenarios. In the first, an analyst is exploring maritime mobility data from the port of Piraeus, issuing SQL queries over a MobilityDB database of ship trajectories. For example, retrieving vessels that entered a specific bounding box during a given time window, or identifying ships whose trajectories exhibit a particular movement pattern. A recommendation system that can recognize the *intent* behind such queries (filtering trajectories by spatiotemporal constraints) can surface related queries from other analysts, helping the user discover relevant patterns without formulating complex spatiotemporal predicates from scratch. In the second scenario, an urban analyst is querying an OpenStreetMap-based database of the Pittsburgh metropolitan area, exploring road network features, points of interest, or neighborhood boundaries across different time periods. Here too, a similarity algorithm that understands that two queries probing adjacent geographic regions or overlapping time intervals are more alike than a pair of unrelated queries, serves as a more useful foundation for recommendations than one that compares only syntactic structure.

In both cases, the similarity algorithm must satisfy two requirements that are in tension: it must capture *semantic intent* – recognizing that two queries probing different time windows or bounding boxes are more similar to each other than to a query counting distinct vehicle types – and it must be *fast enough for interactive use*, as a recommendation system may need to evaluate thousands of query comparisons in fractions of a second to minimize user-perceived latency.

A practical concern, largely overlooked in the query similarity literature, is the *collision-free ratio*: the proportion of queries that a similarity algorithm can uniquely distinguish. An algorithm that collapses many semantically distinct queries into identical representations produces high collision rates that directly degrade recommendation quality. We argue that the collision-free ratio is a first-class evaluation metric and our evaluation framework treats it as such.

TABLE I: Comparison of TESS and GESS with related neural SQL embedding methods. "No query plan required" = does not require query execution plan or database statistics; "No training required" = does not require task-specific supervised training; "Semantic content" = Incorporates the semantic content of columns and values into the query representation; "General similarity" = applicable to produce a pairwise similarity score without adaptations.

Feature	NEO [2]	QueryFormer [3]	PreQR [4]	Paul et al. [5]	**GESS (ours)**	**TESS (ours)**
No query plan required	✗	✗	✓	✗	✓	✓
No training required	✗	✗	✗	✗	✓	✓
Semantic content	✗	✗	✓	✗	✓	✓
General similarity	✗	✗	✓	✗	✓	✓

Contributions

Our work makes the following contributions:

- We propose an evaluation framework for comparing SQL query similarity algorithms in the context of query recommendations, introducing the *collision-free ratio* as a key metric that captures each algorithm's ability to uniquely distinguish semantically different queries.
- We introduce **TESS** (Tree-Embedding SQL Similarity) and **GESS** (Global-Embedding SQL Similarity), a new family of query similarity algorithms based on Natural Language Processing (NLP) vector embeddings of structural and semantic query components.
- We utilize the evaluation framework to compare established query similarity algorithms against TESS and GESS across multiple databases, including spatiotemporal workloads and other benchmarks.
- We summarize all algorithms based on their features and performance, identifying TESS as the best overall choice for measuring query similarity in query recommendation applications.

II. Related Work and Challenges

Algorithms that calculate query similarity can be categorized based on the taxonomy by Arzamasova et al. [6]. **String-based (SB)** approaches consider differences in the literal SQL text. **Witness-based (WB)** approaches compare query output. **Feature-based (FB)** approaches consider query structure and syntax, and **Access area-based (AAB)** approaches consider which parts of the database the queries probe. *Witness-based* similarity is sensitive to the underlying data and disregards structural information. *Access area-based* similarity focuses on predicates and is primarily applicable to simple queries. In the context of query similarity for query recommendations, *feature-based* similarity algorithms make the most sense, as they consider the intent and semantics of the queries.

A. Edit distance-based similarity

A number of *feature-based* similarity algorithms calculate distance among queries based on their edit distance, usually working on a syntax tree representation. This can be effective, but works well only when the queries being compared are already similar. For queries with arbitrary complexity, finding the minimum edit distance is an NP problem [7], and heuristic approximations scale poorly with query complexity. Furthermore, edit distance focuses on syntactical rather than semantic similarity, making it less appropriate for our target application.

B. Global query features

Rather than comparing queries directly, one can extract specific features from every query (*global query features*) and compare those features. This has an inherent performance advantage: one can maintain a pool of pre-computed query features and compare any new query to the entire pool via feature comparison alone.

One notable example is the work by Bordino et al. [8], who analyze a log of search engine queries by representing the query log as a graph connected by temporal adjacency, producing *query flows* that correspond to users with similar goals. They use spectral projection to embed graph nodes into Euclidean space and compute similarity via cosine distance.

C. Embedding-based query representations

A growing body of work applies neural embedding techniques to tree-structured or graph-based representations of SQL queries and query execution plans, motivated primarily by database optimization tasks. Marcus et al. [2] embed database values using word2vec [9] within a tree-structured neural network for join order selection. Zhao et al. [3] propose QueryFormer, a tree-structured Transformer for query *plan* representation augmented with database statistics. Tang et al. [4] propose PreQR, a pre-trained SQL representation model that uses a graph neural network to encode query structure and schema. Paul et al. [5] use query plan encoders to characterize and cluster workloads for database tuning.

While these methods share our interest in embedding SQL queries into vector spaces, they differ from TESS in ways that make a direct comparison inappropriate. Table I summarizes the key distinctions. NEO [2] and QueryFormer [3] operate on *query execution plans* rather than SQL text, and require access to database statistics (e.g., cardinality estimates, cost models) that are unavailable in a general-purpose similarity setting. PreQR [4] requires schema-specific supervised pre-training on labeled query-result pairs, making it inapplicable without task-specific training data. Paul et al. [5] likewise rely on plan-level encodings tied to a specific database engine. Crucially, most of these methods are not designed or evaluated as general-purpose *pairwise similarity metrics*: they produce representations optimized for their respective downstream tasks (join

ordering, cost estimation, workload clustering) and do not expose a similarity score suitable for query recommendation without further task-specific adaptation.

TESS differs from all of the above in three key respects. First, it operates directly on the SQL text without requiring execution plan data, database statistics, or access to the underlying data. Second, it is fully unsupervised and requires no task-specific training, making it immediately applicable across domains, including spatiotemporal databases where labeled data is scarce. Third, it explicitly incorporates the semantic content of column names, table names, and value constants (elements that other embedding approaches discard or treat as opaque identifiers) into its embeddings. These properties make TESS well-suited as a general-purpose query similarity metric where labeled training data is unavailable and low latency is required.

D. Similarity collisions

Most similarity algorithms compare simplified versions of queries, losing information along the way, which can lead to *collisions*: two different queries being evaluated as identical or more similar than they should be. Collisions can occur on *witness-based* algorithms, as different queries can produce the same output, and on *feature-based* algorithms, as many simplify the queries for comparison. In the context of query recommendations, minimizing collisions is essential when choosing a suitable similarity algorithm.

E. Value erasure

A related problem is ignoring the semantic content of parts of the query. Most algorithms do not consider semantic differences between tables, columns, and constants. For example, an algorithm may identify that a '`first_name`' column is different from a '`last_name`' column, but not recognize that they are semantically more similar to each other than to '`product_description`'. Another example would be structurally identical queries that differ only on a value constant within a predicate, like

```
SELECT *
FROM Actors
WHERE name='Anthony Hopkins'
```

vs.

```
SELECT *
FROM Actors
WHERE name='Johnny Depp'
```

A similarity algorithm may ignore all constants when comparing queries, evaluating these as identical. We call this concept **value erasure**, and it is a characteristic that can limit the differentiating potential of similarity algorithms, leading to more collisions.

F. Existing similarity algorithms

We compare our proposed methods with three that are well-studied and well-suited for our target application: **aouiche**, **aligon**, and **makiyama** [10].

Aouiche et al. [11] introduced a metric focused on view selection in data warehouses for simple data cube queries, using only the columns involved in a query as its feature set. Aligon et al. [12] expanded on this by specifying three feature sets based on projection, group-by, and selection join components, combining their similarity values into a single score. Makiyama et al. [13] refined this further by transforming a query into a weighted bag of clause-column token pairs.

In all three algorithms, a pair of queries is transformed into two sets of keywords, and similarity is computed by comparing those sets using Hamming distance (*aouiche*), Jaccard similarity (*aligon*), and cosine similarity (*makiyama*). These algorithms produce continuous similarity values, are simple and fast to compute, and focus on the participating columns and how they are utilized. However, as we demonstrate in our evaluation, all three suffer from significant value erasure, leading to high collision rates that limit their usefulness for query recommendation.

III. New SQL Query Similarity Algorithms

In this section, we present our proposed query similarity algorithms and their variants.

A. Global Embedding SQL Similarity (GESS)

GESS produces a single embedding vector for an SQL query by embedding the query text in its entirety, without further structural analysis. This approach neither extracts SQL structural information nor performs any vector combination. There is no distinction between structural and semantic information; the only parameter is the choice of embedding model. A key advantage is that the embedding model can see the entire query at once, enabling a well-trained language model to jointly reason about the query's structure, semantics, and intent. GESS also represents most query representations that are based on elaborate and expensive machine learning models, like QueryFormer [3], as most of them can capture rich semantic information, but are expensive to run.

B. Tree Embedding SQL Similarity (TESS)

TESS is a *feature-based* approach that produces a global, multi-dimensional query feature vector. Because the feature vector encodes both the structure of a query and all its values, TESS does not undergo *value erasure*, producing largely unique vectors across queries and thereby minimizing the chance of collisions. The largest feature vector we use in our experiments has 768 components.

1) ***Label tree****:* The query is first parsed into a syntax tree, which is then simplified into a tree of plain text labels. Labels fall into two categories: **structural labels**, which correspond to SQL syntactic elements such as `select` or `where`, and **value labels**, which correspond to column references (e.g., `actors.name`) or value constants (e.g., `Brad Pitt`). This representation transforms SQL query comparison into a problem of comparing labeled trees. Figure 1 shows an example label tree, where structural labels are drawn as rectangles and value labels as ellipses. The depth of a label within the tree is significant: labels closer to the root are treated as more important than those deeper in the tree. For instance, queries

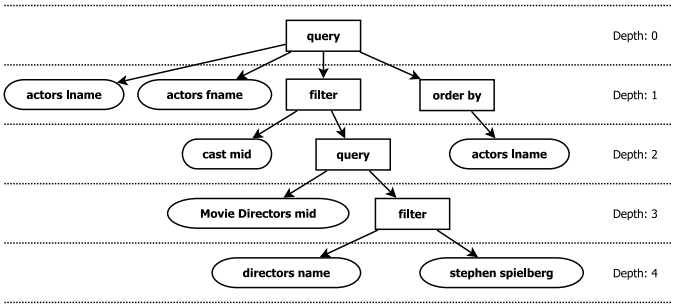

Fig. 1: An SQL query, analyzed into a label tree

that differ in their projected columns or primary `WHERE` predicate are considered more dissimilar than queries that differ only in a deeply nested subquery. Note that the inner joins in Figure 1 are intentionally omitted from the label tree: they are trivial joins based on foreign key relationships and carry no meaningful semantic signal for query similarity.

*2) **Label embeddings**:* NLP vector embedding techniques are applied to each label in the tree to produce a vector that captures its semantic meaning. These per-label vectors are then combined arithmetically into a single vector representing the entire query. Different variants of the algorithm correspond to different ways of collapsing the tree of vectors. The TESS family computes a weighted sum of label embedding vectors V_q, as defined in Equation 1. The weights serve two purposes: they distinguish between structural and value labels, and they attenuate the contribution of labels at greater tree depth. The algorithm is parameterized by the **depth weight** (w_d), which controls how steeply importance decreases with depth, and the **semantic weight** (w_s), which controls the relative importance of value labels versus structural labels. This affects how queries are laid out in the vector space, based on our intuition on customizing the semantic importance of top-level query features and importance of values over structure, depending on the application needs.

$$
\begin{aligned}
V_q &= \sum_{l \in labels} v_l w_l w_d^{d_l} \\
&\quad where \\
v_l &= the\ embedding\ vector\ for\ label\ l \\
w_l &= \begin{cases} 1 & l\ is\ a\ structural\ label \\ w_s & l\ is\ a\ value\ label \end{cases} \\
d_l &= the\ depth\ of\ label\ l \\
w_d &= the\ \textbf{depth weight}\ parameter \\
w_s &= the\ \textbf{semantic weight}\ parameter
\end{aligned}
\tag{1}
$$

C. Variants

Both TESS and GESS rely on producing embedding vectors for text phrases, whether database values, constants, SQL keywords, or an entire query. The choice of embedding method affects how well semantic information is captured. We evaluate two broad classes of embedding methods.

Vocabulary-based: We use the *word2vec* [9] algorithm as implemented by Gensim [14], with the BERT subword tokenizer [15] to decompose each phrase into tokens. The token embeddings are averaged and normalized to produce the final vector. This is a non-contextual method: tokens are treated independently, and word order is disregarded. It is the most computationally efficient option, as token embedding retrieval is an $O(1)$ vocabulary lookup. We trained the word2vec model on the first million sentences of Gensim's *wiki-english-20171001* corpus [16], yielding a vocabulary of 30,522 tokens. We evaluate two variants: **vocab-200** and **vocab-768**, producing 200- and 768-component vectors, respectively.

BERT: We run the phrase through the BERT language model [17] and extract the embedding of the `[CLS]` token as the phrase representation, which is then normalized. This is a contextual method that captures the semantic content of the whole phrase jointly. It is more computationally expensive, as it requires a full forward pass through a large transformer network. We use Hugging Face's *bert-base-uncased* model [15], which produces 768-component vectors.

In total, we evaluate three embedding configurations: **vocab-200**, **vocab-768**, and **bert**.

D. LLM-Based Similarity

We also implement a query similarity algorithm based on a Large Language Model. We use **gpt-4o-mini** [18] with a prompt that includes a description of the database schema and few-shot examples of similar and dissimilar query pairs, producing a distance value for any pair of queries. This algorithm serves as a *gold standard* for semantic query comparison: querying an LLM is the closest automated proxy to asking human annotators to judge query similarity. Its significant drawback is that its computational cost is prohibitive for any performance-sensitive application, as each pairwise comparison requires a separate model inference call. We include it as a reference point for evaluating how closely TESS and GESS approximate human-level semantic judgment.

IV. How to Compare Query Similarity Metrics

A. Cluster separability

One way to evaluate the effectiveness of similarity algorithms, is to assess how different clusters of queries are separable based on the similarity of the queries they contain. Kul et al. [10] conduct a similar analysis, but in their case the query clusters are pre-defined for every dataset they test. For our analysis, we define a small set of syntactically complicated queries (**seed queries**). Then, we apply a set of **simple modifications** to every one of them, to define a query cluster for every *seed query*. Those modifications are applied on SELECT columns (parts of the SELECT clause that are a single column e.g. SELECT id, title ...) and simple comparisons (in the form of column-operator-constant, e.g., title = 'The Matrix'). The modification operations include swapping SELECT columns, swapping columns on comparisons, swapping constants on

comparisons, inserting additional comparisons on predicates, and removing simple comparisons from existing predicates. When we attempt to introduce columns that were not part of the tables referred to by the query, we make sure to include the appropriate table with an inner join operation, based on the foreign keys of the database. We also assert that by introducing these simple joins, as in including tables that are trivially joinable based on simple foreign key constraints, the query should be semantically similar enough to be considered part of the same cluster. All these modifications preserve most of the syntactic structure and complexity of the initial query, so we argue that they should be considered part of the same query cluster. We can then calculate the similarity among all queries, and use clustering metrics to assess how *separable* those clusters are. If a similarity algorithm is effective, the clusters must be clearly defined and separated.

We also argue that a good similarity algorithm should be able to distinguish between queries of the same cluster, and queries of a different cluster, based on the inherent structural similarity of queries within a cluster, and structural difference of queries from different clusters. Thus, we can compare similarity algorithms, by comparing how well the query clusters are defined and separated under each similarity algorithm.

The work by Kul et al. [10] includes three datasets of clusters of queries, but we did not use them in our evaluation, as they are limited in size, do not provide the database schemas, and one of them has parameterized queries (i.e., use "?" instead of specific values in predicates) that are unsuitable for query recommendation applications. Query generation tools like SqlStorm [19], that uses LLMs to produce diverse and complex queries could also be used to produce seed queries for datasets, but it was deemed excessive, given that the datasets we need for evaluation are based on a limited number of seed queries.

B. Collision-free ratio

Another metric to assess similarity algorithms is their ability to distinguish unique queries. A similarity algorithm that only considers the structure of queries will consider different queries that share the same overall structure as identical. The same applies to similarity algorithms that compare only the outputs of queries. Two different queries that happen to produce the same output for a given database will be evaluated as identical. In our case, we identify a collision between two queries A and B if the distances of all queries in the dataset to A are the same as the distances of all queries to B. Specifically, given a distance matrix of all queries, any two rows (or columns) that have the magnitude of their difference less than a very small value (0.00001) are considered a collision. We will call the ratio of the number of non-colliding queries over the overall number of queries the **collision-free ratio**, and we expect that better similarity algorithms will have bigger collision-free ratios, as this metric expresses how well the algorithm captures the semantics of the queries and how well it distinguishes unique queries from one another.

C. Clustering measures

There are numerous metrics that evaluate clustering algorithms, known as Clustering Validity Indices (CVIs). In our case we consider the Silhouette Index [20], the Dunn Index [21], the Davies-Bouldin Index [22] and the Calinski-Harabasz Index [23]. Many CVIs, including Dunn index, Davies-Bouldin index and Calinski-Harabasz index, are widely used, but examine only the worst-case cluster separability. They also require cluster centroids and vectors for all points, which are not available on every similarity algorithm. Silhouette index on the other hand, considers all clusters and all points simultaneously, so it is more resistant to outliers, and operates purely on distances, making it reliable and available for all similarity algorithms. Therefore, we will mainly focus on silhouette index for our evaluation, even though we will present values for other CVIs when available.

D. Multidimensional Scaling (MDS)

Most similarity algorithms produce similarity or distance of a pair of queries, but it is useful to have multidimensional vectors that correspond to every query, for both analysis and visualization purposes. To that end, we will use Multidimensional Scaling (MDS) [24], that can take a distance matrix as input and heuristically produce n-dimensional vectors that comply with that distance matrix as close as possible. The result is not always accurate, as it is stochastic and depends on the geometric properties of these distances and the number of dimensions we choose, but it provides a good spatial approximation of the relationships of the queries.

In order to visualize query distance distribution across the similarity algorithms, and how the query clusters are laid out, we calculate a distance matrix of all queries in the dataset, and use MDS to produce 2-component vectors. We plot these vectors on a scatter plot, coloring the points according to their cluster (the seed query they originated from).

V. Evaluation Setup

We will experimentally evaluate the performance of our two algorithms **GESS** (section III.A) and **TESS** (section III.B) with that of **aouiche**, **aligon**, and **makiyama** (section II.F) and **LLM-based** (section III.D).

A. Algorithm Variants

The *TESS* and *GESS* algorithms are parameterized based on the embedding method they use. Using word2vec embedding with 200-element vectors, we have *TESS(vocab-200)* and *GESS(vocab-200)*, while when we use 768-element vectors we have *TESS(vocab-768)* and *GESS(vocab-768)* respectively. Using the BERT language model as the embedding method, we have the *TESS(bert)* and *GESS(bert)* algorithms.

Every similarity algorithm has the following variants:

- **Original**: The algorithm as originally formulated.
- **Reduced**: The similarity algorithm after we remove all collisions for the given dataset. The number of queries we have to remove because of collisions gives us the *collision-free ratio*.

- **Reanalyzed**: After we calculate a distance matrix of all queries of the dataset, we run MDS to produce 768-component vectors for the queries and recalculate similarity using Euclidean distance.

All these algorithms and their variants produce $9 \cdot 3 = 27$ combinations. We will not explore all of them at the same time, but we will identify which variants are the most useful to compare, through our experiments.

Additionally, we utilize the implementation of *aouiche*, *aligon* and *makiyama* similarity algorithms, as given by Kul et al. [10]. Kul et al added a query pre-regularization step to all three algorithms, which improves the cluster separability they produce. We also conducted experiments with the regularized versions. Our evaluation metrics did not change substantially, so we do not include them in this paper.

B. Evaluation Datasets

For our experiments, we utilize both spatiotemporal and simple databases across different domains. The main tests are performed on a subset of the Piraeus AIS dataset for large-scale maritime data analytics [25], a MobilityDB database with information about ship trajectories around the port of Piraeus. It has 10 tables, 63 columns, and 565812 tuples across all tables. We also constructed a database using OpenStreetMap [26] data from the metropolitan Pittsburgh area (8 tables, 22 columns, 5962963 tuples). For non-spatiotemporal domains, we performed experiments on databases from the BIRD text2sql benchmark [27], namely, the "movies_4" database (17 tables, 53 columns, 455336 tuples), the "olympics" database (11 tables, 32 columns, 879229 tuples), and the "synthea" database (11 tables, 85 columns, 146070 tuples).

The similarity algorithms are compared based on a dataset of clusters of SQL queries. When we construct a dataset, we start by defining a small number of seed queries, and for every one of them, we apply the simple modifications we discussed above, to produce a cluster of similar queries. The more structurally different the seed queries are to each other, the more distant the entire clusters will be from one another. We avoided crafting the seed queries ourselves, to eliminate possible bias. The Piraeus AIS database, as well as the BIRD databases, have a set of SQL queries we can select seed queries from. For the OpenStreetMap database, we used generative AI to generate seed queries that are structurally different.

C. Evaluation Queries

We used the following query workloads in our experiments:

- **piraeus** - The Piraeus AIS database, 14 complicated spatiotemporal seed queries, and 345 queries in total.
- **piraeus-comp** - Subset of the *piraeus* dataset with queries that can be recognized by *aouche*, *aligon*, and *makiyama* implementations (6 seed queries – 175 total queries)
- **pitt** - The OpenStreetMap database (10 seed queries – 3493 queries total).
- **bird-movies_4** - The "movies_4" database from the BIRD benchmark (10 seed queries – 1371 queries total).
- **bird-olympics** - The "olympics" database from the BIRD benchmark (10 seed queries – 1030 queries total).
- **bird-synthea** - The "synthea" database from the BIRD benchmark (10 seed queries – 1873 queries total).

D. Metrics

Throughout our experiments, we compare query similarity algorithms using the following metrics:

Silhouette index *(higher is better)* - The silhouette index is the CVI we consider the most important, as it is well established in the literature, it is less prone to outliers, and works with only distances, since for most algorithms we do not have geometric vectors.

Davies-Bouldin Index and **Calinski-Harabasz Index** *(higher is better)* - The Davies-Bouldin index and Calinski–Harabasz index are CVIs that rely on having geometric points, so they are only available on algorithm variants that involve query vectors. Both are sensitive to outliers.

Intra-cluster normalized distance - The normalized intra-cluster distance indicates how spread apart the queries of a cluster are. It is calculated by dividing the average intra-cluster query distance by the average distance of all pairs of queries in the dataset. This is not directly a metric of separability, but an indication of how a similarity algorithm differentiates similar queries. In fact, it correlates negatively with cluster separability. A small intra-cluster distance can increase cluster separability, but very small values indicate poor separation among similar queries and can lead to collisions.

Triangle violations ratio *(lower is better)* - The ratio of query combinations that violate the triangle inequality. Algorithms that rely on vectors should always have a violations ratio of 0, since the query distances are Euclidean distances of their vectors. For algorithms that have triangle inequality violations, it is less reliable to compare the distances between different pairs of queries.

Collision-free ratio (CFR) *(higher is better)* - The collision-free ratio indicates how well the algorithm captures the semantics of the queries and how well it can distinguish similar queries. The original algorithm variants do not have a collision-free ratio. Only the *reduced* variants have collision-free ratios, since it is calculated based on the number of queries that had to be removed to avoid collisions.

Dunn Index pairwise *(higher is better)* - A variant of the Dunn-Index CVI [21] that we devised, that does not require cluster centroids, operating purely on distances. Since it is not an established metric, we simply consider it as a complementary indication of cluster separability.

E. Algorithm performance

Generating similarity across all pairs of queries in a query pool of size n requires n^2 operations, but each family of similarity algorithms has specific performance implications when adding a new query to an existing pool.

aouiche, **aligon** and **makiyama** require comparing the new query to all existing ones, mandating parsing and extracting features from all of them, which is n operations.

llm-based also requires comparing the new query with all of them, but every comparison needs n LLM inference operations, making the cost prohibitive.

GESS produces a vector for the new query, which is a single operation. Its computational cost is the cost of the embedding algorithm, which can be more expensive if we are using the *bert* embedding method, but it does not depend on n.

TESS produces a vector, which is a single operation. Additionally, since the vector is a combination of vectors of query parts, we can use caching to store the embedding vectors of all columns, common values, common predicates, subqueries, and common clauses, significantly speeding up the calculations. Using caching is paramount in contextual embedding methods, like *bert*, but it may be redundant on vocabulary-based embedding methods like *vocab-200* and *vocab-768*, as they produce vectors through look-up operations.

Constructing a distance matrix for the whole pool is potentially faster on vector-based algorithms, as there exist fast and hardware-accelerated libraries for vector operations.

VI. Evaluation Results

A. Piraeus-comp dataset experiment (Table II, Figure 2, Table III, Figure 3

Setup: We first ran all similarity algorithms on the **piraeus-comp** dataset in order to establish how the algorithms and their variants behave in terms of the evaluation metrics. Table II shows the breakdown of all evaluation metrics for the algorithm variants. The algorithms with our label tree vector combination approach, the *TESS* algorithm variants, are run with $semantic\ weight = 0.5$ and $depth\ weight = 0.5$.

Observations: The *collision-free ratio* of the *aouiche* algorithm is significantly low, with *aligon* and *makiyama* having the second lowest values. *Aouiche* has a *collision-free ratio* of 0.417, which means that most queries, around 58% of them, are considered identical, or near identical, indicating that the algorithm misses much of the semantics of the queries when it compares them. This will be made more apparent on the other datasets.

The reduced variants of *aouiche* and *aligon* always appear to have lower cluster separability, as it's reflected on their CVIs. This is expected, as many queries of a cluster are produced from the seed query by changing a constant value to alternative values. Since *aouiche*, *aligon* and *makiyama*, and probably many other existing algorithms, ignore constants, these queries are going to be considered identical to the seed queries, making the clusters appear more concentrated than they are. As a consequence, it is wrong to include these apparent duplicate queries if we want to assess how well these algorithms perform when it comes to cluster separability, as these duplicates pollute the calculations and significantly inflate the value of the CVIs. In order to address this, going forward, we will only consider the metrics of the reduced variants of all algorithms, as they are guaranteed to not have any collisions, making the comparisons more fair.

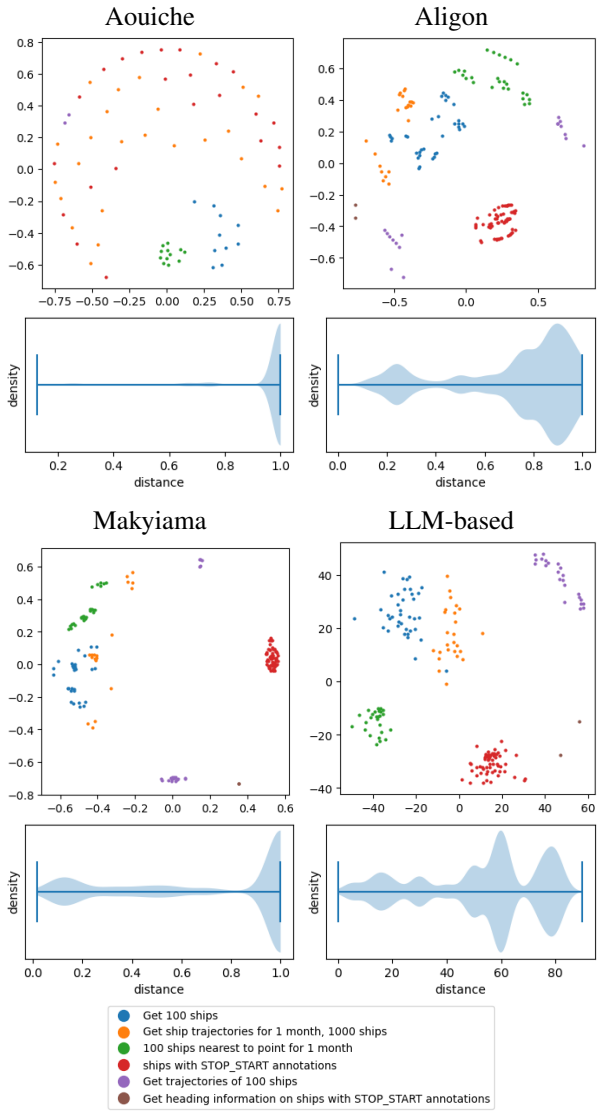

Fig. 2: Piraeus-comp dataset, visualized clusters and distance distribution for *aouiche*, *aligon*, *makiyama* and *LLM-based* similarity algorithms

For most cases, the *reanalyzed* variants of algorithms are decisively similar to the *reduced* variants, therefore, we will omit most *reanalyzed* variants, including only a few, in order to compare CVIs that require vectors.

Figure 2 shows the MDS-based visualizations of the *aouiche*, *aligon*, *makiyama* and *LLM-based* algorithms. The scatter plots display queries as points, color-coded by the seed query that produced them. The legend provides descriptions of the seed queries. The figure displays the distance distribution for every algorithm. We observe that largely, the queries fall into well-separated clusters across all algorithms, as the task is relatively easy, with only six clusters of distant seed queries. We also observe that the distance distributions are heavily right-skewed, indicating a bias towards consistently producing inflated distances, even for seemingly similar queries.

TABLE II: Piraeus-comp dataset, 6 query clusters, original, reduced, and reanalyzed variants of all algorithms

	Silhouette Index	Dunn Index pairwise	Davies-Bouldin Index	Calinski-Harabasz Index	Intra-cluster normalized distance	Violations ratio	Collision-free ratio
aouiche	0.359	0.000			**0.800**	0.000	
aouiche-reduced	0.132	0.500			0.714	0.000	0.417
aouiche-reduced-reanalyzed	0.105	0.564	**3.759**	4.881	0.783	0.000	
aligon	0.468	0.250			0.550	0.002	
aligon-reduced	0.459	0.250			0.574	0.002	0.914
aligon-reduced-reanalyzed	0.440	0.494	1.181	64.856	0.672	0.000	
makiyama	0.597	0.191			0.287	0.007	
makiyama-reduced	0.612	0.191			0.309	0.006	0.914
makiyama-reduced-reanalyzed	0.587	0.325	1.164	226.629	0.459	0.000	
llm	0.591	0.333			0.366	**0.029**	
llm-reduced	0.591	0.333			0.366	**0.029**	**1.000**
llm-reduced-reanalyzed	0.569	0.420	0.745	196.219	0.436	0.000	
TESS(vocab-200)	0.721	**0.866**	0.340	**687.564**	0.212	0.000	
TESS(vocab-200)-reduced	**0.824**	**0.866**	0.340	**687.564**	0.212	0.000	0.960
TESS(vocab-200)-reduced-reanalyzed	0.709	0.339	0.468	563.236	0.437	0.000	
GESS(vocab-200)	0.564	0.159	0.838	130.163	0.367	0.000	
GESS(vocab-200)-reduced	0.576	0.159	0.821	121.770	0.372	0.000	0.960
GESS(vocab-200)-reduced-reanalyzed	0.553	0.159	0.964	115.346	0.465	0.000	
TESS(vocab-768)	0.715	0.853	0.348	655.622	0.217	0.000	
TESS(vocab-768)-reduced	0.821	0.853	0.348	655.622	0.217	0.000	0.960
TESS(vocab-768)-reduced-reanalyzed	0.702	0.332	0.481	535.649	0.448	0.000	
GESS(vocab-768)	0.563	0.166	0.822	130.416	0.373	0.000	
GESS(vocab-768)-reduced	0.574	0.166	0.810	122.477	0.378	0.000	0.960
GESS(vocab-768)-reduced-reanalyzed	0.552	0.165	0.927	116.013	0.475	0.000	
TESS(bert)	0.594	0.296	0.699	515.731	0.262	0.000	
TESS(bert)-reduced	0.594	0.296	0.699	515.731	0.262	0.000	**1.000**
TESS(bert)-reduced-reanalyzed	0.570	0.203	0.810	444.698	0.444	0.000	
GESS(bert)	0.415	0.300	0.988	81.286	0.517	0.000	
GESS(bert)-reduced	0.415	0.300	0.988	81.286	0.517	0.000	**1.000**
GESS(bert)-reduced-reanalyzed	0.399	0.341	1.128	79.508	0.594	0.000	

Another observation we can make is that the *LLM-based* similarity plot appears to have more points and better-defined clusters. This is also reflected on its *collision-free ratio* and *silhouette index*, on Table II, and it is the result of capturing the semantics of the queries better, indicating a superior algorithm.

On Figure 3, we see the visualizations of our algorithm variants, across label-tree-based vs global-based vectors, and across the three embedding methods that we use. We can see that the query points are numerous, similar to *LLM-based* similarity, and also that the distance distributions are more varied, compared to the other algorithms; since we have query vectors and Euclidean distances, we don't have a bias towards inflated distances here. We can also see that the *GESS* variants show much less concentrated clusters. Another important observation is that the outcome for algorithms that use vocab-200 embedding is similar to that for vocab-768 embedding, while having a significantly lower memory footprint and processing cost.

B. Piraeus dataset experiment (Table III)

Since the implementation of the *aouiche*, *aligon*, and *makiyama* algorithms don't take into account syntactic idiosyncrasies of PostGIS and MobilityDB SQL queries, if we include more spatiotemporal complicated queries, we can't compare our similarity algorithms to those baselines. Therefore, we will only show results for our TESS and GESS algorithm variants. We can observe that increasing the number of query clusters reduces the value of most CVIs (Table III), as is expected since we divide the same vector space across more clusters, but the metrics trends we observed on *piraeus-comp* hold here, too. Also, the collision-free ratio is still comparable to having only six clusters.

C. OpenStreetMaps/Pitt dataset experiments (Table IV)

Setup: We ran the similarity algorithms on the *pitt* dataset; the evaluation metrics are in Table IV. We present results only for the *reduced* variants of most algorithms as the base variants suffer from collisions, and the *reanalyzed* variants perform similarly to the *reduced* variants, but require extra analysis and require full recalculation for adding new queries to the set.

TABLE III: Piraeus dataset, 10 query clusters, TESS and GESS algorithms

	Silhouette Index	Dunn Index pairwise	Davies-Bouldin Index	Calinski-Harabasz Index	Intra-cluster normalized distance	Violations ratio	Collision-free ratio
TESS(vocab-200)-reduced	**0.681**	**0.373**	0.420	**733.838**	0.132	**0.000**	0.957
GESS(vocab-200)-reduced	0.567	0.009	**0.954**	176.395	0.281	**0.000**	**0.962**
TESS(vocab-768)-reduced	0.671	**0.373**	0.430	722.621	0.134	**0.000**	0.957
GESS(vocab-768)-reduced	0.567	0.009	0.940	173.430	**0.287**	**0.000**	**0.962**

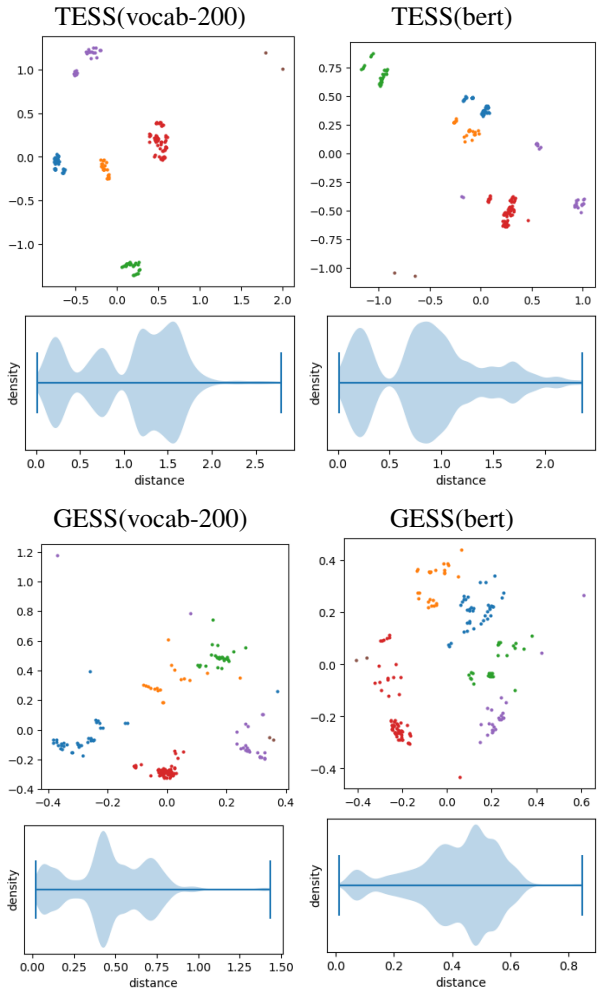

Fig. 3: Piraeus-comp dataset, visualized clusters and distance distribution for our similarity algorithms

We only include *reanalyzed* variants for *makiyama* to compare CVIs that require vectors.

Observations: In contrast to the *piraeus-comp* dataset, *makiyama* had a better silhouette index than *TESS* (0.762 vs. 0.719), with *GESS* between them (0.731), but *TESS* did better on other CVIs. However, even with comparable cluster separability, *aouiche*, *aligon*, and *makiyama* had abysmal *collision-free ratios* (less than 0.1). This means that these algorithms could not differentiate more than 90% of the queries!

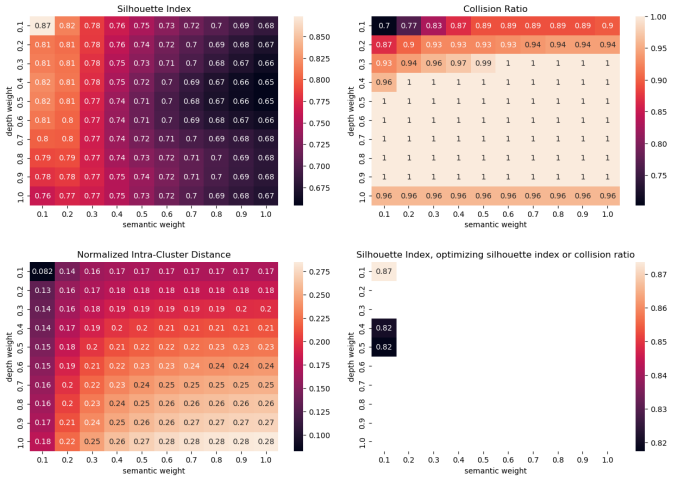

Fig. 4: Piraeus-comp dataset, weights sensitivity analysis, heatmaps of silhouette index, collision-free ratio and normalized intra-cluster distance

D. Other dataset experiments

We also run experiments on the *bird-movie_4*, *bird-olympics*, and *bird-synthea* datasets, which are not spatiotemporal but semantically rich. We observed that both *TESS* and *GESS* appear to be far superior to the other approaches on the collision-free ratio. Our *TESS* algorithm was competitive with *makiyama* on the silhouette index and the Dunn Index, and outperformed it on the Calinski-Harabasz Index (Figure 5).

E. Sensitivity analysis (Figure 4)

Setup: As our *TESS* family of algorithms is parameterized by the *depth weight* and *semantic weight*, it is important to evaluate how the behavior of the algorithm is affected. To that end, we performed a sensitivity analysis on the piraeus-comp dataset, using *vocab-200* and *vocab-768* embeddings. We do a parameter sweep of both weights for values from 0.1 to 1.0, and show how the *silhouette index*, *collision-free ratio*, and *normalized intra-cluster distance* are affected and correlated, a limited form of an ablation study.

We want to identify weight combinations that optimize at least one of these metrics. This way, any weight combination that fails to produce at least one optimal metric can be discarded. These *optimizing configurations* should boost cluster separability metrics beyond their values under $w_s = 0.5$ and $w_d = 0.5$, without sacrificing the high *collision-free ratio*.

TABLE IV: OpenStreetMaps/Pitt dataset, 10 query clusters, TESS and GESS algorithms

	Silhouette Index	Dunn Index pairwise	Davies-Bouldin Index	Calinski-Harabasz Index	Intra-cluster normalized distance	Violations ratio	Collision-free ratio
aouiche-reduced	0.408	0.333			0.412	0.000	0.017
aligon-reduced	0.489	0.086			**0.441**	0.000	0.047
makiyama-reduced	**0.762**	0.159			0.152	**0.056**	0.052
makiyama-reduced-reanalyzed	0.738	0.243	0.529	519.723	0.189	0.000	
TESS(vocab-200)-reduced	0.719	**0.388**	**0.682**	7384.280	0.269	0.000	0.988
GESS(vocab-200)-reduced	0.731	0.331	0.525	**7651.707**	0.249	0.000	**0.990**

TABLE V: Wall-clock similarity calculation performance on the *bird-movie_4* dataset (1371 queries; 1,878,270 query pairs). "Single pair" = time to compute similarity of one pair of queries; "Distance matrix" = time to compute all pairwise distances for the full dataset.

Algorithm	Single query (ms)	Distance matrix (s)
aouiche	0.043	82
aligon	0.052	97.3
makiyama	0.010	19.8
GESS(vocab-200)	0.006	10.45
GESS(vocab-768)	0.015	28.26
GESS(bert)	0.099	185.7
TESS(vocab-200)	0.003	5.98
TESS(vocab-768)	0.013	23.56
TESS(bert)	0.019	35.2

TABLE VI: Wall-clock similarity calculation performance on the *piraeus-comp* dataset (175 queries; 30,450 query pairs). "Single pair" = time to compute similarity of one pair of queries; "Distance matrix" = time to compute all pairwise distances for the full dataset.

Algorithm	Single query (ms)	Distance matrix (s)
aouiche	0.121	3.67
aligon	0.151	4.61
makiyama	0.093	2.83
GESS(vocab-200)	0.021	0.634
GESS(vocab-768)	0.029	0.869
GESS(bert)	1.006	30.631
TESS(vocab-200)	0.006	0.176
TESS(vocab-768)	0.014	0.414
TESS(bert)	0.494	15.035

Observations: We show heat maps of all three metrics across the parameter space for the *piraeus-comp* (Figure 4) dataset. The bottom-right heatmap displays the silhouette index, but only for weight combinations that optimize the *silhouette index* or *collision-free ratio*, as we consider these to be the most important metrics.

As both weights approach zero, the *collision-free ratio* decreases. This is expected: a semantic weight of zero ignores column and constant embeddings entirely, collapsing structurally similar queries to identical vectors, while a depth weight of zero discards everything below the SELECT list. Conversely, the *normalized intra-cluster distance* increases with both weights, since query variations largely involve changing constants and columns at different depths, and these changes affect the aggregate vector proportionally to the weights.

The silhouette index generally increases for smaller weights, because cluster separability is inversely correlated with intra-cluster distance. However, this creates a tension: too small an intra-cluster distance increases collisions, while too large a value reduces separability. We therefore want to jointly maximize all three metrics, and plot their combinations as a function of the weights (Figure 4).

We generally want to keep all three metrics as high as possible, so we can plot the combinations of these values, and how they correspond to the weights.

Takeaway: The weight combination $semantic\ weight = 0.5$ and $depth\ weight = 0.5$ that we used for the main experiments, appears to be a reasonable configuration, but we observe that, depending on the dataset and its size, there are weight combinations that can further increase cluster separability (Figure 5), without sacrificing the *collision-free ratio*, but that would require prior analysis to identify the optimal weights.

F. Wall-Clock Performance (Table V, Table VI)

We ran the similarity calculations for the piraeus-comp and bird-movie_4 datasets multiple times to get an estimate of the runtime performance for the similarity calculations. The experiments run on a personal laptop, with 8 cores at 2.2 GHz, and 16 GB of RAM, mostly on a single core. Since *aouiche*, *aligon*, and *makiyama* do not have any initialization calculations, their runtime reflects the time it takes to calculate similarity across all pairs of queries in the dataset. In contrast, the variants of GESS and TESS have a lot of up-front calculations that have to run before processing any queries, in order to speed-up value embedding calculations. In our experiment configuration, we run initial expensive calculations for analyzing the database before processing any queries. These calculations are only performed once, and scale with the size of the database and complexity of embedding method, but most can be omitted at the expense of the processing time of every query, if initial run time or disk space is a concern. For the *bird-movie_4* database, it took 455 seconds to ini-

TABLE VII: Similarity Algorithm Feature Comparison

Feature	aouiche	aligon	makiyama	TESS	GESS	llm-based
considers semantic content	✗	✗	✗	✓	✓	✓
considers table and column names	✗	✗	✗	✓	✓	✓
differentiates query constants	✗	✗	✗	✓	✓	✓
euclidean space, respecting triangle inequality	✗	✗	✗	✓	✓	✗
only vector distance comparisons	✗	✗	✗	✓	✓	✗
low rate of collisions	✗	✗	✗	✓	✓	✓
non-inflated separability of different queries	✗	✗	✗	✓	✓	✗
configurability of structural vs semantic separability	✗	✗	✗	✓	✗	✗
handles spatiotemporal queries natively	✗	✗	✗	✓	✓	✓
computational complexity of distance matrix	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2)$
computational complexity of adding a new query	$O(n)$	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$
distance calculation	pairwise	pairwise	pairwise	vector distance	vector distance	pairwise
overall performance (section V.E)	fastest	fastest	fastest	faster	fast	prohibitive

tialize for *vocab-200* embedding, 437 seconds for *vocab-768* embedding and 13573 seconds for *bert* embedding. For the *piraeus-comp* database, initialization took 474, 474, and 20708 seconds respectively. When running the similarity calculations, for *vocab-200* and *vocab-768*, the bottleneck was calculating the distance matrix, while producing the query vectors was faster. In contrast, for *bert* embedding, producing the query vectors was significantly slower. Table V and Table VI show the similarity calculation time for all algorithms across both datasets. TESS and GESS have significant start-up time before processing any queries, but for per-query and per-query-pair calculation time, the TESS algorithm with *vocab-200* embedding appears to demonstrate superior performance.

VII. Conclusions (Figure 5, Table VII)

We conclude that the *TESS* family of algorithms have cluster separability comparable to *makiyama*, exceeding it on some datasets, while maintaining superior *collision-free ratio* across the board.

The LLM-based algorithm remains the gold standard for both metrics, but its inference cost makes it impractical at scale. *TESS* and *GESS* are well-performing alternatives that do not share this limitation

Among all algorithms compared, TESS has the best overall properties for query recommendation (Table VII): customizability, scalable performance, geometric consistency, semantic capture, and a low collision rate. For spatiotemporal queries in particular, *TESS* and *GESS* were the only algorithms capable of handling complex PostGIS/MobilityDB syntax.

In summary, this work introduced *Tree Embedding SQL Similarity (TESS)* and *Global Embedding SQL Similarity (GESS)*, a novel family of embedding-based SQL query similarity algorithms well-suited to spatiotemporal query recommendation, together with an evaluation framework based on cluster separability and collision-free ratio.

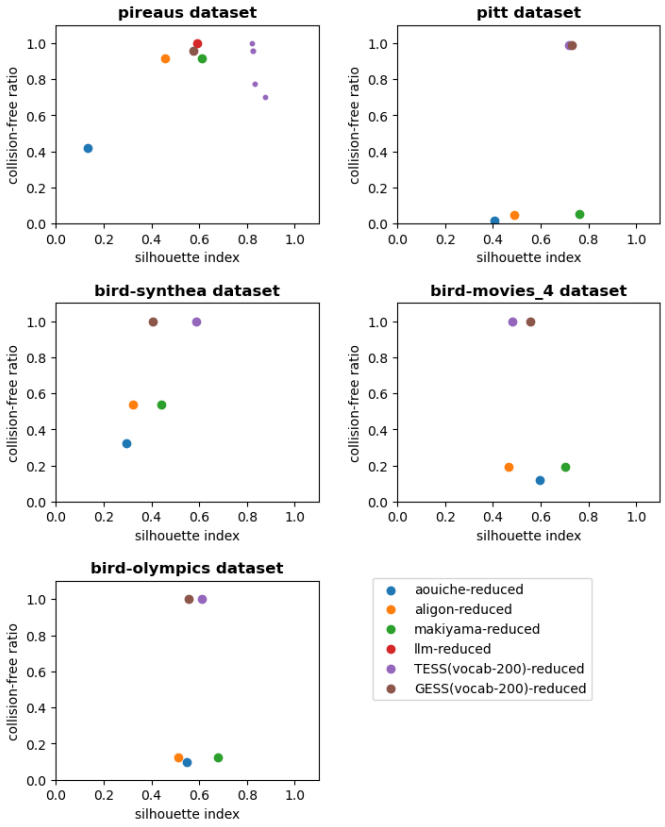

Fig. 5: Silhouette index vs collision-free ratio per dataset

VIII. Limitations and Future Work

Several directions remain open. First, our benchmark constructs query clusters through systematic syntactic modifications of seed queries; while this gives controlled ground truth, evaluation on real user query logs would provide stronger evidence of broader generalizability. Second, normalizing SQL queries to a canonical logical form (e.g., a relational-algebra-

style representation) before embedding could help the method treat logically equivalent queries as more similar, even when they differ in syntactic formulation. Finally, human annotation of query similarity on a subset of our datasets would provide a gold standard independent of LLM judgment, which, while practical, lacks formal guarantees of correctness.

ACKNOWLEDGMENTS

We would like to thank Brian Nixon and Panos K. Chrysanthis for their help with earlier drafts of this paper.

REFERENCES

[1] S. Chu, B. Murphy, J. Roesch, A. Cheung, and D. Suciu, "Axiomatic foundations and algorithms for deciding semantic equivalences of sql queries," *Proceedings of the VLDB Endowment*, vol. 11, 7 2018.

[2] R. Marcus *et al.*, "Neo: A learned query optimizer," *Proceedings of the VLDB Endowment*, vol. 12, pp. 1705–1718, 7 2018.

[3] Y. Zhao, G. Cong, J. Shi, and C. Miao, "Queryformer: A tree transformer model for query plan representation," *Proceedings of the VLDB Endowment*, vol. 15, pp. 1658–1670, 4 2022.

[4] X. Tang, S. Wu, M. Song, S. Ying, F. Li, and G. Chen, "Preqr: Pretraining representation for sql understanding," *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 204–216, 6 2022.

[5] D. Paul, J. Cao, F. Li, and V. Srikumar, "Database workload characterization with query plan encoders," *Proceedings of the VLDB Endowment*, vol. 15, pp. 923–935, 12 2021.

[6] N. Arzamasova, K. Bohm, B. Goldman, C. Saaler, and M. Schaler, "On the usefulness of sql-query-similarity measures to find user interests," *IEEE TKDE*, vol. 32, 10 2020.

[7] K. Zhang, R. Statman, and D. Shasha, "On the editing distance between unordered labeled trees," *Information Processing Letters*, vol. 42, 5 1992.

[8] I. Bordino, C. Castillo, D. Donato, and A. Gionis, "Query similarity by projecting the query-flow graph," *SIGIR 2010 Proceedings*, 2010.

[9] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *ICLR 2013 - Workshops*, 1 2013.

[10] G. Kul, D. T. A. Luong, T. Xie, V. Chandola, O. Kennedy, and S. Upadhyaya, "Similarity metrics for sql query clustering," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, 12 2018.

[11] K. Aouiche, P. E. Jouve, and J. Darmont, "Clustering-based materialized view selection in data warehouses," *LNCS*, vol. 4152 LNCS, 2006.

[12] J. Aligon, M. Golfarelli, P. Marcel, S. Rizzi, and E. Turricchia, "Similarity measures for olap sessions," *Knowledge and Information Systems*, vol. 39, 3 2014.

[13] V. H. Makiyama, J. Raddick, and R. D. C. Santos, "Text mining applied to sql queries: A case study for the sdss skyserver," *Symposium on Information Management and Big Data*, 2015.

[14] R. Řehůřek and P. Sojka, "Software Framework for Topic Modelling with Large Corpora," in *Proc of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. ELRA, May 2010.

[15] T. Wolf *et al.*, "Transformers: State-of-the-art natural language processing," in *Proc of EMNLP: System Demonstrations*. ACL, Oct. 2020.

[16] R. Řehůřek and P. Sojka. (2018) Gensim data. [Online]. Available: https://github.com/piskvorky/gensim-data

[17] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *NAACL HLT 2019*, vol. 1, 10 2018.

[18] O. AI. (2024) gpt-4o-mini. [Online]. Available: https://platform.openai.com/docs/models/gpt-4o-mini

[19] T. Schmidt *et al.*, "Sqlstorm: Taking database benchmarking into the llm era," *Proceedings of the VLDB Endowment*, vol. 18, pp. 4144–4157, 7 2025. [Online]. Available: https://dl.acm.org/doi/10.14778/3749646.3749683

[20] P. J. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis," *Journal of Computational and Applied Mathematics*, vol. 20, 11 1987.

[21] J. C. Dunn, "A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters," *Journal of Cybernetics*, vol. 3, 1 1973.

[22] D. L. Davies and D. W. Bouldin, "A cluster separation measure," *IEEE Transactions on PAMI*, vol. PAMI-1, 1979.

[23] T. Caliński and J. Harabasz, "A dendrite method for cluster analysis," *Communications in Statistics*, vol. 3, 1974.

[24] I. Borg and P. Groenen, "Modern multidimensional scaling: Theory and applications," *Journal of Educational Measurement*, vol. 40, no. 3, 2003.

[25] A. Tritsarolis, Y. Kontoulis, and Y. Theodoridis, "The piraeus ais dataset for large-scale maritime data analytics," *Data in Brief*, vol. 40, 2 2022.

[26] OpenStreetMap contributors, "Planet dump retrieved from https://planet.osm.org ," https://www.openstreetmap.org, 2017.

[27] J. Li *et al.*, "Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls," in *Proc of the NeurIPS*, 12 2024.