Evaluating SQL Selection/Projection over Table Embeddings

Mariam Mellouli
EURECOM
Biot, France
mariam.mellouli@eurecom.fr

Paolo Papotti
EURECOM
Biot, France
paolo.papotti@eurecom.fr

ABSTRACT

Word embeddings are a powerful technique for representing and analyzing textual data in natural language processing (NLP) tasks. Notably, they possess a word-analogy property that represents real-world relationships through geometric operations in the embedding space. We study this property in the setting where embeddings are generated from relational databases. Our study aims to determine whether existing methods to obtain embeddings of relational tables preserve the inherent relationships of the relational model, akin to the word-analogy property in natural language text. By treating the learned vector space itself as an execution substrate, we ask a simple yet unexplored question: can an embedding faithfully stand in for the DBMS when answering basic SQL? To test this hypothesis, we develop a framework that assesses the capability of embeddings of relational data to answer SQL queries involving selection and projection. This framework encompasses the generation of embeddings, the querying of these embeddings through SQL, and the evaluation of the results. Our findings indicate that embedding methods that pretrain on the table can capture and reflect the original table/row/attribute relationships.

VLDB Workshop Reference Format:
Mariam Mellouli and Paolo Papotti. Evaluating SQL Selection/Projection over Table Embeddings . VLDB 2025 Workshop: Tabular Data Analysis (TaDA).

VLDB Workshop Artifact Availability:
The source code, data, and/or other artifacts will be made available

1 INTRODUCTION

Word embeddings are an essential tool in natural language processing (NLP), offering a continuous vector space representation of words that captures both semantic and syntactic relationships. These embeddings are notable for their word-analogy property, where real-world relationships can be effectively represented through geometric operations.

In this study, we focus on methods for generating embeddings from relational databases [8]. Recent works demonstrate that embedding relational data enables several applications, such as data integration [4] and semantic type detection [16], and even novel SQL operators that surpass the limitations of symbolic reasoning, allowing for semantic operations [3]. For instance, queries that leverage word embeddings to perform semantic similarity joins and analogy-based queries, extending the querying capabilities beyond traditional SQL, such as

```
SELECT X.title, Y.title
FROM papers X, papers Y
WHERE X.title < Y.title AND proximity(X.tid, Y.tid) > 0.3
```

where *tids* are for the tuple ids and *proximity* is a function over their embeddings.

Specifically, we investigate whether the relational semantics inherent in databases can be preserved by embeddings, similar to the word-analogy property observed in natural language text. For this study, we develop a framework designed to test whether embeddings generated from relational data can be used to answer simple SQL queries involving selection and projection.

Our contributions are the following:

- We introduce the problem of measuring how embeddings model the relational properties of the underlying data in the setting of relational data.
- We introduce a methodology for the study which pivots on executing a class of SQL queries to probe the embeddings. We deploy the methodology in an evaluation framework.
- We examine multiple methods for generating embeddings from relational tables, measuring the performance of the embeddings against ground truth data retrieved directly from the relational database.
- We analyze the limitations of the existing methods and provide insights into their shortcomings.

We conclude discussing the implications of our research.

Related Work. Several papers have shown how to leverage word embedding techniques in the context of relational databases to capture meaningful relationships [3, 5, 17]. While they exploit relationships encoded in word embeddings for target applications, or analyze properties of table embeddings such as row and column order sensitivity, we focus on examining how well these embeddings preserve relational semantics for the purpose of answering SQL queries, which offers an alternative to traditional text-to-SQL generation approaches [12, 15]. Learning tabular representations has been explored also for more powerful models, such as transformers [1]. However, we focus on word embeddings due to their computational efficiency, which enables the pre-training of the embeddings for the dataset at hand.

2 PRELIMINARIES

Word Analogy In Embeddings. Word analogy tasks in word embeddings identify and model semantic and syntactic relationships between words using their vector representations. These tasks capture both the underlying meanings and contextual nuances of words within a given language corpus.

Proceedings of the VLDB Endowment. ISSN 2150-8097.

A classic example illustrating this concept is the analogy "king - man + woman = queen" [11]. By performing vector arithmetic on word embeddings, it is possible to extract inherent relationships between words. In this example, subtracting the vector representation for "man" from the vector for "king," and then adding the vector for "woman," results in a vector that is close to the vector for "queen." This shows the embeddings' ability to model word relationships through simple geometric operations in the vector space.

Embeddings for Relational Data. Recent advancements in pre-trained language models for NLP tasks have spurred similar progress in embedding relational tables for applications such as table question answering and semantic column type annotation. The majority of these models extend foundational language models, such as BERT, and are specialized to account for the inherent structure of tables, often leveraging vertical attention mechanisms to incorporate information across rows.

TaBERT was one of the pioneering models to extend LMs to tabular data by integrating token-level embeddings with additional positional embeddings, using vertical attention to capture inter-row information with a masked column name prediction objective [14]. Following TaBERT, other models like TURL [6], TAPAS [9], and TaPEx [10] have been proposed. Alongside these transformer-based approaches, other methods focus on representing tables as graphs and applying graph embedding techniques (e.g., Node2Vec) to learn representations for entities like rows, cells, or attributes [8, 13]. These methods produce distinct vector representations for different structural components of the table, which is relevant to our study.

Different downstream applications need distinct types of embeddings, depending on the level of aggregation in the table structure. For instance, semantic column type detection relies on column embeddings, whereas entity matching requires row embeddings [7]. Due to these varying requirements, in this work we distinguish *cell* embeddings from *row* and *attribute* embeddings, according to their aggregation w.r.t. the table structure,

Problem Statement: Relational Semantic Preservation. Given a relational table R composed of attributes A, tuples T, and values V, and an embedding generation method E, the problem is to determine how well E preserves the relational semantics.

Specifically, for a given SQL query Q involving selection and projection, the task is to measure the performance P of the embeddings in terms of precision-recall metrics when these queries are executed over the embeddings instead of the original relational table R.

3 FRAMEWORK

To assess how well table-derived embeddings can answer SQL queries, we develop a framework consisting of three main stages:

(1) **Embedding Generation:** Relational data is transformed into a vector space using existing embedding methods. For our framework, these methods must produce distinct embeddings for individual rows and cell values.

(2) **Embedding-Space Query Emulation:** We introduce a two-step process (Selection and Projection) that translates a restricted class of SQL queries into operations within the embedding space. This process aims to identify relevant row embeddings and then project out desired attribute values.

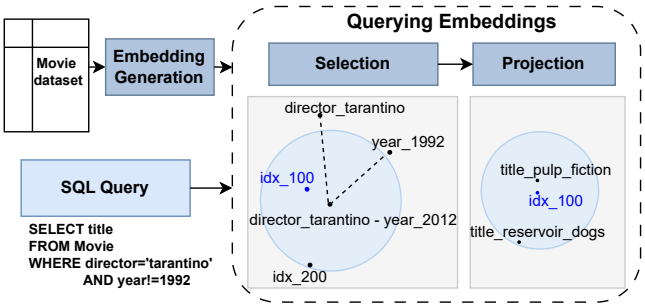

Figure 1: Overview of the framework for executing SQL on table embeddings. The process begins with an input SQL query and a relational dataset. (1) Embedding Generation: The relational data is transformed into embeddings for rows and cell values. (2) Querying Embeddings (Selection): The conditions from the SQL `WHERE` clause are converted into embedding vectors and combined to form a query vector. This vector is used to retrieve the top-k most similar row embeddings (e.g., `idx_100`, `idx_200`). (3) Querying Embeddings (Projection): For each selected row embedding, the framework identifies the closest attribute-value embedding corresponding to the target attribute in the `SELECT` clause.

(3) **Evaluation:** The results obtained from the embedding-space query emulation are compared against the ground truth results from executing the original SQL query on the database.

We now detail the core of our framework — the emulation of SQL queries in the embedding space— as illustrated in Figure 1. Conceptually, the pipeline emulates in the embedding space the classic two-phase query-planner: a coarse geometric filter (selection) followed by a fine-grained value reconstruction (projection).

Selection Step: Constructing the Query Vector. The first step focuses on the conditional part of the SQL query (i.e., the `WHERE` clause) to construct a target query vector, $\mathbf{V}$. This vector $\mathbf{V}$ identifies a region of the embedding space close to the embeddings of rows (tuples) that satisfy the query conditions. We then use $\mathbf{V}$ to retrieve the top-k nearest row embeddings using a similarity metric, where k is a user-defined parameter for the number of results to retrieve.

For this initial framework, we consider SQL queries with `WHERE` clause. This can contain equality conditions on an attribute A_1 with value D_1 (e.g., `A1 = 'D1'`) and inequality conditions on an attribute A_2 with value D_2 (e.g., `A2 != 'D2'`). We assume that the embedding generation process provides embeddings for specific attribute-value pairs, denoted as $\text{emb}(A, D)$, representing the value D in the context of attribute A, e.g., emb(director, 'quentin_tarantino').

The query vector $\mathbf{V}$ is computed as a weighted combination of these attribute-value embeddings: $\mathbf{V} = w_1 \cdot \text{emb}(A_1, D_1) - w_2 \cdot \text{emb}(A_2, D_2)$. As in the word analogy task, the positive term gets $\mathbf{V}$ closer to row embeddings that share the characteristic D_1, while the negative term aims to move it away from those with D_2.

The weights w_1 and w_2 modulate the influence of each condition. The underlying principle is that rarer values (lower frequency) are generally more discriminative and thus should contribute more significantly to query vector $\mathbf{V}$. We derive these weights from the

inverse frequency of the values D_1 and D_2 in the active domain of the attributes scaled to a [0.1, 1.0] range to ensure that no single condition disproportionately dominates the query vector $\mathbf{V}$.

Selection Step: Identifying Candidate Row Embeddings. After computing the query vector $\mathbf{V}$, the next task is to identify row embeddings that are semantically close to it. We assume that the embedding generation process produces a unique vector representation for each row in the table. These row embeddings are identifiable, for instance, by a specific prefix such as "idx_" followed by a row identifier (e.g., "idx_101").

Using cosine similarity, we find the top-k row embeddings that are most similar to $\mathbf{V}$. Let these retrieved row embeddings be: $\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_k$, where each $\mathbf{r}_j$ is an embedding vector starting with the "idx_" prefix (e.g., $\mathbf{r}_1$ = embedding(idx_8)). These k row embeddings represent the candidate tuples most likely to satisfy the original SQL query's conditions.

Projection Step: Retrieving Target Attribute Values. This step extracts the value of a specific target attribute (e.g., "title") for each of the k candidate row embeddings $\mathbf{r}_j$ from the selection step.

Attributes are identifiable by the prefix corresponding to the attribute name followed by the value itself (e.g., "title_PulpFiction", "director_QuentinTarantino"). We then denote the embedding of value D_{target} for the target attribute A_{target} as $\text{emb}(A_{target}, D_{target})$. For each candidate row embedding $\mathbf{r}_j$ (where $j = 1, \ldots, k$):

(1) We search among all available attribute-value embeddings for the target attribute A_{target}. That is, we consider all embeddings of the form $\text{emb}(A_{target}, D_{val})$, where D_{val} is any value occurring in the column for attribute A_{target}.
(2) We find the attribute-value embedding $\text{emb}(A_{target}, D_j^*)$ from this set that is closest (using cosine similarity) to $\mathbf{r}_j$, with $D_j^* = \arg\max_{D_{val}} \text{cos\_sim}(\mathbf{r}_j, \text{emb}(A_{target}, D_{val}))$.

The value D_j^* is then taken as the projected result for the target attribute A_{target} for the j-th candidate row.

Example. Consider the query `SELECT title FROM movie WHERE director = 'quentin_tarantino' AND status != 'released'`

1. Selection Step.

- Target attribute for equality is A_1 = director with value D_1 = 'quentin_tarantino'. Similarly for target attribute for inequality.
- We obtain embeddings: emb(director, 'quentin_tarantino') and emb(status, 'released').
- Weights w_1 and w_2 are calculated based on inverse frequencies (e.g., 'quentin_tarantino' in the 'director' column) and scaled.
- $\mathbf{V}$ is computed: $\mathbf{V} = w_1 \cdot \text{emb}(\text{director}, \text{'quentin\_tarantino'}) - w_2 \cdot \text{emb}(\text{status}, \text{'released'})$.
- We search for row embeddings (prefixed with "idx_") most similar to $\mathbf{V}$. For $k = 3$, retrieved row embeddings are: $\{\mathbf{r}_1 = \text{emb(idx\_8)}, \mathbf{r}_2 = \text{emb(idx\_6)}, \mathbf{r}_3 = \text{emb(idx\_5)}\}$

2. Projection Step. We retrieve (A_{target} = title), for each embedding $\mathbf{r}_j$. For $\mathbf{r}_1$ = emb(idx_8), we search all $\text{emb}(\text{title}, D_{val})$ vectors. If emb(title, 'Django Unchained') is the most similar to emb(idx_8), this is the projected value. We repeat the process for $\mathbf{r}_2$, $\mathbf{r}_3$.

This methodology allows us to test the capability of different table embedding techniques to preserve relational semantics for answering SQL-like queries. By comparing the set of projected results obtained through this embedding-space emulation with the actual results from executing the SQL query on the database, we quantitatively assess the effectiveness of the embedding methods in reflecting the original relational structures.

4 EXPERIMENTAL SET-UP

To evaluate our framework, we conducted experiments using two datasets, embedding generation techniques, and a set of SQL queries.

Datasets. We used two relational datasets:

- **DBLP:** A bibliographic dataset of academic publications. We used a subset with 4 columns: `title`, `authors`, `venue`, and `year`. It contains 56k unique author names and 62k unique publications.
- **Movie:** A dataset of movies with 15 columns, including `title`, `director`, `genres`, and `year` It includes 17k unique director names and 42k unique movie titles.

These datasets provide a mix of textual and categorical data suitable for testing the capabilities of table embeddings.

Embedding Generation. A crucial step in our experiments is the generation of embeddings from the relational datasets. For this purpose, we employed EmbDI [4], a framework specifically designed for creating local embeddings from heterogeneous relational databases, which aligns with the requirements of our proposed query emulation method. We discuss why alternative methods failed at this end of the next section.

The core idea of EmbDI is to first transform a relational table into a graph structure. In this graph:

- Each row (tuple) is represented as a node with a prefix like "`idx__`" followed by the row identifier (e.g., "`idx__101`"). This provides the row embeddings (e.g., $\mathbf{r}_j$) required by our framework.
- Attribute names are also represented as nodes (e.g., with prefixes like "`tt__`" or "`tn__`").
- Individual cell values, in the context of their attribute, are also represented as nodes, e.g., a value D in attribute column A is represented as a node like "`cid__A__D`".

Once the graph is constructed, word embedding algorithms are applied to learn vector representations for these nodes. EmbDI uses random walks on this graph to generate sequences of nodes (sentences), which then serve as input to standard word embedding models like Word2Vec. This process encodes same-row, same-attribute, and other structural relationships, making the resulting embeddings inherently aware of the table's relational structure.[1]

Query Set. We executed SQL queries against the generated embeddings and compared the results with those obtained from the original database. The queries adhere to the structure:

```
SELECT A FROM B WHERE C1 = 'D1' AND C2 != 'D2';
```

Specific instances of `A`, `B`, `C1`, `D1`, `C2`, and `D2` were chosen from the datasets to create diverse test queries.

Evaluation Metrics. We use Precision and Recall, calculated at different cut-off points k (e.g., @5, @10, @20), corresponding to the top-k results retrieved by our framework's selection step.

[1]For generating the embeddings using EmbDI with Word2Vec, we used the following parameters: Embedding Dimension: 300; Window Size (for Word2Vec): 3.

A consideration in our evaluation is the potential for duplicate values in the projected results. While the selection step retrieves unique row embeddings ($\mathbf{r}_j$), the projection step operates independently for each row embedding. Two row embeddings can have the same attribute-value embedding as their closest match for the projection, leading to a value projected twice. Our precision metric, detailed in Appendix A.2, considers the set of unique values projected.

Table 1: Aggregated Precision (P) and Recall (R) scores for the selection/projection task on the Movie and DBLP datasets. Results for table embeddings generated using Word2Vec and Node2Vec (EmbDI framework). Metrics are evaluated at different row top-k retrieval thresholds (k=5, 10, and 20) in the selection step.

		@5		@10		@20	
	Method	R	P	R	P	R	P
Movie	word2vec	0.60	0.80	0.74	0.70	0.90	0.60
	node2vec	0.60	0.70	0.73	0.65	0.82	0.52
DBLP	word2vec	0.42	0.85	0.53	0.8	0.74	0.70
	node2vec	0.40	0.80	0.48	0.62	0.70	0.70

5 EXPERIMENTAL RESULTS AND ANALYSIS

Table 1 reports the aggregated precision and recall scores at $k \in \{5, 10, 20\}$ for queries on the Movie and DBLP datasets using alternative embedding algorithms, Word2Vec and Node2Vec, within EmbDI for generating the table embeddings.

On both dataset, Word2Vec and Node2Vec demonstrate comparable performance overall, with Word2Vec generally maintaining a lead in performance metrics across the different k values. The results indicate that our framework, instantiated with EmbDI embeddings, can achieve promising levels of precision and recall for the targeted SQL query patterns.

A closer examination of individual query results (Appendix A.1) reveals significant variations in performance depending on the query structure, particularly the attribute being projected. Queries projecting on attributes with high-cardinality values (e.g., `title` in the Movie dataset) yielded better outcomes - in these cases, the attribute-value embeddings are likely distinct and well-separated in the embedding space.

Conversely, queries projecting on attributes with low-cardinality values that have high occurrences (e.g., `year` or `status`) often performed poorly. We analyzed the intermediate outputs for these problematic queries and confirmed that the **Selection step** generally succeeded in identifying the correct set of ‘idx‘ row embeddings - the query vector **V** was effective in locating the relevant rows. However, in the **Projection step**, when projecting onto a common value (e.g., ‘year = 2012‘), the embedding for that specific attribute-value pair (e.g., ‘emb(year, ’2012’)‘) might be a "hub" or an average representation that is similarly close to many different row embeddings. Consequently, when trying to find the closest ‘emb(year, D_val)‘ to a specific row embedding $\mathbf{r}_j$, the model may struggle to distinguish the correct year for that particular row. Essentially, the geometric distinctiveness required for accurate projection is diminished for high-frequency, low-cardinality attribute values.

Why Alternative Embedding Methods Fell Short. We now discuss *why* other approaches did not reach comparable quality.

FastText enriches word vectors with character *n*-grams, a design that works well for orthographically similar tokens [2]. In our tabular setting, however, FastText overwhelmingly captures *column*-level regularities while under-representing links *within* a tuple. Concretely, the nearest neighbours of a vector `director_tarantino` are other `director_*` tokens, whereas the corresponding `idx_*` row identifiers are far away in the space. As the first step of our method requires jumping from an attribute–value pair to its host tuple, the selection phase fails outright for FastText embeddings:

TAPAS and TAPEX are pre-train bidirectional transformers over entire tables, injecting structural biases (segment, row, and column embeddings) through attention. Although powerful for natural–language question answering over small tables, they proved unsuitable here for two reasons.

First, both models cap the input at 512 tokens, which translates to only a handful of rows. Sub-sampling defeats the purpose of learning a global embedding space and, empirically, removed many ground-truth tuples from consideration.

Second, they model each word piece is context-dependent; there is no stable embedding we can extract for *“row 37”* or *“director = tarantino”* independent of a specific query. Averaging token vectors, an intuitive workaround, led to highly entangled representations and near-random retrieval.

We also experimented with handing the SQL string to TAPEX’s neural executor. While syntactically valid, the model (i) could not process tables larger than five rows and (ii) returned only single-token answers, preventing evaluation against full projection sets.

6 CONCLUSION

Our study recasts a decades-old relational question — ‘can you answer SQL?’ — in a vector-native setting, charting the first steps toward fully differentiable analytics. By coupling embeddings with a two-phase *selection–projection* procedure, we showed that word analogy reasoning can be repurposed for structured data retrieval.

Yet the experiments also exposed that projection accuracy deteriorates whenever the target attribute has semantically close values (e.g., years or ratings). The cosine distance between such values and their row embeddings becomes indistinguishable, causing the nearest-neighbour step to return spurious cells. projection still suffers from *value-level aliasing*. A natural mitigation is to move from a purely unsupervised projection to a *supervised* one, e.g., by training a small neural network that refines the attribute-value vectors given a handful of labelled query–answer pairs.

The negative findings over other methods to obtain embeddings also point to concrete research avenues. First, future off-the-shelf language or table models could encode tuple identity so that projections can pivot reliably between rows and attributes. Second, embeddings methods for tables should expose *static, disentangled* vectors for both rows and attribute–value pairs.

Overall, our results establish a baseline and outline the technical hurdles that next-generation table embeddings must overcome to deliver fully differentiable SQL over enterprise-scale datasets.

REFERENCES

[1] Gilbert Badaro, Mohammed Saeed, and Paolo Papotti. 2023. Transformers for Tabular Data Representation: A Survey of Models and Applications. *Trans. Assoc. Comput. Linguistics* 11 (2023), 227–249. https://doi.org/10.1162/TACL_A_00544

[2] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching Word Vectors with Subword Information. *TACL* 5 (2017), 135–146.

[3] Rajesh Bordawekar and Oded Shmueli. 2017. Using Word Embedding to Enable Semantic Queries in Relational Databases. In *Proceedings of the 1st Workshop on Data Management for End-to-End Machine Learning* (Chicago, IL, USA) *(DEEM'17)*. Association for Computing Machinery, New York, NY, USA, Article 5, 4 pages. https://doi.org/10.1145/3076246.3076251

[4] Riccardo Cappuzzo, Paolo Papotti, and Saravanan Thirumuruganathan. 2020. Creating Embeddings of Heterogeneous Relational Datasets for Data Integration Tasks. In *SIGMOD*.

[5] Tianji Cong, Madelon Hulsebos, Zhenjie Sun, Paul Groth, and H. V. Jagadish. 2024. Observatory: Characterizing Embeddings of Relational Tables. *Proc. VLDB Endow.* 17, 4 (mar 2024), 849–862. https://doi.org/10.14778/3636218.3636237

[6] Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. 2020. TURL: Table Understanding through Representation Learning. *Proc. VLDB Endow.* 14, 3 (2020), 307–319. https://doi.org/10.5555/3430915.3442430

[7] Muhammad Ebraheem, Saravanan Thirumuruganathan, Shafiq Joty, Mourad Ouzzani, and Nan Tang. 2018. Distributed representations of tuples for entity resolution. *PVLDB* 11, 11 (2018), 1454–1467.

[8] Martin Grohe. 2020. word2vec, node2vec, graph2vec, X2vec: Towards a Theory of Vector Embeddings of Structured Data. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems* (Portland, OR, USA) *(PODS'20)*. Association for Computing Machinery, New York, NY, USA, 1–16. https://doi.org/10.1145/3375395.3387641

[9] Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Martin Eisenschlos. 2020. TaPas: Weakly Supervised Table Parsing via Pre-training. In *ACL*. ACL, 4320–4333. https://doi.org/10.18653/v1/2020.acl-main.398

[10] Qian Liu, Bei Chen, Jiaqi Guo, Morteza Ziyadi, Zeqi Lin, Weizhu Chen, and Jian-Guang Lou. 2021. TAPEX: Table Pre-training via Learning a Neural SQL Executor. https://arxiv.org/abs/2107.07653v3.

[11] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *Advances in Neural Information Processing Systems*, C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger (Eds.), Vol. 26. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf

[12] Simone Papicchio, Paolo Papotti, and Luca Cagliero. 2023. QATCH: Benchmarking SQL-centric tasks with Table Representation Learning Models on Your Data. In *Advances in Neural Information Processing Systems*. http://papers.nips.cc/paper_files/paper/2023/hash/62a24b69b820d30e9e5ad4f15ff7bf72-Abstract-Datasets_and_Benchmarks.html

[13] Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. 2017. Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering* 29, 12 (2017), 2724–2743.

[14] Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. TaBERT: Pretraining for joint understanding of textual and tabular data. *arXiv preprint arXiv:2005.08314* (2020).

[15] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. In *EMNLP*. Association for Computational Linguistics, 3911–3921. https://doi.org/10.18653/v1/D18-1425

[16] Dan Zhang, Madelon Hulsebos, Yoshihiko Suhara, Çağatay Demiralp, Jinfeng Li, and Wang-Chiew Tan. 2020. Sato: contextual semantic type detection in tables. *Proc. VLDB Endow.* 13, 12 (jul 2020), 1835–1848. https://doi.org/10.14778/3407790.3407793

[17] Zixuan Zhao and Raul Castro Fernandez. 2022. Leva: Boosting Machine Learning Performance with Relational Embedding Data Augmentation. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) *(SIGMOD '22)*. Association for Computing Machinery, New York, NY, USA, 1504–1517. https://doi.org/10.1145/3514221.3517891

A MORE EXPERIMENTAL RESULTS

A.1 Detailed Results per Query

Tables 2 (Node2Vec) and 3 (Word2Vec) present a granular view of the performance metrics (Precision P and Recall R at k={5, 10, 20}) for a sample of individual SQL queries executed on the Movie dataset. These detailed results complement the aggregated scores in Table 1 and offer deeper insights into how the proposed selection/projection framework behaves under different query conditions and for different target attributes.

Generally, across both embedding methods, we observe trends consistent with the main findings. Recall (R) tends to increase or stabilize as k (the number of retrieved row embeddings) increases from 5 to 20. This is expected, as a larger retrieval set has a higher chance of capturing more of the ground truth tuples. Precision (P), however, can exhibit more varied behavior; it may decrease if the additional retrieved rows are incorrect, or it may improve/stabilize if the initial selections were highly accurate and further correct items are found. The overall performance patterns between Word2Vec and Node2Vec for specific queries are largely similar, reinforcing the observation that both methods, when used within EmbDI, yield comparable capabilities for this task.

The impact of attribute cardinality on projection accuracy is evident in these detailed results.

- **High-Cardinality Projections (e.g., `SELECT title`):** Queries projecting onto the high-cardinality `title` attribute (e.g., `select title from movie where director = 'quentin_tarantino'`) consistently demonstrate strong performance. For instance, in Table 3, this query achieves P@20 of 0.83 and R@20 of 1.00. This indicates that the selection step effectively identifies relevant rows, and the distinct embeddings for movie titles allow for accurate projection from these selected rows. Similar trends are observed for Node2Vec.
- **Low/Medium-Cardinality Projections (e.g., `SELECT year`, `SELECT status`):** When projecting onto attributes with lower cardinality, such as `year` or `status`, the results can be more mixed, illustrating the "hub" effect. For example, for the query `select status from movie where director = 'quentin_tarantino'` (n=2 expected results), both Node2Vec and Word2Vec achieve R@5 of 0.50 and P@5 of 0.50. This suggests that while one of the two correct rows might be selected and its status correctly projected, the other might be missed or an incorrect status projected, even if the row itself was selected. This highlights that even if a row embedding is correctly retrieved by the selection step, projecting a common, less distinctive attribute value can be difficult.
- **Impact of Query Complexity and Selectivity:** Queries with very specific conditions that result in a small number of ground truth tuples (e.g., `select title ... WHERE director = '...' AND year = '...'` with n=1) can achieve perfect recall and high precision if the query vector successfully isolates the correct row embedding. Conversely, queries involving multiple negations that yield a very large number of expected tuples (e.g., `select title from movie where director != '...' and status != '...' and genres != '...'` with n=4922 or n=42176) demonstrate a significant challenge. In both tables, these queries show R@k = 0.00 for all k. The query vector, formed by subtracting multiple attribute-value embeddings, does not effectively point towards the vast set of correct row embeddings within the top-k results. This underscores the limitations of the current vector arithmetic approach for very broad, negatively defined conditions.

These query results reinforce the conclusion that while the selection step often performs robustly in identifying candidate rows,

Table 2: Performance Metrics for a sample of SQL queries executed over embeddings with Node2Vec. Attribute *n* is the number of expected tuples in the ground truth.

Query	n	R@5	P@5	R@10	P@10	R@20	P@20
`select title from movie where director = 'quentin_tarantino'`	10	0.50	1.00	0.80	1.00	1.00	0.83
`select title from movie where director = 'quentin_tarantino' and status = 'released'`	10	0.50	1.00	0.80	1.00	1.00	0.83
`select title from movie where director = 'quentin_tarantino' and year = 2012.0`	1	1.00	0.25	1.00	0.11	1.00	0.06
`select title from movie where director = 'quentin_tarantino' and year != 2012.0`	9	0.56	1.00	0.78	0.88	1.00	0.75
`select title from movie where director = 'quentin_tarantino' and genres = 'action'`	3	1.00	0.60	1.00	0.38	1.00	0.25
`select title from movie where director = 'quentin_tarantino' and genres != 'action'`	9	0.56	1.00	0.89	1.00	1.00	0.75
`select title from movie where director = 'quentin_tarantino' and status != 'released' and year = 2012.0`	1	1.00	0.25	1.00	0.11	1.00	0.06
`select title from movie where director = 'quentin_tarantino' and status = 'released' and year != 2012.0`	9	0.56	1.00	0.78	0.88	1.00	0.75
`select title from movie where director = 'quentin_tarantino' and status = 'released' and genres = 'action'`	3	1.00	0.60	1.00	0.38	1.00	0.25
`select title from movie where director = 'quentin_tarantino' and status = 'released' and genres != 'action'`	7	0.43	0.60	0.86	0.75	1.00	0.58
`select title from movie where director = 'quentin_tarantino' and status != 'released' and genres = 'action'`	8	0.62	1.00	1.00	1.00	1.00	0.67
`select title from movie where director = 'quentin_tarantino' and status != 'released' and genres != 'comedy'`	2	1.00	0.40	1.00	0.25	1.00	0.17
`select title from movie where director = 'quentin_tarantino' and status = 'released' and genres = 'comedy'`	8	0.62	1.00	0.88	0.88	1.00	0.67
`select title from movie where director = 'quentin_tarantino' and status = 'released' and genres != 'comedy'`	8	0.62	1.00	1.00	1.00	1.00	0.67
`select title from movie where director != 'quentin_tarantino' and status != 'released' and genres != 'action'`	4922	0.00	1.00	0.00	1.00	0.00	1.00
`select title from movie where director != 'quentin_tarantino' and actor_1 != 'samuel_l_jackson' and actor_2 != 'samuel_l_jackson' and actor_3 != 'samuel_l_jackson'`	42176	0.00	1.00	0.00	1.00	0.00	1.00
`select year from movie where director = 'quentin_tarantino'`	10	0.50	1.00	0.80	1.00	1.00	1.00
`select status from movie where director = 'quentin_tarantino'`	2	0.50	0.50	0.50	0.50	0.50	0.50
`select genres from movie where director = 'quentin_tarantino'`	6	0.17	0.50	0.33	0.67	0.67	0.67
`select genres from movie where director = 'quentin_tarantino' and status = 'released'`	5	0.40	0.67	0.60	0.60	0.80	0.67
`select status from movie where director = 'quentin_tarantino' and year = 2012.0`	2	0.50	0.50	0.50	0.50	0.50	0.50
`select status from movie where director = 'quentin_tarantino' and year != 2012.0`	2	0.50	0.50	0.50	0.50	0.50	0.50
`select year from movie where director = 'quentin_tarantino' and genres = 'action'`	3	1.00	0.60	1.00	0.38	1.00	0.30
Average		0.59	0.74	0.74	0.68	0.82	0.58

the projection step's accuracy is highly dependent on the distinctiveness of the target attribute's value embeddings. The framework shows promise, particularly for well-defined queries and high-cardinality projections, but also reveals areas for future improvement, especially in handling low-cardinality projections and complex negative conditions.

A.2 Role of Duplicates

A consideration in our evaluation is the potential for duplicate values in the projected results and how these relate to the ground truth, which may itself contain duplicate attribute values across different tuples. The selection step of our framework retrieves unique row embeddings (e.g., identified by unique `idx_*`). However, the projection step operates independently for each selected row. This means that multiple distinct row embeddings, if they share the same true attribute value, can (and ideally should) project to that same attribute-value embedding. Furthermore, if multiple selected row embeddings are similar enough to the same incorrect attribute-value embedding, this incorrect value could also appear as a duplicate in our projected results. Our precision calculation counts

Table 3: Performance Metrics for a sample of SQL queries executed over embeddings with Word2Vec. Attribute *n* is the number of expected tuples in the ground truth.

Query	n	R@5	P@5	R@10	P@10	R@20	P@20
select title from movie where director = 'quentin_tarantino'	10	0.50	1.00	0.60	1.00	1.00	0.83
select title from movie where director = 'quentin_tarantino' and status = 'released'	10	0.40	1.00	0.60	1.00	1.00	0.83
select title from movie where director = 'quentin_tarantino' and year = 2012.0	1	1.00	0.25	1.00	0.11	1.00	0.07
select title from movie where director = 'quentin_tarantino' and year != 2012.0	9	0.56	1.00	0.56	0.83	1.00	0.75
select title from movie where director = 'quentin_tarantino' and genres = 'action'	3	1.00	0.60	1.00	0.43	1.00	0.25
select title from movie where director = 'quentin_tarantino' and genres != 'action'	9	0.56	1.00	0.67	1.00	1.00	0.75
select title from movie where director = 'quentin_tarantino' and status != 'released' and year = 2012.0	1	1.00	0.25	1.00	0.11	1.00	0.06
select title from movie where director = 'quentin_tarantino' and status = 'released' and year != 2012.0	9	0.33	1.00	0.67	0.86	1.00	0.75
select title from movie where director = 'quentin_tarantino' and status = 'released' and genres = 'action'	3	1.00	0.75	1.00	0.38	1.00	0.25
select title from movie where director = 'quentin_tarantino' and status = 'released' and genres != 'action'	7	0.57	0.80	0.71	0.83	1.00	0.58
select title from movie where director = 'quentin_tarantino' and status != 'released' and genres = 'action'	8	0.62	1.00	1.00	0.89	1.00	0.67
select title from movie where director = 'quentin_tarantino' and status != 'released' and genres != 'comedy'	2	1.00	0.50	1.00	0.25	1.00	0.17
select title from movie where director = 'quentin_tarantino' and status = 'released' and genres = 'comedy'	8	0.50	1.00	0.62	0.83	1.00	0.67
select title from movie where director = 'quentin_tarantino' and status = 'released' and genres != 'comedy'	8	0.62	1.00	1.00	1.00	1.00	0.67
select title from movie where director != 'quentin_tarantino' and status != 'released' and genres != 'action'	4922	0.00	1.00	0.00	1.00	0.00	1.00
select title from movie where director != 'quentin_tarantino' and actor_1 != 'samuel_l_jackson' and actor_2 != 'samuel_l_jackson' and actor_3 != 'samuel_l_jackson'	42176	0.00	1.00	0.00	1.00	0.00	1.00
select year from movie where director = 'quentin_tarantino'	10	0.50	1.00	0.60	1.00	1.00	1.00
select status from movie where director = 'quentin_tarantino'	2	0.50	1.00	1.00	0.40	1.00	0.33
select genres from movie where director = 'quentin_tarantino'	6	0.17	0.50	0.50	0.75	0.67	0.80
select genres from movie where director = 'quentin_tarantino' and status = 'released'	5	0.20	0.33	0.60	0.60	0.60	0.60
select status from movie where director = 'quentin_tarantino' and year = 2012.0	2	1.00	1.00	1.00	0.67	1.00	0.40
select status from movie where director = 'quentin_tarantino' and year != 2012.0	2	0.50	1.00	1.00	0.40	1.00	0.33
select year from movie where director = 'quentin_tarantino' and genres = 'action'	3	1.00	0.60	1.00	0.43	1.00	0.30
Average		0.59	0.76	0.74	0.69	0.88	0.57

such projected duplicates as distinct items in the denominator. This section clarifies this interaction using a specific example.

Consider the query: `SELECT year FROM dblp_scholar WHERE authors = 'm_stonebraker'`. Table 4 details the framework's output for this query, focusing on the projected `year` attribute. The ground truth for this query contains 11 unique 'year' values, which together account for 36 occurrences (tuples) in the `dblp_scholar` table for 'm_stonebraker'.

The "indexes" column in Table 4 lists the actual row identifiers (tuples) from the database that correspond to each unique ground truth 'year' for 'm_stonebraker'. The "found" column indicates, for each of these ground truth instances, whether our framework (specifically, using $k = 20$ in the selection step) successfully selected the corresponding row embedding *and* correctly projected its 'year' value. For example, for the year '1992', which appears for two distinct row indexes (`56143` and `39525`), both are marked as "yes", meaning our framework correctly identified '1992' for both these underlying tuples. Similarly, for the 'nan' (representing missing year data) value, which has 22 occurrences in the ground truth, the "found" column shows "yes" for 9 of these specific row indexes.

Table 4: Results for "select year from dblp_scholar where authors = 'm_stonebraker'"

Value	occurrence	indexes	found
nan	22	**3914**,	**yes**,
		11907,	**yes**,
		13407,	no,
		13730,	no,
		18611,	no,
		19059,	no,
		23862,	**yes**,
		28243,	**yes**,
		29586,	no,
		33942,	no,
		37181,	**yes**,
		38652,	no,
		46946,	**yes**,
		53936,	no,
		60184,	no,
		61396,	no,
		61745,	**yes**,
		62794,	no,
		62919,	no,
		63976,	**yes**,
		64053,	no,
		66317	**yes**
1998	3	1097,	no,
		43374,	no,
		16697	**yes**
1994	2	**54341**,	**yes**,
		22570	no
1992	2	**56143**,	**yes**,
		39525	**yes**
1990	1	**8405**	**yes**
1989	1	16252	no
1976	1	**27120**	**yes**
1975	1	**35929**	**yes**
1980	1	**36691**	**yes**
2002	1	**247**	**yes**
2003	1	**322**	**yes**
11	36	-	20

The final row of Table 4 ("11 | 36 | - | 20") summarizes the overall outcome:

- There are **11 unique** ground truth 'year' values.
- These 11 unique values correspond to **36 total occurrences** (tuples) in the ground truth for 'm_stonebraker'.
- From the $k = 20$ row embeddings selected by our framework, **20 correct projections** of 'year' values were made. These 20 correct projections correspond to identifying **10 out of the 11 unique ground truth 'year' values**.

In this specific instance, the year '1989' (which has 1 occurrence in the ground truth, index 16252) was not found by our model. Therefore, the recall, when calculated based on *unique* ground truth values, is $10/11 \approx 0.909$. The precision calculation considers the set of *unique projected values* that were correct. If, among the values projected from the 20 selected rows, exactly these 10 unique correct 'year' values appeared (even if some, like '1992', were projected multiple times from different selected row embeddings corresponding to distinct ground truth tuples) and no other incorrect 'year' values were projected, then the precision based on unique projected values would be $10/10 = 1.0$.

This example illustrates that our evaluation acknowledges the distinction between unique values and their multiple occurrences. The recall reflects the proportion of unique ground truth values recovered. The precision, as defined, is sensitive to whether any incorrect unique values are projected, regardless of how many times correct values are (correctly) duplicated due to multiple ground truth tuples sharing that value being selected. This approach aims to penalize the introduction of erroneous information while crediting the retrieval of all distinct correct pieces of information.