Optimising Vector Embedding Models for Example Selection in Text-to-SQL Generation

Matthew Draper

Master of Science

Computer Science

School of Informatics

University of Edinburgh

2024

Abstract

A comprehensive survey is conducted to assess the impact of altering the vector embedding model within a standard Text-to-SQL example selection algorithm, focusing on how these changes influence the effectiveness and quality of the selected examples. While this is shown to significantly influence which examples are chosen for few-shot prompting, it has little overall effect on execution accuracy scores. Current approaches to selection are criticised for promoting examples with low similarity to the target gold query, regardless of what embedding model is used. However when considering the results of the survey in aggregate, it is clear that record-setting benchmark scores could be achieved through in-context learning techniques alone. In other words, the experiments show that all the right 'needles in the haystack' exist for directing Text-to-SQL tasks via few-shot prompting; it is just a matter of developing a mechanism geared towards consistently finding them.

Novel methods are then put forward to address the issues identified. This includes the design of a new metric for SQL similarity, along with a standardized format for context-masking SQL queries. An algorithm is proposed that offers a new approach to embedding-based retrieval by encoding the space of masked SQL queries and comparing examples with previously generated 'first guess' predictions for each benchmark question. This results in improved execution accuracy scores on the Spider benchmark when compared to previous methods, and is shown to consistently retrieve high-quality examples for few-shot prompting. Also outlined is a new framework for fine-tuning embedding models aimed at optimising SQL retrieval, further enhancing the ability to quickly retrieve structurally similar queries from a dataset. The research conducted extends the field's understanding of retrieval-based in-context learning techniques, and concludes with definitive criteria that should be considered when designing future Text-to-SQL example selection algorithms.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Matthew Draper*)

Table of Contents

Chapter 1

Introduction

1.1 The Text-to-SQL Problem

Information retrieval from complex database schemas has historically been a non-trivial task. As big data continues to maintain its importance in digital industry, efficient access to key information is becoming increasingly critical. Structured Query Language (SQL) is a declarative programming language for querying relational databases, which has become the standardized reference language for database management and data retrieval tasks.

SQL literacy among the general public, and even within the programming community, is by and large quite low. As a result, companies rely on specialist data professionals to retrieve information from databases and distribute it to other staff members. Database software interfaces such as Microsoft Excel help abstract away from direct SQL query applications; however, businesses often require staff with extensive experience and training in these platforms to reliably store and retrieve company data.

The Text-to-SQL problem aims to bridge the gap between the widespread need to access crucial information from relational databases and the inherent difficulty of forming the correct SQL query to retrieve it. It concerns a natural language processing task, whereby a model is provided some representation of a target database, alongside a relevant question as a plain-text input. An effective solution to the problem would be a model capable of reliably converting well-formed inputs into correct corresponding SQL queries, with a success rate equal to or greater than that of human experts. Although there have been large advances in model performance in recent years, current state-of-the-art techniques remain a significant way off human expert capability.

1.2 Problem History

Originally Text-to-SQL was approached as a sequence-to-sequence based machine translation task [28]. This de-facto approach was superseded by models such as SQLNet [25] and TypeSQL [26] which circumvent the "ordering problem" that is commonly known to limit sequence-to-sequence based architectures [20]. Yet upon the advent of widely-accessible and resoundingly successful large language models (LLMs), a new paradigm for approaching the problem was embraced, as LLMs are now commonly leveraged for conducting Text-to-SQL conversions. Many top-performing submissions published to current Text-to-SQL benchmarks use OpenAI's GPT family of models as the basis for generating predicted SQL queries. Various studies have documented OpenAIs premiere GPT-4 class to be the most effective base engines for code generation tasks [6, 24]. As a consequence, much of the current Text-to-SQL research centres around directing these models toward high execution accuracy scores.

1.3 Potential Beneficiaries

A fully realized general-purpose Text-to-SQL conversion model would be highly valuable across modern industry due to its predicted impact on levels of productivity. Workers that previously did not have the skills to query big data, such as those in the teaching, healthcare and governmental professions, would be suddenly capable of accessing required information through a single interface. By removing the requirement to seek a trained specialist for data-retrieval, large amounts of company time and expenditure could be saved upon integrating a successful Text-to-SQL parser.

1.4 Study Outline

The research conducted in this paper regards example selection techniques for Text-to-SQL, with the objective of reviewing current mechanisms for few-shot prompting. Retrieval-based in-context learning (RetICL) is a primary focus, with particular attention paid to methods that involve the use of vector embedding models (VEMs). By encoding some representation of a Text-to-SQL problem as a vector, similar examples to a target problem can be quickly identified based on their proximity in the vector space. The study aims to determine the limitations of using embedding models for Text-to-SQL example selection, before designing novel RetICL approaches for such a task.

1.5 Problem Statement

It is conjectured that the choice of vector embedding model has a significant influence on which examples are chosen by an embedding-based RetICL system. Studies that incorporate such an approach to RetICL typically fix the choice of embedding model. There is presently no survey in the literature exploring the effect of VEM choice on Text-to-SQL example selection performance. This motivates the first research question:

> **RQ1:** *To what extent does the choice of vector embedding model influence the quality of examples chosen for few-shot Text-to-SQL conversion tasks?*

Once this topic has been addressed, an immediate follow up question arises. Currently, there is no evidence in the literature indicating whether fine-tuning embedding models can improve the quality of retrieved examples and result in improved performance on key Text-to-SQL benchmarks. The second objective of the study is as follows:

> **RQ2:** *To what extent can the fine-tuning of vector embedding models be applied to achieve better quality examples for Text-to-SQL generation?*

These questions form the central motivation for the thesis, and a positive result in either setting could be used to influence future studies in the field that include a ReICL selection mechanism. Any example retrieval procedure that is shown to perform significantly above the current alternatives would be of great value to those approaching few-shot prompting for Text-to-SQL benchmarks such as Spider and BIRD [27, 14].

1.6 Research Hypotheses & Objectives

To preface the study, a set of three research hypotheses are offered. These initial assessments made prior to the investigation motivate the design of future experiments.

1. *Altering the embedding model will result in significantly different examples selected for both the Spider and BIRD benchmarks.*

2. *Few-shot prompting techniques alone may be insufficient to achieve state of the art execution accuracy scores for both the Spider and BIRD benchmarks.*

3. *A fine-tuned embedding model trained on Text-to-SQL examples is likely to result in increased execution accuracy scores when compared against a baseline embedding.*

In order to address these hypotheses, the following research objectives are outlined.

i. *A survey should be conducted for both the Spider and BIRD benchmarks, evaluating eight state-of-the-art vector embedding models and documenting their execution accuracy scores in 1-shot, 3-shot, and 5-shot scenarios.*

ii. *Extended analysis of the survey results should be taken, with aims to answer:*

   - Is there a single embedding model that is more capable at example selection than the alternative options?
   - Why might suggested examples fail to direct the language model towards a successful conversion?
   - How could an embedding model be fine-tuned towards improved example selection capability and higher execution accuracy scores?
   - What is the highest possible execution accuracy score that could theoretically be achieved through the use of example selection techniques alone?

iii. *A framework should be proposed for fine-tuning a chosen embedding system from the survey, demonstrating to what extent supervised fine-tuning techniques could be used benefit VEM-based ReICL selection algorithms.*

Upon fulfilling these objectives, a holistic assessment of the two original research questions should be made by offering provable evidence-based conclusions. These questions remain open problems in the Text-to-SQL field, and it is hoped that the evidence produced in this study will inform future research on these topics.

1.7 Development Notes

The code used to perform these experiments is built on top of the source code provided[1] in the DAIL-SQL study by Gao et al [4]. All implementation is carried out in Jupyter notebooks and additional Python utility scripts, with the codebase divided into 'chapter' directories that correspond with the structure of this document. Results files are distributed alongside the source code and are directly reproducible within the notebooks. The OpenAI GPT suite of language models is chosen to be leveraged across experiments, with the `gpt-3.5-turbo-0125` model used throughout unless stated otherwise.

[1] `https://github.com/BeachWang/DAIL-SQL`

1.8 Related Works

The central source on which this thesis is based is the DAIL-SQL study [4] produced by the Alibaba Group in November 2023. This is the only submission to the Spider and BIRD leaderboards that surveys and develops novel ReICL example selection techniques. Other submissions focus on different refinement approaches, which include: fine-tuning existing specialist Text-to-Code language models [10], constructing a multi-agent framework to solve Text-to-SQL subproblems [22, 11], and extracting the most relevant database rows, columns and tables to include in the prompt [7, 19].

The DAIL-SQL survey documents different components of prompt engineering for Text-to-SQL, before the authors propose a state-of-the-art example selection algorithm. DAIL Selection begins by applying a context-masking function to all plaintext benchmark questions, and then ranks these examples based on their similarity to the masked target question. After ordering the examples by masked-question similarity, candidates must pass a SQL similarity threshold in order to be included in the few-shot prompt. This test measures the proportion of shared tokens between the example query and a previously generated 'first guess' prediction of the target gold SQL. Notably, the DAIL study fixes the vector embedding to the `all-mpnet-base-v2` model across the study [17]. Pseudocode of the full DAIL Selection algorithm is provided in Appendix B.

There is precedent in the literature of successfully fine-tuned embedding models for the purposes of example selection, with the most relevant source being the work by Liu et al. of Microsoft Research [15]. Their paper *"What Makes Good In-Context Examples for GPT-3?"* introduces KATE - a non-parametric selection approach that is tested against several question answering and table-to-text generation benchmarks. KATE utilises the RoBERTa embedding model [29] for example retrieval, the study goes on to fine-tune the embedding for context-dependent tasks, stating in its conclusion

> *"We found that fine-tuning the sentence embeddings for retrieval on task related datasets gave rise to further empirical gains" [15]*

This supports the final research hypothesis in theory while leaving room for new observations and conclusions, as the KATE study does not explore the effects of fine-tuning embeddings for Text-to-Code generation tasks. Since there is no such report on how fine-tuning embeddings could lead to improved example selection in the case of Text-to-SQL, a research gap is identified that is to be addressed across this paper.

Chapter 2

Problem Background & Related Work

2.1 Problem Formulation

Across this paper, a Text-to-SQL *instance* refers to a tuple $(\mathcal{D}, Q, \mathcal{G})$, where:

- $\mathcal{D}$ is the relational database that the problem concerns.
- Q is the plaintext question to be answered, relative to the database.
- $\mathcal{G}$ is the gold SQL query that successfully retrieves the necessary information, relative to the question and the database.

Meanwhile a Text-to-SQL *problem* refers to the binary outcome of whether a conversion model $\mathcal{M}$ is capable of producing an equivalent[1] SQL to the gold query $\mathcal{G}$. Commonly, a commercial large language model is leveraged as the processing engine used to perform the conversions. If this is the case, then Text-to-SQL instances are required to be framed as prompts. This means the input database $\mathcal{D}$ and target question Q must be represented as plaintext in a character-limited entry window. A well-studied approach to enhancing LLM performance for this task is few-shot prompting. By providing k examples from a training set within the prompt, the LLM has the prospect of learning by analogy from the sampled conversions. However, the provided examples must be to some degree relevant to the target problem in order for effective in-context learning to take place. Retrieving examples that are likely to direct the language model towards a successful conversion constitutes the field of retrieval-based in-context learning, which is a well-proven approach to tackling Text-to-SQL tasks.

[1] Of course, several unique SQL queries can retrieve identical records from the database, implying many answers to the same question Q. The stated gold query $\mathcal{G}$ can be thought of as a representative success case, but this does not exclude the existence of other correct answers.

2.2 Current Benchmarks

Text-to-SQL model performance is generally assessed using the execution accuracy (EX) metric. This metric evaluates the proportion of conversions from a test set that produce a correct SQL query. This means the SQL prediction produced by the model need not exactly match the gold query provided in the answer set, but it must retrieve exactly the same information. Other metrics such as valid efficiency score and exact set match score feature in the literature, but are not considered here.

Currently, there exist two primary benchmarks for assessing Text-to-SQL performance, the first of which is Spider [27, 3]. Distributed by Yale University in 2018, the Spider benchmark "consists of 10,181 questions and 5,693 unique complex SQL queries on 200 databases with multiple tables, covering 138 different domains". Spider became the standard benchmark for Text-to-SQL research because of its database variety, and number of labelled train cases. By November 2023, the 90% execution accuracy threshold was surpassed with the submission of the (undisclosed) MiniSeek model, produced by Seek AI. Their submission achieves a 91.2% score, which remains the highest performing submission on the Spider leaderboard.

Given the consistent success of new submissions to Spider, there grew a requirement for an updated, more challenging, Text-to-SQL benchmark. The BIRD benchmark [14] was released in February 2023 and contains "12,751 Text-to-SQL pairs and 95 databases spanning 37 professional domains". BIRD provides a considerably more difficult set of tasks for Text-to-SQL models, as the benchmark focuses on utilising datasets reflective of real life applications and industrial practice. This involves the use of substantially larger databases (with the BIRD dataset requiring 33.4GB of storage), as well as imperfect and erroneous data entries. Problem instances in BIRD also may require additional contextual information, which can be chosen to be supplied to the conversion model, or withheld. This adds a new dimension of difficulty not found in any previous Text-to-SQL benchmark. The current top performing model is the (undisclosed) OpenSearchSQL v2 + GPT4o model, produced by the Alibaba Cloud research group. The submission achieves a score of 72.28%, which remains significantly behind human-expert level performance of 92.96%.

The top open-source performers in their respective benchmarks focus on two different fundamental aspects of the Text-to-SQL problem.

- DAIL-SQL is the highest performing publicly available submission to the Spider benchmark, and is second in the overall leaderboard [4]. The final model achieves an 86.6% execution accuracy score, by leveraging base GPT-4-Turbo and implementing a state-of-the-art approach to example selection. DAIL-SQL aims to select the most relevant few-shot examples to include in the prompt, with the objective of steering the GPT model towards a successful conversion.

- CHESS is highest performing publicly available submission to the BIRD benchmark, and is seventh in the overall leaderboard [19]. It achieves a 66.69% execution accuracy score through a novel approach to schema linking. This concerns providing a concise textual database representation, that distills the key tables and columns necessary to answer the target question. CHESS introduces a three tiered schema-pruning pipeline that proves effective in aiding SQL generation.

It is worth noting that CHESS also outperforms DAIL-SQL on the Spider benchmark with an 87.2% score, however the Spider leaderboard was closed for submissions as of February 2024. A revised Spider 2.0 benchmark offering "a more realistic and challenging benchmark in the era of LLMs" is scheduled for release in Summer 2024.

2.3 Retrieval-Based In-Context Learning Approaches

2.3.1 Example Selection

Few-shot prompting has been a historically successful approach for directing large language models toward desired behaviour in Text-to-Code conversion tasks [12]. By providing a LLM with relevant examples inside the prompt, the model may learn by analogy from the demonstrations without altering the base model weights. This preserves LLM generality, whilst also saving significant time and resources when compared to supervised fine-tuning approaches.

Immediately, several questions are raised as to what makes a *'relevant'* example.

1. What is it that determines an example's relevance?

2. How does one determine whether an example is likely to aid an LLM in it's response or distract it?

3. Do there even exist relevant examples to choose from in the train set?

The Spider benchmark provides 8659 potential training examples to include in prompting, whilst the BIRD benchmark provides 9428 training cases. Any example included in a prompt is referred to as a shot, a prompt that features k demonstration examples is referred to as a k-shot learning problem. It is possible for the value of k to vary in prompt engineering models, with examples only included in the prompt if they satisfy some predefined confidence threshold. A central objective of this study has been to address these three questions in the setting of the Spider and BIRD benchmarks, as well as in the context of Text-to-SQL research as a whole.

In any approach to in-context learning, examples included in a prompt should be selected efficiently and deterministically. A good example $(\mathcal{D}^*, \mathcal{Q}^*, \mathcal{G}^*)$ to suggest should, in theory, have a question similar in sentiment to the target $\mathcal{Q}$, and have an answer query that reflects a similar structure to the target $\mathcal{G}$. It is argued in this paper that oftentimes, this is a conflicting and impossible ideal that is overlooked in the literature, leading to limitations on performance when using RetICL selection methods.

Hence, a central theme across this work is one of trade-offs and compromises, as it is demonstrated how current approaches aiming to satisfy either criterion are obstructed by significant pitfalls. As the field of research progresses, it is hoped that the observations outlined in this study will enrich the understanding of RetICL techniques for Text-to-SQL and encourage the design of mechanisms that consider the question, database, and gold query in future example selection algorithms.

2.3.2 Vector Embedding Retrieval

Various algorithms have been proposed for selecting the best examples for Text-to-SQL problems, many of which rely on the theory and implementation of vector embedding models. A sentence embedding model in particular deterministically maps a natural language input to a high-dimensional vector, where the output's position in the vector space is in some way indicative of the original text's structure, semantics, and meaning. Vector embeddings enable semantic search for similar examples in the training set, as candidates mapped near the target vector can be assumed to be the most comparable.

An example selection algorithm can be thought of as a 'claw machine' of sorts, one that observes the target problem and selects k candidates from a train-set 'prize box'. Under this analogy, the role of the VEM can be thought of as follows:

- The VEM firstly orders all the training instances in the prize box, positioning them in such a way that examples it deems similar are located close together, while dissimilar examples are located further apart.
- Next, the VEM places a ball to represent the target problem inside the prize box abiding by the same positioning rules as deployed in step one.
- The VEM then highlights the k-nearest instances to the ball in the prize box, it then tells the claw machine to select these examples.

The example selection 'claw machine' operates in high-dimensional space, with most models producing vectors ranging from approximately 750 to 1500 dimensions. A VEM is defined by its vector positioning rule, as this is determined the weights that parameterise its deep pre-trained neural network. A decision must be made about what representation of Text-to-SQL instances should be encoded, or in the metaphor, what defines each item in the prize box. This is referred to as the embedding space and is commonly the set of plaintext questions found in each Text-to-SQL instance.

A framework for fine-tuning embedding models is distributed through the python `sentence-transformers` library [2]. It is thought that this could be used to tremendously benefit Text-to-SQL example selection, as sentence embeddings are typically trained on large corpora of natural language text, with limited exposure to SQL problems. In theory, an embedding model specifically trained on Text-to-SQL tasks may be able to suggest examples that better aid SQL generation.

2.4 Prompt Engineering for Text-to-SQL

2.4.1 Prompt Formatting

Throughout the experiments, prompts are formatted to align with the most successful structure identified in the DAIL-SQL survey [4]. When leveraging a large language model for Text-to-SQL tasks, each question in the benchmark must be uniformly and consistently formatted. This involves three primary considerations: how to represent the target database as a plaintext input; how to structure selected examples, deciding what relevant information to include; and how to present the target question for completion. Exemplar 1-shot prompts for both the Spider and BIRD benchmarks are provided in Appendix A, which fully display the formatting approaches to be described.

2.4.2 Database Representation

Code Representation prompting is database representation approach that is integrated across experiments. This is the most successful scheme for prompting `gpt-3.5-turbo` models recorded in Gao et al's survey, and is the method included within the final DAIL-SQL submission [4]. This representation chooses to include all column labels and data types of all tables in the database, along with the foreign key information for each table. In theory, this provides a complete representation of the database structure, so the processing model should not fail due to insufficient database knowledge.

However, in many instances, this results in large amounts of superfluous information included in the prompt, with the input comprised of many tables and columns that are unnecessary for producing the correct SQL query. This is most prevalent in the BIRD benchmark, where databases tend to contain a far larger amount of tables and columns than problems found in Spider. It has been demonstrated how this has the potential to distract a LLM from its objectives, and serves as an active bottleneck to progress [8, 1]. Thus a large sector of Text-to-SQL research focuses on how to best extract relevant tables and columns for a given conversion task. Current state of the art approaches to schema-linking techniques include MAC-SQL [22], RESDSQL [9], and notably CHESS [19]. Modules that extract the most relevant database representations are not included in this study, instead RAT-SQL [21] is used to generate the full textual database representations for prompting.

2.4.3 Example Organization

The DAIL-SQL paper outlines three methodologies for organizing examples within a prompt. The authors' proposed approach, DAIL Organization, only includes the associated question and gold query for each example. This is the approach that is integrated into this study's experiments, meaning that the associated databases for each selected example are not included in the prompts.

It is arguable that this omits a fundamental component that could benefit a language model attempting to learn by analogy from few-shot examples, as both the question and gold query are dependent on their corresponding database. However in many instances, especially in higher shot scenarios, including each example's full database representation exceeds the text input limits of many LLMs. A recommendation for a future study would be to utilise Full Information Organization (which includes question, query and

database for each example) alongside existing schema-linking and feature-extraction techniques. This would ensure each example identified includes all the necessary information to best impact retrieval-based in-context learning.

When using Code Representation prompting and DAIL Organization, a typical input provided to the language model will adhere to the following structure.

```
/* Some example questions and corresponding SQL queries are provided
based on similar problems: */

/* Answer the following :
${EXAMPLE QUESTION}
${EXAMPLE GOLD SQL} */

/* Given the following database schema : ${DATABASE SCHEMA} */
/* Answer the following : ${TARGET QUESTION} */
SELECT
```

2.4.4 K-Shot Prompt Pipeline

All conversion tasks for Spider and BIRD benchmarks must first be formatted as a plain-text input, with the context window limited to 16,385 tokens per prompt (approximately 65,540 bytes of text). Constructing a prompt requires many sub-modules for obtaining a database representation, selecting examples and question formatting. Psuedocode is provided to illustrate in abstract the central components of prompt construction.

Algorithm 1 Construct Prompt

Require: $\mathcal{D}$: Target Database, Q: Target Questions, k: Number of Examples, train_set: Benchmark Train Set, embedding_model: VEM

function CONSTRUCT_PROMPT($\mathcal{D}$, Q, k, train_set, embedding_model)

$formatted\_database \leftarrow$ `db_to_text`($\mathcal{D}$)

$selected\_examples \leftarrow$ `get_examples`(Q, k, train_set, embedding_model)

$formatted\_examples \leftarrow$ `examples_to_text`(selected_examples)

$formatted\_question \leftarrow$ `"Answer the following: "` + Q + `" SELECT "`

$prompt \leftarrow formatted\_examples + formatted\_database + formatted\_question$

return *prompt*

To preface future chapters, pseudocode is also offered for a basic example selection algorithm, where candidates are determined by embedding the plaintext question space. This algorithm constitutes one of the most common approaches to retrieval-based in-context learning for Text-to-SQL tasks. Any alterations from this central mechanism will be outlined in full, with accompanying pseudocode located in Appendix B.

Algorithm 2 Select Examples + Question Embedding

Require: Q: Target Question, k: Number of Examples, train_set: Benchmark Train Set, embedding_model: VEM

function GET_EXAMPLES(Q, k, train_set, embedding_model)
    $train\_questions \leftarrow train\_set[\text{“}question\text{”}]$
    $train\_embeddings \leftarrow embedding\_model.\texttt{encode}(train\_questions)$
    $target\_embedding \leftarrow embedding\_model.\texttt{encode}(Q)$
    $distances \leftarrow \texttt{compute\_distances}(target\_embedding, train\_embeddings)$
    $nearest\_neighbors \leftarrow \texttt{closest\_k\_indices}(distances, k)$
    $selected\_examples \leftarrow train\_set[nearest\_neighbors]$
    return $selected\_examples$

2.5 Vector Embedding Models for Example Selection

2.5.1 Vector Embedding Model Architectures

SBERT-Based Models: The leading architecture for producing vector embeddings derives from the landmark paper “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding” released by Devlin et al. in May 2019 [2]. BERT (standing for Bidirectional Encoder Representations from Transformers) utilizes a deep bidirectional transformer architecture to pre-train a language model on a large corpus of text. This bidirectionality allows BERT to consider the context from both the left and the right side of a token, enabling a richer understanding of language compared to previous models.

The standard BERT architecture is unsuited to the task of semantic search. To evaluate a notion of similarity between two sentences, BERT’s cross-encoder architecture requires inputting both candidates together into its deep birectional transformer network.

This means that identifying the most similar sentence in a dataset to a target would require completing this computation pairwise for all candidate examples, resulting in a significant computational burden when dealing with large datasets. Reimers and Gurevych describe this fundamental issue of employing BERT for semantic search.

> "BERT uses a cross-encoder: Two sentences are passed to the transformer network and the target value is predicted. However, this setup is unsuitable for various pair regression tasks due to too many possible combinations. Finding in a collection of 10,000 sentences the pair with the highest similarity requires with BERT 49,995,000 inference computations. On a modern V100 GPU, this requires about 65 hours." [16]

In their paper "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks" [16], Reimers and Gurevych propose the SBERT framework. By applying a pooling step, outputs from the BERT architecture are converted to meaningful vectorised sentence embeddings. The semantic similarity between embeddings can be quickly evaluated using common linear algebra metrics such as euclidean distance and cosine distance.

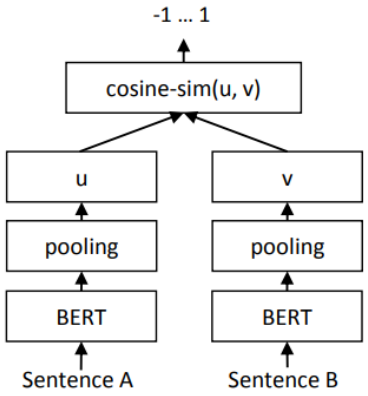

Figure 2.1: SBERT Sentence Similarity Evaluation Pipeline [16]

With this method, the time required to identify the most similar example to a target from a 10,000 entry training set is drastically reduced from an estimated 65 hours to just 5 seconds. The SBERT framework is distributed via the popular python `sentence-transformers` library, which provides open access to a variety of sentence embedding models and associated functions. The library also includes tools to fine-tune the underlying BERT model, producing optimized embeddings for targeted objectives.

Through the huggingface central platform, many pre-trained embedding models based on the SBERT architecture are accessible for public use. Of these, the most frequently downloaded models include `bert-base-nli-mean-tokens` [2, 16], `stsb-roberta-base` [29, 16], `all-mpnet-base-v2` [17], `all-MiniLM-L6-v2` [23] and `all-MiniLM-L12-v2` [23].

OpenAI Models: Current state-of-the-art models include those available via the OpenAI API, where three closed-source embedding models are made accessible through paid access: `text-embedding-ada-002`, `text-embedding-3-small`, and `text-embedding-3-large`. OpenAI has shared detailed insights about uses cases and benchmark performance of these models. However, specific architectural details and the exact training methodologies are not disclosed in public documentation.

2.5.2 Embedding Spaces

When using ReICL techniques for few-shot prompting, there are three fundamental considerations that influence the choice of examples selected. First and foremost, the structure of the example selection algorithm will dictate how and which examples are ultimately chosen from the train set. For instance, a completely uninformed ReICL selection algorithm might always return the first training example, regardless of the target problem's context. In this study, only retrieval algorithms that incorporate sentence embeddings are considered, with the singular objective of selecting the most relevant examples to steer towards successful SQL generation.

For such algorithms, it is hypothesized that the choice of VEM will also have an impact on the results. Embedding models generate unique vector projections for encoded Text-to-SQL instances, so altering the embedding can cause examples to be positioned closer to or farther from the projected target question in the new vector space. Since it is the k-nearest neighbors of the target vector that are selected for few-shot prompting, it is conjectured that the choice of VEM has considerable influence on which training examples are nominated.

Finally, the choice of how Text-to-SQL instances are represented directly influences the resultant projection space. This choice is critical to the retrieval of effective examples, and is referred to as the embedding space. Sentence embedding models are most effective when encoding natural language sentences with standard grammar and conventional structure. Historically, this has led to the common choice of using plaintext questions as the embedding space for Text-to-SQL tasks. This abstracts away from all information that could be inferred from the associated database structure, as examples are chosen exclusively based on their level of semantic similarity to the target question.

Note that any example selection algorithm only has access to the target database $\mathcal{D}$, and the target question Q, along with the training set of provided examples. If the primary goal of RetICL is to select the examples most 'similar' to the target gold query, this must be inferred exclusively from $\mathcal{D}$ and Q. This limits the potential options for choosing an embedding space, and gives rise to significant obstructions that are outlined as a central argument of this thesis.

Question Embeddings: An example of question-embedding based retrieval is demonstrated using the first problem from the Spider benchmark test-set. The top three examples identified when using the `all-mpnet-base-v2` VEM are as follows:

Target Question:	How many singers do we have?
Gold SQL:	`SELECT count(*) FROM singer`
Example Question	**Example SQL**
How many artists do we have?	`SELECT count(*) FROM artist`
How many artists are there?	`SELECT count(*) FROM artist`
How many songs have a shared vocal?	`SELECT count(DISTINCT title) FROM vocals AS T1 JOIN songs AS T2 ON T1.songid = T2.songid WHERE TYPE = "shared"`

Table 2.1: Question Embedding Retrieval: Spider 3-Shot Examples

Evidently, the first two suggestions have semantically similar questions to the target, and their queries share a matching SQL structure with the true gold query - these examples can be considered RetICL success stories. However, the third example asks a very different question in terms of data-retrieval, which is reflected in a drastically different SQL to the true gold query. This poor-quality example seems to be chosen based on the question's semantic similarity to the term 'singers' found in the target question. This highlights a fundamental problem in using plaintext questions as the embedding space: contextually relevant examples can be promoted for few-shot prompting whilst taking a query that has little to no relevance for encouraging correct SQL generation.

Masked Question Embeddings: As displayed, a common problem when encoding plaintext questions is that contextual database information has a significant influence on the examples retrieved. Hence it would be more desirable to choose questions that are structurally similar to the target, as opposed to semantically similar. By context-masking the instance questions this problem can be circumvented. RESD-SQL [9] is a schema-

linking module deployed to identify database context-dependent tokens in a question for masking. Algorithm 3 in Appendix B provides pseudocode for an example selection algorithm that chooses examples by proximity in the masked-question embedding space. Some SBERT models are engineered to handle sentences that include designated context-masked tokens [16, 17]; the generic token to replace context-dependent token labels is <mask>, whilst the token used to replace quoted values is <unk>.

To continue with the first example from the Spider benchmark test set, the associated context-masked question would be "How many <mask> do we have?". All questions from the training set undergo context-masking, and it is this set of abstracted question strings which forms the embedding space. When using the `all-mpnet-base-v2` VEM, the top three nearest examples to the masked target question are as follows:

Target Masked Question:	How many <mask> do we have?
Gold SQL:	`SELECT count(*) FROM singer`
Example Masked Question	**Example SQL**
How many <mask> do we have?	`SELECT count(*) FROM Aircraft`
How many <mask> do we have?	`SELECT count(*) FROM Employee`
How many <mask> do we have?	`SELECT count(*) FROM Flight`

Table 2.2: Masked Question Embedding Retrieval: Spider 3-Shot Examples

The three selected examples in the dataset share identical sentences after context-masking and thereby have a common vector encoding. The examples also share a common SQL structure with the true gold query. Still, it should be noted that there is no guarantee that sharing an identical masked question to the target problem implies a identically structured gold query. In this case, embedding the context-masked question space yields arguably perfect ReICL performance, Section 3.4 will illustrate how this is misrepresentative of the approach's capacity to generate high-quality examples.

2.6 Advanced Example Selection Techniques

2.6.1 Similarity Thresholds

The DAIL Selection mechanism introduced by Dao et al. [4] proposes a similarity threshold approach for Text-to-SQL example selection. Potential example nominations are firstly ordered based on their masked-question embedding distances, which serves

as an initial heuristic rather than a definitive measure of example quality. The candidates then must pass a quality assurance metric in order for them to be included in the prompt. The DAIL algorithm uses a notion of SQL similarity to define its threshold function (outlined in Section 2.6.3). In brief, the DAIL-SQL method determines whether to include an example in the prompt by comparing the example SQL to a previously attempted conversion, which serves as a 'first guess' to the true gold SQL structure. In the DAIL study, these pre-predicted SQLs are generated using the Graphix-T5 language model [13]. The idea being that a pre-prediction may not ultimately be a correct answer, but it should share many of the key structural components of the gold SQL.

2.6.2 SQL Context Masking

DAIL Masking: To evaluate the similarity between two SQLs, the DAIL approach first extracts the skeleton from the queries. RESD-SQL [9] is used as a schema-linking module to identify database-specific clauses in the query, before these tokens are replaced with an underscore. At the end of the masking, only SQL keywords, operators, and underscores remain. This can be percieved as a liberal approach to context-masking SQLs, as largely different queries can share a common mapping. In this study, this approach to context-masking SQLs is referred to as *DAILMask*.

Proposed Masking: The DAIL masking mechanism does not account for the number of unique columns or tables referenced in a SQL query, instead choosing to abstract over this information. With this in mind, a new SQL masking procedure is proposed, called *SQLMask*. This procedure takes a regex-oriented approach to masking column and table identifiers across the SQL, making it highly efficient when compared to *DAILMask*. By assigning consistent generic identifiers to each table and column referenced in a query, *SQLMask* preserves the number of unique tokens whilst abstracting away from database context. An example from the Spider test set is offered in Table 2.3 to demonstrate the differences between the masking approaches. A selection of eight further examples is provided in Appendix C.

SQL:	`SELECT DISTINCT Country FROM singer WHERE Age > 20`
DAIL MASK:	`select distinct _ from _ where _`
SQL MASK:	`SELECT DISTINCT col1 FROM table1 WHERE col2 > num`

Table 2.3: Example SQL context-masking Comparison

2.6.3 SQL Similarity Measures

DAIL Measure: In the DAIL Selection algorithm, the Jaccard similarity metric is used as a threshold function to determine whether to include a candidate example in the prompt. This is a common natural language similarity metric which calculates the proportion of shared token labels found across two sentences. In this context of measuring the similarity between SQL queries, it will be referred to as *DAILSim*.

$$DAILSim(SQL_1, SQL_2) = \frac{\text{Number of common tokens}}{\text{Total unique tokens}}$$

Proposed Measure: Applying *DAILMask* to a pair of queries before calculating their *DAILSim* value seems to be entirely inadequate for determining sensible notions of SQL similarity. Many distinct query tokens are mapped to an underscore under *DAILMask*, whilst the *DAILSim* measure is defined via the number of unique tokens shared between the queries - this creates a notable issue. Also, the Jaccard similarity measure between two SQLs only considers individual token labels, without accounting for their position and overall structure. To ensure that structure is considered when deciding SQL similarity, the tree similarity of edit distance (TSED) metric is identified. Song et al. outline the novel metric [18], which first computes the syntax tree edit distance between two queries and then normalizes to a value in the range zero to one. This metric is agnositic to token labels, so to balance token label similarity and token structure similarity, the *SQLSim* metric is proposed. This is the chosen metric for SQL similarity used across the study, to be used in conjunction with *SQLMask*.

$$SQLSim(SQL_1, SQL_2) = \frac{DAILSim(SQL_1, SQL_2) + TSED(SQL_1, SQL_2)}{2}$$

Table 2.4 provides an example that highlights the differences between the DAIL approach and the proposed method for measuring SQL similarity. The example is a clear demonstration of two SQLs that are dissimilar in structure and complexity but are treated as perfect matches under the DAIL approach. Eight additional demonstrations comparing the similarity scoring functions are provided in Appendix C.

SQL_1	`SELECT name, country, age FROM singer ORDER BY age DESC`
DAILMask(SQL_1)	`select _ from _ order by _ desc`
SQLMask(SQL_1)	`SELECT col1, col2, col3 FROM table1 ORDER BY col3 DESC`
SQL_2	`SELECT T1.Name FROM people AS T1 JOIN poker_player AS T2 ON T1.People_ID = T2.People_ID ORDER BY T2.Earnings DESC`

DAILMask(SQL_2)	`select __ from __ order by __ desc`
SQLMask(SQL_2)	`SELECT alias2.col1 FROM table1 AS alias1 JOIN table2 AS alias2 ON alias1.col2 = alias2.col2 ORDER BY alias2.col3 DESC`
DAILSim(*DAILMask*(SQL_1), *DAILMask*(SQL_2)) = 1.0	
SQLSim(*SQLMask*(SQL_1), *SQLMask*(SQL_2)) = $0.369\dot{4}$	

Table 2.4: SQL Queries and Similarity Metrics

2.6.4 Pre-Predicted SQL Embeddings

SQLSim, when used alongside the proposed masking *SQLMask*, is a stricter and seemingly more complete approach to evaluating notions of SQL similarity than the method deployed in the DAIL study. However, manually identifying the best examples for a target based on *SQLSim* score remains too computationally expensive when working with a moderately sized dataset. This is a case where vector embedding models could provide a high-speed compromised solution. Though VEMs are trained with natural language sentences, it is anticipated that an embedding-based approach could be fine-tuned to map SQL queries to meaningful vectors after sufficient training examples.

The second half of this study investigates the potential of embedding the masked SQL space rather than the standard masked-question space for ReICL, and whether this can lead to the identification of higher quality examples. Of course, it is not possible to embed the true target gold query in a Text-to-SQL problem, but to take inspiration from the DAIL Selection algorithm, it is possible to embed a pre-predicted 'first guess' SQL. This runs the risk of choosing examples that reinforce bad behaviour exhibited by erroneous pre-prediction queries. Hence, the study aims to answer two pivotal questions concerning example selection algorithms which prioritize a notion of SQL similarity.

1. **Embedding Pre-Predicted SQLs:** If we embed a masked pre-predicted SQL for a target problem, and obtain the nearest neighbors in the embedded SQL space, will this outperform traditional question embedding approaches? To what extent does this reinforce bad translation from the first prediction?

2. **Embedding Gold SQLs:** If we embed the true gold SQL under *SQLMask*, do we retrieve consistently high quality examples? Does the inclusion of highly similar examples to the true gold query guarantee increases in benchmark performance?

Chapter 3

RQ1: Survey of Vector Embedding Model Performance

3.1 Experiment Outline

To address the first research question, a comprehensive evaluation of state-of-the-art embedding models is conducted against both the Spider and BIRD benchmarks. A total of twenty-four experiments are recorded, evaluating the performance of eight surveyed VEMs; these include the five SBERT-based models and the three OpenAI models stated in Section 2.5.1. Each model is assessed by three independent k-shot experiments, where $k \in \{1,3,5\}$. Otherwise, all experiments adhere to the following parameters.

Parameter	**Value**
Language Model	`gpt-3.5-turbo-0125`
Selection Algorithm	GET_EXAMPLES_MASK (Algorithm 3)
Embedding Space	Context-Masked Questions
Database Representation	Code Representation
Example Organization	DAIL Organization
Schema Linker for Question Masking	RESD-SQL

Table 3.1: Embedding Model Experiment Fixed Parameters List

To preface the main survey results, two forms of baseline experiments are recorded for each benchmark. These provide informative insights into how well `gpt-3.5-turbo` can perform without the aid of informed retrieval based in-context learning techniques.

- **Zero Shot Experiment:** Benchmark questions are posited to the language model without any examples included in the prompt. The results of this experiment for the Spider and BIRD benchmarks are offered in Table 3.2.

Benchmark	Spider	BIRD
EX	0.720	0.439

Table 3.2: Comparison of Zero-Shot Spider and BIRD EX Scores

- **Random Selector Experiments:** Examples are chosen at random from the training dataset, with k-shot experiments conducted for $k \in \{1,3,5\}$. The results for the Spider and BIRD benchmarks in each case are offered in Table 3.3.

k	Spider	BIRD
1	0.734	0.412
3	0.734	0.409
5	0.737	0.419

Table 3.3: Comparison of Random Selector k-shot Spider and BIRD EX Scores

By comparing the execution accuracy scores of Tables 3.2 and 3.3, it is evident that providing random examples results in a negligible increase in performance for Spider and is actually detrimental to performance for BIRD. This provides evidence that poor examples can result in decreased performance and misalign a language model's behaviour when attempting to complete Text-to-SQL conversions.

3.2 Spider Results

Survey: The results of the 8 VEM survey for the Spider benchmark are displayed in Table 3.4. Prompt and evaluation files for each experiment are distributed with the source code, with all results fully reproducible via the corresponding Jupyter notebook.

k	BERT	RoBERTa	mpnet-v2	Mini-L6	Mini-L12	te3-small	te3-large	ada002
1	0.760	0.765	0.764	**0.773**	0.752	0.761	0.761	0.770
3	0.766	**0.779**	0.767	0.767	0.766	0.771	0.777	0.775
5	0.777	0.775	0.776	0.766	0.772	0.781	**0.784**	0.783

Table 3.4: Research Objective 1 Survey: Spider EX Scores

It is evident from the survey that employing informed ReICL example selection techniques can increase execution accuracy for Spider Text-to-SQL conversions by up to 6.4% above the zero-shot baseline. However, for each value of k, changing the VEM does not appear to significantly impact the final EX score, with all outcomes falling within a maximum distance of 2.1% to the top-scoring VEM in their k-shot tier. Evidently, no singular VEM outperforms all other alternatives for all values of k, with distinct models setting the record scores for 1-shot, 3-shot, and 5-shot experiments respectively. Thus, altering the VEM shows only a slight effect on the EX score, but this does not fully address how much influence VEM choice has on example selection.

Aggregated Results: A significant result is observed when considering the Spider experiment results in aggregate. Counting how many benchmark questions were correctly answered across the experiments provides a lower bound of the best theoretical score that can be achieved through example selection techniques alone. The number of problems that could be answered with just a single example is surprisingly impressive, as when considering the eight 1-shot experiments in the survey **896 / 1034 (86.7%)** of the benchmark questions were seen to be answered correctly. Extending this to all twenty-four k-shot experiments, this proportion increases to **924 / 1034 (89.7%)**. Both figures exceed the final score from the DAIL-SQL study (86.6%) and surpass any known submission that leverages `gpt-3.5-turbo`. While altering the VEM may result in a negligible difference to the overall EX score, these results imply that there is a discernible difference in which questions are answered correctly across the experiments.

Example Variety Analysis: To illustrate this point further, the proportion of mutually correctly answered questions for every pair of experiments is recorded. This proportion is known as the *Jaccard index* of two experiments' pairwise outcomes.

$$\textit{Jaccard Index} = \frac{\textit{Number of common correct answers}}{\textit{Total number of unique correct answers in both sets}}$$

For example, consider the results from the 1-shot `bert-base-nli-mean-tokens` experiment and the 1-shot `stsb-roberta-base` experiment. The number of questions that the BERT-equipped experiment answered correctly, the number of questions that the RoBERTa experiment answered correctly, and the number of questions that both experiments answered correctly are given in Table 3.5. From this information, we can infer the Jaccard index between the two experiments results.

Survey Result	Value
Number of correct BERT answers	786
Number of correct RoBERTa answers	791
Number of common correct answers	743
Total number of unique correct answers in both sets	$786 + 791 - 743 = 834$
BERT / RoBERTa Experiment Results Jaccard Index	$743/834 \approx 0.891$

Table 3.5: BERT and RoBERTa 1-shot Experiments Jaccard Index Computation

This indicates that approximately 11% of the correctly answered questions for the 1-shot BERT and RoBERTa experiments are answered correctly by only one model but not the other. This difference is a more relevant measure to the study, and is formally referred to as *Jaccard distance*. Since the embedding model is the only independent variable between the experiments, the observed discrepancy must be due to VEM influence on example selection, with different examples being included in the prompts.

Pairwise Jaccard scores are recorded for each set of experiments, stratified by k. The average Jaccard distance for each k represents the expected proportion of correct answers that differ between any two given experiments, indicating how often one model correctly answers a question that the other does not. Across the strata, it is seen that approximately 12% of the correct answers differ between any two experiments, as indicated by the average Jaccard distance values of 0.123 for 1-shot, 0.125 for 3-shot, and 0.119 for 5-shot experiments.

k	Average Jaccard Index	Average Jaccard Distance
1	0.877	0.123
3	0.875	0.125
5	0.881	0.119

Table 3.6: Research Objective 1 Survey: Spider Average Jaccard Scores & Distances

This consistent average Jaccard distance highlights that, even though the models achieve similar overall execution accuracies, the specific questions they answer correctly can often vary. This 12% discrepancy must be attributed to the choice of vector embedding model, and the consequent variation in examples retrieved during the selection process. Hence, further evidence is shown that the VEM choice can have a substantial impact on the performance language models when deployed for few-shot learning.

Example Quality Analysis: To understand how successful question embedding based approaches are at example selection, the results from 1-shot experiments are used to form an evaluation matrix, documenting question success versus example quality. An example is classified as 'good' if the SQLSim value between the example query and its true target gold SQL is above 0.85. The threshold for a 'good' example classification is chosen to match the threshold deployed in the DAIL Selection algorithm. The evaluation matrix given in Table 3.7 classifies all 1-shot experiment outcomes from the survey.

	Good Example	**Bad Example**
Correct	1815	4495
Incorrect	174	1788

Table 3.7: Research Objective 1 Survey: Spider Evaluation Matrix

The matrix offers important insights into the effect of few-shot prompting for Text-to-SQL conversions. We see that providing a 'good' example significantly increases the chance of a correct conversion, but does not guarantee a positive outcome. Also notable is the sheer number of 'bad' examples selected. In Table 3.7, 76% of cases are seen to include a 'bad' quality example. An exhaustive search of the Spider training set for each benchmark question reveals that 66% of Spider problems have at least one 'good' example that could be nominated. When lowering the *SQLSim* threshold to 0.75, this increases to 92%, but these strong potential candidates are still mostly going unselected.

To further this claim, the mean example quality is calculated for each experiment by measuring the *SQLSim* score between the nominated example SQLs and the target gold query. This displays that, across the board, examples chosen in the survey have SQLs that are structurally dissimilar to their target, and are unlikely to significantly aid a language model in its conversion attempts. It is argued that this is not particularly the fault of the embedding models, but instead the algorithm that deploys them. Selecting examples based on semantic similarity to the target plaintext question often fails to retrieve examples that share any meaningful similarity in SQL structure.

k	**BERT**	**RoBERTa**	**mpnet-v2**	**Mini-L6**	**Mini-L12**	**te3-small**	**te3-large**	**ada002**
1	0.534	0.543	**0.545**	0.527	0.540	0.448	0.461	0.470
3	**0.529**	0.527	0.525	0.511	0.528	0.446	0.455	0.469
5	0.518	0.512	0.517	0.501	**0.522**	0.439	0.449	0.464

Table 3.8: Research Objective 1 Survey: Spider Average Example Similarity Scores

This highlights a critical insight from the study that reveals a fundamental flaw in current ReICL algorithms: the failure of question similarity-based example retrieval.

- **Masked Question Similarity $\not\Rightarrow$ Similar SQL:** When directly studying the prompts that have been produced for Text-to-SQL tasks across the survey, it is clear that choosing examples based on masked-question similarity generally does not promote examples with similar SQLs to the target gold query.

  Successful conversions tend to include examples with a higher SQLSim score relative to the gold query, with success cases having an average example similarity score of 0.62 compared to 0.48 for failure cases. This reinforces that the proposed *SQLSim* metric is a strong indicator as to whether an example is suitable for few-shot prompting, and a reminder of the failings of question embedding retrieval.

  If the aim of ReICL is to identify examples with a high level SQL similarity to the target, then embedding the masked-question space evidently does not achieve this goal. The reason behind this disparity, it is argued, is a straightforward conclusion from the definition provided in Section 2.1 of a Text-to-SQL problem. The gold query for any Text-to-SQL problem depends on both the original question and the target database. Masked question structure can be a useful heuristic for the example SQL's composition, but it has little to no correlation with the relevant database's structure. This means that you can have identical masked questions, yet have completely different SQL structures if the two examples relate to different databases. Often, examples are promoted that are largely different in SQL structure to the target, which can distract a language model and result in unsuccessful conversions. Hence embedding the masked-question space is **not** encouraged for ReICL example selection in future studies.

Spider Summary: The aggregated survey results suggest that example selection techniques are an underexplored avenue for the Spider benchmark, particularly when using smaller or outdated language models. The possibility of reaching an 89.7% success rate demonstrates that all the necessary 'needles in the haystack' exist within the Spider training set to achieve state-of-the-art execution accuracy scores. However, using question embedding-based retrieval to is argued to be significantly limited in its effectiveness at finding suitable examples. Regardless of the embedding model used, on average selected examples tend to be of an extremely low quality relative to the target gold. The question, then, is how to develop a ReICL mechanism that can identify the best possible candidates with a high degree of consistency.

3.3 BIRD Results

Survey: Similar results are seen for the BIRD benchmark, albeit with lower success rates due to a large increase in question complexity in comparison to Spider problems. The results of the eight VEM survey for the BIRD benchmark are found in Table 3.9.

k	BERT	RoBERTa	mpnet-v2	Mini-L6	Mini-L12	te3-small	te3-large	ada002
1	0.437	0.428	0.428	0.420	0.437	0.435	0.432	**0.438**
3	0.430	0.429	0.434	0.434	0.448	**0.450**	0.430	0.441
5	0.444	0.432	0.427	0.436	**0.448**	0.441	0.437	0.437

Table 3.9: Research Objective I Survey: BIRD EX Scores

Once again, EX scores do not deviate significantly among the experiments[1], yet here there is less evidence of a linear relationship between the number of few-shot examples used and execution accuracy score. What becomes most apparent is the fact that 18 / 24 of experiments actually perform worse than the zero-shot baseline. Selected examples appear to deter the language model from making successful conversions, leading to a decrease in overall execution accuracy scores. Problems in the BIRD training set tend to be more complex and varied in SQL structure, so an inappropriate recommendation looks to be seriously detrimental to the probability of a successful outcome.

Aggregated Results: A high aggregated EX score is also observed across the BIRD experiments, with a combined **1021 / 1534 (66.6%)** of questions seen to be answered correctly over the twenty-four experiments. A ReICL system capable of replicating this score would achieve eighth place on the BIRD leaderboard. This may be challenging to fully realize, as there is a greater disparity between aggregate and average performance on the BIRD benchmark in comparison to the results observed for Spider. Table 3.10 demonstrates this extended gap between typical performance and optimal performance.

Benchmark	Average Score	Aggregate Score	Agg - Avg
Spider	0.770	0.897	0.127
BIRD	0.436	0.670	0.234

Table 3.10: Average vs Aggregated Score Comparison for Spider and BIRD Benchmarks

[1]The evaluation script for BIRD differs from Spider and has a timeout element for long-running prediction vs gold comparisons. This results in a small amount of potential variability between individual experiment runs, dependent on local hardware and processing resources.

Example Variety Analysis: Average Jaccard index and distance scores are also recorded for the BIRD results, this time revealing a notable difference in the proportion of correct answers shared between experiments. The figures found in Table 3.11 indicate that approximately 30% of correct answers differ between any two given BIRD experiments, in contrast to the roughly 12% of cases observed for Spider.

k	Average Jaccard Index	Average Jaccard Distance
1	0.700	0.300
3	0.697	0.303
5	0.707	0.293

Table 3.11: Research Objective 1 Survey: BIRD Average Jaccard Scores & Distances

The larger average Jaccard distance scores suggests that changing the VEM has a notable impact on which questions are answered correctly. Not only are BIRD questions generally harder for large language models to answer, they also appear more difficult to reliably choose few-shot examples for. Overall, changing the embedding model seems to have a considerable impact on individual BIRD conversion tasks, but less so on the overall execution accuracy score.

Example Quality Analysis: An evaluation matrix is also produced for the 1-shot BIRD survey results. Once again, the evidence shows that a 'good' example is more likely to lead to a success case rather than a failure, cementing the merit of effective example selection techniques. But what is especially prevalent is the number of 'bad' examples chosen across BIRD experiments. In Table 3.12, 96% of cases include an example of low similarity to the target gold. A manual search of the training set reveals that 66% of BIRD benchmark questions have a 'good' example in the training set that could assist the language model, and upon lowering the threshold to 0.75 this increases to a considerable 88%. But once again these examples are often not being correctly identified, indicating that the selection of poor examples for few-shot prompting is likely a major bottleneck to achieving higher BIRD execution accuracy scores.

	Good Example	Bad Example
Correct	352	4948
Incorrect	138	6834

Table 3.12: Research Objective 1 Survey: BIRD Evaluation Matrix

Considerably low average example *SQLSim* scores are also recorded across the experiments. What should be noted alongside the figures in Table 3.13 is the example similarity scores for the random selector experiment. For the 1-shot random example experiment the average example SQLSim score to the target was 0.423, for the 3-shot random experiment the average score was 0.426, and for the 5-shot random experiment the average score was 0.424. This shows that examples chosen by masked-question embedding retrieval are, on average, only slightly more relevant to the conversion task than a randomly selected query.

k	BERT	RoBERTa	mpnet-v2	Mini-L6	Mini-L12	te3-small	te3-large	ada002
1	0.468	0.459	**0.480**	0.466	0.470	0.458	0.466	0.452
3	0.468	0.454	**0.481**	0.467	0.468	0.456	0.464	0.457
5	0.470	0.452	**0.481**	0.466	0.467	0.453	0.462	0.455

Table 3.13: Research Objective 1 Survey: Spider Average Example Similarity Scores

BIRD Summary: The results from the BIRD study reinforce many of the conclusions already drawn in Section 3.2. When considering the aggregated results of the survey, it is apparent that a ReICL approach that always selects the right examples could result in impressive levels of performance. But it is also shown that typical examples selected via a masked-question embedding approach often have little relevance to the target problem. For the BIRD benchmark, this notably results in many experiments producing lower execution accuracy scores than a zero-shot prompting approach.

On average, examples chosen for BIRD have even lower *SQLSim* scores to their respective gold queries than examples chosen for Spider. Considering that for 66% of BIRD problems, there exists at least one 'good' example in the train set to recommend, this must be a fault of the example selection algorithm as opposed to any individual embedding model. The surveys across Chapter 3 build a case against choosing examples based on notions of question similarity, as embedding based retrieval is argued to be misused in this context. Moving forwards[2], the focus of the research is directed towards leveraging embedding models towards more successful example retrieval through the design of new ReICL algorithms and a different choice of embedding space.

[2]Further investigations into the BIRD benchmark are not conducted in the second half of this study. Instead, newly designed ReICL mechanisms aimed at reducing the gap between average and aggregate performance are evaluated against Spider. It is hoped and anticipated that a successfully engineered ReICL mechanism for Spider will be able to translate favourable performance when tested using BIRD.

Chapter 4

RQ2: Supervised Fine-Tuning of Vector Embedding Models

4.1 Supervised Fine-Tuning Pipeline

The results of Chapter 3 demonstrate that all the right training examples exist in both the Spider and BIRD benchmarks to achieve state-of-the-art performance through few-shot prompting methods alone. However, it remains unclear whether a mechanism can be developed to reliably identify these high-quality examples. One unexplored approach to improving the quality of selected examples could be VEM fine-tuning. Ideally, a fine-tuned embedding model would learn to cluster relevant examples near the target problem in the projected vector space. If fine-tuning causes high-quality examples to map closer to the target, they are more likely to be selected as the nearest neighbor.

To use the `sentence-transformer` library's fine-tuning suite, one must first decide on an embedding space. Next, a loss function needs to be chosen that encourages the selection of good examples while discouraging bad ones. In this study, the CoSENT [5] loss function is chosen for this purpose, which requires training examples in the form *(sentenceA, sentenceB, score)*. The score function is expected to be a similarity measure within the range $[-1, 1]$, where a score of 1 indicates a notion of perfect similarity between the sentence pair, and a score of -1 indicates complete dissimilarity.

Once a training dataset of such triples is provided, the `sentence-transformer` library has the facility to fine-tune an existing SBERT-based VEM. Given enough examples, this should produce refined embeddings, with vector positions in the output space

reflecting the similarity scores from the training data. This underscores the importance of establishing an accurate measure of similarity for sentences in the embedding space, which in turn limits the possible approaches to fine-tuning for Text-to-SQL.

4.2 Fine-Tuning of Question Embeddings

One might first consider how to fine-tune a VEM when using context-masked questions as the embedding space. A central lesson from Chapter 3 was that very little about SQL structure can be inferred from the benchmark question alone, as gold queries equally depend on the target database. This makes it difficult to design a general-purpose VEM fine-tuning framework, as a universal notion of sentence similarity cannot exist between two masked-question strings for the purposes of Text-to-SQL. Two examples with identical masked questions could take very different gold queries. Under any VEM encoding, the two instances will share a vector embedding, and there is no possible way of distinguishing which of the two cases should be included in a prompt. This makes it impossible to create a consistent and reliable training methodology.

An overfit approach to fine-tuning question embeddings for the Spider benchmark is detailed in Appendix D as part of this study. The method embeds the standard question space in order to minimize vector overlap, and uses a very basic scoring metric. The Spider experiments from Section 3.2 are utilized, in essence creating an 'answer sheet' for example recommendation. It can perhaps be viewed as a brute-force approach, with the objective of promoting known successful 1-shot examples for each question, and dissuading known fail cases. The aim is to reproduce the observed 86.7% 1-shot aggregate score by manually pointing to which examples will result in a correct conversion.

The fine-tuned embedding produced via this approach leads to one of the highest EX scores observed in the study, but can be seen to promote a backwards approach to the traditional machine learning paradigm. The survey used to generate the training data demanded a huge amount of runtime that is not expected to be replicated by other studies. Also, the prompts that led to a success case when leveraging the `gpt-3.5-turbo` model may not result in a successful conversion if using a different LLM. Most importantly, the approach massively overfits to answer Spider benchmark questions exclusively, providing no room for extrapolation to other Text-to-SQL problems. Thus it is deemed to lack scientific rigor, and is not included in the main body of the study.

4.3 Fine-Tuning of SQL Embeddings

A limiting factor in tackling Text-to-SQL benchmarks is the relative lack of labeled training examples. Both Spider and BIRD contain fewer than 10,000 training examples, which may be insufficient for conducting effective fine-tuning of a large language model. However, a much larger dataset could be generated to fine-tune a vector embedding model instead. Rather than using examples in isolation, examples are considered *pairwise* for model training. By pairing examples in the train-set, over 44 million combinations can be potentially generated with for training. To align with the embedding fine-tuning pipeline, some form of similarity score must be provided with each pair.

A natural conclusion to this idea would be to fine-tune an embedding model using pairwise *SQLSim* scores from the train-set. This would require a ReICL mechanism that embeds the context-masked SQL space, which has not been explored before. Embedding the SQL space could resolve many potential problems identified with using pre-predicted SQLs as the key component of a ReICL mechanism.

- **Retrieval Speed:** Identifying the k most similar examples to a target SQL would require a linear search of the train-set for every benchmark question. This would require a massive computational overhead, and ultimately slow retrieval rates. By embedding the masked SQL and obtaining the k-nearest neighbors, nearby examples will be retrieved much quicker.
- **Avoiding Overfit:** The nearest neighbors in the embedding space are not guaranteed to be the most similar SQLs in the train-set. This could offset potential concerns on whether retrieved examples will take on the negative characteristics of incorrect pre-predictions.
- **Previous Embedding Training:** Embedding models are trained on large natural language corpus, with little exposure to the full variety of possible SQL query structures. By providing a vast quantity of SQL pairs with *SQLSim* values, it is hoped targeted fine-tuning can 'rewrite' some of the predisposed behaviour of a vector embedding model, and encourage the linking of queries based on their SQL similarity over their linguistic similarity.

Complete pseudocode is provided in Appendix B for a selection algorithm that identifies examples through embedding the context-masked pre-predicted SQL space.

4.4 Embedding Fine-Tuning Experiment

In this section, six experiments are conducted against the Spider benchmark to address several key themes of the study. Where relevant, the experiments employ `all-mpnet-base-v2` as the base embedding model used to encode the masked SQLs. Experiments are conducted for $k \in \{1,3,5\}$ as seen before in the previous chapter.

1. **Pre-Pred SQL Similarity + Manual Selection:** The top k examples by *SQLSim* are selected by linear search, relative to the pre-predicted SQL query[1].

2. **Pre-Pred SQL Similarity + Embedding Selection:** The top k closest examples in the embedding space are selected, relative to the pre-predicted SQL query.

3. **Pre-Pred SQL Similarity + Fine-Tuned Embedding Selection:** The top k closest examples in the embedding space are selected, relative to the pre-predicted SQL query (using a fine-tuned embedding model).

4. **SQL Similarity + Manual Selection:** The top k examples by *SQLSim* are selected by linear search, relative to the true gold SQL query.

5. **SQL Similarity + Embedding Selection:** The top k closest examples in the embedding space are selected, relative to the true gold SQL query.

6. **SQL Similarity + Fine-Tuned Embedding Selection:** The top k closest examples in the embedding space are selected, relative to the pre-predicted SQL query (using a fine-tuned embedding model).

By conducting these experiments, the benefits and drawbacks of selecting examples based on some notion of SQL similarity should become clear. Each experiment looks to provide a different insight into ReICL methods that focus on SQL similarity selection, with some key points of interest highlighted below.

- Experiments 1/2 and 4/5 examine how much performance is lost when transitioning from expensive manual selection to efficient embedding based retrieval. On average, manually determining the top five examples for a target using *SQLSim* takes over one-hundred times longer than embedding-based retrieval. If examples of similar quality to the true top k candidates can be quickly retrieved without an exhaustive search, then the ReICL mechanism can be considered a success.

[1]The file of pre-predicted SQLs are sourced directly from the DAIL-SQL study [4], available at `https://github.com/BeachWang/DAIL-SQL/tree/main/results`

- Experiments 2/3 and 4/6 look to explore whether a fine-tuned embedding system can retrieve better quality examples than the equivalent standard base model. The `all-mpnet-base-v2` model is fine-tuned using 300,000 distinct training tuples[2], each of the form *(SQL_1, SQL_2, score)*. These SQLs are context-masked using *SQLMask* and the similarity score is obtained using *SQLSim*.

- Experiments 4, 5, and 6 each attempt to find the *k* best examples by comparing candidates to the gold queries for each Spider task. Of course, this is an unrealizable construction that assumes the selection model has access to the true benchmark answers. This is conducted to assess how well `gpt-3.5-turbo` can perform when provided with examples of a guaranteed quality. This set of experiments can be viewed in contrast to experiments 1, 2, and 3 which offer practical solutions for SQL similarity based example selection by using pre-predictions.

Results: The experiment outcomes display some of the highest Spider benchmark scores recorded across the study, with the best performance achieved when manually providing the true top examples by *SQLSim* score. It is also clear that using an embedding model to speed up example retrieval leads to minimal compromise in EX score, in fact this ultimately improves performance in the set of pre-prediction experiments.

k	DAIL	Pre-Pred Manual	Pre-Pred Embed.	Pre-Pred SFT	Gold Manual	Gold Embed.	Gold SFT
1	0.763	0.788	0.775	0.774	0.806	0.779	0.789
3	0.773	0.783	0.784	0.781	0.810	0.799	0.795
5	0.780	0.789	0.791	0.791	0.810	0.809	0.804

Table 4.1: Comparison of Spider EX Performance Across Different Selection Methods

The results in Table 4.1 suggest that selecting examples based on masked pre-predicted SQL similarity is an effective approach to example selection. All such experiments outperform the best results from the question embedding survey in Chapter 3. In other words, the highest outcome for each *k* when embedding the question space is lower than the worst outcome observed when using pre-predicted SQL selection. This approach also beats out the DAIL Selection algorithm [4] at each level of *k* when reproduced locally, whilst retrieving examples of far higher similarity to the gold query (Tables 4.1 & 4.2). There is still an approximate 2% to 3% disparity between each

[2]Fine-tuning was conducted for 8 epochs using Google Colab's A100 GPU. The final model achieved a loss of 0.0985 after the complete training procedure. The fine-tuned VEM has been published to huggingface and is now accessible through the `sentence-transformers` library by calling "`model = SentenceTransformers(s2593817/sft-sql-embedding)`".

pre-prediction experiment and its corresponding gold query experiment at each tier. This gap makes clear that practical implementations are quite a way off in their ability to select the best possible examples from the training sets. Of course, if the set of pre-predictions was of a higher quality then this problem would be less prevalent.

Example Quality Analysis: Disappointingly, the fine-tuning procedure results in lower execution accuracy scores than the base model in both pre-prediction and gold query selection experiments. This may appear like the embedding fine-tuning results in worse examples being selected, but this is actually not the case. For each experiment, the mean example quality is calculated by measuring the average *SQLSim* score between the examples SQLs retrieved by the selection algorithm and the target gold query. Table 4.3 illustrates that, on average, examples offered up by a fine-tuned embedding model tend to take a higher similarity score than examples chosen by the standard VEM.

k	DAIL	Pre-Pred Manual	Pre-Pred Embed.	Pre-Pred SFT	Gold Manual	Gold Embed.	Gold SFT
1	0.581	0.724	0.718	0.723	0.841	0.818	0.829
3	0.571	0.724	0.715	0.721	0.837	0.811	0.824
5	0.566	0.722	0.711	0.719	0.834	0.804	0.820

Table 4.2: Comparison of Average Example Similarity Scores Relative to Gold Queries

Evidently, the fine-tuned embedding model is very effective at quickly obtaining SQLs similar to a target. This is a major success of the study. Introducing a new way of quickly retrieving similar SQLs to a target from a dataset is likely the most practical takeaway from this research, with wide-ranging applications. Inspecting the results files of the fine-tuned gold embedding experiment shows that the nominated examples are indeed structurally similar to the target gold, demonstrating that the mechanism works as intended. Unfortunately, this does not directly result in improved EX scores.

A contributing factor as to why this may be the case for the pre-predicted SQL experiments is the issue of overfit. To prove this, average example similarity is recorded relative to the pre-prediction queries, as opposed to the gold queries. When comparing Table 4.2 with Table 4.3, it becomes clear that examples recommended in experiments 1, 2 and 3 have much higher similarity to the pre-predicted query than they do with the target gold. This outcome is expected by design, but the large gap in average similarities indicates that SQLs are being overfit to the prediction rather than the target. Consequently, for incorrect pre-predictions, selected examples are likely to exacerbate any faults of the original prediction and reduce the chances of successful conversion.

The fine-tuned VEM tends to select examples that are even closer to the pre-predicted query than the base VEM's nominations, which has made the issue of overfit more pronounced. This means that embedding the pre-predicted query using the standard `all-mpnet-base-v2` model is the most successful realizable approach, as it allows for fast retrieval and reduces the effects of overfitting to flawed pre-predicted queries.

k	Pre-Pred Manual	Pre-Pred Embed.	Pre-Pred SFT
1	0.925	0.889	0.901
3	0.920	0.882	0.894
5	0.914	0.874	0.889

Table 4.3: Comparison of Average Example Similarity Scores Relative to Pre-Predictions

Summary: The experiments have demonstrated that combining *SQLMask* with embedding-based retrieval is a highly effective method for identifying SQL queries with a structure similar to the target query. A large number of training instances can be generated from relatively few SQLs to fine-tune an embedding model, which has been shown to further improve the quality of selected examples. Still, presenting a language model with highly similar examples to the target gold query does not guarantee successful conversion. Fine-tuning can also result in selecting examples that are overfit to a flawed pre-prediction, leading to lower EX scores compared to the base model.

That said, achieving a score of 81% on the Spider benchmark with the prior-generation `gpt-3.5-turbo` base model is a huge success. According to Table 2 from the DAIL-SQL paper [4], the upper limit for `gpt-3.5-turbo` using 5-shot DAIL selection is 79.6%. This assumes the selector can reference the gold query, as in experiments 4, 5, and 6. The embedded gold SQL selection algorithm extends this upper limit to 80.9%, suggesting this approach could be superior to the state-of-the-art DAIL method if tested with the premier `gpt-4-turbo` or `gpt-4o` language models.

Upon review, the results files for the gold query embedding experiments exhibit the consistent retrieval of examples that are relevant to the Text-to-SQL conversion tasks. It is theorized that a chain-of-thought prompting style, that fully explains why these selected examples are relevant to generating the target, would significantly increase EX scores. Any continuation of the study would focus on developing new prompting techniques to best support any language model learning by analogy from these examples.

Chapter 5

Conclusion

5.1 Study Overview

The research conducted has produced a series of results that can inform future Text-to-SQL example selection studies, especially if utilising embedding models for retrieval. Aggregated results from a large survey of embedding models make clear for the first time that sufficient examples exist in both the Spider and BIRD training sets to achieve near state-of-the-art performance through few-shot prompting techniques alone. This is particularly notable as such scores can be achieved while using outdated language models that are less capable than the current top of the range commercial offerings.

Altering the embedding model has been shown to impact which examples are selected, but has relatively little impact on overall EX scoring and average example quality. Algorithms that select examples primarily based on notions of question similarity are criticised for producing examples of low relevance to the target gold query. In response, a ReICL mechanism has been proposed that embeds the context-masked SQL space, which results in some of the highest scores observed across the paper, as well as one of the highest recorded execution accuracies to date for any Spider submission leveraging the `gpt-3.5-turbo` model. This can be viewed as a milestone success for the study.

The method outlined provides a concrete and effective way of retrieving structurally similar SQLs to a target, with the quality of selected examples being further enhanced when using a fine-tuned embedding model. The novel context-masking script *SQLMask* establishes a standard format for SQL queries and is designed to complement the *SQLSim* metric. When used alongside embedding based retrieval, this certainly appears

to be the current best way of retrieving similar SQLs from a dataset, which could have wide-ranging applications. Significantly, practical implementations which embed masked pre-predicted SQL queries outperform the state-of-the-art DAIL Selection algorithm when tested with `gpt-3.5-turbo` on the Spider benchmark.

5.2 RQ1 & RQ2 Conclusions

A brief conclusion is provided to each of the central research questions that lie at the heart of the paper to summarize the key takeaways and inform future works.

RQ1: *To what extent does the choice of vector embedding model influence the quality of examples chosen for few-shot Text-to-SQL conversion tasks?*

The survey of embedding models conducted in Chapter 3 made clear that altering the model often results in different examples selected for prompting. However very little variation in overall EX scores or example quality is seen amongst experiments. This led to the central conclusion that the choice of embedding model is **not** as important as the choice of embedding space or the design of the retrieval algorithm. In the survey, examples are selected by determining which examples have questions that are most the semantically and linguistically similar to the target question. It's argued how this is not a good approach to choosing examples for Text-to-SQL, as a problem's question can have very little correlation with the problem's gold SQL. This method of selection abstracts entirely from the database related to the target problem, resulting in many cases where the examples chosen by a VEM have queries with almost no structural similarity to the target gold.

A positive outcome of this inconsistent and highly variable approach to example selection was the broad range of example SQLs exhibited throughout the survey. Over 89% of Spider problems and 66% of BIRD problems have been shown to be answerable using few-shot prompting methods alone. This surpasses all preconceived notions of what a `gpt-3.5` class model could achieve through few-shot prompting. The challenge lies in developing a single mechanism capable of choosing all the right examples to achieve this ideal performance. It is believed that no single question embedding mechanism could retrieve all these examples while still being able to generalize to new unseen Text-to-SQL problems. Therefore, focus should instead be assigned to designing new retrieval algorithms that avoid using question similarity as the primary metric for example quality, and instead uses it as a guiding heuristic.

RQ2: *To what extent can the fine-tuning of vector embedding models be applied to achieve better quality examples for Text-to-SQL generation?*

The fine-tuning procedure outlined in Chapter 4 led to the retrieval of examples with higher average *SQLSim* scores to the target query when compared to the base model. This is believed to have established the most effective and efficient approach to selecting similarly structured SQL queries from a dataset. The retrieval method works in conjunction with *SQLMask*, which introduces a new standard format for SQL queries by abstracting away from database-dependent labels. By embedding the masked SQL space, similar queries to a target can be quickly identified in a context-agnostic fashion.

Disappointingly, the fine-tuned embedding experiments yield lower Spider execution accuracy scores than their base model counterparts, even if they produce examples of a higher similarity to the true gold query. This indicates that there is more to Text-to-SQL few-shot prompting than simply providing examples with highly relevant SQLs. I.e. In higher-shot scenarios, the selected examples could have a cumulative effect beyond their individual influence on the language model. This aspect has not been considered when designing selection algorithms in this study. It is believed that the examples nominated from the embedded gold SQL experiments should be good enough to direct correct conversions if using a stronger language model, especially if used alongside a chain-of-thought prompting approach, and schema-linking modules that can provide concise database representations.

5.3 Future Works

The primary suggestion from this study is to repeat the experiments of Chapter 4 using a premier commercial language model such as `gpt-4-turbo`. This would require a larger financial outlay than was deemed feasible for this study but would provide valuable insights into whether the proposed techniques can truly rival the top-performing DAIL-SQL submission [4]. Additionally, it would clarify to what extent incorrect answers can be attributed to language model limitations versus few-shot prompting limitations. If a highly capable language model is unable to answer correctly even when manually provided with the k most similar training queries to the gold, this may challenge the conventional wisdom of what constitutes a good example.

Another reccomendation would be to repeat the experiments of Chapter 4 against the BIRD benchmark. This was not conducted in this study as no file of pre-predictions for BIRD problems is included in the DAIL source code repository, so no meaningful cross-comparison can be made. Average example similarity scores from the Chapter 3 BIRD experiment were notably poor, so it would be of interest to see how well a LLM would respond when prompted with example queries of a guaranteed level of relevance.

Given that all the examples necessary for top performance have been shown to exist in the Spider and BIRD train sets, any alternative mechanism of consistently retrieving them would be highly valuable. It is concluded that a move away from embedding-based retrieval may be best for future selection algorithms, and any truly effective selection algorithm should consider question structure, database structure, as well as some notion of expected SQL structure too. It is unlikely that all of this information could be captured by a single sentence embedding model, which are primarily designed for basic natural language inputs.

5.4 Closing Remarks

Example selection for Text-to-SQL is evidently a non-trivial task. Attempting to predict the best training cases to include in a prompt without a prior knowledge of the gold SQL structure is very difficult. Embeddings are often used as an efficient mechanism of identifying potentially similar examples, but in this study it has been shown that they are ineffective at retrieving quality candidates when encoding the question space.

It is conjectured that Text-to-SQL benchmarks could see leaderboard topping submissions that focus on prompt engineering techniques alone. The pre-predicted SQL techniques outlined in Chapter 4 can be used to successfully retrieve examples with a strong degree of similarity to the provided ‘first guess’ query. Combining this with a chain-of-thought prompting method, which explains how the example golds are derived from their respective databases, could open a promising path for future research.

Although language models cannot solely rely on few-shot demonstrations for successful conversions, ReICL techniques for Text-to-SQL should not be underestimated, as this report has revealed that there is untapped potential that can be unlocked for the Spider and BIRD benchmarks - simply by deploying effective example selection.

Bibliography

[1] Ben Bogin, Matt Gardner, and Jonathan Berant. Global reasoning over database structures for text-to-SQL parsing. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3659–3664, Hong Kong, China, November 2019. Association for Computational Linguistics.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[3] Catherine Finegan-Dollak, Jonathan K. Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. Improving text-to-sql evaluation methodology. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 351–360. Association for Computational Linguistics, 2018.

[4] Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. Text-to-sql empowered by large language models: A benchmark evaluation. *Proc. VLDB Endow.*, 17(5):1132–1145, may 2024.

[5] Xiang Huang, Hao Peng, Dongcheng Zou, Zhiwei Liu, Jianxin Li, Kay Liu, Jia Wu, Jianlin Su, and Philip S. Yu. Cosent: Consistent sentence embedding via similarity ranking. *IEEE/ACM Trans. Audio, Speech and Lang. Proc.*, 32:2800–2813, may 2024.

[6] Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. A survey on large language models for code generation. *arXiv preprint arXiv:2406.00515*, 2024.

[7] Wenqiang Lei, Weixin Wang, Zhixin Ma, Tian Gan, Wei Lu, Min-Yen Kan, and Tat-Seng Chua. Re-examining the role of schema linking in text-to-sql. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6943–6954, 2020.

[8] Wenqiang Lei, Weixin Wang, Zhixin Ma, Tian Gan, Wei Lu, Min-Yen Kan, and Tat-Seng Chua. Re-examining the role of schema linking in text-to-SQL. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6943–6954, Online, November 2020. Association for Computational Linguistics.

[9] Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. Resdsql: decoupling schema linking and skeleton parsing for text-to-sql. In *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'23/IAAI'23/EAAI'23. AAAI Press, 2023.

[10] Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, and Hong Chen. Codes: Towards building open-source language models for text-to-sql. *arXiv preprint arXiv:2402.16347*, 2024.

[11] Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, and Hong Chen. Codes: Towards building open-source language models for text-to-sql. *arXiv preprint arXiv:2402.16347*, 2024.

[12] Jia Li, Ge Li, Chongyang Tao, Huangzhao Zhang, Fang Liu, and Zhi Jin. Large language model-aware in-context learning for code generation. *arXiv preprint arXiv:2310.09748*, 2023.

[13] Jinyang Li, Binyuan Hui, Reynold Cheng, Bowen Qin, Chenhao Ma, Nan Huo, Fei Huang, Wenyu Du, Luo Si, and Yongbin Li. Graphix-t5: mixing pre-trained transformers with graph-aware layers for text-to-sql parsing. In *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence*

and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence, AAAI'23/IAAI'23/EAAI'23. AAAI Press, 2023.

[14] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin C.C. Chang, Fei Huang, Reynold Cheng, and Yongbin Li. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2024. Curran Associates Inc.

[15] Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. What makes good in-context examples for gpt-3? *arXiv preprint arXiv:2101.06804*, 2021.

[16] Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China, November 2019. Association for Computational Linguistics.

[17] Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. Mpnet: masked and permuted pre-training for language understanding. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc.

[18] Y. Song, S. Ezzini, X. Tang, C. Lothritz, J. Klein, T. Bissyande, A. Boytsov, U. Ble, and A. Goujon. Enhancing text-to-sql translation for financial system design. In *2024 IEEE/ACM 46th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 252–262, Los Alamitos, CA, USA, apr 2024. IEEE Computer Society.

[19] Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. Chess: Contextual harnessing for efficient sql synthesis. *arXiv preprint arXiv:2405.16755*, 2024.

[20] Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. Order matters: Sequence to sequence for sets. *arXiv preprint arXiv:1511.06391*, 2015.

[21] Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. RAT-SQL: Relation-aware schema encoding and linking for text-to-SQL parsers. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7567–7578, Online, July 2020. Association for Computational Linguistics.

[22] Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Qian-Wen Zhang, Zhao Yan, and Zhoujun Li. Mac-sql: Multi-agent collaboration for text-to-sql. *arXiv preprint arXiv:2312.11242*, 2023.

[23] Wenhui Wang, Hangbo Bao, Shaohan Huang, Li Dong, and Furu Wei. MiniLMv2: Multi-head self-attention relation distillation for compressing pretrained transformers. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli, editors, *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 2140–2151, Online, August 2021. Association for Computational Linguistics.

[24] Yiqing Xie, Alex Xie, Divyanshu Sheth, Pengfei Liu, Daniel Fried, and Carolyn Rose. Codebenchgen: Creating scalable execution-based code generation benchmarks. *arXiv preprint arXiv:2404.00566*, 2024.

[25] Xiaojun Xu, Chang Liu, and Dawn Song. Sqlnet: Generating structured queries from natural language without reinforcement learning. *arXiv preprint arXiv:1711.04436*, 2017.

[26] Tao Yu, Zifan Li, Zilin Zhang, Rui Zhang, and Dragomir Radev. TypeSQL: Knowledge-based type-aware neural text-to-SQL generation. In Marilyn Walker, Heng Ji, and Amanda Stent, editors, *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 588–594, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

[27] 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. Spider: A large-scale human-labeled dataset for complex and cross-domain

semantic parsing and text-to-SQL task. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii, editors, *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921, Brussels, Belgium, October-November 2018. Association for Computational Linguistics.

[28] Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*, 2017.

[29] Liu Zhuang, Lin Wayne, Shi Ya, and Zhao Jun. A robustly optimized BERT pre-training approach with post-training. In Sheng Li, Maosong Sun, Yang Liu, Hua Wu, Kang Liu, Wanxiang Che, Shizhu He, and Gaoqi Rao, editors, *Proceedings of the 20th Chinese National Conference on Computational Linguistics*, pages 1218–1227, Huhhot, China, August 2021. Chinese Information Processing Society of China.

Appendix A

Exemplar Benchmark Prompts

A.1 Spider Example Prompt

The following prompt is generated for the first question in the Spider dev set, namely: "How many singers do we have?", from the 'concert_singers' database, with gold query "SELECT count(*) FROM Singers". The prompt derives from the *all-mpnet-base-v2* 1-shot experiment, using DAIL-organisation and CR database representation.

PROMPT:

```
/* Some SQL examples are provided based on similar problems: */
\\/* Answer the following: How many aircrafts do we have? */
\\SELECT count(*) FROM Aircraft
\\
\\/* Given the following database schema: */
\\CREATE TABLE "stadium" (
\\"Stadium\_ID" int,
\\"Location" text,
\\"Name" text,
\\"Capacity" int,
\\"Highest" int,
\\"Lowest" int,
\\"Average" int,
\\PRIMARY KEY ("Stadium\_ID")
\\)
\\
```

```
\\CREATE TABLE "singer" (
\\"Singer\_ID" int,
\\"Name" text,
\\"Country" text,
\\"Song\_Name" text,
\\"Song\_release\_year" text,
\\"Age" int,
\\"Is\_male" bool,
\\PRIMARY KEY ("Singer\_ID")
\\)
\\
\\CREATE TABLE "concert" (
\\"concert\_ID" int,
\\"concert\_Name" text,
\\"Theme" text,
\\"Stadium\_ID" text,
\\"Year" text,
\\PRIMARY KEY ("concert\_ID"),
\\FOREIGN KEY ("Stadium\_ID") REFERENCES "stadium"("Stadium\_ID")
\\)
\\
\\CREATE TABLE "singer\_in\_concert" (
\\"concert\_ID" int,
\\"Singer\_ID" text,
\\PRIMARY KEY ("concert\_ID","Singer\_ID"),
\\FOREIGN KEY ("concert\_ID") REFERENCES "concert"("concert\_ID"),
\\FOREIGN KEY ("Singer\_ID") REFERENCES "singer"("Singer\_ID")
\\)
\\
\\/* Answer the following: How many singers do we have? */
\\SELECT
```

A.2 BIRD Example Prompt

The following prompt is generated for the first question in the BIRD dev set, namely: "What is the highest eligible free rate for K-12 students in the schools in Alameda County? Eligible free rate for K-12 = 'Free Meal Count (K-12)' / 'Enrollment (K-12)'", from the 'california_schools' database, with gold query "Free Meal Count (K-12)' / 'Enrollment (K-12)' FROM frpm WHERE 'County Name' = 'Alameda' ORDER BY (CAST('Free Meal Count (K-12)' AS REAL) / 'Enrollment (K-12)') DESC LIMIT 1". The prompt derives from the *all-mpnet-base-v2* 1-shot experiment, using DAIL-organisation and CR database representation, with additional context included.

PROMPT:

```
/* Some SQL examples are provided based on similar problems: */
\\
\\/* Answer the following: What is the highest infant mortality rate
\\ per thousand of the countries whose inflation is under 3? */
\\ SELECT MAX(T2.Infant\_Mortality) FROM economy AS T1 INNER JOIN
\\ population AS T2 ON T1.Country = T2.Country WHERE T1.Inflation < 3
\\
\\/* Given the following database schema: */
\\CREATE TABLE frpm
\\(
\\    CDSCode                                    TEXT not null
\\        primary key,
\\    'Academic Year'                            TEXT  null,
\\    'County Code'                              TEXT  null,
\\    'District Code'                            INTEGER         null,
\\    'School Code'                              TEXT  null,
\\    'County Name'                              TEXT null,
\\    'District Name'                            TEXT null,
\\    'School Name'                              TEXT null,
\\    'District Type'                            TEXT null,
\\    'School Type'                              TEXT null,
\\    'Educational Option Type'                  TEXT null,
\\    'NSLP Provision Status'                    TEXT null,
```

```
\\    'Charter School (Y/N)'                          INTEGER    null,
\\    'Charter School Number'                         TEXT  null,
\\    'Charter Funding Type'                          TEXT null,
\\    IRC                                             INTEGER    null,
\\    'Low Grade'                                     TEXT  null,
\\    'High Grade'                                    TEXT null,
\\    'Enrollment (K-12)'                             REAL      null,
\\    'Free Meal Count (K-12)'                        REAL       null,
\\    'Percent (\%) Eligible Free (K-12)'              REAL       null,
\\    'FRPM Count (K-12)'                             REAL       null,
\\    'Percent (\%) Eligible FRPM (K-12)'              REAL       null,
\\    'Enrollment (Ages 5-17)'                        REAL       null,
\\    'Free Meal Count (Ages 5-17)'                   REAL       null,
\\    'Percent (\%) Eligible Free (Ages 5-17)'         REAL       null,
\\    'FRPM Count (Ages 5-17)'                        REAL       null,
\\    'Percent (\%) Eligible FRPM (Ages 5-17)'         REAL       null,
\\    '2013-14 CALPADS Fall 1 Certification Status' INTEGER    null,
\\    foreign key (CDSCode) references schools (CDSCode)
\\)
\\
\\CREATE TABLE satscores
\\(
\\    cds         TEXT not null
\\        primary key,
\\    rtype       TEXT  not null,
\\    sname       TEXT null,
\\    dname       TEXT null,
\\    cname       TEXT null,
\\    enroll12    INTEGER         not null,
\\    NumTstTakr  INTEGER          not null,
\\    AvgScrRead  INTEGER          null,
\\    AvgScrMath  INTEGER          null,
\\    AvgScrWrite INTEGER          null,
\\    NumGE1500   INTEGER          null,
\\--     PctGE1500   double      null,
```

```
\\        foreign key (cds) references schools (CDSCode)
\\)
\\
\\CREATE TABLE schools
\\(
\\    CDSCode     TEXT not null
\\        primary key,
\\    NCESDist    TEXT  null,
\\    NCESSchool  TEXT  null,
\\    StatusType  TEXT  not null,
\\    County      TEXT not null,
\\    District    TEXT not null,
\\    School      TEXT null,
\\    Street      TEXT null,
\\    StreetAbr   TEXT null,
\\    City        TEXT null,
\\    Zip         TEXT null,
\\    State       TEXT  null,
\\    MailStreet  TEXT null,
\\    MailStrAbr  TEXT null,
\\    MailCity    TEXT null,
\\    MailZip     TEXT null,
\\    MailState   TEXT  null,
\\    Phone       TEXT null,
\\    Ext         TEXT  null,
\\    Website     TEXT null,
\\    OpenDate    DATE        null,
\\    ClosedDate  DATE        null,
\\    Charter     INTEGER    null,
\\    CharterNum  TEXT  null,
\\    FundingType TEXT null,
\\    DOC         TEXT  not null,
\\    DOCType     TEXT not null,
\\    SOC         TEXT  null,
\\    SOCType     TEXT null,
```

```
\\    EdOpsCode   TEXT  null,
\\    EdOpsName   TEXT null,
\\    EILCode     TEXT  null,
\\    EILName     TEXT null,
\\    GSoffered   TEXT null,
\\    GSserved    TEXT  null,
\\    Virtual     TEXT  null,
\\    Magnet      INTEGER   null,
\\    Latitude    REAL      null,
\\    Longitude   REAL      null,
\\    AdmFName1   TEXT null,
\\    AdmLName1   TEXT null,
\\    AdmEmail1   TEXT null,
\\    AdmFName2   TEXT null,
\\    AdmLName2   TEXT null,
\\    AdmEmail2   TEXT null,
\\    AdmFName3   TEXT  null,
\\    AdmLName3   TEXT null,
\\    AdmEmail3   TEXT null,
\\    LastUpdate  DATE        not null
\\)
\\
\\/* Answer the following: What is the highest eligible free rate
\\ for K-12 students in the schools in Alameda County? Eligible
\\ free rate for K-12 = `Free Meal Count (K-12)` / `Enrollment (K-12)` */
\\SELECT
```

Appendix B

Example Selection Algorithms

Pseudocode for retrieval-based in-context learning example selection algorithms discussed across Chapters 3 and 4 are outlined in this Appendix. Also included is an outline of the DAIL-SQL embedding algorithm as defined by Gao et al. in "Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation"

B.1 Embedded Masked Question Selector

As featured across all vector embedding model survey experiments across Section 3.2 *"Spider Results"*, and Section 3.3 *"BIRD Results"*.

Algorithm 3 Select Examples + Masked Question Embeddings

Require: Q: Target Question, $\mathcal{D}$: Target Database, k: Number of Examples, train_set: Benchmark Train Set, embedding_model: VEM

function GET_EXAMPLES_MASK(Q, $\mathcal{D}$, k, train_set, embedding_model)
    $train\_questions \leftarrow train\_set[\text{"}question\text{"}]$
    $masked\_questions \leftarrow \texttt{mask\_questions\_with\_schema\_link}(train\_questions, train\_set)$
    $train\_embeddings \leftarrow embedding\_model.\texttt{encode}(masked\_questions)$
    $masked\_target \leftarrow \texttt{mask\_questions\_with\_schema\_link}(Q, \mathcal{D})$
    $target\_embedding \leftarrow embedding\_model.\texttt{encode}(masked\_target)$
    $distances \leftarrow \texttt{compute\_distances}(target\_embedding, train\_embeddings)$
    $nearest\_neighbors \leftarrow \texttt{closest\_k\_indices}(distances, k)$
    $selected\_examples \leftarrow train\_set[nearest\_neighbors]$
    return $selected\_examples$

B.2 Embedded Pre-Predicted SQL Selector

As featured in Experiments 2 & 3 of Section 4.4 *"Embedding Fine-Tuning Experiment"*

Algorithm 4 Select Examples + Masked Pre-Pred SQL Embedding

Require: pred_sql: Target Pre-Prediction, $\mathcal{D}$: Target Database, k: Number of Examples, train_set: Benchmark Train Set, embedding_model: VEM

function GET_EXAMPLES_PRE_PRED(pred_sql, k, train_set, embedding_model)
    $train\_sqls \leftarrow train\_set[\text{"}query\text{"}]$
    $masked\_sqls \leftarrow \texttt{SQLMask}(train\_sqls)$
    $train\_embeddings \leftarrow embedding\_model.\texttt{encode}(masked\_sqls)$
    $masked\_pred \leftarrow \texttt{SQLMask}(pred\_sql)$
    $target\_embedding \leftarrow embedding\_model.\texttt{encode}(masked\_pred)$
    $distances \leftarrow \texttt{compute\_distances}(target\_embedding, train\_embeddings)$
    $nearest\_neighbors \leftarrow \texttt{closest\_k\_indices}(distances, k)$
    $selected\_examples \leftarrow train\_set[nearest\_neighbors]$
    return $selected\_examples$

B.3 Embedded Gold SQL Selector

As featured in Experiments 5 & 6 of Section 4.4 *"Embedding Fine-Tuning Experiment"*

Algorithm 5 Select Examples + Masked Gold SQL Embedding

Require: $\mathcal{G}$: Target Gold Query, $\mathcal{D}$: Target Database, k: Number of Examples, train_set: Benchmark Train Set, embedding_model: VEM

function GET_EXAMPLES_GOLD($\mathcal{G}$, k, train_set, embedding_model)
    $train\_sqls \leftarrow train\_set[\text{"}query\text{"}]$
    $masked\_sqls \leftarrow \texttt{SQLMask}(train\_sqls)$
    $train\_embeddings \leftarrow embedding\_model.\texttt{encode}(masked\_sqls)$
    $masked\_gold \leftarrow \texttt{SQLMask}(\mathcal{G})$
    $target\_embedding \leftarrow embedding\_model.\texttt{encode}(masked\_gold)$
    $distances \leftarrow \texttt{compute\_distances}(target\_embedding, train\_embeddings)$
    $nearest\_neighbors \leftarrow \texttt{closest\_k\_indices}(distances, k)$
    $selected\_examples \leftarrow train\_set[nearest\_neighbors]$
    return $selected\_examples$

B.4 Manual Pre-Predicted SQL Selector

As featured in Experiment 1 of Section 4.4 *"Embedding Fine-Tuning Experiment"*

Algorithm 6 Select Examples + Manual Pre-Pred SQL Similarity

Require: pred_sql: Target Pre-Prediction, $\mathcal{D}$: Target Database, k: Number of Examples, train_set: Benchmark Train Set

```
function MANUAL_EXAMPLES_PRE_PRED(pred_sql, k, train_set)
    train_sqls ← train_set["query"]
    masked_sqls ← SQLMask(train_sqls)
    masked_pred ← SQLMask(pred_sql)
    best_examples ← [ ]
    for i, candidate in enumerate(masked_sqls) do
        score ← SQLSim(masked_pred, candidate)
        if len(best_examples) < k then
            best_examples.append((score, train_set[i]))
            # Sort the candidate list to ensure the k^th highest scorer is
            always at the end of the list
            best_examples.sort(key = score, reverse = True)
        else
            # If example is in top k candidates by SQLSim score, remove
            the old k^th highest scorer and insert the new example
            if score > best_examples[-1]) then
                best_examples[-1] = (score, train_set[i])
                best_examples.sort(key = score, reverse = True)
                return [example for score, example in best_examples]
```

B.5 Manual Gold SQL Selector

As featured in Experiment 4 of Section 4.4 *"Embedding Fine-Tuning Experiment"*

Algorithm 7 Select Examples + Manual Gold SQL Similarity

Require: $\mathcal{G}$: Target Gold Query, $\mathcal{D}$: Target Database, k: Number of Examples, train_set: Benchmark Train Set

```
function MANUAL_EXAMPLES_GOLD(G, k, train_set)
    train_sqls ← train_set["query"]
    masked_sqls ← SQLMask(train_sqls)
    masked_gold ← SQLMask(G)
    best_examples ← [ ]
    for i, candidate in enumerate(masked_sqls) do
        score ← SQLSim(masked_gold, candidate)
        if len(best_examples) < k then
            best_examples.append((score, train_set[i]))
            # Sort the candidate list to ensure the k^th highest scorer is
            always at the end of the list
            best_examples.sort(key = score, reverse = True)
        else
            # If example is in top k candidates by SQLSim score, remove
            the old k^th highest scorer and insert the new example
            if score > best_examples[-1]) then
                best_examples[-1] = (score, train_set[i])
                best_examples.sort(key = score, reverse = True)
                return [example for score, example in best_examples]
```

B.6 DAIL Selector

Adapted from Appendix A.1 of Gao et. al's *"Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation"* [4]

Algorithm 8 DAIL Selection

Require: Q: Target Question, train_set: Set of Training Examples, k: Number of Examples, pre_pred: Target SQL Pre-Prediction, embedding_model: VEM

```
function GET_EXAMPLES_DAIL(pred_sql, k, train_set, embedding_model)
    # Generate sentence embeddings for train set questions and target
    masked_questions ← question_mask_with_schema_link(train_set["question"])
    train_embeddings ← embedding_model.encode(masked_questions)
    masked_target ← question_mask_with_schema_link(Q)
    target_embedding ← embedding_model.encode(masked_target)
    # DAIL Mask the train set queries & target pre-prediction
    masked_sqls ← DAILMask(train_set["query"])
    masked_pred ← DAILMask(pre_pred)
    # Use question similarity as heuristic for good examples
    sort train_set by cosine_similarity(target_embedding, train_embeddings[i])
    # Use DAILSim as threshold measure for example quality
    for i, example in enumerate(train_set) do
        if DAILSim(pre_pred, masked_sqls[i]) ≥ 0.85 then
            top_examples.append(example)
        else
            backup_examples.append(example)
    ranked_examples ← top_examples + backup_examples
    selected_examples ← ranked_examples[0 : k]
    return selected_examples
```

Appendix C

SQL context-masking & Similarity Examples

C.1 *DAILMask* vs *SQLMask* Masking Examples

EXAMPLE 1	
SQL:	`SELECT DISTINCT Country FROM singer WHERE Age > 20`
DAIL MASK:	`select distinct _ from _ where _`
PROP. MASK:	`SELECT DISTINCT col1 FROM table1 WHERE col2 > num`

EXAMPLE 2	
SQL:	`SELECT Name, Country, Age FROM singer ORDER BY Age DESC`
DAIL MASK:	`select _ from _ order by _ desc`
PROP. MASK:	`SELECT col1 , col2 , col3 FROM table1 ORDER BY col3 DESC`

EXAMPLE 3	
SQL:	`SELECT Song_Name, Song_release_year FROM singer WHERE Age = (SELECT min(Age) FROM singer)`
DAIL MASK:	`select _ from _ where _ = ( select min ( _ ) from _ )`
PROP. MASK:	`SELECT col1 , col2 FROM table1 WHERE col3 = (SELECT min(col3) FROM table1)`

EXAMPLE 4	
SQL:	`SELECT country , count(*) FROM singer GROUP BY country`
DAIL MASK:	`select __ , count ( __ ) from __ group by __`
PROP. MASK:	`SELECT col1 , count(*) FROM table1 GROUP BY col1`

EXAMPLE 5	
SQL:	`SELECT s.Song_Name FROM singer AS s WHERE s.Age > (SELECT avg(Age) FROM singer)`
DAIL MASK:	`select __ from __ where __ > ( select avg ( __ ) from __ )`
PROP. MASK:	`SELECT alias1.col1 FROM table1 AS alias1 WHERE alias1.col2 > (SELECT avg(col2) FROM table1)`

EXAMPLE 6	
SQL:	`SELECT Location, Name FROM stadium WHERE Capacity BETWEEN 5000 AND 10000`
DAIL MASK:	`select __ from __ where __ between __ and __`
PROP. MASK:	`SELECT col1 , col2 FROM table1 WHERE col3 BETWEEN num AND num`

EXAMPLE 7	
SQL:	`SELECT max(Capacity) , avg(Capacity) FROM stadium`
DAIL MASK:	`select max ( __ ) , avg ( __ ) from __`
PROP. MASK:	`SELECT max(col1) , avg(col1) FROM table1`

EXAMPLE 8	
SQL:	`SELECT Name, Capacity FROM stadium ORDER BY Average DESC LIMIT 1`
DAIL MASK:	`select __ from __ order by __ desc limit __`
PROP. MASK:	`SELECT col1 , col2 FROM table1 ORDER BY col3 DESC LIMIT num`

EXAMPLE 9	
SQL:	`SELECT T1.Name, T1.Capacity FROM stadium AS T1 JOIN concert AS T2 ON T1.Stadium_ID = T2.Stadium_ID WHERE T2.Year >= '2014' GROUP BY T1.Stadium_ID ORDER BY count(*) DESC LIMIT 1`
DAIL MASK:	`select _ from _ where _ group by _ order by _ ( _ ) desc limit _`
PROP. MASK:	`SELECT alias1.col1 , alias1.col2 FROM table1 AS alias1 JOIN table2 AS alias2 ON alias1.col3 = alias2.col3 WHERE alias2.col4 >= str GROUP BY alias1.col3 ORDER BY count(*) DESC LIMIT 1`

C.2 *DAILSim* vs *SQLSim* Metric Examples

EXAMPLE 1	
SQL_1	`SELECT count(*) FROM singer`
DAILMask(SQL_1)	`select count ( _ ) from _`
SQLMask(SQL_1)	`SELECT count(*) from table1`
SQL_2	`SELECT count(*) FROM Templates`
DAILMask(SQL_2)	`select count ( _ ) from _`
SQLMask(SQL_2)	`SELECT count(*) FROM table1`
DAILSim(*DAILMask*(SQL_1), *DAILMask*(SQL_2)) = 1.0	
SQLSim(*SQLMask*(SQL_1), *SQLMask*(SQL_2)) = 1.0	

EXAMPLE 2	
SQL_1	`SELECT country , count(*) FROM singer GROUP BY country`
DAILMask(SQL_1)	`select _ , count ( _ ) from _ group by _`
SQLMask(SQL_1)	`SELECT col1 , count(*) FROM table1 GROUP BY col1`
SQL_2	`SELECT count(*) , city FROM employee GROUP BY city`
DAILMask(SQL_2)	`select count ( _ ) , _ from _ group by _`
SQLMask(SQL_2)	`SELECT count(*) , col1 FROM table1 GROUP BY col1`

DAILSim(*DAILMask*(SQL_1), *DAILMask*(SQL_2)) = 1.0
SQLSim(*SQLMask*(SQL_1), *SQLMask*(SQL_2)) = 0.931

EXAMPLE 3	
SQL_1	`SELECT count(*) FROM singer`
DAILMask(SQL_1)	`select ( _ ) from _`
SQLMask(SQL_1)	`SELECT count(*) FROM table1`
SQL_2	`SELECT grade FROM Highschooler GROUP BY grade HAVING count(*) >= 4`
DAILMask(SQL_2)	`select _ from _ group by _ having _`
SQLMask(SQL_2)	`SELECT col1 from table1 GROUP BY col1 HAVING count(*) >= num`
DAILSim(*DAILMask*(SQL_1), *DAILMask*(SQL_2)) = $0.66\dot{6}$	
SQLSim(*SQLMask*(SQL_1), *SQLMask*(SQL_2)) = 0.394	

EXAMPLE 4	
SQL_1	`SELECT name , country , age FROM singer ORDER BY age DESC`
DAILMask(SQL_1)	`select _ from _ order by _ desc`
SQLMask(SQL_1)	`SELECT col1 , col2 , col3 FROM table1 ORDER BY col3 DESC`
SQL_2	`SELECT template_id , version_number , template_type_code FROM Templates`
DAILMask(SQL_2)	`select _ from _`
SQLMask(SQL_2)	`SELECT col1 , col2 , col3 FROM table1`
DAILSim(*DAILMask*(SQL_1), *DAILMask*(SQL_2)) = 0.5	
SQLSim(*SQLMask*(SQL_1), *SQLMask*(SQL_2)) = $0.708\dot{3}$	

EXAMPLE 5	
SQL_1	`SELECT count(*) FROM singer`
DAILMask(SQL_1)	`select count ( _ ) from _`
SQLMask(SQL_1)	`SELECT count(*) FROM table1`

SQL_2	`SELECT count(*) FROM concert WHERE YEAR = 2014 OR YEAR = 2015`
DAILMask(SQL_2)	`select __ from __ where __ = __`
SQLMask(SQL_2)	`SELECT count(*) FROM table1 WHERE col2 = num`
DAILSim(*DAILMask*(SQL_1), *DAILMask*(SQL_2)) = $0.6\dot{3}$	
SQLSim(*SQLMask*(SQL_1), *SQLMask*(SQL_2)) = 0.627	

EXAMPLE 6	
SQL_1	`SELECT count(*) FROM singer`
DAILMask(SQL_1)	`select count ( __ ) from __`
SQLMask(SQL_1)	`SELECT count(*) FROM table1`
SQL_2	`SELECT count(*) FROM student AS T1 JOIN has_pet AS T2 ON T1.stuid = T2.stuid JOIN pets AS T3 ON T2.petid = T3.petid WHERE T1.sex = 'F' AND T3.pettype = 'dog'`
DAILMask(SQL_2)	`select count ( __ ) from __ where __ group by __ desc limit __`
SQLMask(SQL_2)	`SELECT count(*) FROM table1 AS alias1 JOIN table2 AS alias2 ON alias1.col1 = alias2.col1 JOIN table3 AS alias3 ON alias2.col2 = alias3.col2 WHERE alias1.col3 = str AND alias3.col4 = str`
DAILSim(*DAILMask*(SQL_1), *DAILMask*(SQL_2)) = $0.4\dot{6}$	
SQLSim(*SQLMask*(SQL_1), *SQLMask*(SQL_2)) = $0.24\dot{6}$	

EXAMPLE 7	
SQL_1	`SELECT avg(age) , min(age) , max(age) FROM singer WHERE country = 'France'`
DAILMask(SQL_1)	`select avg ( __ ) , min ( __ ) , max ( __ ) , from __ where __ = __`
SQLMask(SQL_1)	`SELECT avg(col1) , min(col1) , max(col1) , FROM table1 WHERE col2 = str`

SQL_2	`SELECT document_id , template_id , Document_Description FROM Documents WHERE document_name = "Robbin CV"`
DAILMask(SQL_2)	`select __ from __ where __ = __`
SQLMask(SQL_2)	`SELECT col1 , col2 , col3 FROM table1 WHERE col4 = str`
DAILSim(*DAILMask*(SQL_1), *DAILMask*(SQL_2)) = 0.381	
SQLSim(*SQLMask*(SQL_1), *SQLMask*(SQL_2)) = 0.642	

EXAMPLE 8	
SQL_1	`SELECT DISTINCT T1.Fname FROM student AS T1 JOIN has_pet AS T2 ON T1.stuid = T2.stuid JOIN pets AS T3 ON T3.petid = T2.petid WHERE T3.pettype = 'cat' OR T3.pettype = 'dog'`
DAILMask(SQL_1)	`select distinct __ from __ where __ or __`
SQLMask(SQL_1)	`SELECT DISTINCT alias1.col1 FROM table1 AS alias1 JOIN table2 AS alias2 on alias1.col2 = alias2.col2 JOIN table3 AS alias3 ON alias3.col3 = alias2.col3 WHERE alias3.col4 = str OR alias3.col4 = str`
SQL_2	`SELECT petid , weight FROM pets WHERE pet_age > 1`
DAILMask(SQL_2)	`select __ from __ where __ > __`
SQLMask(SQL_2)	`SELECT col1 , col2 FROM table1 WHERE col3 > num`
DAILSim(*DAILMask*(SQL_1), *DAILMask*(SQL_2)) = 0.7	
SQLSim(*SQLMask*(SQL_1), *SQLMask*(SQL_2)) = 0.182	

Appendix D

Fine-Tuning Question Embeddings (Extended)

D.1 Experiment Construction

In this appendix, an artifact of the project research is documented in full. This has been evaluated as a preliminary investigation into embedding fine-tuning supplimentary to the main report, which reinforces limitations found when embedding the question space.

In this section, the primary objective is to develop a framework that can replicate the aggregated results from the RQ1 embedding model survey. When discussing aggregate performance in Section 3.2, we saw 86.7% of Spider questions can be answered correctly when provided a single training example. It would be desirable to train an embedding model that always chooses a good 1-shot example where possible; any single procedure capable of achieving an execution accuracy score of 86%+ whilst leveraging `gpt-3.5-turbo` would be seen as a triumphant success.

A backwards way of approaching this problem would be to use the results from the Chapter 3 survey to form a training set for a embedding model. Informally, the philosophy behind the proposed method is *"If you see this Spider question, then choose an example we **already know** will provide a right answer"*. This opposes the general paradigm where fine-tuning examples are sourced from the train-set distributed with the benchmark. Also, to build a dataset in such a fashion assumes some implicit knowledge of the gold query, which is ill-advised. Hence the framework is not a rigorous science and instead serves as a first foray into fine-tuned example selection methods.

To achieve this 'answer sheet' approach, the best way forward is to embed the plain-text question space. If the aim is to directly pick out known good examples from the training set, context-masking the candidates complicates this process, as many examples share a common masked question. When two examples share a masked question, they also share a common embedding vector. If one of these is a good example to suggest for a Spider task while the other is not, the selection mechanism cannot distinguish between the two. Therefore, for this specific application, embedding the standard question space is more appropriate than using a context-masked approach.

In alignment with the fine-tuning pipeline set out in Section 4.1, training examples must be *(sentenceA, sentenceB, score)* tuples. The aim is to push certain examples closer to the target question in the embedding space, so the sentence pairs must be of the form *(target question, example question)*. Examples are sourced from the 1-shot experiments conducted in Chapter 3, as they are directly constructive to the conversion process (or at least not significantly inhibitive). It is more difficult to assess the influence of examples included in 3-shot and 5-shot prompts directly, as some examples may aid the language model while others might be superfluous or even detrimental to the conversion.

To encourage the desired behaviour in an embedding model, a primitive approach to fine-tuning is outlined as follows.

- For every successful 1-shot conversion in the RO1 survey, define a *positive* training example of the form *(target question, example question, 1)*.
- For every unsuccessful 1-shot conversion in the RO1 survey, define a *negative* training example of the form *(target question, example question, 0)*.

When training an embedding using the CoSENTLoss function, a similarity score of 1 within and example implies a notion of perfect similarity whilst a score of 0 implies a notion of no similarity. The outcomes from the 8 survey experiments yields 7073 positive examples and 2223 negative examples for model training. The `all-mpnet-base-v2` model is subject to five epochs of fine-tuning on this dataset, achieving a final loss of 0.4105. This is accessible via the `sentence-transformers` library by calling `"model = SentenceTransformers(s2593817/sft-question-embedding)"`. Using this optimized model to anchor Spider example selection leads to an improvement in performance when compared to the base model, especially for $k = 1$ as designed.

k	all-mpnet-base-v2	all-mpnet-base-v2 **(SFT)**
1	0.764	0.804
3	0.767	0.789
5	0.776	0.789

Table D.1: Comparison of Model Performance

The fine tuned embedding model achieves 80.4% success rate for $k = 1$ which is significantly higher than the base model success rate, as well as all 1-shot outcomes from the survey. The fall in performance for the higher shot experiments can be expected, as the training data is assembled specifically to target success in the $k = 1$ case. However an improvement for both $k = 3$ and $k = 5$ is still observed over the base model.

The new embedding clearly promotes some of the 'good' examples marked out in the train set, but does not select all of them. This results in a large 6.3% disparity when compared to the aggregated results. There remains over sixty questions that are known to have a successful 1-shot prompt which fail when generating examples via the fine-tuned embedding. There are several reasons for this happening, which are unlikely to be resolved by any further fine tuning. The main problem being is that assigning a positive *(target question, example question, 1)* tuple to the train set in no way guarantees that the target and example questions map to the same vector. The fine-tuning procedure simply looks to identify a model that is a minimizes the CoSENT loss function; there is no assurance that a 'good' example becomes a nearest neighbor to its target in the fine-tuned embedding space. Many bad examples that are linguistically very similar to the target question remain the nearest neighbor after fine-tuning, a single training example encouraging a different choice isn't enough. Also, the train set often promotes multiple different examples for the same question. The recommendations aren't made in isolation, each changes the fabric of the underlying BERT model in an overlapping and potentially conflicting manner. These problems would become even harder to control if looking to optimize 3-shot prompting, 5-shot prompting and beyond, where one must consider the holistic effect of the examples on top of the individual influences.

In summary, this approach is ultimately impractical and overfit to the Spider benchmark. It serves as an informative first exploration in how fine-tuning embeddings can lead to marginal gains in performance, but it has also demonstrated that simply showing an embedding examples to target does not guarantee their selection for prompting.