TRANSQL: A Transformer-based Model for Classifying SQL Queries

Shirin Tahmasebi[†], Amir H. Payberah[†], Ahmet Soylu[¶], Dumitru Roman[§], Mihhail Matskin [†]
[†]KTH Royal Institute of Technology, Sweden [¶]Oslo Metropolitan University, Norway [§]SINTEF AS, Norway
[†]{shirint, payberah, misha}@kth.se [¶]ahmet.soylu@oslomet.no [§]dumitru.roman@sintef.no

***Abstract*—Domain-Specific Languages (DSL) are becoming popular in various fields as they enable domain experts to focus on domain-specific concepts rather than software-specific ones. Many domain experts usually reuse their previously-written scripts for writing new ones; however, to make this process straightforward, there is a need for techniques that can enable domain experts to find existing relevant scripts easily. One fundamental component of such a technique is a model for identifying similar DSL scripts. Nevertheless, the inherent nature of DSLs and lack of data makes building such a model challenging. Hence, in this work, we propose TRANSQL, a transformer-based model for classifying DSL scripts based on their similarities, considering their few-shot context. We build TRANSQL using BERT and GPT-3, two performant language models. Our experiments focus on SQL as one of the most commonly-used DSLs. The experiment results reveal that the BERT-based TRANSQL cannot perform well for DSLs since they need extensive data for the fine-tuning phase. However, the GPT-based TRANSQL gives markedly better and more promising results.**

***Index Terms*—SQL Classification, BERT, GPT**

I. INTRODUCTION

There are two broad categories of computer languages: General-Purpose Languages (GPL) and Domain-Specific Languages (DSL). GPLs are applicable across various domains; for example, XML and Python are samples of markup and programming GPLs, respectively. In contrast to GPLs, DSLs are specially designed for specific problem domains. Despite the low applicability outside their domain, DSLs enable domain-expert users (who have little knowledge outside their domain) to express their solutions in a more abstract and easier-to-learn language [1].

Usually, domain experts who use DSLs write DSL scripts similar to their previously-written ones [2]. Thus, it is immensely beneficial to have a tool to help expert users to search among their existing DSL scripts and reuse them for writing new ones rather than writing them from scratch. This increases the reuse opportunity and makes writing new DSLs much easier and faster. Accordingly, the first step in designing such tools is to propose a model for identifying similar DSL scripts, which is the main focus of this paper.

One way to identify similar scripts is to compare the *embeddings* of DSL scripts. To this end, we take advantage of transformer-based approaches, which have shown promising results [3], [4], [5]. One prevalent transformer-based technique for making embeddings is BERT [6], a pre-trained model which can be fine-tuned using downstream tasks. Although BERT-based models show remarkable performance in different domains, they all rely on a critical point: having enough data for fine-tuning the model, which is not always the case. For example, while working with DSLs, the lack of data is always a challenge. One approach to address the lack of data problem is to use large language models such as GPT-3 [7], [8], [9]. The main idea behind these models is that if a huge model is trained on massive and quality datasets, it is possible to mitigate the need to fine-tune the model for domain-specific tasks.

In this paper, we propose TRANSQL, a model for classifying DSL scripts (particularly SQL queries) when there is a lack of data (i.e., in a few-shot context). We focus on SQL queries as an example of DSLs for two reasons [10]: (1) SQL is one of the most popular DSLs, and (2) Unlike most of the other DSLs, there exist available public datasets for SQL. To the best of our knowledge, this is the first work considering the few-shot context of SQLs for classifying them. To this end, we first embed the SQL scripts and then classify them accordingly. We leverage two language models for embedding DSLs: BERT and GPT-3 models. In the BERT-based TRANSQL, we use two different downstream tasks for detecting similar SQLs. We consider these approaches as baselines for our work.

Our experiments show that by using GPT-3 to embed SQL queries and having a proper policy for selecting GPT-3 *in-context examples*, it is possible to achieve noticeable accuracy in the classification task without doing any fine-tuning. In addition, the GPT-based approach significantly outperforms the BERT-based baselines.

Contributions. The main contributions of our work include:

- Introducing TRANSQL, a transformer-based model for classifying SQL queries;
- Using BERT and GPT-3 for embeddings and classifying SQL queries;
- Tackling the lack of data in building the model by leveraging the few-shot learning capability of GPT-3; and
- Evaluating the influence of employing different strategies for in-context examples selection on the few-shot learning capability of the GPT-based TRANSQL.

II. PROBLEM DEFINITION

This work aims to build a classification model for identifying similar SQL queries. An immediate question here is: What is meant by *similarity* between SQL queries? In other words, based on which metrics can two SQL queries be considered similar? In this regard, there exist several approaches for measuring similarities between SQL queries, such as *Fragment-based*, *Tuple-based*, and *Access-area-based*. Details of these approaches, along with their drawbacks, are described in Section III-B. Due to the critical drawbacks of these approaches, in TRANSQL, we present a new approach by taking the full-text of SQL queries into account. This way, the queries' fragments and structures are involved in measuring the similarity. To this end, we first use an embedding model to obtain numerical representations of the full-text SQL queries; then, we leverage the cosine-similarity score of their numerical representations to measure the similarity between the queries. Several well-known and commonly-used embedding models exist for making numerical representations from the text. We briefly review them in Section III-A.

Our approach to identify similar SQL queries has two steps:

1) **Making query embedding:** Here, we define a performant embedding model to calculate the numerical representation of SQL queries. Let us assume there are three queries, q_1, q_2, and q_3, where q_1 and q_2 are similar in terms of their structure and functionality, but q_2 and q_3 are different. Thus, we expect the embedding model to generate embeddings for q_1 and q_2 closer to each other compared to the generated embeddings for q_2 and q_3. Moreover, we use their cosine-similarity score to measure how close the embeddings are to each other.
2) **Making a quality classifier:** In this step, we aim to make a classifier that receives query embeddings as input and predicts the class label. However, due to the inherent nature of DSLs, their usage is very domain-specific, and there may not be enough available data for training the models. Therefore, the classifier must not rely on massive data for learning. In other words, the classifier needs to perform well in few-shot contexts.

III. Background

In this section, we first give a brief overview of embedding models. Then, we present different approaches for measuring query similarity underlining the drawbacks of each one.

A. Embedding Models

Words and sentences are input to the Natural Language Processing (NLP) models. We should embed the words to numerical fixed-size vectors, known as *embeddings* to give them to NLP models. Similarly, for sentences, which are sequences of words, a sequence of word embeddings can be used as their representation [11]. There are two broad categories of embedding models [11], [12]: context-free embeddings and contextual embeddings. Context-free embeddings, such as GloVe [13] and Word2Vec [14], map words to fix embeddings regardless of the context the words are used. In contrast to context-free embeddings, the main idea behind contextual embeddings is that the meaning of words depends on their surrounding words and context. Hence, contextual models, such as ELMo [12] and BERT [6], consider the context for calculating the word embeddings.

Most recent contextual embedding models are based on *transformer* architecture, which is an *attention*-based encoder-decoder architecture [15]. These novel contextual models are divided into two main groups by their structure: *encoder-based* models and *decoder-based* models.

Encoder-based Models: The base architecture of the encoder-based models consists of a *multi-layer encoder-only transformer*. One of the ground-breaking models in this group is BERT [6]. The basic idea followed by BERT is to pre-train a general transformer-based model on significantly large datasets and then fine-tune the pre-trained model using downstream tasks on relatively smaller datasets. The pre-trained BERT can be used in various downstream tasks, such as question-answering [16] and sentence classification [17]. It is worth mentioning that the critical assumption in the downstream phase is that there exists enough data for fine-tuning the model.

Decoder-based Models: The base architecture of the models in this group is a *multi-layer decoder-only transformer*. GPT as a decoder-based model is a game-changer that revolutionized the area of few-shot learning models [7], [8], [9]. GPT has different variants, among which GPT-3 is the largest, most capable, and performant. The main idea behind GPT-3 is that if a model is large enough, by training it on comprehensive and quality datasets, it can perform well in other tasks even without further fine-tuning [9]. Moreover, GPT-3 has shown good performance on one-shot and few-shot learning, thanks to its *in-context learning* capability.

In-context learning is the ability of a language model to do a task by inferring from a set of examples as its input. In order to illustrate this, let us say that model $\mathcal{M}$ is a model with in-context learning capability [18], [19]. Then, given "`thanks` $\rightarrow$ `tack, hello` $\rightarrow$ `hej, mint` $\rightarrow$ `mynta, otter` $\rightarrow$" as input and without any down-stream task fine-tuning, we expect $\mathcal{M}$ to learn that the task is English-Swedish translation and generate "`utter`" as output.

Formally, if context $\mathcal{C}$ consists of k examples, then it can be formulated as $\mathcal{C} = \{x_1, y_1, x_2, y_2, \cdots, x_k, y_k\}$, where, given input x, model $\mathcal{M}$ produces output y with the following probability [19]:

$$P_{\mathcal{M}}(y|\mathcal{C}, x) = \prod_{t=1}^{T} p(y_t|\mathcal{C}, x, y_{<t}) \qquad (1)$$

B. Query Similarity

One valid question for measuring the query similarity is the metrics for comparing them. There exist several approaches for measuring query similarity, among which the most frequent ones are as follows [20], [21], [22]:

1. Fragment-based, in which *query fragments*, such as attribute names, table names, join types, and where predicate, are used to distinguish queries. The main drawback of this approach is that it does not consider in which clause of the query the fragments are used.

2. Tuple-based (or, *witness-based*), in which each query is characterized by the rows (tuples) of the database accessed by it. This approach has several critical drawbacks. First, for databases with a high changing rate, it is necessary to re-execute the query to extract the returned tuples, which is computationally expensive and inefficient. Second, semantically different queries may return the same tuples at some time. Third, semantically similar queries may return different tuples because of having different filter conditions.

3. Access-area-based, in which each query is represented by the range of the attributes to which it has access. Therefore, the similarity between each pair of queries is measured based on the overlapping between their access ranges. The main drawback of this approach is that it is likely to have two semantically-similar queries with two completely different filter conditions.

IV. Solution

We propose our model, TranSQL, in which we use two different language models, i.e., BERT and GPT-3, for embedding and classifying SQL queries. In this section we explain the BERT-based TranSQL, and the GPT-based TranSQL.

A. *BERT-based* TranSQL

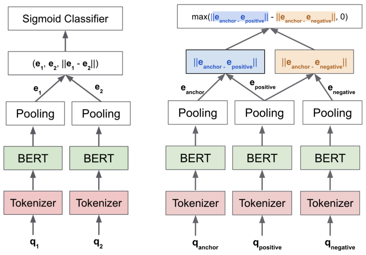
Different variants of BERT have shown satisfactory results in classification and clustering tasks. For example, CoCluBERT [5] is a BERT-based model for clustering Python source code based on their functionality. It uses CuBERT [23] for embedding source code and Siamese and Triplet networks for fine-tuning. Inspired by CoCluBERT, we propose the BERT-based TranSQL that consists of two steps: (i) an embedding step and (ii) a classification step.

The embedding step: Here, we leverage BERT for making query embeddings. To do so, we split the query q into multiple tokens, add the `[CLS]` token at the beginning of the tokens and give them as input to the BERT model. For example, we tokenize the query `select * from T` as `[CLS][select][*][from][T]`.

Two ways of using BERT for sentence embedding are to extract the embedding of `[CLS]` token or to use a pooling layer (e.g., mean or max pooling) to aggregate the embedding of all the tokens. Sentence-BERT (SBERT) [3] is a BERT variant that uses pooling approaches for detecting similarities between input sentences. We leverage SBERT in our model to embed the queries. SBERT fine-tunes a pre-trained BERT model on two downstream tasks:

- Pair-wise Sentence Similarity: In this network, depicted in Figure 1(a), the model gets two sentences as input and predicts if they are from the same class.
- Triple-wise Sentence Similarity: In this downstream task, BERT is fine-tuned using the Triplet network, as illustrated in Figure 1(b). Here, the model is fed with three input sentences: (1) an anchor sentence, (2) a positive one chosen from the same class as the anchor, and (3) a negative one chosen from any class other than the anchor's class. The objective is to maximize the similarity between the anchor and positive embeddings and minimize the similarity between the anchor and negative embeddings.

In the BERT-based TranSQL, we take advantage of two pre-trained variants of BERT in SBERT: CuBERT [23], a pre-trained model on source code, and CodeBERT [24], a bimodal pre-trained model for programming and natural languages. We fine-tune both variants using the tasks mentioned above.



(a) Siamese (b) Triplet

Fig. 1. Two network architectures used for fine-tuning BERT model for sentence similarity; Siamese and Triplet, depicted in (a) and (b), respectively.

TABLE I
GPT-based Embeddings Summary

	Ada	Babbage	Curie	Davinci
Text Similarity	text-similarity-ada	text-similarity-babbage	text-similarity-curie	text-similarity-davinci
Text Search	text-search-ada	text-search-babbage	text-search-curie	text-search-davinci
Code Search	code-search-ada	code-search-babbage		

The classification step: After fine-tuning SBERT (based on CuBERT or CodeBERT), we leverage it for the classification phase. To this end, we give a query as input to the fine-tuned CuBERT or CodeBERT and then use the embedding of `[CLS]` token for classifying the query. We make a simple feed-forward network with a softmax function as a classifier.

B. *GPT-based* TranSQL

Since we may not have enough data to fine-tune the BERT model, we use GPT-3, which has the few-shot learning capability, as the second approach in the TranSQL. Unlike the BERT-based TranSQL, which has two separate steps for embedding and classifying the queries, the GPT-based TranSQL unifies these two steps: the model gets a query q, together with a set of in-context examples as input, and returns the label of q. However, to predict q's label, the model should make the embedding of q and all the in-context examples.

OpenAI offers three families of embedding models for GPT, each designed for specific functionality, including text-search, text-similarity, and code-search. The difference between these three families comes from the dataset and the tasks used for pre-training them. Moreover, each family consists of, at most, four separate embeddings of different sizes and capabilities: Ada (1024 dimensions), Babbage (2048 dimensions), Curie (4096 dimensions), and Davinci (12288 dimensions).

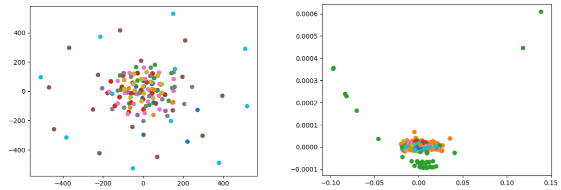
Among these models, Davinci is the largest and the most capable one; however, it is the most expensive and the slowest model. In contrast to Davinci, Ada is the smallest and the least capable one, but it is the least expensive and fastest. It is worth mentioning that not all three task families (i.e., text-search, text-similarity, and code-search) consist of all four embeddings. To make it clear, we demonstrate all the available embedding models in Table I. In the GPT-based TranSQL, we take advantage of three embeddings, which are all highlighted in Table I.

The first step in using the GPT-based TRANSQL is to choose the in-context examples for each input query q for passing to the model. As mentioned in Section III-A, in-context examples refer to examples given as input to language models (e.g., GPT-3) to help the model understand the task it is supposed to do without any task-specific fine-tuning steps. To let the model do the inference for all labels equally, it is important to balance the label distribution of in-context examples, which means that we need to take the same number of examples from each class. Let us represent the number of examples selected from each class as k, which is a hyper-parameter. Then, to balance the label distribution, k cannot be greater than the number of instances in the smallest class. Therefore, if $n_{smallest}$ represents the number of instances in the smallest class, then $k \leq n_{smallest}$.

After deciding about k, the number of in-context examples for each label, a critical question is how to select these examples. In other words, do we need a specific metric for choosing the in-context examples, or can we choose them randomly? To answer these questions, we hypothesize that the performance of GPT-3 is sensitive to the quality of the given in-context examples. This means that a reasonable strategy to choose the in-context examples can improve the performance compared to a case when we select them randomly. Therefore, to evaluate our hypothesis, we consider the following two ways for choosing the in-context examples in the GPT-based TRANSQL, named *top-k similar* and *centroid-based* as explained below:

- **Top-*k* Similar:** In this approach, for each query q, we iterate through all the labels and calculate the cosine-similarity of q's embedding and the embedding of all instances for each label. Then, we select the k queries with the highest cosine-similarity score for each label. Therefore, at the end of the algorithm, the in-context examples contain the top k most similar queries from each label.
- **Centroid-based:** Here, first, for each label, the average of its members embeddings is calculated and considered as its centroid. Then, given an input query q, we first calculate the similarity score of q's embedding and the embedding of all centroids. Then, the label corresponding to the centroid with the most similarity score to q is selected as $most\_similar\_label$. Now, from the instances with the label of $most\_similar\_label$, we select the top k most similar ones to q. We also select the top k least similar ones to q from all the other labels different other $most\_similar\_label$.

After choosing the in-context examples, we give them to the model for classification. For the classification, we take advantage of another API offered by OpenAI [25]. Since this classification API receives its input (a query) and the in-context examples set in the textual format, it uses any of the embedding methods mentioned earlier (i.e., Ada, Babbage, Curie, and Davinci) to obtain their numerical representations. To distinguish the embedding method used by the classification API and the embedding method used for selecting in-context examples (in the top-k similar and centroid-based approaches), we call the former one *classifier embedding* and the latter one *selection embedding*. Our solution uses the same embedding model in both the classifier and selection embeddings. For example, if the selection embedding is Ada, we also use Ada for the classifier embedding. Table II summarizes the classifier and selection embedding methods we use in the GPT-based TRANSQL.



(a) CuBERT Trained on Siamese and Triplet Networks (b) CodeBERT Trained on Siamese and Triplet Networks

Fig. 2. The dots represent the 2D query embedding vectors–using CuBERT and CodeBERT. The color of the dots represents the class to which they belong.

TABLE II
GPT-BASED STRATEGIES

No.	Classifier Embedding	In-context Set	Selection Embedding
1	Ada	Random	-
2	Ada	Top-k Similar	*code-search-ada*
3	Ada	Centroid-based	*code-search-ada*
4	Babbage	Random	-
5	Babbage	Top-k Similar	*code-search-babbage*
6	Babbage	Centroid-based	*code-search-babbage*
7	Davinci	Random	-
8	Davinci	Top-k Similar	*text-similarity-davinci*
9	Davinci	Centroid-based	*text-similarity-davinci*

V. EXPERIMENTS

In this section, we first briefly explain our experiment configurations and the dataset. Then, we analyze the results and describe the main takeaways from our experiments.

A. Configurations and Dataset

We implemented TRANSQL in Python, which is publicly available on Github[1]. As a dataset, we use Bombay [26] that consists of student answers to the mid-term exams of a database course for two years. The exams consist of 14 separate query-writing tasks (14 classes). From all the answers, the distinct, logically-, and syntactically-correct ones are filtered and added to the dataset. In this dataset, the number of instances for the smallest class is four, meaning there are very few samples for some classes; thus, it is a proper dataset for our experiments on few-shot contexts.

[1] https://github.com/ShirinTahmasebi/Query-Embedding

B. Results and Analysis

Here, we present and analyze the results of the BERT-based TRANSQL and the GPT-based TRANSQL.

BERT-based TRANSQL: The embedding results of the BERT-based solution is depicted in Figure 2. Each dot in the plots represents the 2D-reduced query embeddings, and its color shows the label of the class to which it belongs. As we see, due to a lack of data, the BERT-based embeddings mentioned in Section IV-A (CuBERT and CodeBERT) are unable to learn query embeddings properly and identify different classes. Since the fine-tuned BERT cannot distinguish different classes, it is not feasible to use the BERT-based solution for classifying the queries.

GPT-based TRANSQL: As mentioned in Section IV-B, we leverage three embeddings in the GPT-based TRANSQL, highlighted in Table I. To examine how well these embeddings perform on our dataset, first, we embed the SQL queries using these three embedding methods. The results, illustrated in Figure 3, demonstrate that all three GPT-based embedding models reach an acceptable embedding, meaning that classes are almost distinguishable.

Our GPT-based strategies for the classification task are described in Section IV-B, and also summarized in Table II. In Figure 4, we compare these strategies in terms of precision, recall, and F1 score. As Figure 4 shows, the top-*k* similar and centroid-based approaches for selecting in-context examples outperform the random selection approach. Specifically, in terms of precision, as we see in Figure 4(a), leveraging top-*k* similar approach, instead of randomly selection, leads to an improvement[2] of about 8.45%, 13.8%, and 2.1% for Ada (comparing strategies #1 and #2), Babbage (comparing strategies #4 and #5), and Davinci (comparing strategies #7 and #8) models, respectively.

Similarly, the precision improvement percentage for the centroid-based approach is about 22.5%, 33.8%, and 3.2% for Ada (comparing strategies #1 and #3), Babbage (comparing strategies #4 and #6), and Davinci (comparing strategies #7 and #9) models. This provides evidence that, while working with GPT-3, the way of in-context example selection matters, and leveraging different approaches for in-context example selection significantly impacts the classification task. Moreover, the improvement percentage for Ada is more than that for Babbage, which is also more than that for Davinci. For example, by using the centroid-based approach instead of random selection, the improvement percentages in Ada and Davinci are 22.5% and 3.2%, respectively. This brings us to a conclusion that, by using a less expensive and much faster model such as Ada (compared to Davinci), it is still possible to reach promising results, provided that a proper in-context selection approach is chosen.

Key Takeaways: Based on the results depicted in Figure 2, Figure 3, and Figure 4, the following takeaways can be extracted from all of the conducted experiments:

- GPT-based embedding models outperform BERT-based embedding models in a few-shot context.
- The way of choosing in-context examples is effective in the final performance of the GPT-3 classifier.
- In our experiments, the smallest class of our dataset has only four instances. This number is the limiting factor for deciding how many instances from each class we can put in the in-context examples set (as mentioned in Section IV-B, $k \leq n_{smallest}$). However, this also reveals an interesting point. Since the performance of TRANSQL is acceptable by having only four instances from each class, we can conclude that this approach applies to most few-shot contexts.
- If a proper approach is leveraged for in-context selection, it is possible to take advantage of faster and cheaper embedding models.

VI. RELATED WORK

This section briefly overviews some of the recent applications of BERT-based and GPT-based language models.

A. BERT-based Clustering and Classification Tasks

In [27], the authors show that BERT-based models can be good few-shot learners by leveraging a proper prompt. Hence, they have used DistilBERT [28] and RoBERTa [29] with a properly designed prompt template, which converts the text classification task to a Question-Answer (QA) task. In [5], the authors proposed CoCluBERT for clustering Python source code based on their functionality. There are several works that focus on classification and clustering text and source codes; however, to the best of our knowledge, no previous research has investigated the classification or clustering of DSLs (SQL in particular), taking their lack of data into account.

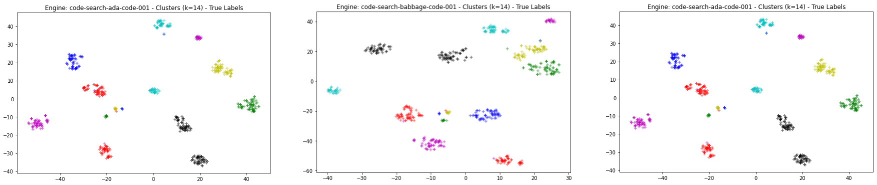
B. GPT-based Few-shot Learning Tasks

In [30], the authors hypothesize that GPT-3 performance for classification tasks highly depends on its given in-context examples. Hence, they propose an approach for data augmentation, with the help of which they can create high-quality in-context examples. Another interesting application of GPT-3 is proposed in [31], in which the authors propose a system for generating email responses. Although a few works are leveraging the few-shot learning capability of GPT-3 for different applications and domains, no study has investigated using GPT-3 for the classification of DSLs and SQL queries.

VII. CONCLUSION AND FUTURE WORK

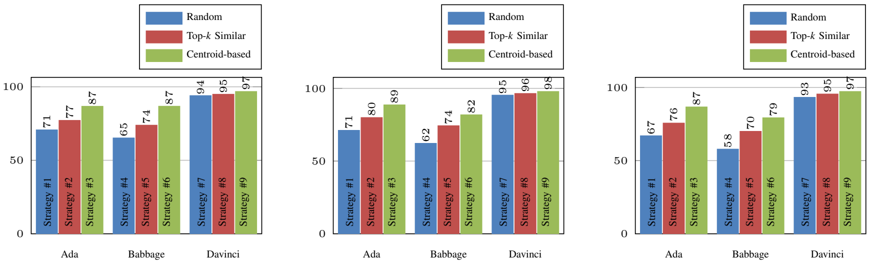
This work explored several solutions for classifying Domain-Specific Language (DSL), particularly SQL queries. To this end, we presented TRANSQL and used two language models, i.e., BERT and GPT-3, to embed SQL queries and classify them. Our findings confirmed that BERT-based TRANSQL requires large data for their fine-tuning step, so they cannot perform well in few-shot contexts. However, since

[2] Improvement percentage from value a to b is calculated using: $\frac{b-a}{a} \times 100$



(a) *code-search-ada-code-001* (b) *code-search-babbage-code-001* (c) *text-similarity-davincci-code-001*

Fig. 3. The dots represent the 2D query embedding vectors. The color of the dots represents the class to which they belong.



(a) Precision (b) Recall (c) F1-score

Fig. 4. Precision, recall, and F1-score of all GPT-based strategies are depicted in (a), (b), and (c), respectively.

GPT-3 is a quality few-shot learner, the GPT-based TRANSQL outperforms the BERT-based TRANSQL. Moreover, our experiments revealed that, in the GPT-based TRANSQL, the strategy for in-context example selection plays a crucial role in the performance of the classification task, bringing about up to 33% improvement in the task performance. We believe that this work is a stepping stone for studying the few-shot context of DSLs. In addition, going beyond SQL queries and changing the focus from SQL to other DSLs is another direction for future work.

ACKNOWLEDGMENT

This work received partial funding from the EC through the projects DataCloud (101016835) and enRichMyData (101070284).

REFERENCES

[1] M. Fowler, *Domain-specific languages*. Pearson Education, 2010.

[2] T. Degueule et al., "Melange: A meta-language for modular and reusable development of dsls," in *ACM SIGPLAN SLE*, 2015, pp. 25–36.

[3] N. Reimers et al., "Sentence-bert: Sentence embeddings using siamese bert-networks," *arXiv preprint arXiv:1908.10084*, 2019.

[4] N. Peinelt et al., "tbert: Topic models and bert joining forces for semantic similarity detection," in *Annual Meeting of the ACL*, 2020, pp. 7047–7055.

[5] M. Hägglund et al., "Coclubert: Clustering machine learning source code," in *ICMLA*. IEEE, 2021, pp. 151–158.

[6] J. Devlin et al., "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[7] A. Radford et al., "Improving language understanding by generative pre-training," *OpenAI blog*, 2018.

[8] ——, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[9] T. Brown et al., "Language models are few-shot learners," *NeurIPS*, vol. 33, pp. 1877–1901, 2020.

[10] A. Alexandrov et al., "Representations and optimizations for embedded parallel dataflow languages," *ACM Transactions on Database Systems (TODS)*, vol. 44, no. 1, pp. 1–44, 2019.

[11] N. Smith, "Contextual word representations: A contextual introduction," *arXiv preprint arXiv:1902.06006*, 2019.

[12] M. Peters et al., "Deep contextualized word representations," in *Conference of the North American*. ACL, Jun. 2018, pp. 2227–2237.

[13] J. Pennington et al., "Glove: Global vectors for word representation," in *EMNLP*, 2014, pp. 1532–1543.

[14] T. Mikolov et al., "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[15] A. Vaswani et al., "Attention is all you need," *NeurIPS*, vol. 30, 2017.

[16] W. Yang et al., "End-to-end open-domain question answering with bertserini," *arXiv preprint arXiv:1902.01718*, 2019.

[17] A. Cohan et al., "Pretrained language models for sequential sentence classification," *arXiv preprint arXiv:1909.04054*, 2019.

[18] S. Xie et al., "An explanation of in-context learning as implicit bayesian inference," *arXiv preprint arXiv:2111.02080*, 2021.

[19] J. Liu et al., "What makes good in-context examples for gpt-3?" *arXiv preprint arXiv:2101.06804*, 2021.

[20] J. Akbarnejad et al., "Sql querie recommendations," *VLDB Endowment*, vol. 3, no. 1-2, pp. 1597–1600, 2010.

[21] N. Khoussainova et al., "Snipsuggest: Context-aware autocompletion for sql," *VLDB Endowment*, vol. 4, no. 1, pp. 22–33, 2010.

[22] N. Arzamasova and K. Böhm, "Scalable and data-aware sql query recommendations," *Information Systems*, vol. 96, p. 101646, 2021.

[23] A. Kanade et al., "Learning and evaluating contextual embedding of source code," in *ICML*. PMLR, 2020, pp. 5110–5121.

[24] Z. Feng et al., "Codebert: A pre-trained model for programming and natural languages," *arXiv preprint arXiv:2002.08155*, 2020.

[25] "OpenAPI - Classificaiton Endpoint," https://beta.openai.com/docs/guides/classifications.

[26] G. Kul et al., "Similarity metrics for sql query clustering," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 12, pp. 2408–2420, 2018.

[27] Z. Chen et al., "Better few-shot text classification with pre-trained language model," in *ICANN*. Springer, 2021, pp. 537–548.

[28] V. Sanh et al., "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," *arXiv preprint arXiv:1910.01108*, 2019.

[29] Y. Liu et al., "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[30] S. Balkus et al., "Improving short text classification with augmented data using gpt-3," *arXiv preprint arXiv:2205.10981*, 2022.

[31] J. Thiergart et al., "Understanding emails and drafting responses–an approach using gpt-3," *arXiv preprint arXiv:2102.03062*, 2021.