TRANSQLATION: TRANsformer-based SQL RecommendATION

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*—The exponential growth of data production emphasizes the importance of database management systems (DBMS) for managing vast amounts of data. However, the complexity of writing Structured Query Language (SQL) queries requires a diverse range of skills, which can be a challenge for many users. Different approaches are proposed to address this challenge by aiding SQL users in mitigating their skill gaps. One of these approaches is to design recommendation systems that provide several suggestions to users for writing their next SQL queries. Despite the availability of such recommendation systems, they often have several limitations, such as lacking sequence-awareness, session-awareness, and context-awareness. In this paper, we propose TRANSQLATION, a session-aware and sequence-aware recommendation system that recommends the fragments of the subsequent SQL query in a user session. We demonstrate that TRANSQLATION outperforms existing works by achieving, on average, 22% more recommendation accuracy when having a large amount of data and is still effective even when training data is limited. We further demonstrate that considering contextual similarity is a critical aspect that can enhance the accuracy and relevance of recommendations in query recommendation systems.**

I. INTRODUCTION

Given the exponential growth in the amount of data being produced, it is undeniable that having powerful tools for managing such voluminous data is of paramount importance. One such tool that plays a fundamental role in data management is Database Management Systems (DBMS). A recent survey by StackOverflow [1] about the top-used DBMSs in 2022 revealed that several relational DBMSs, i.e., MySQL, PostgreSQL, SQLite, and Microsoft SQL Server, were among the top five most frequently-used DBMSs. This fact leads to the result that most database users rely on Structured Query Language (SQL) to access and query their data. However, writing SQL queries can be challenging for different users, as it demands diverse skills [2], [3], including: (i) expertise in SQL syntax [2], (ii) familiarity with the database schema [2], [4], and (iii) a comprehensive understanding of the application domain [3].

Since many users are deficient in at least one of the skills mentioned above, a considerable amount of research has been conducted to aid SQL users in mitigating such skill gaps. These research works can be classified into several categories, a few of which are as follows:

1) A popular research trend in this regard is to develop tools for converting users' requirements expressed in Natural Language (NL) to SQL queries, a.k.a. NL2SQL. This approach is particularly advantageous for novice users encountering challenges in comprehending the complex syntax of SQL [5], [6], [7], [8], [9], [10], [11].
2) Another approach is to design tools for clustering and identifying similar SQL queries based on different query similarity metrics [12], [13].
3) Another practical approach is to design recommendation systems that utilize previous queries submitted by a user to suggest the subsequent SQL query. This approach offers notable benefits; first, it enables the personalization of suggestions based on a user's query history during a particular user session; second, it benefits users with varying proficiency levels [14], [2], [15], [16], [17], [3].

From the three approaches mentioned above, using recommendation systems presents significant advantages in addressing the skill gap challenges faced by SQL users. Such systems also provide notable benefits to a broad spectrum of users by offering personalized suggestions. The output of such recommendation systems can be either a full SQL query or specific parts of the query. Recommending a full SQL query is challenging, given the rigid and complex syntax of SQL; thus, a considerable amount of research has been dedicated to recommending specific parts of the query, named *fragments*, rather than recommending the full query [14], [18], [2], [15]. In other words, most works focus on recommending the next SQL query's fragments, which are the *attributes* and the *table names* used in a query. We explain the concept of query fragments in more detail in Section II; however, to clarify, let us illustrate it using an example. Given an SQL query as `SELECT ATTR_1, ATTR_2 FROM TABLE_1 WHERE ATTR_3=1`, the set of fragments for this query would be: {`ATTR_1, ATTR_2, TABLE_1, ATTR_3`}. Recommending only query fragments may have limitations in isolation, as it does not cover the entire query construction. However, when combined with other query structure recommendation systems, users can utilize a two-stage process: first, they select their query structure, and then they choose fragments to fill the query structure. This approach enhances the overall SQL query creation process [4], [19], [3].

In query recommendation systems, users' query history can provide valuable insight into their preferences and objectives, which can be used to enhance the relevance of future recommendations. In other words, by considering the *context* of a user's current session, known as *session-awareness*, a system

Author accepted manuscript version of the publication in
2023 IEEE International Conference on Big Data (BigData)
Published version: https://doi.org/10.1109/BigData59044.2023.10386277

can provide more personalized recommendations tailored to their current needs. Moreover, by analyzing the sequence of queries in a user session, referred to as *sequence-awareness*, the system can capture the temporal dependencies between queries and better understand the user's intentions. Therefore, incorporating session-awareness and sequence-awareness into query recommendation systems provides the potential for having more effective and personalized recommendations that align better with the users' goals [19], [3].

There are different types of recommendation systems, such as collaborative filtering, content-based filtering, and hybrid methods, which combine collaborative filtering and content-based filtering to take advantage of their strengths [20], [21]. However, these methods often fall short regarding sequential and context-aware recommendations [20], [21]. A potential solution to overcome these limitations and provide sequential and context-aware recommendations is to use Natural Language Processing (NLP) techniques. These techniques leverage sequential models to encode a user's historical interactions into a vector representation that can be used to provide personalized recommendations [20], [21]. By considering the sequence of events and modeling the context in which the user interacts with the system, these models can provide more accurate and contextually relevant recommendations.

According to the available literature, only a few studies have considered session-awareness and sequence-awareness for providing effective and personalized suggestions [19], [3]. However, these studies employed one-hot encoding to represent user session queries, missing out on capturing semantic similarities and context-awareness. Notably, considering contextual and semantic similarity can improve the relevance of suggestions in various applications [20], [21]. Nevertheless, to date, no research has utilized the contextual and semantic similarities between queries to enhance the accuracy of recommendations. Therefore, the lack of considering contextual similarity is a critical limitation in query recommendation systems.

In this paper, we propose TRANSQLATION, a session-aware and sequence-aware recommender system that suggests fragments of the next SQL query in a user session by considering contextual and semantic similarities between queries. We propose a BERT-based architecture for TRANSQLATION that takes a user session as input and recommends the fragments of the subsequent SQL query in the same session. By leveraging BERT, we can take advantage of its sequence-awareness and context-awareness capabilities. We evaluate TRANSQLATION on two frequently-used datasets and demonstrate that, in case of having a large amount of data, TRANSQLATION outperforms existing works by 22% on average. Moreover, even when training data is limited, TRANSQLATION can still outperform the baselines due to its transferability. Overall, TRANSQLATION has significant potential to improve the accuracy and relevance of recommendations in query recommendation systems.

II. PRELIMINARIES

The main goal of TRANSQLATION is to propose a session-based, sequence-aware model for SQL query recommendations. This section begins with a formal definition of key concepts and terms related to the problem and then provides a brief overview of BERT, the language model used in TRANSQLATION.

A. Terminology

DEFINITION 1 (*Query*)**.** A query, denoted by Q, is composed of $|Q|$ tokens, and the sequence of the tokens in Q is shown as $(t_1, t_2, \cdots, t_{|Q|})$. As an example, assume Q_1 represents a query statement as: `SELECT ATTR_1, ATTR_2 FROM TABLE_1`. Then, the sequence of tokens of Q_1 is: (`SELECT`, `ATTR_1`, `ATTR_2`, `FROM`, `TABLE_1`).

DEFINITION 2 (*Fragments*)**.** Given a query Q, its fragments are the set of all attributes and table names in Q, represented by $\texttt{fragments}(Q)$. Accordingly, the fragment set of Q_1 is equal to: $\texttt{fragments}(Q_1)$ = {`ATTR_1`, `ATTR_2`, `TABLE_1`}.

DEFINITION 3 (*Session*)**.** A user session is a sequence of queries submitted by the user in chronological order. The following notation represents a session: $S = (Q_1, \cdots, Q_{|S|})$, where $|S|$ denotes the session length, i.e., the number of queries submitted by the user. The use of a numerical subscript to denote the order of queries in a session, such as i in Q_i, indicates the sequence-awareness. The sequence of queries in a user session often reflects the intention of the user [22], [17]. Moreover, if Q_t represents a query in the user session S and $t \leq |S|$, then Q_{t+1} and Q^*_{t+1} show the actual and recommended next query in the user session, respectively.

DEFINITION 4 (*Log File*)**.** A log file is a sequence of user sessions. The number of user sessions in the log file defines the length of the log file. Therefore, a log file L, with length $|L|$, is represented as $L = (S_1, \cdots, S_{|L|})$.

DEFINITION 5 (*Fragment Recommendation*)**.** If a log file L comprises of $|L|$ user sessions, i.e., $L = (S_1, \cdots, S_{|L|})$, then, the current user session is represented by $S_i = (Q_1, \cdots, Q_t)$, where $1 \leq i \leq |L|$. The primary objective in the fragment recommendation is to predict the set of fragments of the next query, $\texttt{fragments}(Q^*_{t+1})$.

DEFINITION 6 (*Accuracy*)**.** To evaluate the prediction accuracy of $\texttt{fragments}(Q^*_{t+1})$, similar to [3], we divide it into the table and attribute accuracy. Assuming Q_{t+1} as the actual next query, we calculate the table accuracy by considering only the table names present in $\texttt{fragments}(Q_{t+1})$ and $\texttt{fragments}(Q^*_{t+1})$. Similarly, we calculate the attribute accuracy by considering only the attribute names in $\texttt{fragments}(Q_{t+1})$ and $\texttt{fragments}(Q^*_{t+1})$. The formula is as follows:

$$\frac{|\texttt{fragments}(Q_{t+1}) \cap \texttt{fragments}(Q^*_{t+1})|}{|\texttt{fragments}(Q_{t+1})|} \quad (1)$$

B. BERT Language Model

BERT [23] is a fundamental model leveraged in the design of TRANSQLATION. It is a language model pre-trained on significantly large datasets, which can be fine-tuned using smaller datasets to perform various downstream tasks such as question-answering [24] and sentence classification [25]. BERT utilizes several special tokens to mark different parts of input sequences to facilitate the pre-training and fine-tuning processes. The most commonly-used special tokes are `[CLS]` and `[SEP]`. The `[CLS]` token is employed as the first token in the input sequence, and its output is often used in a classification task. The `[SEP]` token is utilized to differentiate between two sequences. BERT has different specialized variants, each targeting specific objectives. One widely-used variant is RoBERTa [26], which modifies the pre-training process of BERT to enhance its performance and is used as the base model for many other language models, such as CodeBERT [27], a RoBERTa-based language model for code-related tasks.

III. TRANSQLATION

As defined in Section II, the objective of the fragment recommendation is to predict the set of fragments of the next query of a user session. Figure 1 shows the architecture of TRANSQLATION. Below, we first elaborate on the different components of this architecture, and then we provide further details about the training and fine-tuning process.

A. Component Architecture

The architecture of TRANSQLATION's fragment prediction is depicted in Figure 1. The insight behind TRANSQLATION is as follows:

1) We create a representation for every fragment that has appeared in the user session so far. To do so, we **format the input** and feed it to the BERT language model. By using BERT, TRANSQLATION becomes aware of the sequence of queries in each user session, making it both sequence-aware and context-aware.
2) The obtained representations from BERT are fed to **the recommender layer**, which is a binary classifier.
3) The recommender layer estimates the likelihood of each fragment being included in the next query.

In what follows, we explain each of these steps in more detail.

Input Formatting. This step aims to format and prepare the input for being fed to the language model. To this end, each query in the current user session is first tokenized, and (i) a classification token (e.g., `[CLS]` for BERT and `<s>` for RoBERTa) is inserted before each fragment, and (ii) a separation token (e.g., `[SEP]` for BERT and `</s>` for RoBERTa) is appended at the end of it. We use $\tilde{Q}_i$ to denote the formatted input of query Q_i. For example, if Q_i is `SELECT ATTR_1 FROM TABLE_1`, then $\tilde{Q}_i$ will be `SELECT [CLS] ATTR_1 FROM [CLS] TABLE_1 [SEP]`.

After creating such augmented representation for each query in a user session, they are concatenated to form the final input representation (Figure 1). However, it is essential to consider that the maximum length of BERT input tokens is 512. To ensure compliance, if concatenating a query to the previous ones results in a representation that exceeds the 512 token limit, we remove a query from the beginning of the concatenation, allowing the input length to stay within 512. Finally, we feed the concatenation of such formatted and augmented representations to the BERT language model. Accordingly, BERT returns the encoded representation of all tokens ($\{\mathtt{R}_1, \mathtt{R}_2, \cdots\}$). Among all the encoded representations, the representation of the classification tokens ($\mathtt{R}_{\mathtt{[CLS]}}$) is extracted and passed to the subsequent layer. The representations of the classification tokens capture the contextual information of the whole user session and are used to make predictions for the fragments of the next query.

Recommender Layer. The recommendation layer is a binary classifier comprised of a basic linear neural network that accepts the encoded representation of the classification tokens from the language model. As shown in Figure 1, if there are c classification tokens, then the input to the recommendation layer is $\{\mathtt{R}'_1, \mathtt{R}'_2, \cdots, \mathtt{R}'_c\}$. For each input, the output of the recommendation layer (represented by $\{\ \hat{\mathtt{Y}}_1, \hat{\mathtt{Y}}_2, \cdots, \hat{\mathtt{Y}}_c\ \}$) is the probability of the corresponding fragment being included in the subsequent query. More specifically, having W and b as the parameters of the recommender layer, then: $\hat{\mathtt{Y}}_i = \mathtt{sigmoid}(W\mathtt{R}'_i + b) \ : \ 1 \leq i \leq c$. According to the predicted probabilities, $\hat{\mathtt{Y}}_i$, a straightforward strategy is employed to determine whether the fragments are expected to appear in the next query. Specifically, if the value of $\hat{\mathtt{Y}}_i$ exceeds 0.5, it is inferred that the corresponding fragment will appear in the next query.

B. Model Learning

To train and evaluate TRANSQLATION, we leverage SQL log files to create train and test datasets. The procedure for creating both datasets is the same. Let L be the log file from which the train (or test) dataset is created, where L comprises $|L|$ user sessions, i.e., $L = (S_1, S_2, \cdots, S_{|L|})$. For each session S_i in L, the queries $(Q_1, Q_2, \cdots, Q_t)$ are iterated over and formatted according to the input formatting method described in Section III-A. Let $(\tilde{Q}_1, \tilde{Q}_2, \cdots, \tilde{Q}_t)$ represent the formatted and augmented queries of S_i. Then, we create the inputs and their corresponding ground-truth labels as follows:

$$\forall j \in [1, t) : \begin{cases} \mathtt{input}_j = \mathtt{concat}(\tilde{Q}_1, ..., \tilde{Q}_j) \\ \mathtt{label}_j = \mathtt{fragments}(Q_{j+1}) \end{cases} \quad (2)$$

To clarify the procedure of creating the datasets, assume a session consisting of three queries denoted as Q_1, Q_2, and Q_3, alongside their respective formatted versions denoted as $\tilde{Q}_1$, $\tilde{Q}_2$, and $\tilde{Q}_3$. These queries and their formatted versions are represented in the first and second columns of Table I respectively.

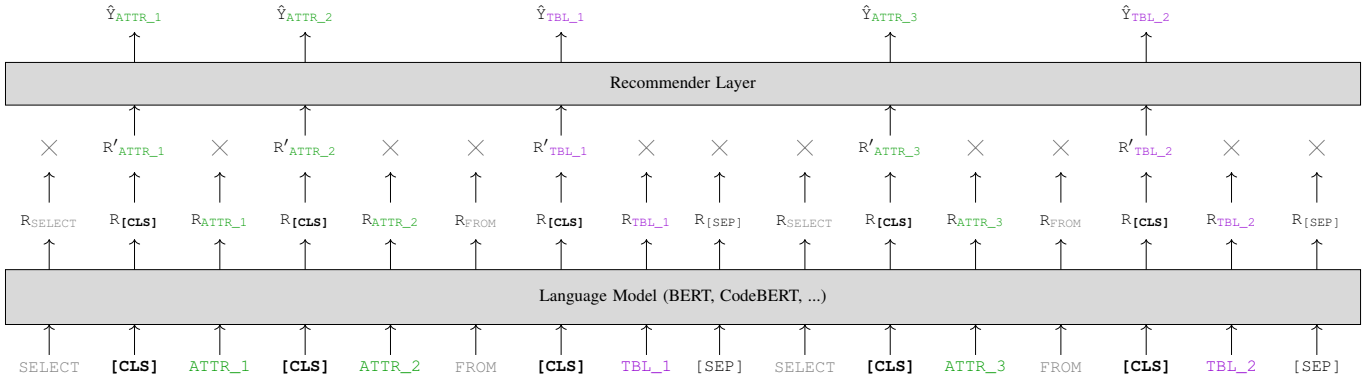


Fig. 1: Architecture of the Fragment Prediction Component

TABLE I: An Example of a User Session

Q_1: SELECT ATTR_1 FROM TBL_1 WHERE ATTR_3 = 1	**$\tilde{Q_1}$:** SELECT **[CLS]** ATTR_1 FROM **[CLS]** TBL_1 WHERE **[CLS]** ATTR_3 = 1 **[SEP]**
Q_2: SELECT ATTR_3 FROM TBL_2	**$\tilde{Q_2}$:** SELECT **[CLS]** ATTR_3 FROM **[CLS]** TBL_2 **[SEP]**
Q_3: SELECT ATTR_1, ATTR_2 FROM TBL_1	**$\tilde{Q_3}$:** SELECT **[CLS]** ATTR_1, **[CLS]** ATTR_2 FROM **[CLS]** TBL_1 **[SEP]**

Accordingly, for such a user session, the inputs and labels are created according to Table II:

IV. Experiments

In this section, we describe the datasets, introduce the baselines, and provide an overview of the experiments conducted to evaluate the effectiveness of TRANSQLATION in predicting the fragments of the user's next SQL query.

A. Datasets

For training and evaluating TRANSQLATION, we need to select several SQL log files for creating the train and test datasets according to the procedure mentioned in Section III-B. However, since TRANSQLATION is a session-aware and sequential recommendation system, the SQL log files should satisfy two requirements: (1) they need to be session-based, and (2) the sequence of queries in each session should be specified.

In this regard, we extract all the SQL log files used in similar related work [18], [14], [2], [15], [4], [12], [28], [19], [17], [16], [3] and filter them based on the two mentioned requirements, resulting in having only two SQL log files: Sloan Digital Sky Survey (SDSS) and SQLShare [29]. SDSS is an astronomical survey containing data for over three million astronomical objects, and SQLShare is collected from the SQLShare platform, a database-as-a-service platform where users can upload data and write queries to interact with it.

For both SDSS and SQLShare, we use the pre-processed data provided by [3], [19]. The data has several attributes, including session ID and query text. In both datasets, the number of user sessions is 11317. However, the average number of queries per user session in SQLShare and SDSS is 12 and 86, respectively. Thus, SDSS is about seven times larger than SQLShare. Such a huge difference makes these two datasets a perfect match for our work. By evaluating the performance of TRANSQLATION on both large and small datasets, we better understand how well the model generalizes across different data sizes and characteristics. This also helps us determine the scalability of the approach and its potentiality for being used in real-world scenarios, where the size and characteristics of data may vary significantly.

B. Baseline

To choose suitable baselines for comparison with TRANSQLATION, we conducted a thorough review of various related works [18], [14], [2], [15], [4], [12], [28], [19], [17], [16], [3]. Nevertheless, these works utilize diverse resources for their recommendations, such as log files, database data, and database schema. Given that TRANSQLATION relies on log files as the primary resource for recommendation, we can only compare it with other methods that also rely on log files. After filtering the related works based on this criterion, we ended up with five suitable ones as [2], [4], [19], [17], [3]. Among these, [2], [17] are not appropriate as baselines due to their limitations: the former lacks the capability to suggest new fragments–merely selecting fragments from the input, while the latter focuses on selecting the range or the value for attributes in where clauses. Hence, as baselines, we have chosen [4], [19], [3]. Notably, in [3], the authors proposed two different approaches, namely the convolutional sequence-to-sequence model (ConS2S) and the transformer-based model (Workload-aware), both of which are considered in our evaluations.

C. Fragment Prediction

This section presents the evaluation results of TRANSQLATION on SDSS and SQLShare log files. The train, test, and validation datasets are derived from the log files used in [3], following the approach explained in Section III-A. The accuracy of TRANSQLATION is assessed according to the metrics defined in Section II and summarized

TABLE II: Examples of Inputs and Labels for the User Session

	Input	Label
1	$\texttt{input}_1 = \vec{Q_1}$	$\texttt{label}_1 = \texttt{fragments}(Q_2) = \{$ ATTR_3, TBL_2 $\}$
2	$\texttt{input}_2 = \texttt{concat}(\vec{Q_1}, \vec{Q_2})$	$\texttt{label}_2 = \texttt{fragments}(Q_3) = \{$ ATTR_1, ATTR_2, TBL_1 $\}$

in Table III. Here, we analyze the evaluation results for both log files. The source code of TRANSQLATION is publicly available on GitHub[1].

SDSS. As explained in Section IV-A, since SDSS is a large dataset, evaluating TRANSQLATION on SDSS gives us an insight into TRANSQLATION's ability to generalize and perform well when provided with sufficient training data. In our study, we introduced two models: TRANSQLATION-BERT and TRANSQLATION-CodeBERT, based on BERT-base and CodeBERT, respectively. We chose BERT-base to evaluate how changing the input format for a basic transformer-based variant affects performance and CodeBERT to measure how using this input format for a structure-aware language model improves performance.

Rows 1 to 5 in Table IIIa describe the results of these models. The results confirm that both TRANSQLATION-BERT and TRANSQLATION-CodeBERT significantly outperform the baseline using a proper input format. Specifically, TRANSQLATION improves table accuracy by up to 15% and attribute accuracy by up to 30% [2].

SQLShare. SQLShare is a smaller dataset than SDSS, and using it enables us to evaluate TRANSQLATION's performance with limited training data. Given that TRANSQLATION-BERT outperforms TRANSQLATION-CodeBERT for SDSS, we chose BERT as the base language model for SQLShare and proposed the TRANSQLATION-BERT model. Row 1 of Table IIIb shows the evaluation result of TRANSQLATION-BERT on SQLShare. The results indicate that while TRANSQLATION achieves comparable results to the baseline, it does not outperform it due to having limited training data. Thus, to improve TRANSQLATION's performance on such small datasets, we hypothesized that fine-tuning the model on a large dataset and transferring it to a small dataset could enhance its performance.

We proposed TRANSQLATION-BERT-Transferred, which uses the TRANSQLATION-BERT model fine-tuned on SDSS as the base model and then fine-tuned it on SQLShare. The evaluation results in row 2 of Table IIIb show that TRANSQLATION-BERT-Transferred outperforms the baseline in terms of both table and attribute accuracy, improving them by 8% and 2%, respectively. Thus, thanks to the scalability and transferability of TRANSQLATION, it can outperform the baseline even with limited training data.

[1]https://github.com/ShirinTahmasebi/TranSQLation

[2]The improvement percentage of two values v_1 and v_2 is calculated as: $\frac{v_1 - v_2}{(v_1+v_2)/2} \times 100$

Freezing BERT Layers. As mentioned earlier, in TRANSQLATION-BERT, we have used the BERT-base, in which all of its 12 layers are trained during the fine-tuning phase by default. However, a common approach is to freeze some of the layers to prevent them from being updated during fine-tuning [30], [31], [32]. This helps the model focus on learning task-specific features and optimizing only the last few layers, which results in reducing the overall training time and computation cost and potentially improving the performance by focusing on more task-specific features.

Here, we analyze the effect of freezing the BERT layers on the performance of TRANSQLATION when trained on SDSS. Specifically, we investigate the impact of freezing each of the 12 BERT layers in a bottom-up order on the model's performance. Consequently, we obtain 12 models, each with different frozen layers. Then, we train each of the 12 models for four epochs and capture the loss value four times per epoch, resulting in 16 captured loss values per model. These loss values for each of the 12 models are depicted in Figure 2a. Moreover, the training time for each of the 12 models for each epoch is plotted in Figure 2b. We also ensure that these models do not overfit by evaluating their accuracy in predicting attributes and tables on the test dataset–presented in Figure 2c and Figure 2d, respectively.

After analyzing Figure 2a, Figure 2c, and Figure 2d, it can be inferred that the number of frozen layers significantly impacts the total loss value, loss convergence rate, models' prediction accuracy, and models' training time. Freezing all 12 layers and only training the recommender layer results in model underfitting, as the loss value increases significantly, and the accuracy on both attribute and table prediction reduces dramatically. Conversely, training all the layers improves the loss value and models' accuracy for both table and attribute prediction, despite increasing training time. An interesting finding is that the training time decreases significantly by freezing four layers, but the loss value improves, and the accuracy for both table and attribute prediction increases. Furthermore, by freezing four layers, the accuracy is even higher than the model in which all the layers are trained. Therefore, the insight gained from this experiment is that by freezing some of the BERT model layers, the model can learn task-specific features better and, in some cases, achieve better accuracy while spending less time training.

V. RELATED WORK

Figure 3 illustrates a timeline presenting the most prominent SQL query recommendation systems. This figure indicates that the idea of designing such systems emerged in 2009. Since then, this topic has attracted significant attention from

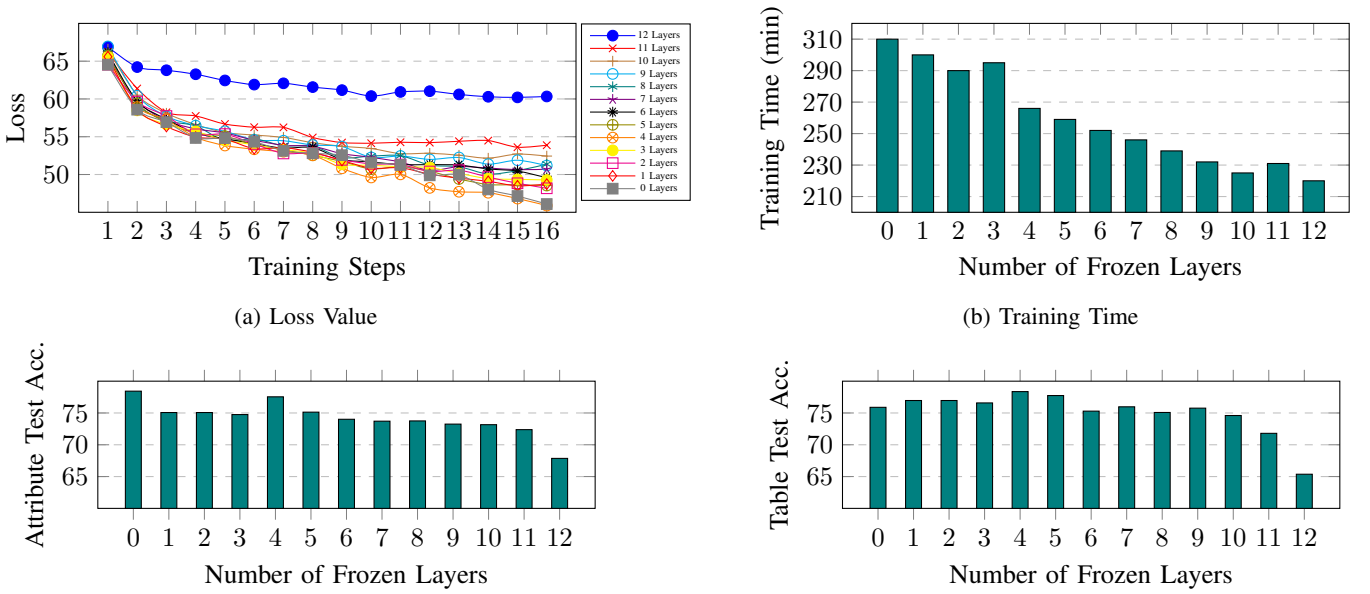
TABLE III: Accuracy of Fragment Prediction (The accuracy of baselines are reported according to [19], [3].)

(a) Results on SDSS

	Model	Table Accuracy	Attribute Accuracy
1	TRANSQLATION-BERT	75.89	78.43
2	TRANSQLATION-CodeBERT	70.99	74.34
3	Workload-aware [3]	65	58
4	ConS2S [3]	65	56
5	QueRIE [4]	46	26

(b) Results on SQLShare

	Model	Table Accuracy	Attribute Accuracy
1	TRANSQLATION-BERT	66.10	60.40
2	TRANSQLATION-BERT-Transferred	70.16	67.28
3	Workload-aware [3]	64	66
4	ConS2S [3]	46	55
5	QueRIE [4]	16	26



(a) Loss Value

(b) Training Time

(c) Attribute Accuracy

(d) Table Accuracy

Fig. 2: The Impact of Freezing BERT Layers in TRANSQLATION-BERT on (2a) Loss Value, (2b) Training Time, (2c) Attribute Accuracy, and (2d) Table Accuracy

researchers investigating ways to enhance the efficiency of these systems.

This section aims to review and compare previous works on SQL query recommendation systems, in order to demonstrate their evolution from 2009 to the present day. To accomplish this, we extracted the primary features and requirements of such systems and used them to categorize and compare the works. Furthermore, we discuss the limitations and shortcomings of previous works that we aim to address in ours.

The requirements and features used for comparing the existing works are as follows:

- Session-awareness: Incorporating session-awareness can be advantageous in providing personalized and relevant suggestions by taking into account other queries within a user session. Thus, this feature can be used as a point of comparison when evaluating different SQL query recommendation systems.
- Sequence-awareness: Considering sequence-awareness can enable the system to capture temporal dependencies between queries, allowing for a better understanding of users' intentions.
- Contextual Similarity: Contextual similarity has been shown to enhance the quality of recommendations in many recommendation systems by considering the semantic and contextual similarities between items [20], [21] As such, this feature is also used as a comparison metric when investigating SQL query recommenders.
- Recommendation Type: Recommendations may come in different forms. Specifically, they can either take the form of auto-completion of the query that is currently being written or suggestions for the next query.
- Basis of Recommendation: SQL query recommenders leverage different sources to provide suggestions. The three most frequently-used sources are database data, database schema, and query logs.

In what follows, the SQL query recommendation systems, mentioned in Figure 3, are described. Moreover, we present a summary of the comparison of these systems based on the aforementioned features and requirements in Table IV.

In the pioneering paper of [14], the authors designed a system for recommending join predicates–including join tables and join conditions. The sources of such recommendations are query logs and database schema. This system receives two inputs; input specification–the tables used in `WHERE`, and output specification–attributes in `SELECT` clauses. Then, given these two specifications as the system's input, the system

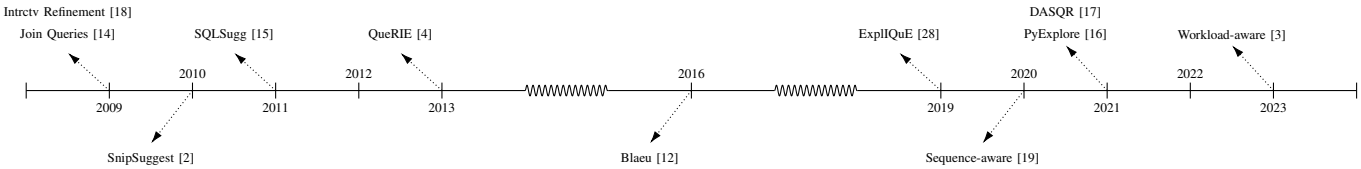


Fig. 3: Timeline of SQL Query Recommendation Systems

TABLE IV: Summary of SQL Recommendation Systems

	Session-aware	Sequence-aware	Contextual Similarity	Recommendation Type		Based on		
				Auto-completion	Next Query	Data	Schema	Logs
Join Queries [14]	✗	✗	✗	✓	✗	✗	✓	✓
Interactive Refinement [18]	✗	✗	✗	✗	✓	✓	✓	✗
SnipSuggest [2]	✗	✗	✗	✓	✗	✗	✗	✓
SQLSugg [15]	✗	✗	✗	✓	✗	✓	✓	✗
QueRIE [4]	✓	✗	✗	✗	✓	✗	✗	✓
Blaeu [12]	✗	✗	✗	✗	✓	✓	✗	✗
ExplIQuE [28]	✗	✗	✗	✗	✓	✓	✓	✗
PyExplore [16]	✗	✗	✗	✓	✗	✓	✗	✗
DASQR [17]	✓	✗	✗	✗	✓	✗	✗	✓
Sequence-aware [19] Workload-aware [3]	✓	✓	✗	✗	✓	✗	✗	✓
TRANSQLATION	✓	✓	✓	✗	✓	✗	✗	✓

generates a join query graph based on which it recommends join predicates. Therefore, the output of the system is a join query graph. The authors evaluated their work on a dataset named 'AT&T Proprietary Database,' which is not publicly available. The evaluation metric is the accuracy of the suggested join tables. However, this work has several critical limitations. First, the recommendations are based on a static join query graph that does not update by the addition of new data to the database. Second, it only recommends the join tables and conditions without focusing on the query structure or any other query parts. Third, it is neither session-aware nor sequence-aware.

The main focus of [18] is to address the *many/few answers* problem. This problem states that, while submitting queries to databases, for some queries, too many or very few tuples are returned. In such situations, the proposed innovative model aids users in refining their queries to return a reasonable number of tuples. Specifically, this model receives a query as input and returns a refined set of recommendations for adjusting the ranges of `WHERE` attributes. The recommendations are according to the database schema and data. However, the challenge is that multiple ways exist for narrowing down or expanding the ranges of `WHERE` attributes. Thus, in this work, the policy used for choosing a proper way to refine the ranges is to interact with users. The limitations of this work can be outlined as follows: first, it may require resubmitting queries, which can cause a heavy workload for most databases. Second, it only recommends the ranges of `WHERE` attributes without considering other aspects of the query structure. Third, the refinement process heavily relies on user involvement. Fourth, it does not take user sessions or the sequence of queries in each session into account for recommendations.

SnipSuggest [2] is a context-aware auto-completion system for SQL queries. In this case, context-aware denotes that the suggestions given by the system depend on the query written thus far. The main approach in this insightful work is as follows; first, a Directed Acyclic Graph (DAG) is created based on the log of submitted SQL queries, which is called *workload DAG*. Subsequently, depending on what the user is typing, the nodes of the workload DAG are ranked according to the probability of their appearance in the continuation of the query. Finally, the most probable nodes are provided as suggested fragments to the user. The recommendations are generated based on query log files. This model is evaluated on Sloan Digital Sky Survey (SDSS)[3] dataset. The evaluation metric is the accuracy of the recommended fragments. The limitations of this approach can be summarized as follows: first, the recommendations rely on a static DAG that is created only once and not updated as new data is added to the database. Second, the system does not consider user sessions or the sequence of queries within each session for generating recommendations.

In [15], the authors proposed an SQL recommendation system, named SQLSugg, which takes the current partial query written thus far as input; then, as output, the system recommends the fragments for the same query. The suggestions are based on database schema and data. The approach has two steps: (a) an offline step, in which two sets of graphs–schema and data graphs, known as templates[4]–are created and indexed. (b) an online step in which users type the query keywords. Then, the keywords are mapped to database attributes. Based on the matching between keywords and attributes, the most relevant schema graphs are selected. These selected graphs are ranked based on the number and relevance of the matched

[3]http://skyserver.sdss.org/dr16/en/home.aspx

[4]The template word here differs from what it means in our work.

data graphs. The model is evaluated on two datasets: (1) DBLP, a dataset of publication records, and (2) DBLife, a dataset of activity information of top people in the database community. To evaluate their approach, they asked experts to score the relevance of their suggestions. However, this approach has several critical limitations, including its reliance on having a static database schema and data, failure to consider user sessions, and neglect of the sequence of queries in each session.

In [4], the authors proposed a model, named QueRIE, which takes a full query from users as input. Then, as output, it selects the most probable query in the log file and returns it as the recommended next query. This work is evaluated on the SDSS dataset. Notably, this work considers the concept of *user sessions*; to do so, it leverages the queries within each session to create a vector representation for each session. Then, by comparing these vectors, it can identify similar user sessions. The limitation of this work is that it does not consider the sequence of queries in each user session for recommendations.

Blaeu [12] is an interactive system that facilitates data exploration and refinement. As input, Blaeu takes the users' initial query and clusters the data based on the result set of the query. The system then presents interactive cluster maps to users, allowing them to navigate and zoom into areas of interest. Moreover, Blaeu provides users with a query that can be used for selecting that area of data. The authors proposed several algorithms for creating the data clusters and evaluated their accuracy on two datasets: (1) US Bureau of Transportation Statistics, which describes delays of US internal flights during January 2010. (2) Hollywood films, which describes a few economic indicators for 785 movies released between 2007 and 2012. The limitations of this work are: first, generating data maps requires resubmitting queries, which can be computationally expensive for large databases. Second, users need to have a high level of involvement in the process. Third, it does not consider user sessions and sequence of queries for recommendation.

ExplIQuE [28] is a framework for query refinement recommendations to assist users in improving their queries. The model takes a full query as input and produces clustered result sets, accompanied by the `WHERE` clauses needed to retrieve each cluster. The recommendations are based on data records rather than logs and database schema. The system is evaluated on a dataset for bacteria growth on solid plates. The limitations of this work can be summarized as follows: the approach presented is only applicable when the database schema and data are static and do not change. Furthermore, the recommendation system does not consider user sessions or the sequence of queries in each session for recommendations.

PyExplore [16] is a framework that takes users' initial query with `WHERE` clause. Then, as output, it recommends a query with refined `WHERE` clause based on the data. The method involves measuring the correlation between attributes in the database and dividing them into several groups. Each group is then represented by one attribute, and a decision tree is created per representative attribute. Each node in each of these decision trees defines the split point of the corresponding attribute at that level. Then, the data rows of the database are clustered based on the leaves of the created decision tree. Thus, when users type an input query, the input query is mapped to a node of the decision tree. By navigating from the mapped node to the root, sample data is selected for each node and recommended to the user. The framework is evaluated on several datasets: CORDIS[5], SDSS, Movies[6], Car Sales (IBM)[7], Intel Lab Data[8]. For measuring the performance, expert users are asked to score the recommendations. The limitations of this work include the assumption that the database schema and data are static and never change, the absence of user session and sequence information for recommendation.

In the inspiring paper of [17], the authors provide a data-aware query recommendation system, called DASQR, capable of suggesting complete queries, query templates, and query fragments. The system is considered data-aware as it takes into account actual data values when recommending filtering conditions and predicates. In this work, the methods used for query representation are (1) feature-based, (2) tuple-based, and (3) access-area-based. The similarity between queries is evaluated based on the utilized representation approach. In case of using the first and second representation approach, cosine similarity is used as the metric. In case of using the third representation approach, two metrics–named overlap and closeness–are proposed for evaluation. The limitation of this work is that it does not consider the sequence of queries in each user session for recommendations.

In the insightful papers of [19], [3], an innovative recommendation system is introduced, which takes a full query as input and recommends the templates and fragments for the next query as output. This system is capable of providing sequence-aware and session-based suggestions at both the query fragment and query template levels. For the query fragment level, they exploited several models based on the encoder-decoder architecture–such as Sequence-to-Sequence (Seq2Seq) CNN, Seq2Seq RNN, and transformers–to predict the next query; then, the predicted query is parsed to extract its fragments. Also, for the query template level, the same Seq2Seq models are used as a classification task. In both levels, one-hot encoding is leveraged as the way of query vectorization. They have evaluated their methods on two open-source datasets–SDSS and SQLShare. The evaluation metrics for fragment prediction are precision–which equals the number of correct fragment predictions over the number of total fragment predictions, and recall, which equals the number of correct fragment predictions over the number of total target fragments. The evaluation metric for template prediction is prediction accuracy. A noteworthy advantage of this system is that it is session-aware and sequence-aware. However, a critical limitation of this work is the utilization of one-hot encoding for embedding queries, which does not consider

[5]https://data.europa.eu/euodp/en/data/dataset/cordisH2020projects
[6]https://www.kaggle.com/datasets/rounakbanik/the-movies-dataset
[7]https://www.kaggle.com/thatbrock/ibm-watson-saleswinloss
[8]http://db.csail.mit.edu/labdata/labdata.html

word similarities in queries and consequently leads to reduced performance.

VI. Conclusion and Future Work

In this paper, we proposed TranSQLation, a recommendation system for suggesting SQL query fragments within a user session. TranSQLation is designed to be session-aware, sequence-aware, and context-aware, addressing limitations in existing SQL recommenders. Our results showed that TranSQLation outperforms existing solutions, achieving a 22% performance boost when trained on ample data and remaining effective even with limited data. As a future work, we aim to predict query templates and integrate them into TranSQLation for an enhanced recommender. This combination will allow users to choose a template for their next query and fill it in with recommended fragments.

Acknowledgment

This work received partial funding from the EC through the projects DataCloud (101016835) and enRichMyData (101070284). The computations were enabled by resources provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS) and the Swedish National Infrastructure for Computing (SNIC) partially funded by the Swedish Research Council through grant agreements no. 2022-06725 and no. 2018-05973. Also, we would like to express our sincere gratitude to Eugenie Y. Lai, who generously shared her expertise, codebase, and datasets with us.

References

[1] "Stackoverflow survey - 2022," "https://survey.stackoverflow.co/2022/#most-popular-technologies-database", accessed: 2023-03-08.

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

[3] E. Lai et al., "Workload-aware query recommendation using deep learning," *26th International Conference on Extending Database Technology (EDBT)*, pp. 53–65, 2023.

[4] M. Eirinaki et al., "Querie: Collaborative database exploration," vol. 26, no. 7. IEEE, 2013, pp. 1778–1790.

[5] L. Dong and M. Lapata, "Coarse-to-fine decoding for neural semantic parsing," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, vol. 1. Association for Computational Linguistics, 2018, pp. 731–742.

[6] T. Yu et al., "SyntaxSQLNet: Syntax tree networks for complex and cross-domain text-to-SQL task," in *Proceedings of EMNLP*. Association for Computational Linguistics, 2018.

[7] T. Shi et al., "Incsql: Training incremental text-to-sql parsers with non-deterministic oracles," *CoRR*, 2018.

[8] B. Wang et al., "RAT-SQL: Relation-aware schema encoding and linking for text-to-SQL parsers," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2019.

[9] J. Guo et al., "Towards complex text-to-SQL in cross-domain database with intermediate representation," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2019.

[10] D. Choi et al., "Ryansql: Recursively applying sketch-based slot fillings for complex text-to-sql in cross-domain databases," *Computational Linguistics*, vol. 47, no. 2, pp. 309–332, 2021.

[11] L. Dou et al., "Unisar: A unified structure-aware autoregressive language model for text-to-sql semantic parsing," *International Journal of Machine Learning and Cybernetics*, 2023.

[12] T. Sellam et al., "Cluster-driven navigation of the query space," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 5, pp. 1118–1131, 2016.

[13] S. Tahmasebi et al., "Transql: A transformer-based model for classifying sql queries," in *21st IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2022, pp. 788–793.

[14] X. Yang et al., "Recommending join queries via query log analysis," in *International Conference on Data Engineering*. IEEE, 2009, pp. 964–975.

[15] J. Fan et al., "Interactive sql query suggestion: Making databases user-friendly," in *International Conference on Data Engineering*. IEEE, 2011, pp. 351–362.

[16] A. Glenis et al., "Pyexplore: Query recommendations for data exploration without query logs," in *International Conference on Management of Data*, 2021, pp. 2731–2735.

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

[18] C. Mishra and N. Koudas, "Interactive query refinement," in *International Conference on Extending Database Technology: Advances in Database Technology*, 2009, pp. 862–873.

[19] E. Lai et al., "Sequence-aware query recommendation using deep learning," *VLDB Endowment*, vol. 12, 2020.

[20] F. Sun et al., "Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer," in *ACM international conference on information and knowledge management*, 2019, pp. 1441–1450.

[21] G. Moreira et al., "Transformers4rec: Bridging the gap between nlp and sequential/session-based recommendation," in *ACM Conference on Recommender Systems*, 2021, pp. 143–153.

[22] J. Aligon et al., "A collaborative filtering approach for recommending olap sessions," *Decision Support Systems*, vol. 69, pp. 20–30, 2015.

[23] J. Devlin et al., "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of NAACL-HLT*, vol. 1, 2018, pp. 4171–4186.

[24] W. Yang et al., "End-to-end open-domain question answering with BERTserini," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*. Association for Computational Linguistics, 2019.

[25] A. Cohan et al., "Pretrained language models for sequential sentence classification," in *Proceedings of EMNLP-IJCNLP*. Association for Computational Linguistics, 2019, pp. 3693–3699.

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

[27] Z. Feng et al., "CodeBERT: A pre-trained model for programming and natural languages," in *EMNLP*. Association for Computational Linguistics, 2020.

[28] M. Le Guilly et al., "Explique: Interactive databases exploration with sql," in *ACM International Conference on Information and Knowledge Management*, 2019, pp. 2877–2880.

[29] S. Jain et al., "Sqlshare: Results from a multi-year sql-as-a-service experiment," in *International Conference on Management of Data*, 2016, pp. 281–293.

[30] D. Grießhaber et al., "Fine-tuning BERT for low-resource natural language understanding via active learning," in *Proceedings of the 28th International Conference on Computational Linguistics*. International Committee on Computational Linguistics, 2020.

[31] J. Lee et al., "What would elsa do? freezing layers during transformer fine-tuning," *arXiv preprint arXiv:1911.03090*, 2019.

[32] X. Jiao et al., "Lightmbert: A simple yet effective method for multilingual bert distillation," *arXiv preprint arXiv:2103.06418*, 2021.