Sequence-Aware Query Recommendation Using Deep Learning

Eugenie Y. Lai
University of British Columbia
eugenie.lai@alumni.ubc.ca

Mostafa Milani
University of Western Ontario
mostafa.milani@uwo.ca

Omar AlOmeir
University of British Columbia
oomeir@cs.ubc.ca

Rachel Pottinger
University of British Columbia
rap@cs.ubc.ca

ABSTRACT

Users interact with databases management systems by writing sequences of queries. Those sequences encode important information. Current SQL query recommendation approaches do not take that sequence into consideration. Our work presents a novel sequence-aware approach to query recommendation. We use deep learning prediction models trained on query sequences extracted from large-scale query workloads to build our approach. We present users with contextual (query fragments) and structural (query templates) information that can aid them in formulating their next query. We thoroughly analyze query sequences in two real-world query workloads, the Sloan Digital Sky Survey (SDSS) and the SQLShare workload. Empirical results show that the sequence-aware, deep-learning approach outperforms methods that do not use sequence information.

PVLDB Reference Format:
. . *PVLDB*, 12(xxx): xxxx-yyyy, 2020.
DOI: https://doi.org/10.14778/xxxxxxx.xxxxxxx

1. INTRODUCTION

Database management systems (DBMSs) provide essential infrastructure for users to access and interact with high volumes of data. End users often interact with such database applications through SQL query sessions: a sequence of queries posted by a user to achieve their intent. Hence both field-specific and database-related expertise are required since users need to understand the database schema and SQL syntax to pose meaningful questions.

Recommending information about query statements is one of the ways to help users formulate SQL queries. There are many different approaches. The QueRIE framework [13, 1] uses collaborative filtering and recommends historical queries from workloads. SnipSuggest [22, 21] models each query as a vertex in a directed acyclic graph (DAG) and autocompletes an input query with query fragments (e.g., tables, attributes) at an interactive speed based on probability scores. SQLSugg [14] generates queryable templates, i.e., undirected graphs with attributes as vertices.

However, these approaches have two common limitations. The **first** common limitation is that the existing approaches consider queries individually and ignore query sequences. Since each session is a snapshot of a user's knowledge exploration [6, 13], the sequential changes in the query statement are a sequence of Q&A's that reveals users' thought process. Hence query sequences and the sequential changes in query sessions can contain valuable information in revealing user intent and making recommendations. To illustrate, we use an example from the SQLShare [19] workload, collected from a database-as-a-service platform, the SQLShare, where human users upload their data and write queries to interact.

EXAMPLE 1. *Changes in queries in a user session tell a story. Figure 1 shows a session from the SQLShare workload with queries over a database about genomics experiments. The user starts by counting the number of unique experiment types (Q_1), then explores the gene and type used in each experiment (Q_2), and ends the sequence by asking the number of genes used in each type of experiments, where the gene count is greater than a threshold (Q_3). Those three queries represent a sequential exploratory pattern.* □

Instead of using natural sequences, the existing approaches rely on a single, human-selected metric to compute query relevance and make either whole-query or query fragment recommendations. Recommending whole queries is challenging as the model-generated queries are likely complex and contain syntax errors. To solve these problems, the QueRIE framework turns queries into vectors and finds equivalent raw queries in the workload using cosine similarity. SQLSugg measures attribute importance using the number of distinct values. Then it maps attributes to SQL keywords based on attribute importance and generate ASPJ queries clause-by-clause. Instead of whole queries, SnipSuggest recommends tables, attributes, functions, and conditions as the user types a query. It assigns a probability score to the edges between query vertices based on the queries' overall popularity in the workload. However, none of the existing approaches consider queries at the session level.

The **second** common limitation of existing query recommendation systems is that the existing approaches rely on

Proceedings of the VLDB Endowment, Vol. 12, No. xxx
ISSN 2150-8097.
DOI: https://doi.org/10.14778/xxxxxxx.xxxxxxx

```
Q1: SELECT COUNT(DISTINCT type) FROM Experiment
Q2: SELECT gene, type FROM Experiment ORDER BY type
Q3: SELECT type, COUNT(gene) FROM Experiment
    GROUP BY type HAVING COUNT(gene) > 1
```

Figure 1: A sequence of queries in the SQLShare workload.

manually selected features to represent query statements. Such human intervention leads to inevitable information loss when representing a query's semantic meaning and user intent. The QueRIE framework encodes queries as vectors, where each dimension is either a table or an attribute. SnipSuggest models each query as a set of tables, attributes, functions, and conditions. SQLSugg uses tables and attributes to generate and rank queryable templates. Since queries can have distinct meanings even if they have similar selected features and vice versa, recommendations based on such representations become arbitrary. These query representations are also oblivious to relationships between the words in queries. We use an example from the Sloan Digital Sky Survey (SDSS) [34, 35] workload, collected from the SDSS database that stores images, spectra, and catalog data for more than three million astronomical objects.

EXAMPLE 2. *Figure 2 shows queries Q_4 and Q_5 from two different sessions in SDSS, and the query Q in the current user session. The QuerRIE framework compares queries based on fragments such as tables name, and therefore finds Q more similar to Q_4 rather than Q_5 as the same set of tables appear in Q and Q_4. However, considering the structure of the queries, Q is more similar to Q_5. They only differ in a pair of tables,* SpecObj *and* SpecPhoto*, which means Q_5's session perhaps can be more effective for query recommendations for the current user compared to Q_4's session.* □

Example 2 shows that using manually picked features, such as table name, for comparing queries can result in second-rate recommendations. In this example, we also need to compare queries using their structural properties to be able to give a preferred recommendation. Selecting the right set of features for query recommendation is difficult. The example shows that it is not trivial which features summarize the syntactic properties of queries. In general, the choice of query features depends on the application and the type of queries in the workload. As such, this calls for solutions with automatic feature selection that can adapt to different workloads and queries.

In this work, we aim to address these concerns by modelling the query recommendation task as a natural thought process of the end-users. To do that, we use whole query statements and query session sequences in SQL query workloads, a collective knowledge exploration history of past users in the form of query sessions. We seek to give users contextual and structural information about their next-step queries. Specifically, given a query, our approach predicts two things of the next query: query fragments and query templates. Query fragments are the database-dependent components, e.g., tables and attributes, while query templates are a composite of SQL keywords (Figure 4).

We leverage the sequential feature in knowledge exploration using deep learning prediction models. Neural networks obviate the need for human intervention such as feature selection since they have the capacity to hold and learn

```
Q4: SELECT DISTINCT obj.specclass, obj.z
    FROM SpecObj obj, SpecLine ln
    WHERE obj.specobjid = ln.specobjid
    AND obj.specclass = 3
Q5: SELECT top 10 pt.specclass, pt.z FROM SpecPhoto pt
    WHERE pt.zErr NOT IN
        (SELECT DISTINCT p.zErr
         FROM SpecPhoto p, SpecLine ln
         WHERE p.specobjid = ln.specobjid)
Q: SELECT top 5 obj.specclass, obj.z FROM SpecObj obj
   WHERE obj.z NOT IN
     (SELECT DISTINCT j.z FROM SpecObj j, SpecLine ln
      WHERE j.specobjid = ln.specobjid)
```

Figure 2: Q_5 and Q are structurally similar as both are nested top-k queries, but Q_4 and Q are more similar based on the tables they access, as they query SpecObj but Q_5 queries SpecPhoto.

from large volumes of workload data at all levels [25], e.g., word, query, and session level. Instead of using manually picked properties, neural networks provide suitable inductive biases catered to queries [45], i.e., our models use workload data to exploit patterns and decide what syntactic properties are relevant to our query recommendation task.

We model our sequence-aware query recommendation problem as a query prediction problem for a given user input. A closely related NLP area is conventional interactive language models such as chatbots whose goal is to respond to user input [44, 37]. Instead of recommending whole queries, we divide our query prediction task into query fragment prediction and query template prediction. Given a query statement, we use a sequence-to-sequence (seq2seq) model to predict query fragments and then use a classification model to predict query templates. The seq2seq architecture is drawn from the space of sentence-level NLP [28, 25, 32, 39, 41] and query representation learning in databases [18, 20]. We use a standard classification model from NLP [17].

We first train the seq2seq models to predict the successor query and parse the model-generated query to extract query fragments. Applying fine-tuning, we use the trained seq2seq models for query template prediction as a classification task. The trained seq2seq models contain compressed representations of the training data and hence provide a starting point for downstream tasks [11, 18]. Fine-tuning has been abundantly used in NLP for text summarization [29], text classification [17], and text generation1 [12].

Deep learning models exploit the intrinsic structure in data differently depending on the choice of neural networks [23, 45]. We follow the common practices in the related topics and explore three seq2seq models, recurrent neural networks (RNNs) with gated recurrent units (GRUs), the transformer, and convolutional neural networks (CNNs). The models extract features in different ways: RNNs with GRUs recognize the sequential dependencies between tokens in a query; the transformer focuses on the relatedness between tokens regardless of time and distances; CNNs automatically identify n-grams in a query. We apply the models at the word level.

We leverage whole queries and introduce additional challenges, e.g., sequence-aware query recommendation. The following is a summary of our contributions:

- We formally define a new approach that provides query structural and contextual information to guide DBMS

users to formulate next-step queries (Section 2).

- We conduct a thorough analysis of real-world workload data to show the sequential changes in query syntactic properties (Section 4).
- We adapt a broad set of neural network models to our problem and show our modelling workflow (Section 5).
- We empirically evaluate our approach (Section 6).

We also give model background (Section 3), discuss related work (Section 7), and conclude (Section 8).

2. PROBLEM DEFINITION

We first present some preliminary definitions before explaining the query recommendation problem.

DEFINITION 1. *[Query] A query $Q = (t_1, ..., t_{|Q|})$ is a sequence of tokens t_i from a vocabulary V. We consider a vocabulary that contains words from SQL queries. We define v to denote the size of V and $\mathcal{Q}$ to denote the collection of all queries over V.* □

EXAMPLE 3. *For the query "SELECT * FROM PhotoTag", Q_1 = (SELECT, *, FROM, PhotoTag) is a query with a vocabulary of words, and Q_2 = (S, E, L, E, ...) is a query with a vocabulary of characters.* □

DEFINITION 2. *[Query Encoding] For a token t_i in a query Q, we define $\boldsymbol{e}_i \in \{0,1\}^v$ as the one-hot encoding of t_i, i.e. a vector of bits for tokens in V where the bit that corresponds to t_i is 1 and all the other bits are 0.* □

EXAMPLE 4. *The query encoding of t_1 = SELECT and t_2 = * w.r.t. a vocabulary $V = \{t_1, t_2, t_3, t_4\}$ is the following one-hot encoding: $\boldsymbol{e}_1 = [1\ 0\ 0\ 0]$ and $\boldsymbol{e}_2 = [0\ 1\ 0\ 0]$.* □

DEFINITION 3. *[Session, Query Subsequence, and Workload] A (user) session $\mathcal{S} = (Q_1, ..., Q_{|\mathcal{S}|})$ is a sequence of queries. A query subsequence refers to a pair $\langle Q_i, Q_{i+1} \rangle$ of consecutive queries in a session. A query workload $\mathcal{W}$ is a set of sessions $\{\mathcal{S}_1, ..., \mathcal{S}_m\}$.* □

In our query recommendation setting, we assume a user poses a sequence of queries that are recorded in a session $\mathcal{S}^*$. The goal in our recommender system is to help the user in writing the next query, Q^*_{i+1}, considering the last posed query, Q^*_i. A direct approach is to recommend an entire query statement. However, this is often not practical for several reasons. First, we often have limited knowledge about the user's intention based on the last query posed by the user. Second, suggesting an entire query may not necessarily help the user to compose the next query, especially if the predicted next query has very complex syntax that may be difficult to understand. Another problem with this approach is that recommending an entire query requires models that can generate error-free queries (both syntax error and logical error), which is a challenging task.

For the above reasons, we define the query recommendation problem as a query property prediction task. Instead of predicting an entire query, we predict certain properties of the next query. This will better help the user compose the query. We particularly focus on two query properties: the structure of the next query, which we specify with query templates, and the fragments of the next query such as table names, attribute names, and built-in functions.

```
SELECT j.target, CAST(j.estimate AS VARCHAR) AS estimate
FROM Jobs j, Status s,
        (SELECT DISTINCT target, queue FROM Servers r
         WHERE r.queue NOT IN (SELECT MIN(queue)
                               FROM Servers
                               GROUP BY target))
WHERE j.outputtype LIKE '%QUERY%'
```

Figure 3: Sample query Q

```
SELECT Attribute, Function(Attribute AS VARCHAR)
FROM Table, Table,
  (SELECT DISTINCT Attribute, Attribute FROM Table
   WHERE Attribute NOT IN (SELECT Function(Attribute)
                           FROM Table
                           GROUP BY Attribute))
WHERE Attribute LIKE Literal
```

Figure 4: The query template for query Q

DEFINITION 4. *[Query Fragment and Template] Given a query Q, a* query fragment *is either a table, an attribute, or a function in Q. We use tables(Q), attributes(Q), functions(Q), and literals(Q) to respectively refer to the sets of these fragments in Q. The template of Q, denoted by template(Q), is the statement obtained from Q by replacing tables, attributes, function names, and literals in Q with Table, Attribute, Function, and Literal resp., and removing aliases.* □

EXAMPLE 5. *Figure 4 shows template(Q), the template of the query Q in Figure 3. The fragments of Q are tables(Q) = {Jobs, Status, Servers}, attributes(Q) = {target, estimate, queue, outputtype}, functions(Q) = {CAST, MIN}, and literals(Q) = {%QUERY%}.* □

We use a query workload to collect information about the queries and effectively predict the next query's properties. We define the query recommendation as query property prediction tasks.

DEFINITION 5. *Let $\mathcal{W} = \{\mathcal{S}_1, ..., \mathcal{S}_m\}$ be a workload over a database D, and $\langle Q_i, Q_{i+1} \rangle$ be a query subsequence where Q_i is the last query the user posed to D, and Q_{i+1} is the next query that the user will pose. The query template prediction problem is to predict template(Q_{i+1}). Similarly, the query fragment prediction problems are to predict the sets of fragments tables(Q_{i+1}), attributes(Q_{i+1}), functions(Q_{i+1}), and literals(Q_{i+1}).* □

The prediction tasks in Definition 5 are the intrinsic building blocks of our query recommendation solution. A summarization of our solution and the application of template and fragment prediction is as follows: assume the user starts a session by posing an initial query which presumably is not the final intended query. To compose the next query, we provide recommendations in two ways. First, we predict the next query's template to serve as a basis for the the user to further modify and compose the next query. Second, we predict the fragments in the next query and present them to the user while she fills the fragments in the template. Our solution provides a list of top-k templates and fragments. We build the models in Section 5. We empirically evaluate our solution in Section 6 and show the efficacy of our recommendations as k increases.

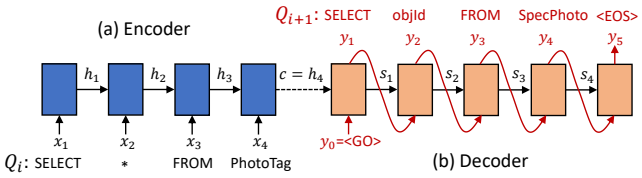


Figure 5: An RNN seq2seq model that takes query Q in a session and predicts the next query Q^*.

3. SEQUENCE-TO-SEQUENCE MODELS

We model sequence-aware query recommendation as a query prediction problem for a given query input. A closely related field is NLP. Sequence-to-sequence (seq2seq) refers to models for solving supervised learning problems where both the input and the target are sequences of tokens. These models are used in a wide variety of applications in NLP such as machine translation, headline generation, and text summarization [39]. Seq2seq models with the state-of-the-art performance are predominantly neural network models.

In this section, we provide the necessary background about seq2seq models that we apply in Section 5. We review the common encoder-decoder architecture of seq2seq neural networks models in Section 3.1, then we explain three seq2seq models in Sections 3.2, 3.3, and 3.4, which we will apply for fragment prediction. We review a fine-tuning that we use to build new models for classification problems to address the template prediction task.

3.1 Encoder-Decoder Seq2Seq Architecture

Sequence-to-sequence (seq2seq) models aim to map an input sequence $(\boldsymbol{x_1}, ..., \boldsymbol{x_n})$ to a target sequence $(\boldsymbol{y_1}, ..., \boldsymbol{y_m})$, where $\boldsymbol{x_i}, \boldsymbol{y_j} \in \mathbb{R}^d$ are d-dimensional vectors representing the input and target tokens. In these models, the encoder reads the input, token-by-token or in-parallel, and computes hidden states $\boldsymbol{h_1}, ..., \boldsymbol{h_n}$. The decoder receives a context vector $\boldsymbol{c} = g(\boldsymbol{h_1}, ..., \boldsymbol{h_n})$ that represents the input sequence, and it generates an output sequence.

For query recommendation application, an input sequence $(\boldsymbol{x_1}, ..., \boldsymbol{x_n})$ represents a sequence of tokens $(t_1, ..., t_n)$ in an input SQL query Q_i from a session $\mathcal{S}$ where each $\boldsymbol{x_i} = \boldsymbol{e_i}$ is the one-hot representation of token t_i. The target $(\boldsymbol{y_1}, ..., \boldsymbol{y_m})$ represents the next query Q_{i+1} in the session $\mathcal{S}$. The seq2seq models compare their output query to the target query to improve the encoder's ability to summarize queries with numeric vectors and the decoder's ability to use the vectors to map the input to the target. This seq2seq architecture is used as an autoencoder in [18] for query representation learning, where $\boldsymbol{x_i} = \boldsymbol{y_j}$, and the main focus is the context vector $\boldsymbol{c}$. Instead, our work uses this architecture to predict the next query, where $\boldsymbol{x_i} \neq \boldsymbol{y_j}$, and we focus on the decoder output sequence (see details in Section 5). In addition to the RNN model explored in [18, 20], we adapted the transformer and CNN models in Section 5.

3.2 RNN Seq2seq Model with GRUs

Recurrent Neural Networks (RNNs) are neural networks for processing sequential data [39, 8]. An RNN takes an input sequence $(\boldsymbol{x_1}, ..., \boldsymbol{x_n})$ one token at a time and generates a hidden state $(\boldsymbol{h_1}, ..., \boldsymbol{h_n})$. An element $\boldsymbol{h_i}$ of the hidden state is computed at each time step i and contains the semantic meaning of the sequence tokens seen so far. The output $\boldsymbol{y_j}$ has the same size as the input $\boldsymbol{x_i}$.

Standard RNNs suffer from vanishing and exploding gradient problems, which impair their ability to handle long-term temporal dependencies. LSTM networks resolve these issues by extending RNN with a memory cell and a set of gates that decide to input, output, or forget information. Similar to LSTM units, Gated Recurrent Units (GRUs) also use gates to control information flow but without separate memory cells. RNNs with GRUs have been shown to outperform LSTM networks on sequence modelling tasks while being computationally more efficient [9, 23].

An RNN seq2seq model consists of an RNN encoder and an RNN decoder that are connected through the context vector $\boldsymbol{c}$ [3, 4]. Figure 5 shows such a model with an input and output queries. The context vector $\boldsymbol{c}$ encodes all the information from the input sequence and allows the model to map sequences of different sizes, whereas standard RNN that has input and output of the same size. The decoder uses the final $\boldsymbol{h_n}$ from the encoder to set the context vector as its initial state ($\boldsymbol{c} = \boldsymbol{h_n}$) and the standard RNN formula with the encoder outputs to estimate the conditional probability. Similar to the encoder, the decoder RNN also workd iteratively. At the j-th iteration, it generates the output token $\boldsymbol{y_j}$ using $\boldsymbol{s_{j-1}}$ and $\boldsymbol{y_j}$ from the previous iteration. In our query recommender, we implement the encoder and decoder with one layer RNNs with GRUs [9]. In our application of RNN seq2seq model, we pass the decoder's output to a softmax layer to generate the probabilities of each token in the output query. We apply the cross-entropy loss and use the Adam optimizer with default settings.

3.3 The Transformer

Transformers are seq2seq models designed to solve two problems faced by the RNN seq2seq models: long training time due to recurring structure of the RNN encoder and drop in performance due to long temporal dependencies [41]. Instead of sequentially processing an input sequence token-by-token as in an RNN seq2seq model, transformers take an input sequence as a whole and read the tokens in-parallel.

Transformers apply *the attention mechanism* to compute more meaningful representations by scoring tokens based on their relatedness to others. The attention mechanism is first introduced in [3] where the authors extend the RNN seq2seq model with context vectors $\boldsymbol{c_j}$ for each output token $\boldsymbol{y}_j$. Unlike the context vector $\boldsymbol{c}$ in the recurrent model that is equal to the last hidden state $\boldsymbol{h_n}$, $\boldsymbol{c_j}$ is a weighted sum of the all the hidden states and is defined as $\boldsymbol{c_j} = \sum_{j \in [1,m]} (\alpha_{ji} \times \boldsymbol{h_i})$. The weight α_{ji} is a probability that the token $\boldsymbol{y}_j$ is aligned to, or translated from, an input token $\boldsymbol{x}_i$. This allows these models to "attend to" certain parts of the input while generating each output token. For example, in Figure 6, the context vector $\boldsymbol{c_4}$ with a large weight α_{44} means the output $\boldsymbol{x}_4$ mainly attends to the hidden state $\boldsymbol{h_4}$ of the input $\boldsymbol{y}_4$ and the table name PhotoTag. This implies that users who access SpecPhoto in a query often query PhotoTag next.

Transformers apply different forms of attention that we briefly explain here and refer the readers to [41] for more detail. Transformers use self-attention to compute which input tokens we should attend to while we calculate the hidden representation of an input token. Similarly, self-attention is applied to previously computed output tokens when we calculate the current output token using masking that ignores the next tokens. Transformers also attend to the input tokens when computing the outputs as in the conventional

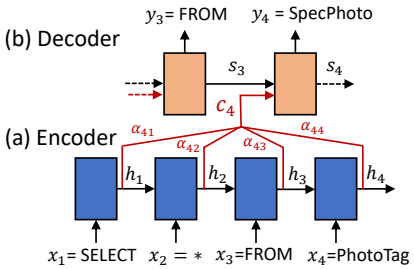


Figure 6: Attention mechanism in seq2seq models.

attention mechanism shown in Figure 6. To compute attention, transformers use the dot product attention mechanism, where the tokens to attend to are represented with keys and values and the current token is represented as a query (not to be confused with the queries in our recommendation solution). Keys, values and queries are matrices and computing attention is reduced to matrix multiplication which can be performed efficiently. Transformers further refine attention by adding a concept of multi-head attention, which gives the model multiple sub-representations of the input by expending its ability to focus on different tokens [41].

We use a standard transformer model [41, 24]. The encoder is consist of six identical layers, and each has two sub-layers. The first sub-layer implements a multi-head self-attention mechanism, and the second is a fully connected feed-forward network. The decoder has a similar structure. In addition to the two sub-layers in each encoder layer, the decoder includes a middle sub-layer that performs multi-head attention over the encoder output matrix. The self-attention layer is slightly different from the encoder as the current and future tokens are masked w.r.t. a position so the decoder is only allowed to attend to the tokens prior to a position. The vector output of the decoder stack is passed to a fully connected neural network and a softmax layer to estimate the probability of the words in our vocabulary. We use the Adam optimizer with default settings.

3.4 Convolutional Seq2seq Model

CNNs are feed-forward neural networks that can effectively recognize patterns in data, and are used mainly in vision for image and video recognition, analysis, and classification. More recently they are also used in sequential data analysis such as NLP and time series' analysis. A CNN applies convolving filters to extract local patterns or features. A convolving filter is a fixed size kernel matrix that is used to slide through the input and compute the dot product. The application of CNNs in NLP enables the model to automatically identify n-grams in input and create a semantic representation. Unlike RNNs, CNNs' performance is independent of the input length and allows parallel computation.

CNNs are also used in a seq2seq model that we apply for our query recommendation tasks [15, 40]. In the model's encoder, the input sequence $\boldsymbol{x_i} = (\boldsymbol{x_1}, ..., \boldsymbol{x_n})$ is embedded in a distributional space as a sequence $\boldsymbol{x_i}' = (\boldsymbol{x_1'}, ..., \boldsymbol{x_n'})$ with $\boldsymbol{x_i'} \in \mathbb{R}^m$. The encoder and decoder in CNN seq2seq share a multi-layer structure. Each layer contains a one dimensional convolution followed by a non-linear activation function. Stacking multiple layers allows the model to capture long temporal dependencies in the input. The output of the l-th layer is denoted as $\boldsymbol{h^l} = (\boldsymbol{h_1^l}, ..., \boldsymbol{h_n^l})$ for the decoder, and $\boldsymbol{z^l} = (\boldsymbol{z_1^l}, ..., \boldsymbol{z_n^l})$ for the encoder. The layers in encoder and decoder have a similar structure. Gated linear units (GLU) are used to enable the networks to control which inputs of the current context are relevant.

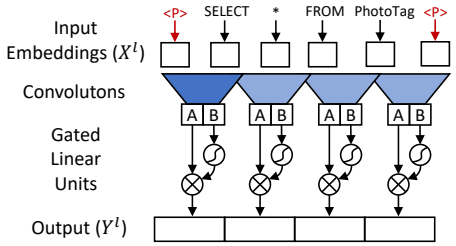


Figure 7: One layer of a CNN Seq2seq Model.

The decoder output $\boldsymbol{h^L}$ in the last layer L is passed through a fully connected layer and a softmax layer to compute the probability distribution to generate the output sequence $\boldsymbol{y_j}$. CNN seq2seq architecture also relies on multi-step attention that applies a separate attention mechanism in each layer of the decoder and improves learning through a normalization strategy (see [15] for more detail).

3.5 Fine-Tuning Seq2Seq Models

We use a fine-tuning technique to build a prediction model for the query template prediction task, which is a classification problem. Fine-tuning is widely used in many computer vision applications [38, 16, 30] and it has been recently adapted for NLP [17]. Computer vision models, including image classification, object detection, semantic segmentation, and image captioning, are rarely trained from scratch and instead they are fine-tuned from general models that are pre-trained using large common datasets to work with a particular dataset. In NLP, this is applied by pre-training a language model (LM) from a large corpus of text, such as Wikitext extracted from Wikipedia, then fine-tuning it for a target dataset, and finally fine-tuning the model for a particular task on the same dataset [17]. In Section 5, we explain the application of fine-tuning for template prediction task.

4. WORKLOADS AND ANALYSIS

We describe the two, real world, large-scale workloads (Section 4.1 and 4.2), we analyze them (Section 4.3), and explain the implications of our analysis on data sampling, model selection, and model evaluation (Section 5).

4.1 SDSS Workload

The Sloan Digital Sky Survey (SDSS) database stores images, spectra, and catalog data for more than three million astronomical objects with 87 tables, 46 views, 467 functions, 21 procedures, and 82 indices [34]. SDSS offers several online data access tools. Its user groups, ranging from high school students to astronomers, have varied user intent and diverse levels of knowledge in astronomy and SQL. SDSS provides a publicly-available workload that is the result of user interactions collected since 2001. We used the archived workload data that spans from 2002 to 2011 and extracted the following information for each SQL query entry:

- The **query**, the raw query statement extracted from the "SqlStatement.statement" attribute.
- The **start time**, from the "SqlLog.theTime", a date-time value that indicates the start time of the query.

- The **session ID**, from the "SessionLog.sessionID" attribute. Each session is a mix of SQL query entries and webhit entries that are sorted first by client IP address and then by time. When the time gap between successive hits from the same client IP passes 30 minutes, a new session is created.
- The **session class label**, extracted through a series of joins described in [46]. There are seven session classes. We use browser sessions, identified as query sessions created by human users [34, 35].

4.2 SQLShare Workload

The SQLShare workload is collected from SQLShare [19], a database-as-a-service platform [19], where human users upload their data and write queries to interact. Its users also range from students to domain experts. It also provides a publicly-available workload, which provides exemplars for query sequences over short-term, user-uploaded datasets in various domains, ranging from biomedical to ocean sciences. We extracted the following information for each query entry:

- The **query**, the raw query statement in the "Query" attribute.
- The **start time**, from the "start_time" attribute.
- The **session ID**, identified using SDSS's definition for query sessions [34, 35]. We sorted data first by the "Owner" attribute and then by time. The value of the "Owner" attribute is an ID that indicates the user who posed the query entry. We created a new session when two successive queries from the same owner are more than 30 minutes apart.

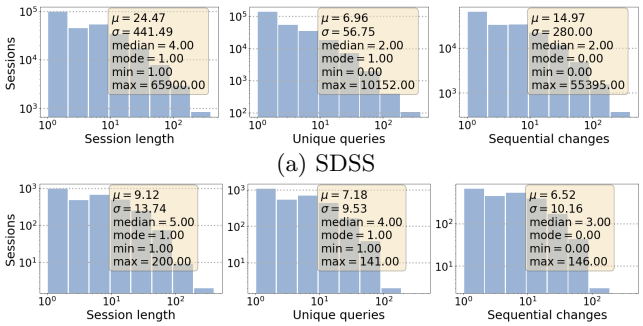
Since users upload their own datasets to SQLShare, the query sessions can operate on their individual datasets and be oblivious to the datasets used in other sessions by other users. This characteristic makes the SQLShare workload a collection of individual workloads with a variety of schemas, rather than one schema used in the SDSS workload. This difference in the SDSS and SQLShare service is reflected in the analysis in Section 4.3 and then leads to the differences in how we handled the workload data in Section 5.1.

4.3 Workload Analysis

We separately analyze the SDSS and SQLShare workload to illustrate the importance of query subsequences and using query statements as a whole in understanding user intent and query recommendation. We show the session statistics (Section 4.3.1), explore sequential changes in syntactic features (Section 4.3.2) and query templates (Section 4.3.3). Since we focus on query fragments and templates, we replaced the numeric literals with a <NUM> token.

4.3.1 User Session Analysis.

We group the workload data by session and extract the following statistics to understand how much queries vary in a session: Sessions length is the number of queries in a session, Unique queries is the number of unique queries in a session, and Sequential changes is the number of times a successive query differs from the one preceding it in a session. This value reflects the scenarios where users switch back and forth between queries, e.g., only three unique queries but six consecutive changes in a session.



(a) SDSS

(b) SQLShare

Figure 8: Session statistics

The SDSS workload has close to 18 million sessions, 932 million webhits, and 194 million SQL entries in total across seven classes. Browser sessions are generated by human users [34, 35]. There are 267,577 browser sessions with 6.3 million SQL entries. 76.8% of the sessions have more than one SQL entry, and 69.6% have more than one unique query. The SQLShare workload has 2,930 sessions with 26,727 queries. 79.9% of the sessions have more than one query, 77.3% have more then one unique query, and approximately 64% have more than one sequential change.

Our analysis of the SDSS and SQLShare session statistics (Figure 8) shows the frequent changes in queries, and majority of the human users post a variety of queries in sequence to achieve session intent. Although SDSS and SQLShare have similar session statistics in median, mode, and minimum, on average, SDSS has longer sessions but fewer unique queries per session. One possible reason derives from the use of the SDSS service. For instance, SDSS users often repeatedly pose the same query statement with changes in numeric literals to browse different images. The SDSS sessions' standard deviation and maximum values of all three session properties are much higher than SQLShare. One possible explanation lays in the complexity of the schema in the workloads. Since the SDSS workload has 87 tables and 46 views, SDSS users have access to more than three million astronomical objects to explore and thus, tend to pose more queries in a session, compared to the SQLShare users who interact with user-uploaded datasets. These observations suggest user needs for the assistance to formulate queries, especially for users who are not familiar with the SQL language and the database.

4.3.2 Query Subsequence Analysis

Our work aims to use query subsequences defined in Definition 3 to capture the sequential changes in query fragments and template. We analyze query syntactic properties to understand the changes in query subsequences. We used the ANTLR parser to generate the Abstract Syntax Trees (AST) of queries and extracted six syntactic properties [46]: Word count, Function count, Table count, Predicate count, and Selected columns are subsequently the numbers of words, functions, unique tables, predicates, and selected columns in a query, and Predicate column refers to the number of table names in the predicates of a query.

EXAMPLE 6. *For Q in Figure 3,* Word count *= 36,* Function count *= 1,* Table count *= 3,* Predicate count *= 2* (NOT IN, LIKE), Selected columns *= 5 (Jobs.target, Jobs.estimate,*

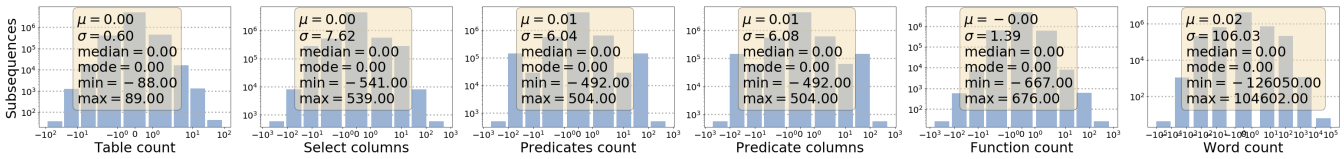


Figure 9: Changes in the syntactic properties of SDSS query subsequences.

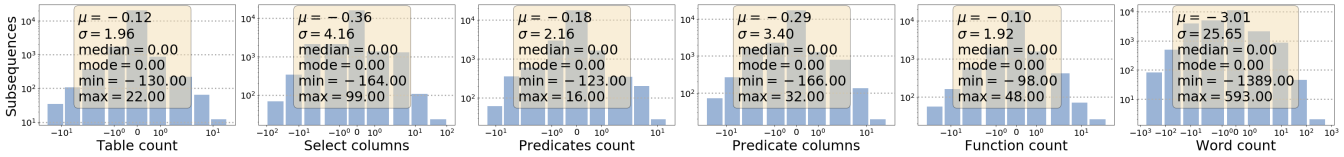


Figure 10: Changes in the syntactic properties of SQLShare query subsequences.

Servers.target, Servers.queue, Jobs.outputtype), and Predicate column *= 1 (Servers).* □

To capture the sequential changes in query statements, we used the differences in the six syntactic features between two consecutive queries of a session. We obtained query subsequences as the following. We first grouped queries by session ID, sorted queries by the start time, and dropped the sessions with only one query entry. In each pair, the two queries are from the same session and are consecutive based on the start time, and the first query has an earlier start time than the second. For every pair, we took the difference between the two queries in the six syntactic features.

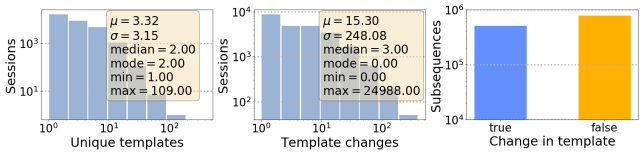
We obtained query subsequences from the SDSS and SQLShare workload separately. From our analysis on the SDSS workload (Figure 9), out of 6 million subsequences, 7.69% use more tables in their second queries, 13.87% select more attributes, 13.52% use more predicates, 13.65% include more tables in the predicates, 10.38% use more functions, and 16.26% become longer. The percentage is similar for the subsequences that decrease in our syntactic measurements.

In the SQLShare workload (Figure 10), out of 23,797 query subsequences, 4.83% increase their table count in the second query, 11.87% select more attributes, 9.22% use more predicates, 9.45% include more tables in their predicates, 8.82% use more functions, and 12.71% get longer. Compared to SDSS, queries in the SQLShare workload sequentially become less complex based on our syntactic properties. 9.45% of the subsequences use fewer tables in their second query, 19.86% select fewer attributes, 16.71% use fewer predicates, 17.78% include fewer tables in the predicates, 11.82% use fewer functions, and 40.32% become shorter. A possible implication is that SQLShare users tend to pose more targeted and simplified queries as they interact with the application.

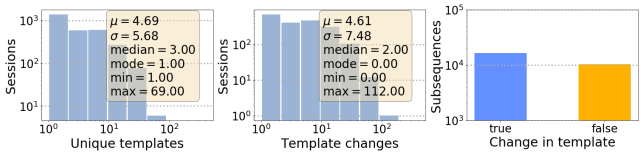
Compared to SQLShare, the sequential changes in the syntactic properties of the SDSS workload have a more dramatic value range and standard deviation in the number of selected attributes, predicates, tables in the predicates, and word count. We attribute that to the fundamental differences in the applications. For instance, SDSS database has 87 tables and 46 views available to users, while the datasets uploaded to SQLShare by human users would likely be simpler. Hence SDSS users would have a greater variety of query fragments when formulating queries.

4.3.3 *Query Template Analysis.*

Our work aims to help users write queries by recommending query templates defined in Definition 4. Query templates



(a) SDSS



(b) SQLShare

Figure 11: Template statistics

consist of SQL keywords and reflect the SQL structure of query statements. Given a query statement Q, we obtained $template(Q)$ by replacing tables, attributes, functions, and literals with the corresponding tokens. In this analysis, we used the whole SQLShare workload and randomly sampled 25% of the sessions in the SDSS workload and analyzed templates at the session and subsequence level to assess user needs for query structural information: Unique templates is the number of unique templates in a session, Template changes is the number of sequential changes in query templates within a session, and Change in template is whether a query subsequence has two distinct templates.

Sessions use a variety of query templates (Figure 11). In the SDSS workload, 78.6% of sessions use more than one unique template, and 64.2% of sessions change query templates more than once. The number is 68.3% and 55% respectively for the SQLShare workload. We also assessed the change in templates in query subsequences. 39.5% of the SDSS query sequences have two unique templates, while the number is 61.5% for SQLShare. Overall, the SDSS sessions have fewer unique templates but more sequential changes in templates than the SQLShare sessions. Although SQLShare sessions are overall shorter, they use more unique templates. These observations align with the session analysis, where we show that SDSS sessions are longer but with less variety in query statements, which can be explained by the differences in the workloads discussed in Section 4.3.1.

5. MODELLING WORKFLOW

We used the workload analysis to guide our data preprocessing (Section 5.1), model selection (Section 5.2), train-

ing (Section 5.3), and evaluation (Section 5.4). In this section, we address the unique challenges that comes with our sequence-aware query recommendation problem.

5.1 Data Preprocessing

We took the following steps to construct datasets from the SDSS and SQLShare workload for building the models. First, to maximize the seq2seq model performance, we processed query statements for the models to easily learn the semantic meaning and construct sensible predicted queries. Since our approach uses model-generated queries, the models operate at the word level to obtain a collection of sensible tokens as vocabulary.

Second, we extracted query subsequences from query sessions to allow our models to capture the sequential changes in query statements. Similarly to Section 4.3.2, we first sorted SQL query entries by their session ID and start time and dropped sessions with only one query entry. Since we focus on capturing the changes, for each session, we then dropped the consecutive query statement duplicates. We obtained 3.8 million SQL query entries and 289,370 unique query statements from the SDSS workload and 22,719 query entries and 21,625 unique statements from SQLShare. We obtained 3.6 million subsequences from the SDSS workload and 20,773 subsequences from the SQLShare workload.

From our SDSS workload analysis, we found redundant query statements within and across sessions, leading to duplicates in query subsequences. In training, the models would give more weight to the query statements that appear more often. Although it is unlikely that a query that appears ten more times carries more information about query semantics, we keep the duplicated subsequences in our datasets to allow the models to focus on more popular query statements for our prediction tasks. We randomly sampled 25% of the SDSS query sessions to obtain a manageable subset. Our final dataset extracted from the SDSS workload has 896,861 query subsequences with 46,336 unique query statements. Finally, we assigned a template class to each query statement for our template prediction problem. The extracted SDSS dataset has 17,007 unique templates, while SQLShare has 5,229 unique templates.

5.2 Model Selection

We model query recommendation as a query prediction problem for a given query. A closely related field to handling SQL queries is NLP. Our problem is similar to a text prediction problem, where interactive language models such as chatbots are trained with subsequences of phrases to respond to user input. Instead of traditional NLP models that rely on feature engineering, we use deep learning models that automatically exploit the intrinsic features in data.

We assessed three deep learning architectures described in Section 3. RNNs recognize text as a sequence of tokens, and GRUs help to capture dependencies between tokens in long sequences using a gating mechanism. However, their performance diminishes as sequences elongate, and their recurring structure leads to long training time. Thus we adapted transformers that capture the relatedness between tokens regardless of the distance between tokens using an attention mechanism. Rather than taking in tokens one-by-one, the transformer reads tokens in-parallel. CNNs are another family of deep learning architectures that are known to be competitive in NLP. They are feed-forward neural networks that automatically identify local patterns such as n-grams.

However, SQL queries are also different from natural languages. SQL queries follow a certain grammar depending on the DBMS application and have syntactic errors if the grammar is violated. The direct, model-generated queries from seq2seq models can contain syntactic errors. Recommending whole queries may not be helpful since queries can also be complex and hard for human users to understand and modify, especially when the user is not familiar with the schema and syntax. Hence to make our query prediction problem tractable and simplify the recommended information for users to understand and use, we separated our query prediction problem into query fragment prediction and template prediction.

5.3 Model Training

We aim to justify the importance of query subsequences and deep learning architectures in query recommendation. To isolate the effect of our sequence-aware approach, we built two sets of models with different tasks using the same neutral network architectures. The difference lays in how the query subsequences in the training set are constructed.

We trained the sequence-aware (seq-aware) models with a query prediction task. Given a query subsequence $\langle Q_i, Q_{i+1} \rangle$ in the training set, we used Q_i as the model input sequence and Q_{i+1} as the target sequence. We train the sequence-blind (seq-blind) models with a reconstruction task. Given a query subsequence $\langle Q_i, Q_i \rangle$ in the training set, we used Q_i as the model input sequence and Q_i as the target sequence. In training, we first built the vocabulary as a collection of word tokens from the training set. We used the standard cross-entropy loss in training. Cross-entropy loss decreases as the predicted probability converges to the actual token for a given step. The application of the trained models for query fragment prediction has two steps (cf Figure 12). First, we input to the models the last query Q_i^* in the current user session, and we receive a output query Q_{i+1}^*. Second, we parse Q_{i+1}^* and report its fragment as the prediction results.

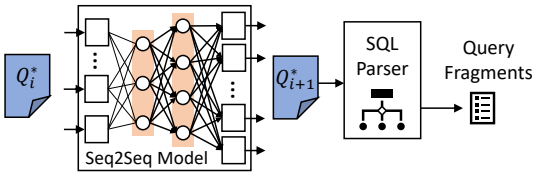


Figure 12: Query fragment prediction.

We built classification models described in Section 3.5 for template prediction. The seq-aware classification models were trained with $\langle Q_i, template(Q_{i+1}) \rangle$, while the seq-blind classification models were trained with $\langle Q_i, template(Q_i) \rangle$. In addition, we aim to justify the efficacy of fine-tuning in template prediction. Figure 13 shows the overview of fine-tuning a classification model for our template prediction task. To apply fine-tuning, we used the trained encoder from the seq2seq models and augmented the classification model. The trained encoder has learned to extract features from query statements that are relevant to query prediction and is used to turn the input query into a vector representation. In comparison to the augmented classification models, we trained the classification models alone using both seq-aware and seq-blind training set. We also used cross-entropy loss.

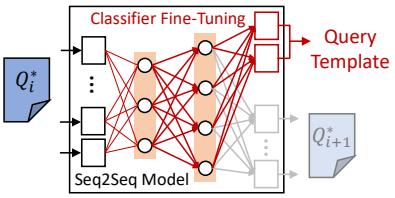


Figure 13: Query template prediction.

5.4 Model Evaluation

Our work aims to help users write queries by recommending query fragments and templates as Definition 4. In evaluation, we parsed the model-generated next query to obtained the predicted fragments and string literals. Since all the queries in the SDSS workload operate on the same schema that is complete and publicly available, we were able to separate the tables, attributes, and functions by parsing the queries and then checking against the schema. However, since only selected schemas used in the SQLShare workload are available, we combined the fragments in the evaluation for fragment prediction using the SQLShare workload.

To assess the performance of deep learning models, We compared the seq2seq models to the binary fragment-based collaborative filtering approach employed by the QueRIE framework [13, 1]. The QueRIE framework first turns queries into vectors. Given Q_i, this approach constructs vectors based on the tables and attributes used in Q_i. It then recommends historical queries in the workload that are the closest to the user input query using cosine similarity. Since it requires access to the schema, we were able to assess this approach using the SDSS workload but not SQLShare.

We aim to help users with limited experience explore without overwhelming them with less relevant information. Following [13, 1, 46], we use different k values to further assess performance based on the fragments and templates of the top-k recommended queries. We use beam search decoding to obtain the top-k queries from the seq2seq models. Beam search is a decoding method generates top-B candidate output sequences with the highest score at each step; where B is known as the beam width [43, 27, 42]. At each step, beam search considers all possible tokens and selects the B most likely extensions. The search is repeated until a limited time or until all top-B sequences terminate. This allows for multiple sequences to be explored in parallel instead of a single sequence in greedy decoding.

In our case, beam width B is equal to k. In particular, we use beam search in fragment prediction task to find k next queries with highest score. We then extract the query fragments and return to the user. An alternative to beam search is to use the probabilities returned by the softmax function at each step of a greedy decoder to find top-k fragments. Our experimental evaluation shows that applying beam search outperforms this alternative approach, specially for larger values of k. This is because the softmax function often returns higher probabilities for a few top tokens and the probabilities drop for the remaining top k tokens. This suggests that probabilities at each step do not provide significant support for recommending top-k fragments.

6. EXPERIMENTAL EVALUATION

We evaluate the efficacy of the combination of deep learning models and query subsequences in query recommendation. In particular, the methods take a query Q_i as input and predict the query fragments (Section 6.2) and template (Section 6.3) of the next query Q_{i+1}. Using the SDSS and SQLShare workload data, we compare the performance of two baselines, sequence-aware (seq-aware) models, sequence-blind (seq-blind) models, and binary fragment-based collaborative filtering approach in the QueRIE framework [13, 1]. We acknowledge that the QueRIE framework concerns with a different query recommendation problem.

6.1 Setup

6.1.1 Data Split

We used the SQLShare workload and our extracted SDSS workload to evaluate the model performance on both fragment prediction and template prediction. Our SDSS workload has 896,861 query subsequences with 46,336 unique query statements, while the SQLShare workload has 20,773 subsequences with 21,625 unique statements. For both workloads, we used a (80/10/10) random split for the train, validation, and test sets, respectively.

6.1.2 Methods Compared

Our goal is to show that the seq-aware seq2seq methods outperform existing techniques. Thus we compared the various models from Section 5.3, where seq-aware models are trained with a prediction task using query subsequences, while seq-blind models are trained with a reconstruction task. To assess the effectiveness of deep learning models, we include the binary fragment-based collaborative filtering approach in the QueRIE framework [13, 1]. Additionally, we include two simple baselines for each prediction task. To predict the fragments in Q_{i+1}, the baseline$_1$ uses the most popular fragments, and the baseline$_2$ uses the fragments in Q_i. For template prediction, the baseline$_1$ uses the most popular templates, and the baseline$_2$ uses the template in Q_i. We augmented the trained encoder from the seq2seq models with the fully-connected classification model in Section 3.5. We assessed the performance of the fully-connected classification model to evaluate the efficacy of fine-tuning.

6.1.3 Hyper-Parameter Tuning

We tuned the hyper-parameters based on the seq-aware models for the fragment prediction task using the SDSS workload. Following [46, 41, 15], we constrained the range of the hyper-parameters of each model to make the tuning tractable. For the RNN models, we assessed the number of hidden layers in $\{1, 6\}$ and hidden size in $\{150, 350\}$. For the transformer models, we tested the number of layers in $\{3, 10\}$, hidden size in $\{256, 1028\}$. For the CNN models, we tested the same range for number of layers and hidden size with kernel size in $\{3, 5\}$. We tested the hidden size in $\{300, 2000\}$ for each classification model using the SDSS workload. We tested dropout in $\{0.1, 0.5\}$ for all models. The hyper-parameters are selected based on the best validation loss. We report the performance of the best models and used the same set of hyper-parameter to train the seq-aware models for the SQLShare workload and the seq-blind models for both workloads.

6.1.4 Performance Metrics

For fragment prediction, we report the test average recall and F-measure on the fragments and literals to evaluate

Method		SQLShare				SDSS							
		F-measure		Recall		F-measure				Recall			
	seq	fra	lit	fra	lit	tab	att	fun	lit	tab	att	fun	lit
baseline_1	-	0.1398	0.0744	0.0834	0.0554	0.5687	0.1773	0.5156	0.0529	0.5685	0.1139	0.5159	0.0354
baseline_2	-	0.6052	0.5564	0.6981	0.5002	0.4619	0.5493	0.3796	0.2862	0.5457	0.5499	0.5413	0.2755
QueRIE	-	-	-	-	-	0.5797	0.4641	0.5631	0.0598	0.4988	0.4727	0.4326	0.0556
rnn	×	0.6168	0.5115	**0.7256**	0.4351	0.5734	0.5714	0.5648	0.2669	0.4779	0.5695	0.4042	0.2622
transformer	×	0.5945	0.5652	0.6655	0.4489	0.5869	0.5645	0.5975	**0.2986**	0.4820	0.5611	0.4322	**0.2918**
cnn	×	0.4024	0.3840	0.2162	0.1304	0.5784	0.5604	0.6013	0.2981	0.4750	0.5620	0.4421	0.2907
rnn	✓	0.6183	0.4489	0.5984	0.3296	0.7226	0.6507	**0.6832**	0.1758	0.6335	0.6526	**0.5510**	0.1743
transformer	✓	**0.6482**	**0.6304**	0.6260	**0.5520**	0.7675	**0.6646**	0.5273	0.1895	0.7050	**0.6648**	0.3811	0.1874
cnn	✓	0.3878	0.1966	0.3681	0.1348	**0.7703**	0.6303	0.2083	0.0348	**0.7539**	0.6223	0.1370	0.0337

Table 1: Query fragment prediction using greedy decoding, where fra, lit, tab, att, and fun are fragments, string literals, tables, attributes, and functions respectively. seq shows if the model is seq-aware with ✓ and seq-blind with ×. baseline_1 uses the most popular fragments, whereas baseline_2 uses the fragments in Q_i.

model performance and the test average precision, recall, and F-measure to evaluate performance of the top-k parameter. F-measure is computed by $\frac{2\times\text{Precision}\times\text{Recall}}{\text{Precision}+\text{Recall}}$, where precision is the number of correct fragment predictions over the number of total fragment predictions, and recall is the number of correct fragment predictions over the number of total target fragments. For template prediction, we report the test average accuracy to evaluate model performance and the test average recall for performance varying k.

6.2 Fragment Prediction

6.2.1 Model Performance

Table 1 shows the fragment prediction results. We used greedy decoding described in Section 5.4 to obtain the model-predicted query. For the SQLShare workload, the baseline_2 outperforms the baseline_1 for fragments and literals due to the nature of the SQLShare service. SQLShare users upload datasets and only have access to the uploaded data. Consequently, SQLShare query sessions operate on a variety of datasets with different schemas. In such a setting, it is easy for the baseline_2 to perform well as it has the context of the next query and is oblivious to other schemas in the training set. The seq2seq models have access to all the schemas in the training set and have to learn from the workload to distinguish the schemas and recommend relevant fragments and literals. Although the baseline_2 and seq-blind rnn achieve high recall in fragment prediction, the seq-aware transformer outperforms others with high precision.

For the SDSS workload, since SDSS has the same schema available to all users, queries across all sessions operate on the same set of fragments. The baseline_1 achieves high performance in table and function prediction, while the baseline_2 performs well in attribute and literal prediction as they rely on the context of the query. For table and attribute prediction, all of the seq2seq models improve upon the baselines and the QueRIE framework. The seq-aware cnn performs the best in table prediction, while the seq-aware transformer performs the best in attribute prediction. The seq-aware rnn obtains the best performance in function prediction. In string literal prediction, all the seq-blind models perform similarly to the baseline_2, and the seq-blind transformer achieves the best performance.

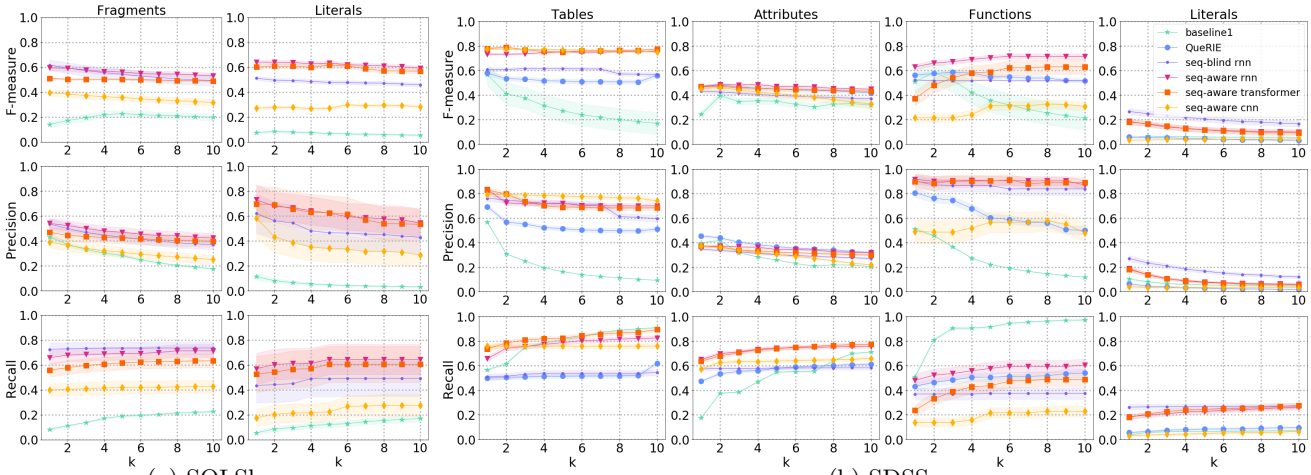
6.2.2 Evaluation of the Top-k Parameter

We increased the number of recommended queries k to evaluate the methods as users request more fragments (Figure 14). We varied k in $\{1, 16\}$ and found that the trend remains unchanged when $k > 10$. We used beam search decoding described in Section 5.4 to obtain the top k model-generated queries from the seq2seq models. We found that the seq-blind rnn outperforms other seq-blind models. Thus to simplify Figure 14, we reported the performance of the baseline_1, seq-blind rnn, seq-aware rnn, seq-aware transformer, and seq-aware cnn. Figure 14a shows the results on the SQLShare workload. All the seq2seq models outperform the baseline_1. In fragment prediction, although the seq-blind rnn achieves high recall, the seq-aware rnn and transformer perform better in precision and outperform others as k increases. They also outperform the seq-blind rnn by 0.1 for all measures in literal prediction.

Figure 14b shows the results on the SDSS workload, where a shortlist of tables and functions are very popular among users. In table, attribute, and function prediction, although the baseline_1 obtains high recall, all the seq2seq models outperform the baseline in precision and F-measure. All the seq-aware models outperform the QueRIE framework in table, attribute, and literal prediction. Specifically, all the seq-aware models achieve and maintain high performance in table prediction. In attribute prediction, the seq-aware rnn and transformer preform the best as k increases. The seq-aware rnn also maintains high precision in function prediction and performs the best overall. In literal prediction, all the seq2seq models except CNNs outperform the baseline_1, and the seq-blind rnn performs the best.

6.2.3 Discussion

We observed the following: (1) CNNs perform better on the SDSS workload than on SQLShare. Since a number of different schemas are used in the SQLShare workload, the query statements consist of more diverse vocabulary and n-grams. This characteristic makes it more difficult for the CNN architecture to find local patterns and predict fragments. (2) CNNs outperform others by far in table prediction on the SDSS workload. The SDSS workload queries share the schema, and tables often appear as a group in the WHERE clause. For these reasons, CNNs are more likely to capture tables. (3) The baseline_2 and seq-blind rnn and transformer have similar results on both workloads. For SDSS, the seq-blind models outperform the baseline_2 as the



(a) SQLShare (b) SDSS

Figure 14: Fragment prediction performance using beam search decoding as k increases. k is the number of queries recommended by the methods. baseline$_1$ uses the most popular fragments. The shadow shows the 95% confidence interval.

Method	seq	SQLShare	SDSS
baseline$_1$	-	0.1146	0.3323
baseline$_2$	-	0.4531	0.6156
QueRIE	-	-	0.4154
fully-connected	×	0.3975	0.3185
rnn	×	0.4549	0.3269
transformer	×	0.4497	0.3397
cnn	×	0.1727	0.3173
fully-connected	✓	0.4659	0.6949
rnn	✓	0.4739	0.7299
transformer	✓	**0.5100**	**0.7395**
cnn	✓	0.2169	0.6687

Table 2: Query template prediction accuracy. seq shows if the model is seq-aware with ✓ and seq-blind with ×. baseline$_1$ uses the most popular template, whereas baseline$_2$ uses the template of Q_i.

seq-blind models can learn from other queries used for the same schema. For SQLShare, it is more challenging for the seq-blind models to generalize the learning. (4) Varying k, the seq-blind rnn and transformer perform similarly to the QueRIE framework. Both approaches use query similarity to make recommendations, but the seq-blind models use whole queries instead of the hand-picked features in the QueRIE framework. (5) Overall, in table, attribute, and function prediction, the seq-aware transformer outperforms others. Given long queries, the transformer architecture can relate the word tokens in a query regardless of how far they are apart from each other. This characteristic enables the transformer models to learn and preserve information better in comparison to the RNN-GRU architecture. (6) Regarding the generalization of model performance on different workloads, the seq-aware transformer is able to apply the attention mechanism and recognize the relatedness among individual word tokens in a query.

6.3 Template Prediction

6.3.1 *Model Performance*

Table 2 shows the template prediction accuracy of the models. The baseline$_2$ performs better than the baseline$_1$ for both workloads. For the SQLShare workload, the seq-blind rnn and transformer perform similar to baseline$_2$. Except for the CNN model, all the seq-aware models improve upon the baseline$_2$. For the same architecture, the seq-aware models also outperform their seq-blind counterparts. For SDSS, all the seq-aware models drastically improve upon the baseline$_2$. For both workloads, the seq-aware rnn and transformer consistently perform well with the seq-aware transformer achieving the best accuracy. We acknowledge that the QueRIE framework does not consider SQL structure.

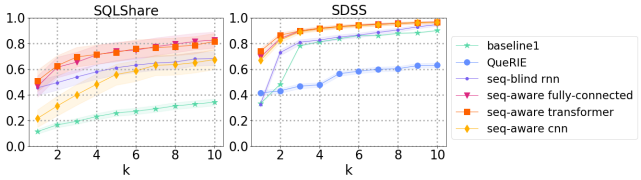


Figure 15: Template prediction recall as k increases. The shadow shows the 95% confidence interval. baseline$_1$ uses the most popular templates.

6.3.2 *Evaluation of the Top-k Parameter*

Figure 15 shows model recall with k in {1, 10}, where k is the number of recommended templates. We evaluated the quality of the recommended templates as its number increases. We included the baseline$_1$, seq-blind rnn, seq-aware fully-connected, seq-aware transformer, and seq-aware cnn. For both workloads, the RNN and transformer architecture achieve the best recall and perform similar in both seq-blind and seq-aware models. For the SQLShare workload, the best recall is obtained by all the seq-aware models except the seq-aware cnn. When $k = 1$, the difference between the recall of the seq-aware and seq-blind models is less than 0.06. However, the seq-aware models improve by over 0.1 at $k = 2$ and are able to keep improving at a steady rate from there. For SDSS, all the seq-aware models outperforms others by far when $k = 1$. The seq-blind rnn and transformer are able to pick up from $k = 2$. The difference in recall becomes smaller as k increases. All of the seq-blind and seq-aware models consistently perform better than the baseline$_1$.

6.3.3 Discussion

We found the following: (1) Although Table 2 shows that for the SDSS workload, the baseline$_2$ outperforms all the seq-blind models at $k = 1$, the seq-blind models achieve a recall of over 0.7 at $k = 2$. The seq-blind models make template predictions based on syntactic similarity. This suggests that the seq-blind models are able to recognize syntactically similar queries in the workloads and recommend relevant templates. (2) Similar to its performance in fragment prediction, the CNN architecture performs better on the SDSS workload than on SQLShare. Again, the additional complexity in the SQLShare workload makes it difficult for CNNs to identify local patterns at the word level. (3) For both workloads, the difference between the recall of the seq-aware transformer and the seq-aware fully-connected model becomes smaller as k increases. This suggests that fine-tuning is effective when users request a small set of template recommendation, e.g., $k < 4$.

In summary, the seq-aware seq2seq models achieve the best performance in both query fragment prediction and query template prediction. Our results show the effectiveness of the combined use of deep learning architectures and query subsequences in query recommendation.

7. RELATED WORK

We review some of the related work on query composition, representation learning, and recommendation systems.

7.1 Facilitating Query Composition

Query recommendation is one way to help users write queries. There are many approaches. SnipSuggest [22] models each query as a vertex and uses query popularity to build a directed acyclic graph (DAG); SQLSugg [14] generates undirected graphs as queryable templates; QueRIE [13] uses collaborative filtering to make recommendations based on summarized query sessions. These existing methods use selected features, e.g., tables and attributes, to model query statements, and ignore query sequences in sessions.

The work in [2] uses collaborative filtering to recommend sets of queries as OLAP sessions based on previous sessions. Its treatment of whole sessions instead of individual queries means that the full sequence of queries is taken into account. Sessions are extracted from workloads, ranked based on similarity, and fitted to create a session that most resembles the user's future steps in the current session. [31] provides a framework for recommending next steps in a data analysis context, where sessions are modelled as trees. Similar past sessions are compared using a tree edit distance metric. Abstract generalized actions are recommended instead of concrete actions, this is similar in concept to the templates we recommend. These two works are similar in terms of goals but used in different contexts so it is difficult to compare.

Recent work has been using deep learning techniques for improvements in facilitating query composition from aspects other than query recommendation. In [46] character-level and word-level CNNs and LSTMs are used to predict query performance to guide users write more efficient queries, while [26] shows query tree visualization to help users understand SQL fragments to formulate queries. While these solutions apply similar techniques to ours, they facilitate query composition from different perspectives.

7.2 Sequence Representation Learning

Neural networks are effective tools for learning the underlying explanatory factors and useful representation from data for sequence modeling task such as machine translation and natural language understanding [39, 15, 41, 17]. Using neural network for representation learning has two main advantages. First, it reduces the cost of feature engineering and enables automatic feature learning from raw data in supervised or unsupervised approaches. Second, it allows models to work with heterogeneous data as they do not rely on hand-picked features. These advantages are important in our query recommendation problem. Feature engineering for queries is a challenging task and is not always applicable to different query workloads. [18, 20] employ LSTM autoencoder to express queries in a standardized form for query composition tasks such as error prediction and query similarity comparison. Our work differentiates as discussed in Section 3.1. We use seq2seq models to capture query sequences for effective query recommendation.

7.3 Deep Learning in Recommender Systems

Recommender systems estimate users' preferences and interests to recommend them favorable items. These systems apply three types of strategies [36]. Content-based systems use items' properties and users' information to match items and users. In collaborative filtering, the recommendations are made by learning from the past user-item interactions, e.g. the history of visited or liked items. Hybrid systems apply a combination of the two strategies.

Recently, plenty of research has been done on deep learning in recommender systems. [10] applies neural networks to recommend videos on YouTube; [7] uses deep learning models to recommend application on GooglePlay; [33] presents an RNN-based system to recommend Yahoo news (see [45] for a related survey). Deep learning has driven a revolution in recommender systems and the systems that use deep learning have shown significant improvements over traditional recommender systems. Our work is the first to use deep learning in a recommender system for SQL queries.

8. CONCLUSION AND FUTURE WORK

We introduced a sequence-aware deep learning query recommendation approach that addresses two main weaknesses of the existing query recommendation systems, namely ignoring the sequential data in query sessions and manual feature selection to compare queries. We applied and compared three main seq2seq models for fragment prediction and tuned them for template prediction. We evaluated the models on two real-word workloads, SDSS and SQLShare, and show major improvements over the existing methods for query recommendation and prediction.

Many steps can be taken to extend our work. We could incorporate query result information to customize recommendations. This semantic search in query workloads based on query intentions [5]. Automatic feature selection in these models make it easier to apply the models in heterogeneous settings. A possible research direction is to apply and assess the models in this paper with workloads of different query languages. We also plan to consider a setting where the models are trained on a workload over a database and transferred to make recommendation on a different database. This requires transfer learning and tuning the models to make effective recommendations.

9. REFERENCES

[1] J. Akbarnejad, G. Chatzopoulou, M. Eirinaki, S. Koshy, S. Mittal, D. On, N. Polyzotis, and J. S. V. Varman. SQL querie recommendations. *PVLDB*, 3(2):1597–1600, 2010.

[2] J. Aligon, E. Gallinucci, M. Golfarelli, P. Marcel, and S. Rizzi. A collaborative filtering approach for recommending OLAP sessions. *Decision Support Systems*, 69:20–30, 2015.

[3] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[4] J. Bastings. The annotated encoder-decoder with attention, 2018.

[5] G. Borodin and Y. Kanza. Search-by-example over sql repositories using structural and intent-driven similarity. *DKE*, 128:101811, 2020.

[6] U. Cetintemel, M. Cherniack, J. DeBrabant, Y. Diao, K. Dimitriadou, A. Kalinin, O. Papaemmanouil, and S. B. Zdonik. Query steering for interactive data exploration. In *CIDR*, 2013.

[7] H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, R. Anil, Z. Haque, L. Hong, V. Jain, X. Liu, and H. Shah. Wide and deep learning for recommender systems. In *DLRS*, page 7–10, 2016.

[8] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[9] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[10] P. Covington, J. Adams, and E. Sargin. Deep neural networks for youtube recommendations. In *RecSys*, page 191–198, 2016.

[11] A. M. Dai and Q. V. Le. Semi-supervised sequence learning. In *NIPS*, pages 3079–3087, 2015.

[12] S. Edunov, A. Baevski, and M. Auli. Pre-trained language model representations for language generation. *arXiv preprint arXiv:1903.09722*, 2019.

[13] M. Eirinaki, S. Abraham, N. Polyzotis, and N. Shaikh. QueRIE: Collaborative database exploration. *TKDE*, 26(7):1778–1790, 2014.

[14] J. Fan, G. Li, and L. Zhou. Interactive sql query suggestion: Making databases user-friendly. In *ICDE*, pages 351–362, 2011.

[15] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin. Convolutional sequence to sequence learning. *arXiv preprint arXiv:1705.03122*, 2017.

[16] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.

[17] J. Howard and S. Ruder. Universal language model fine-tuning for text classification. In *ACL*, pages 328–339, 2018.

[18] S. Jain, B. Howe, J. Yan, and T. Cruanes. Query2vec: An evaluation of nlp techniques for generalized workload analytics. *arXiv preprint arXiv:1801.05613*, 2018.

[19] S. Jain, D. Moritz, D. Halperin, B. Howe, and E. Lazowska. SQLShare: Results from a multi-year sql-as-a-service experiment. In *SIGMOD*, page 281–293, 2016.

[20] S. Jain, J. Yan, T. Cruane, and B. Howe. Database-agnostic workload management, 2018.

[21] N. Khoussainova, M. Balazinska, W. Gatterbauer, Y. Kwon, and D. Suciu. A case for A collaborative query management system. In *CIDR*, 2009.

[22] N. Khoussainova, Y. Kwon, M. Balazinska, and D. Suciu. Snipsuggest: Context-aware autocompletion for sql. *PVLDB*, 4(1):22–33, 2010.

[23] R. Kiros, Y. Zhu, R. Salakhutdinov, R. S. Zemel, R. Urtasun, A. Torralba, and S. Fidler. Skip-thought vectors. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *NIPS*, pages 3294–3302, 2015.

[24] G. Klein, Y. Kim, Y. Deng, J. Senellart, and A. M. Rush. Opennmt: Open-source toolkit for neural machine translation. In *Proc. ACL*, 2017.

[25] Q. Le and T. Mikolov. Distributed representations of sentences and documents. In *ICML*, pages 1188–1196, 2014.

[26] F. Li and H. V. Jagadish. Constructing an interactive natural language interface for relational databases. *PVLDB*, 8(1):73–84, Sept. 2014.

[27] J. Li and D. Jurafsky. Mutual information and diverse decoding improve neural machine translation. *arXiv preprint arXiv:1601.00372*, 2016.

[28] J. Li, T. Luong, and D. Jurafsky. A hierarchical neural autoencoder for paragraphs and documents. In *ACL*, pages 1106–1115, 2015.

[29] Y. Liu and M. Lapata. Text summarization with pretrained encoders. *arXiv preprint arXiv:1908.08345*, 2019.

[30] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, pages 3431–3440, 2015.

[31] T. Milo and A. Somech. Next-step suggestions for modern interactive data analysis platforms. In *SIGMOD*, pages 576–585, 2018.

[32] K. Pichotta and R. J. Mooney. Using sentence-level lstm language models for script inference. *arXiv preprint arXiv:1604.02993*, 2016.

[33] M. Quadrana, A. Karatzoglou, B. Hidasi, and P. Cremonesi. Personalizing session-based recommendations with hierarchical recurrent neural networks. In *RecSys*, page 130–137, 2017.

[34] M. J. Raddick, A. R. Thakar, A. S. Szalay, and R. D. C. Santos. Ten years of skyserver i: Tracking web and sql e-science usage. *Computing in Science Engineering*, 16(4):22–31, 2014.

[35] M. J. Raddick, A. R. Thakar, A. S. Szalay, and R. D. C. Santos. Ten years of skyserver ii: How astronomers and the public have embraced e-science. *Computing in Science Engineering*, 16(4):32–40, 2014.

[36] F. Ricci, L. Rokach, and B. Shapira. Recommender systems: Introduction and challenges. *Recommender Systems Handbook*, page 1, 2015.

[37] I. V. Serban, A. Sordoni, Y. Bengio, A. Courville, and J. Pineau. Building end-to-end dialogue systems using

generative hierarchical neural network models, 2015.

[38] A. Sharif Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *CVPR workshops*, pages 806–813, 2014.

[39] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks, 2014.

[40] B. Trevett. Convolutional sequence to sequence learning, 2018.

[41] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. 2017.

[42] T. Wang, D. J. Wu, A. Coates, and A. Y. Ng. End-to-end text recognition with convolutional neural networks. In *ICPR*, pages 3304–3308, 2012.

[43] S. Wiseman and A. M. Rush. Sequence-to-sequence learning as beam-search optimization. In *EMNLP*, pages 1296–1306, 2016.

[44] A. Xu, Z. Liu, Y. Guo, V. Sinha, and R. Akkiraju. A new chatbot for customer service on social media. In *CHI*, page 3506–3510, 2017.

[45] S. Zhang, L. Yao, A. Sun, and Y. Tay. Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys*, 52(1):1–38, 2019.

[46] Z. Zolaktaf, M. Milani, and R. Pottinger. Facilitating sql query composition and analysis. In *SIGMOD*, 2020.