

**Workload-Aware SQL Query Recommendation Using
Retrieval-Augmented Generation**

by

Ehsan Soltan Aghai

B.Sc. Computer Engineering, Sharif University of Technology, 2020

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL
STUDIES

(Computer Science)

The University of British Columbia

(Vancouver)

April 2025

© Ehsan Soltan Aghai, 2025

The following individuals certify that they have read, and recommend to the Faculty of Graduate and Postdoctoral Studies for acceptance, the thesis entitled:

Workload-Aware SQL Query Recommendation Using Retrieval-Augmented Generation

submitted by **Ehsan Soltan Aghai** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Science**.

Examining Committee:

Rachel Pottinger, Professor, Computer Science, UBC
Supervisor

Raymond Ng, Professor, Computer Science, UBC
Supervisory Committee Member

Abstract

Writing effective SQL queries remains a major obstacle for non-expert users who need to explore and analyze data. While recent advances in deep learning have enabled limited forms of SQL query recommendation, existing systems typically focus on predicting partial query structures or rely on schema-specific features. In this thesis, we present a retrieval-augmented generation (RAG) framework for recommending full SQL queries based solely on previous user queries in a session. At the core of our approach is a transition-aware dual encoder trained to retrieve the most likely next query template by capturing both semantic similarity and structural transitions across queries. This retrieval is followed by a language model that generates the full SQL query, conditioned on the retrieved template, recent query history. Our method requires no access to the database schema and adapts naturally to evolving workloads. Compared to traditional rule-based or collaborative recommendation systems, it offers a more flexible and interpretable solution that models user behavior over time. Empirical results show that our system produces contextually relevant queries, improving usability for users with limited SQL experience.

Lay Summary

Writing SQL queries to retrieve data from databases can be difficult for users without technical training. This thesis presents a system that helps users compose complete and accurate SQL queries by learning from how others have written queries in the past. It uses a two-step approach: first, it finds examples of similar queries from previous sessions. Then, it generates a new query based on those examples and the user's recent actions. Unlike traditional tools that rely on fixed templates or database-specific rules, this system adapts to changing user needs and does not require knowledge of the database structure. The result is a more flexible and user-friendly way to support data exploration, especially for non-expert users. Experiments show that the system produces relevant queries, helping users find the information they need more effectively.

Preface

This thesis is original, unpublished, independent work by the author, Ehsan Soltan Aghai.

Throughout my thesis writing, and with the approval of my supervisor, I utilized Grammarly to offer suggestions for grammar, style, and coherence. Additionally, I utilized ChatGPT as a tool to help me clarify generic complex concepts. While the GenAI tools served as a valuable supplementary resource, all final interpretations, conclusions, and analyses remain my own, ensuring that the work reflects my original thought process and academic integrity.

Furthermore, GenAI was also employed as part of the methodology to generate full SQL queries within a retrieval-augmented generation framework. This technical use of GenAI is fully documented and discussed in Chapter 5 of the thesis. I decided to use OpenAI, as a leading GenAI service provider, to support this GenAI-based experimental analysis, which is further detailed in Chapter 6 .

Table of Contents

Abstract	iii
Lay Summary	iv
Preface	v
Table of Contents	vi
List of Tables	ix
List of Figures	x
Acknowledgments	xi
1 Introduction	1
2 Problem Formulation	4
3 Preliminary	6
3.1 Sequence Models for Query Understanding	6
3.1.1 Recurrent Neural Networks (RNNs)	6
3.1.2 Long Short-Term Memory (LSTM)	8
3.2 Transformer Architecture	8
3.3 Encoder-Decoder Models for Sequence Prediction	9
3.4 Large Language Models (LLMs)	9
3.4.1 BERT	10

3.4.2	BART	10
3.4.3	MiniLM	10
3.4.4	GPT	11
3.5	Model Adaptation Techniques	11
3.5.1	Fine-Tuning	11
3.5.2	Few-Shot Learning	11
3.5.3	Retrieval-Augmented Generation (RAG)	12
3.6	Summary	13
4	Related Work	14
4.1	Natural Language Query Recommendation	15
4.1.1	Problem	15
4.1.2	Approach	16
4.1.3	Data-driven Techniques Review	16
4.2	SQL Query Recommendation	21
4.3	Text-to-SQL Systems	23
4.3.1	Model Architecture Innovations	23
4.3.2	Retrieval-Augmented Generation	25
5	Methodology	27
5.1	Key Definitions	29
5.2	Overview of the RAG Framework	31
5.3	Query Template Retrieval	32
5.3.1	Template Abstraction	32
5.3.2	Dual Encoder Architecture	33
5.3.3	Auxiliary Transition Classifiers	34
5.3.4	Vector Database Indexing and Retrieval	37
5.3.5	Transition-Aware Reranking	38
5.3.6	Full SQL Query Generation	39
6	Experimental Analysis	41
6.1	Experimental Setup	41
6.1.1	Dataset: SQLShare Workload	41
6.1.2	Baseline Methods	43

6.1.3	Hyperparameters	44
6.2	Empirical Evaluation	45
6.2.1	Evaluation Metrics	46
6.2.2	Results and Analysis	46
7	Conclusion and Future Work	50
	Bibliography	52

List of Tables

Table 6.1	Key Hyperparameters for the Transition-Aware RAG Model . .	45
Table 6.2	Template Match Accuracy	47
Table 6.3	Fragment F ₁ Scores by Fragment Type	48
Table 6.4	Normalized Average Edit Distance (lower is better)	49

List of Figures

Figure 3.1	RNN Model Architecture [29].	7
Figure 3.2	LSTM Model Architecture [28, 29]	8
Figure 3.3	A general (hypothetical) Retrieval-Augmented Generation (RAG) framework for SQL query recommendation.	12

Acknowledgments

This thesis would not have been possible without the support of many individuals.

I would like to express my deepest gratitude to my supervisor, Prof. Rachel Pottinger, for her incredible support, guidance, and patience throughout my master's journey. Her thoughtful feedback and encouragement were invaluable at every stage of this work.

I am also sincerely thankful to Prof. Raymond Ng, for serving on my supervisory committee and for offering insightful comments that helped improve the quality and focus of this thesis.

I gratefully acknowledge Compute Canada for providing the computational resources essential to this research.

To my partner, thank you for your love, support, and steady belief in me, especially through the most challenging times.

Lastly, I'm forever grateful to my family for their encouragement and unwavering support throughout this journey.

Chapter 1

Introduction

In today's data-driven world, the volume of information generated is immense. This surge has led to an increasing number of users who, while not database experts, need to interact with databases to extract meaningful insights. Structured Query Language (SQL) remains the standard tool for such interactions, offering powerful capabilities to query and manipulate data. However, crafting effective SQL queries is no simple task. It demands a solid grasp of the application domain, an understanding of the underlying database schema, and proficiency in SQL syntax and concepts. Many users, especially those without formal training in database management, find themselves struggling to retrieve and analyze data efficiently.

Consider a marketing analyst aiming to segment customers based on recent purchasing behaviors. They might start with a basic SQL query to view all sales:

```
SELECT * FROM sales
```

Realizing the need for more specificity, they might refine their SQL query to focus on recent transactions:

```
SELECT * FROM sales WHERE purchaseDate >= '2025-01-01'
```

Further, they might decide to aggregate the data to see total sales per region:

```
SELECT region, SUM(amount) FROM sales WHERE  
purchaseDate >= '2025-01-01' GROUP BY region}
```

This step-by-step process highlights the challenges users face. Each step requires not only knowledge of SQL syntax but also an understanding of how to translate analytical needs into precise SQL queries. Mistakes can lead to SQL queries that are either incorrect or produce misleading results.

Traditional SQL query recommendation systems have typically relied on rule-based techniques or predefined templates. While straightforward to implement, such methods often struggle to adapt to the diversity of real-world workloads and the evolving needs of users [6, 12]. They also lack the flexibility to model nuanced transitions or personalize recommendations based on a user’s session history.

Recent work has begun exploring the use of deep learning in SQL query recommendation. For example, Lai et al. [21] propose a workload-aware method that leverages deep learning to predict query templates and query fragments. However, their system does not generate full SQL queries and instead treats the problem as a two-step classification task. Overall, the application of deep learning to this domain remains limited, with most models focusing on either partial prediction or relying heavily on schema-specific features.

At the same time, large language models and retrieval-augmented generation (RAG) [24] have shown promising results in other areas. Despite their success, these techniques have not been thoroughly explored for SQL query recommendation. In particular, no existing work has yet demonstrated how RAG can be used to generate full SQL queries based purely on a user’s session SQL query history, without access to the underlying database schema.

To address this gap, we propose a RAG-based system that generates the next SQL query by first retrieving a likely SQL query structure based on previous user SQL queries and then generating the full SQL query using a language model conditioned on that structure and recent context. This hybrid approach balances precision and flexibility, offering high-quality predictions without sacrificing efficiency.

Contributions. This thesis presents a new approach to SQL query recommendation based on recent user SQL query history, using a RAG framework. The method is divided into two stages: first, a transition-aware dual encoder is used to retrieve the most likely next SQL query template from a vector database built on past query templates. Second, a large language model generates the full SQL query by conditioning on the retrieved template, recent queries, and predicted semantic

transitions. One key contribution of this work is the integration of semantic transitions, such as adding filters into both training and inference. These transitions help the system better model how users evolve their queries over time. Compared to previous work, this method provides a more structured, interpretable, and semantically guided approach to recommend complete SQL queries.

Thesis Outline. The rest of this thesis is organized as follows:

- Chapter 2 formally defines the SQL query recommendation problem addressed in this thesis.
- Chapter 3 introduces the background concepts that support the rest of the thesis.
- Chapter 4 reviews related work on SQL query recommendation, natural language query suggestion, and text-to-SQL systems, with an emphasis on techniques that motivate or relate to our approach.
- Chapter 5 presents the overall approach developed in this thesis.
- Chapter 6 describes the experiments and the evaluation setup, along with a discussion of the results.
- Chapter 7 concludes the thesis and outlines possible directions for future work.

Chapter 2

Problem Formulation

We consider the task of recommending the next SQL query in a user session based on the most recent queries previously issued by the same user. Let $\mathcal{Q}^{(u)} = \{Q_1^{(u)}, Q_2^{(u)}, \dots, Q_n^{(u)}\}$ denote the sequence of n SQL queries written during session u , where each $Q_i^{(u)}$ is a structured SQL statement representing a step in the user’s analytical process.

While sessions may vary in length, we focus on the subset of this problem where only the two most recent queries, $Q_{i-1}^{(u)}$ and $Q_i^{(u)}$, are used as input. This restriction is motivated by both computational efficiency and the observation that recent queries carry the strongest signal for predicting a user’s next SQL query [8, 21]. Given this pair, our objective is to generate the most likely next query $Q_{i+1}^{(u)}$, representing a natural continuation of the user’s session.

Formally, we aim to learn a function:

$$\Phi: (Q_{i-1}^{(u)}, Q_i^{(u)}) \mapsto \hat{Q}_{i+1}^{(u)},$$

where $\hat{Q}_{i+1}^{(u)}$ is the predicted next query generated by the system.

In addition to this input query pair, we assume access to a historical query workload:

$$\mathcal{W} = \{\mathcal{Q}^{(1)}, \mathcal{Q}^{(2)}, \dots, \mathcal{Q}^{(M)}\},$$

where each $\mathcal{Q}^{(j)}$ is a previously recorded query session. This workload is used only during training to learn patterns of user behavior and inform retrieval and

generation components. At inference time, the model operates solely on the two most recent queries without accessing the broader workload.

We do not assume access to database schemas, table definitions, or user metadata. This design choice supports generalization across heterogeneous databases and deployment in schema-agnostic environments. Queries are represented as raw SQL strings.

This problem formulation enables a general-purpose SQL recommendation system that leverages session-level query dynamics. Our approach is presented in Chapter 5.

Chapter 3

Preliminary

This chapter introduces the core machine learning concepts needed to understand the methods used in this thesis for SQL query recommendation. Since our target audience may not have a background in machine learning, we provide intuitive explanations alongside technical details.

We begin with foundational sequence models like Recurrent Neural Networks (RNNs), followed by more recent architectures such as Transformers and encoder-decoder models. These models form the basis of large language models (LLMs), which are then adapted to new tasks using fine-tuning, prompt tuning, or Retrieval-Augmented Generation (RAG). In Chapter 4, we discuss how these models and adaptation strategies are used in the literature.

3.1 Sequence Models for Query Understanding

SQL sessions can be seen as sequences of queries, each query building on the previous. Therefore, we start by discussing how machine learning models handle sequential data.

3.1.1 Recurrent Neural Networks (RNNs)

RNNs [19, 38] are designed to process input sequences, one step at a time, while remembering information about earlier steps. For example, if a user issues a series of queries, an RNN can potentially “remember” the earlier ones while analyzing

the current one.

Given a sequence of inputs x_1, x_2, \dots, x_T , the RNN maintains a hidden state h_t that gets updated at each step:

$$h_t = \tanh(W_{xh}x_t + W_{hh}h_{t-1} + b_h), \quad (3.1)$$

where W_{xh} (Weight matrix between the input and hidden layer), W_{hh} (Weight matrix for the recurrent connection), and b_h (bias vector) are learnable parameters, and \tanh is a nonlinear function that squashes values to a fixed range. Figure 3.1 shows the architecture of a basic RNN, where each hidden state passes information through the sequence.

The model uses $y_t = \tanh(W_{hy}h_t + b_y)$ to make predictions which is the output (e.g. the next token in a query).

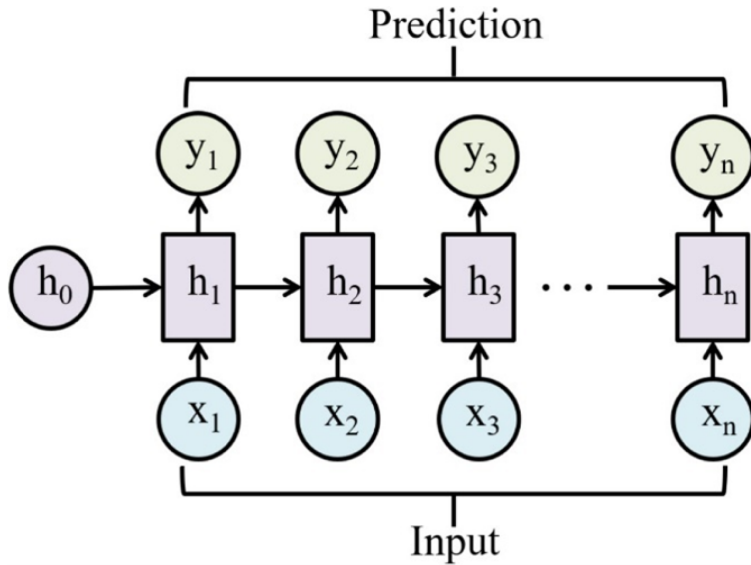


Figure 3.1: RNN Model Architecture [29].

Although RNNs are intuitive, they struggle to retain information across long sequences, which limits their usefulness in tasks involving longer query histories.

3.1.2 Long Short-Term Memory (LSTM)

LSTM networks [16] address the memory limitations of RNNs by introducing a memory cell and gating mechanisms that control how information is added or forgotten:

- **Forget gate:** Removes irrelevant information.
- **Input gate:** Accepts new information.
- **Output gate:** Decides what to pass forward.

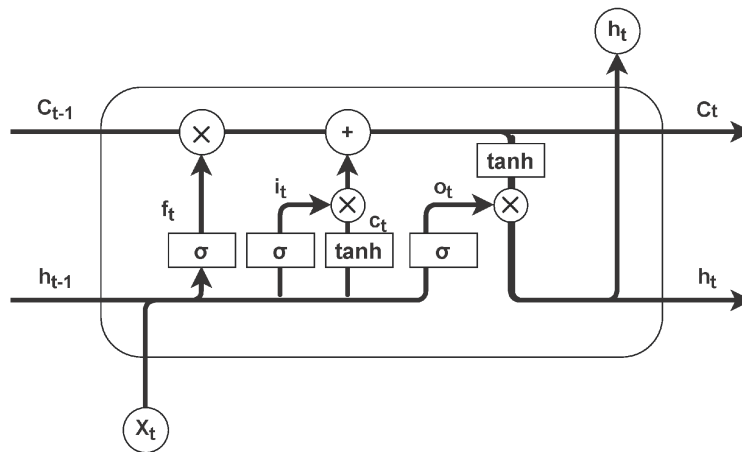


Figure 3.2: LSTM Model Architecture [28, 29]

The internal structure of an LSTM, including its memory cell and gating mechanisms, is illustrated in Figure 3.2. This architecture allows LSTMs to remember dependencies over longer spans, which is useful in multi-step analytical sessions.

3.2 Transformer Architecture

Transformers [41] represent a significant shift from RNN-based models. Instead of processing inputs one-by-one, transformers handle the entire sequence simultaneously using a mechanism called self-attention.

The main idea behind self-attention is to allow the model to weigh the importance of different parts of the input when producing an output. This is especially

useful when parts of a SQL query, or a session, depend on each other in non-sequential ways.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \quad (3.2)$$

where Q , K , and V represent the input tokens transformed into matrices, and d_k is a scaling factor.

Transformers have become the backbone of most modern language models due to their ability to capture long range dependencies and scale efficiently.

3.3 Encoder-Decoder Models for Sequence Prediction

The encoder-decoder architecture is a flexible structure that is often used for tasks such as translation and summarization. The **encoder** reads the input sequence (e.g. the previous SQL query) and produces a fixed-length representation that captures its meaning. The **decoder** then uses this representation to generate the output sequence (e.g. the next query), one token at a time.

The overall probability of generating a sequence y_1, \dots, y_T from the input x_1, \dots, x_N is given by:

$$P(y_1, \dots, y_T | x_1, \dots, x_N) = \prod_{t=1}^T P(y_t | y_{1:t-1}, x_{1:N}). \quad (3.3)$$

This formulation enables the model to condition each generated token on both the input and the previously generated output. The encoder-decoder architecture forms the basis of several models discussed in the sections. This architecture is used in many models that we will describe next.

3.4 Large Language Models (LLMs)

LLMs [49] are deep neural networks trained on massive datasets to understand and generate human-like language. They serve as the foundation of modern AI systems like ChatGPT [32], and they can also be applied to structured tasks such as generating SQL queries.

LLMs typically fall into one of three categories, based on how they process and

generate text:

- **Encoder-only models** (e.g. BERT [9]) are designed to read and understand input text. They are useful for tasks like classification where the model needs to extract meaning from the input but not generate output text.
- **Decoder-only models** (e.g. GPT [5, 36]) generate output text by predicting one token at a time, based on the previous tokens. Even though they lack a separate encoder, they implicitly "encode" context from previously generated tokens and use that to decode the next one. This behavior makes them well-suited for language generation tasks, such as dialogue, story writing, or SQL query generation.
- **Encoder-decoder models** (e.g. BART [23]) first encode the entire input sequence into a contextual representation, and then decode from this representation to produce the output. This architecture is ideal for tasks where the output depends on understanding the full input, such as translation, summarization, or SQL query prediction from previous queries.

These distinctions are important because different types of LLMs are better suited for different tasks.

3.4.1 BERT

BERT [9] is trained to predict missing words in a sentence, making it useful for understanding the structure and meaning of queries.

3.4.2 BART

BART [23] combines the BERT encoder with a GPT-style decoder. It is trained to recover corrupted input, making it suitable for tasks such as generating SQL from incomplete or modified inputs.

3.4.3 MiniLM

MiniLM [44] is a compact version of BERT, trained to retain performance while running faster, useful when resources are limited.

3.4.4 GPT

The GPT models [5, 36] are trained to predict the next word in a sequence. GPT-3 introduced few-shot learning, where it can perform new tasks by showing only a few examples at inference time.

3.5 Model Adaptation Techniques

3.5.1 Fine-Tuning

Fine-tuning [10] is a method used to adapt a large pre-trained model to a new, more specific task. The model is first trained on a general dataset, and then it is retrained on a smaller, task-specific dataset. This second round of training, on the new dataset, is what we refer to as fine-tuning.

During fine-tuning, all of the model’s internal weights are updated so that it can better perform the target task. For instance, if we start with a pre-trained BERT model and fine-tune it on a classification task, the model learns to adjust its internal representations to improve its classification accuracy.

Fine-tuning works well when the new dataset is similar to the pre-training data, and it often leads to high performance with relatively little task-specific data.

3.5.2 Few-Shot Learning

Few-shot learning [5] is a way to use large pre-trained language models without any additional training. Instead of updating the model’s parameters (as in fine-tuning), we simply provide a few examples of the desired task directly in the input prompt at inference time. This allows the model to generalize and perform the task based on these in-context examples.

For instance, if we want the model to complete a SQL query, we can provide a few pairs of input-output queries as part of the prompt, followed by a new input query. The model then uses the pattern in the examples to generate the corresponding output.

This method contrasts with fine-tuning, which requires retraining the model, and prompt tuning, which requires learning prompt embeddings. Few-shot learning is highly efficient and especially useful when:

- The model is very large and expensive to retrain.
- There is limited labeled data for the new task.
- Flexibility is needed to try different tasks quickly.

Few-shot learning became widely popular with GPT-3 [5], which demonstrated strong performance on a wide range of tasks using only a handful of examples at inference time.

3.5.3 Retrieval-Augmented Generation (RAG)

RAG [24] combines retrieval and generation into a single system. This approach is especially useful for our problem because SQL queries often follow patterns, and being able to reference similar past examples can guide better predictions.

- The **retriever** looks through a large collection of previous queries and selects the ones that are most similar to the current query context.
- The **generator** then uses both the current input and the retrieved examples to generate the next query.

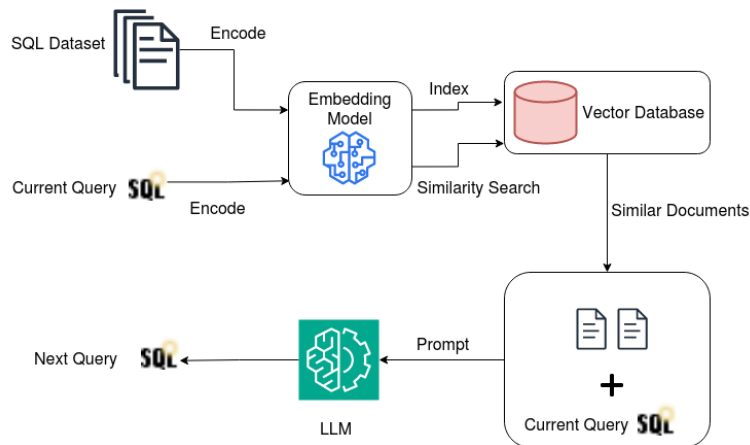


Figure 3.3: A general (hypothetical) Retrieval-Augmented Generation (RAG) framework for SQL query recommendation.

As shown in Figure 3.3, the retrieval step gives the system access to information beyond what it has memorized during training. This acts like a dynamic memory. Rather than relying entirely on fixed model weights, the system can "look up" helpful examples at runtime. These examples provide useful patterns that guide the generator toward a more accurate and context-aware prediction.

RAG is particularly helpful when training data is limited. By grounding the generation process in relevant examples, RAG improves both fluency and accuracy, making it the most effective solution explored in this thesis.

3.6 Summary

This chapter introduced the machine learning concepts necessary to understand the rest of the thesis. We covered:

- Sequence models (RNNs, LSTMs, Transformers)
- Architectures (encoder, decoder, encoder-decoder)
- Large language models (BERT, BART, GPT, MiniLM)
- Adaptation techniques (fine-tuning, few-shot learning, RAG)

These concepts provide the foundation for the SQL query recommendation techniques presented in the next chapters.

Chapter 4

Related Work

Two different strategies have emerged to tackle the problem in the context of SQL recommendation based on previous queries. The first method involves recommending the next SQL query based on the user’s prior history of written SQL queries. This straightforward method relies on the explicit patterns in the user’s SQL query history to provide relevant suggestions.

However, an alternative and more recent approach has arisen with the advancements in large language models and general artificial intelligence. This method enables users to express their queries in natural language, and the system responds by recommending the subsequent query also in natural language. This user-friendly approach aims to bridge the gap between users and the system, making the query formulation process more intuitive and accessible. Nevertheless, it is important to note that this alternative approach involves an additional component responsible for translating the natural language queries into their corresponding SQL counterparts. This translation step is crucial to ensure seamless communication between the user and the underlying SQL-based system, allowing for accurate query execution and retrieval of desired results.

In this section, we first describe various natural language query recommendation problems and identify those that are related to the SQL recommendation problem. Additionally, we discuss natural language recommendation techniques based on their approaches and review recent works relevant to our methodology. Next, we explore a variety of techniques specifically employed in addressing the

SQL recommendation problem. Finally, we examine state-of-the-art techniques in text-to-SQL problems and identify approaches that can be transferred to the SQL recommendation problem.

4.1 Natural Language Query Recommendation

4.1.1 Problem

Natural Language (NL) query recommendation is the task of recommending the next NL query based on the user’s prior history of written NL queries. We can categorize works on NL query recommendations based on either the problem they address or the technique they employ. There exist three distinct problems in the literature on natural language query recommendation:

- **NL Query Auto-Completion** [1, 34] is the process of suggesting a full NL query for the partial NL queries as the user types in real-time. The goal of NL query auto-completion is to help users save time and improve their search experience by predicting their intended NL query and preventing typing errors.
- **NL Query Suggestion** [8, 30, 31, 50] is the process of presenting users with a set of recommended NL queries based on the user’s past NL queries. The goal of NL query suggestions is to provide users with relevant suggestions.
- **NL Query Reformulation** [18, 43] is the process of modifying a user’s initial NL query in order to match the user’s information needs better and improve the effectiveness of search results. The goal of NL query reformulation is to help users overcome search barriers, such as ambiguity or lack of precision, and provide them with more relevant and accurate search results.

According to the above definitions, NL query auto-completion is not relevant to our problem. Thus, we will focus on NL query reformulation and, particularly, on NL query suggestion. NL Query reformulation involves recommending an alternative representation of the current NL query to achieve better search results. In contrast, NL query suggestion is applicable in a broader range of use cases, where the next

NL query recommendation can be either closely similar to the current NL query or related to the context of past NL queries within the session.

4.1.2 Approach

Building on this understanding of NL query reformulation and suggestion, we now proceed to categorize NL query recommendation techniques according to their approach, exploring how each method supports the varied needs of query enhancement as following:

- **Interestingness-based** systems assess the interestingness of meaningful and actionable information that is produced by different information discovery methods. The measure can be objective (e.g. information gain) or subjective (e.g. Unforeseen values that are unexpected for the users) and most of the classic techniques are based on similarity measures (e.g. collaborative filtering) for NL query recommendation [45].
- **Data-driven** systems can be well adapted to the user preference and generate personalized NL query recommendations based on prior in-context NL queries [35, 37]. Since the computing power has increased widely, deep learning approaches outperform classic information retrieval techniques extensively. Therefore, we look for state-of-the-art deep learning-based methods to transfer cutting-edge knowledge from NL to SQL query recommendation.

Interestingness-based methods are not our focus in this section, as we require more data-intensive techniques to adequately incorporate the various preferences of different users and effectively utilize previous in-context NL queries. Therefore, we will direct our focus to data-driven techniques in the next subsection.

4.1.3 Data-driven Techniques Review

In this subsection, we review notable data-driven techniques in natural language query recommendation, focusing on those with novel contributions. While many studies in this area use similar methods with minor model adjustments, the techniques highlighted here introduce significant advancements relevant to our method-

ology. These selections help us understand how innovative approaches can enhance the accuracy and relevance of SQL query recommendations, reflecting the latest pivotal research in the field.

Conversational Query Understanding Using Sequence to Sequence Modeling [37]

To enable chatbots and digital assistants to understand conversations effectively, it is crucial to go beyond the capabilities of conventional search engines, which excel in responding to open domain queries but are limited to stateless search. In this paper, the authors address the concept of conversational queries, which rely on the ongoing conversation context. They propose a solution called context-aware query reformulation, aiming to fulfill the user’s information needs within the conversation by understanding and incorporating the relevant context.

The context-aware query reformulation task takes conversation history (previous queries and answers) and the current query as inputs. The desired output is a reformulated query that integrates contextually relevant information, extending beyond the content of the current query. The authors focus on utilizing only the previous query from the conversation history for this problem. Thus, they formulate the problem as a sequence-to-sequence task, where the objective is to learn how to generate target sequences based on source sequences using a set of source-target sequence pairs.

Furthermore, the research incorporates attention techniques to enhance the performance of the sequence-to-sequence model. The attention mechanism helps the model focus on relevant parts of the source sequence during decoding, improving the quality of the generated target sequence.

Relation to Our Work: This paper frames query reformulation as a sequence-to-sequence task, modeling consecutive queries as source-target pairs. We adopt a similar framing for SQL query recommendation, but instead of using an encoder-decoder architecture, we employ a dual-encoder setup to encode both the current and next queries into a shared embedding space. Their insight into leveraging contextual history for predicting the next query informed our decision to treat past SQL queries as context and to model the causal transitions between them through learned representations.

Learning to Attend, Copy, and Generate for Session-Based Query Suggestion [8]

This paper focuses on query suggestion in search sessions, where users often reformulate their queries to convey complex information needs. Search engines can assist users by providing relevant query suggestions, enhancing the efficiency of the search process. Query suggestion becomes particularly valuable when users are unfamiliar with the terminology and ontologies required to formulate precise queries. It expedites the search process by enabling deeper exploration of the current search direction or offering alternative paths to different aspects of the search query.

The paper highlights patterns in query reformulation, such as term addition, removal, and retention. Notably, a significant portion of search terms (approximately 62%) is reused from a user's previous queries, with the majority of these repetitions (over 67%) coming from the least frequently used terms. These terms, defined as those that appear rarely within the overall query history but are repeatedly used by individual users, are crucial to the user's information needs.

The authors note that traditional sequence-to-sequence models are not effective for query suggestion due to two limitations. Firstly, these models consider input as a sequence of words without incorporating query-level information. Secondly, they tend to neglect low-frequency terms, even though retention terms fall into this category and play a crucial role in query reformulation patterns. To address these shortcomings, the authors propose a method for session-based query suggestions in this paper.

The session-based query suggestion model takes the previous session queries as input and generates the suggestion for the next query as a sequence of words. It is based on the sequence-to-sequence framework but incorporates three mechanisms to overcome the aforementioned limitations:

1. **Attend:** A query-aware attention mechanism is employed to generate the next query, considering the session context.
2. **Copy:** To manage term retention, a copy mechanism is implemented, allowing the decoder to replicate session context and handle out-of-vocabulary words.

3. Generate: This component generates new words to address the failures of traditional sequence-to-sequence models and prevent missing out on novel terms.

The model, referred to as ACG, is trained using a multi-objective learning process, taking into account the various mechanisms involved in the query suggestion task.

Relation to Our Work: This work highlights the shortcomings of traditional sequence-to-sequence models that treat input purely as word-level sequences, overlooking structural patterns and query-level semantics. This insight directly influenced our decision to incorporate transition-aware classifiers into our model. By explicitly modeling structural changes, such as added or removed SQL clauses, we ensure the system captures query evolution at a higher level of abstraction. While ACG uses mechanisms like copying and attention to preserve key terms across sessions, our method formalizes structural change through transition labels and integrates this information into both retrieval and generation stages.

Using BERT and BART for Query Suggestion [30]

This research paper addresses the problem of query suggestion in modern search engines, which aims to refine or explore different aspects of the search context. The objective is to recommend the most relevant query based on the session context and user intent. The problem is similar to the ACG paper, where prior session queries serve as input and the output is the next query in the session.

The transformer architecture, similar to the general sequence-to-sequence model, is employed in this study, consisting of an encoder and decoder. The encoder represents the session information, while the decoder generates the next query. However, the transformer introduces a self-attention mechanism within the encoder, enabling efficient information transfer between the encoder and decoder. Pre-trained transformers have demonstrated superior performance in various natural language processing (NLP) tasks, surpassing other techniques. Hence, this research fine-tunes two pre-trained transformers for the query suggestion problem.

Relation to Our Work: This paper demonstrates the effectiveness of fine-tuning pre-trained transformer models for query suggestion, reinforcing the value

of transfer learning in query modeling tasks. We adopt a similar approach by leveraging transformer-based models, such as MiniLM or CodeBERT, as the foundation for our dual encoders. However, instead of using an encoder-decoder structure to directly generate full SQL queries, we adopt a two-step approach due to computational constraints. Specifically, we fine-tune a smaller model to retrieve the most likely next SQL template, and then use a large language model to generate the final query. This setup allows us to maintain generation quality while reducing the computational burden of end-to-end fine-tuning on larger models.

Improving Sequential Query Recommendations with Immediate User Feedback [35]

This paper dives into interactive next-query recommendation systems, where the system responds to an initial query by providing results and a query recommendation for the user’s next step. The user then provides feedback by accepting or rejecting the recommended query, and the system adapts based on this feedback, aiming to maximize the total reward (user acceptance). While state-of-the-art techniques utilize sequence-to-sequence models and session queries, they often lack effective adaptation to user feedback. The authors seek to enhance query recommendation performance through the integration of immediate user feedback.

The proposed method, Exponentially Weighted Exploration, and Exploitation with Transformer-based Experts (Exp3-TE), combines cutting-edge transformer-based autoregressive models (experts) for causal language modeling with a next query recommendation algorithm that dynamically adjusts to immediate user feedback using a multi-armed bandit approach. Unlike traditional multi-armed bandits with fixed arm sets, this algorithm incrementally builds the candidate’s arm set. Additionally, it incorporates multiple experts and combines their query recommendations to improve overall model performance.

Relation to Our Work: Although our method does not incorporate real-time user feedback or online adaptation, this work emphasizes the importance of making query recommendations more responsive to user-specific behavior and intent. Inspired by that goal, we incorporate a transition-aware reranking step that refines retrieved SQL templates by aligning them with predicted structural changes in the session. Instead of relying on external feedback, our system uses internal signals,

such as predicted additions or removals of SQL components, to adjust the ranking of candidates. This allows us to simulate a form of adaptive behavior, improving the alignment between recommendations and the user’s evolving query intent.

4.2 SQL Query Recommendation

SQL query recommendation, as previously mentioned, involves recommending the next SQL query based on a user’s history of queries. Similar to the categorization discussed in the previous section, SQL recommendation methods can be broadly classified into interestingness-based and data-driven approaches. These techniques are designed to help users formulate relevant and accurate SQL queries by leveraging their historical interactions and the specific content of their queries.

- **Interestingness-based:** In this category, SQL recommendation methods primarily fall into two main categories:
 - **Collaborative filtering** is a method that predicts a user’s interests by collecting preference information from many users. This technique, exemplified by platforms like Netflix which recommend movies based on users’ similar viewing histories, effectively captures and utilizes patterns of collective behavior. In the context of SQL query recommendation, collaborative filtering methods, such as those outlined by Arzamasova and Böhm [3], leverage these user behaviors to identify similarities in SQL query preferences. User-based collaborative filtering compares the SQL query history of a target user with that of others to find people with similar patterns. The target user is then suggested SQL queries that are popular among these similar users. Conversely, item-based collaborative filtering analyzes user SQL query histories to find similarities directly between SQL queries themselves, suggesting to the target user queries that are similar to those they have previously executed or shown interest in.
 - **Content-based** methods, as distinct from collaborative filtering, focus on analyzing the content and attributes of individual SQL queries to make recommendations. Unlike collaborative filtering, which lever-

ages user behavior patterns, content-based methods (e.g. Eirinaki et al. [12]) extract specific features from SQL queries such as keywords, table names, or attributes. For each user, a query profile is created using these extracted features. Recommendations are then made by matching SQL queries with profiles similar to those in the user’s query history. One of the key advantages of content-based methods is their immunity to the cold-start problem, as they do not rely on user interaction data but rather on the content of the queries themselves, allowing for personalized recommendations even when historical user data is limited.

- **Data-driven** methods have become increasingly popular recently, particularly through the application of cutting-edge neural networks. In these methods, complex patterns and representations are learned from SQL query data, specifically using advanced deep learning techniques. In this area, recurrent neural networks (RNNs), reinforcement learning, and models based on transformers are frequently used. Deep learning models are appropriate for query recommendation tasks because they can efficiently extract sequential dependencies and contextual information from query sequences. In this category, two recent works stand out. In Lai et al. [21], the authors employ sequence-to-sequence models to extract information from SQL query sequences. However, their approach divides the recommendation task into two sub-tasks: template classification and fragment prediction, and as a result, they do not recommend full SQL queries. On the other hand, Meduri et al. [27] takes a different approach and utilizes reinforcement learning techniques to recommend the next SQL query.

Relation to Our Work: Our approach builds on recent data-driven methods for SQL query recommendation, particularly the work of Lai et al. [21], which highlights the importance of modeling query evolution through structured reformulations like clause additions or refinements. Like their method, we separate query templates from schema-specific fragments. However, we extend this foundation in several important ways. Rather than relying on template classification, we employ a retrieval-based framework using a dual encoder, which offers greater flexibility and generalization. We also introduce semantic transition types and train clause-

level classifiers to explicitly model how different parts of a SQL query are expected to evolve. Most notably, we complete the recommendation pipeline by generating full SQL queries using a large language model (LLM), guided by both the retrieved template and recent query context. This integration allows us to produce executable and coherent queries aligned with user intent. By combining retrieval, transition modeling, and generation, our framework leverages LLMs to capture richer patterns in query sessions, ultimately enhancing the accuracy and usefulness of recommendations.

4.3 Text-to-SQL Systems

The text-to-SQL problem entails generating an SQL query that is semantically equivalent to a given natural language query (NLQ) targeted at a particular Relational Database with a defined schema. The goal is to produce a valid SQL query that retrieves results from the database, aligning precisely with the user’s intent as stated in the NLQ.

However, this task comes with challenges on both the natural language (NL) and SQL fronts. The ambiguity and complexity of natural language can lead to multiple interpretations of a specific query, making accurate translation more difficult. Moreover, SQL is a structured language with strict grammar and limited expressiveness, which can result in complex SQL queries even for seemingly straightforward NL queries.

Additionally, the success of executing a SQL query relies on avoiding syntactical or semantic errors. If the translated SQL query contains any such issues, it becomes impossible to execute, hindering the retrieval of desired results. Building on this, we can categorize text-to-SQL research into three distinct categories based on the approach taken to address these challenges:

4.3.1 Model Architecture Innovations

This category focuses on the design and development of specialized neural network architectures tailored to improve the parsing and understanding of natural language queries and their translation into SQL. These architectures may incorporate advanced mechanisms such as attention layers or employ novel neural network

designs that better capture the intricacies of language and SQL syntax. To delve deeper within the model architecture innovations category, these systems can further be divided based on the nature of the decoder output into three main categories:

- **Sequence-based** approaches tackle the text-to-SQL problem as a sequence-to-sequence transformation where the network is trained to produce an SQL query in the form of a sequence of tokens based on the given NL query. While these methods simplify the text-to-SQL conversion, they do not inherently address the issue of generating syntactically correct queries. Nonetheless, the approach has gained significant traction, particularly with the rise of large pre-trained transformer models. This technique was adopted by Seq2SQL [51] as one of the first text-to-SQL systems.
- **Sketch-based slot filling** systems such as SQLNet [46] and Hydranet [26] are designed to streamline the process of generating SQL queries by breaking it down into smaller sub-tasks. This is achieved by creating a query sketch with specific parts left blank, which are later filled in to form the complete SQL query. However, implementing this approach can lead to an increase in the complexity of the neural network, as it may require handling each part of the query separately. Additionally, extending this method to handle complex SQL queries becomes challenging.
- In **grammar-based** approaches, instead of producing straightforward tokens as in the sequence-to-sequence method, a sequence of grammar rules is generated. These rules are subsequently applied to construct the SQL query. This approach has been introduced to address challenges present in other methods, such as syntactic and semantic errors, as well as the ability to handle complex queries effectively. Prominent examples of grammar-based approaches include IRNet [15] and RAT-SQL [42], which were among the most efficient techniques before being surpassed by the emergence of new large pre-trained transfer models.

4.3.2 Retrieval-Augmented Generation

Retrieval-augmented generation (RAG) is a technique that integrates traditional machine learning models with a retrieval component to enhance performance by accessing a database of relevant examples. This approach, often utilized in various domains, combines the capabilities of neural networks with external knowledge sources to improve the accuracy and relevance of generated content [24]. In the context of text-to-SQL translation, RAG leverages databases of NLQ-SQL pairs, using them to aid in generating SQL queries. By retrieving relevant examples from a dataset, the model can use the context of previously solved examples to form a more accurate translation of the new query, enhancing the model’s performance with practical, example-based learning. Notable advancements in this category include the following works, which illustrate significant progress in retrieval-augmented techniques used to tackle this problem:

ReFSQL: A Retrieval-Augmentation Framework for Text-to-SQL Generation [47]

This paper introduces “ReFSQL”, a retrieval-augmented framework aimed at improving the text-to-SQL generation process by addressing the gap between specific structure knowledge and general linguistic knowledge. This framework consists of two main components: (1) a structure-enhanced retriever, and (2) a generator. The retriever is designed to identify and incorporate SQL samples that share similar structural characteristics, thereby providing a context that improves the model’s understanding and generation capabilities. Additionally, the paper presents a novel method of Mahalanobis contrastive learning [25] that effectively bridges the gap between specific and general knowledge by guiding the model towards a distribution closely resembling that of the training examples. The framework demonstrated robustness and effectiveness across multiple datasets, achieving state-of-the-art performance, particularly when combined with other backbone models like the flan-T5.

Retrieval-augmented GPT-3.5-based Text-to-SQL Framework with Sample-aware Prompting and Dynamic Revision Chain [14]

This paper introduces a novel text-to-SQL framework that leverages a retrieval-augmented prompting approach to generate SQL queries from natural language questions. This approach uses large language models (LLMs) with GPT-3.5 and incorporates two key techniques: (1) sample-aware prompting, and (2) a dynamic revision chain. The sample-aware prompting involves retrieving and using SQL queries that share similar intentions to the input question, enhancing the LLM's understanding and response accuracy. The dynamic revision chain method iteratively refines the generated SQL queries based on fine-grained feedback, including execution results and natural language explanations of SQL queries. This iterative process helps in adjusting the queries to increase their accuracy and adaptability to the user's intent.

In summary, recent research in text-to-SQL technology shows significant progress in translating natural language queries into SQL. Innovations highlighted in the discussed papers, such as retrieval-augmented frameworks, few-shot learning with large language models, and creative prompt design, each contribute to improving this field. These advancements aim to help users without extensive SQL knowledge interact with databases. We will use these techniques to create novel methods for tackling the SQL recommendation problem in the next section.

Chapter 5

Methodology

Over recent years, there has been substantial enhancement in large language models, leading to their widespread application across diverse domains. These models are now employed in various contexts, such as chatbots, content generation, language translation, sentiment analysis, text summarization, question-answering systems, and personalized recommendations [2, 4, 7, 20, 22, 40, 48, 52]. Yet, there has been a lack of exploration in using LLMs for recommending SQL queries based on past interactions.

Motivated by this gap, we initially attempted to solve the next SQL query recommendation task by directly fine-tuning a large language model. Specifically, we fine-tuned GPT-2 [36], a decoder-based Transformer model designed for text generation, to take the user’s current query as input and predict the next likely SQL query. While the model demonstrated a strong ability to produce fluent and SQL-like output, it showed some important shortcomings that made it unreliable for our use case. It often generated incomplete or overly long queries, produced syntax errors, and lacked consistency across predictions. For example, given the input:

```
SELECT salary FROM employees WHERE department = 'sales'
```

the model might generate:

```
SELECT salary FROM employees WHERE  
department = 'marketing' AND
```

which is syntactically incomplete. These issues were mainly due to GPT-2’s limited size, which constrained its ability to model the complexity of SQL generation. Although larger models like GPT-3.5 [5] are far better suited for this task, we were unable to fine-tune them due to computational constraints, which limited our ability to achieve more reliable results. This is a common limitation faced by many academic and research projects that don’t have access to large-scale infrastructure, so we focused on methods that are more practical and reproducible under standard resource budgets.

These challenges highlighted an important insight: while LLMs offer powerful generative capabilities, using them directly for SQL recommendation, without structure or schema guidance, results in unreliable outputs. As a result, we transitioned to a more structured approach that leverages the strengths of both traditional and generative methods. Specifically, we decomposed the task into two stages: (1) predicting the structure of the next SQL query, referred to as its *query template*, and (2) using this predicted template to guide the generation of the full SQL query. This two-stage setup not only incorporates structural constraints to ensure syntactic validity, but also preserves the flexibility of LLMs to generate complete, diverse queries tailored to the user’s past behavior.

An important aspect of our approach is how we retrieve the next query template. Rather than treating each SQL query in isolation, we look at how queries evolve within a session. Much like how users reformulate search queries by adding, removing, or reusing terms, SQL users also tend to modify their queries in predictable ways, such as by adding a filter, changing a grouping, or dropping an ordering clause. These kinds of changes aren’t always captured well by traditional sequence-to-sequence models [21], which often treat input as a flat sequence of tokens and ignore query-level structure.

This insight was strongly influenced by work in session-based natural language query suggestion [8], which highlights the importance of modeling how queries evolve over time. That work shows that most edits in search sessions follow consistent reformulation patterns, like reusing terms from previous queries or introducing small but important changes. Inspired by this, we introduce the idea of modeling *query transitions*, the structural differences between two consecutive SQL queries in a session. These transitions help capture how a user’s intent changes at a higher

level, beyond just token-level edits.

To support this, we define a set of semantic transition types that describe common changes between queries, such as adding a `WHERE` clause or removing an aggregate function. We use these transitions both during training, to improve supervision, and at inference time, to guide retrieval and generation.

The rest of this chapter introduces the key concepts that our system is built on: query templates, query fragments, and query transitions. We then describe the full architecture of our method, which follows a retrieval-augmented generation (RAG) framework. At a high level, the system first retrieves likely next query templates using neural embeddings, and then fills in those templates using a language model guided by the user’s last two SQL queries.

5.1 Key Definitions

To support our structured approach, we introduce three core concepts: *query templates*, *query fragments*, and *query transitions*. These definitions help us abstract query structure, isolate schema-specific details, and model how queries change over time.

Definition 1 (Query Template [21]) *Given a SQL query, Q_i , we define its **query template**, $T(Q_i)$, as a tree structure derived from the abstract syntax tree (AST) of Q_i , where all schema-specific elements are replaced with general placeholders. Specifically:*

- *Table names are replaced with `TAB`*
- *Column names are replaced with `ATT`*
- *Literal values (both numbers and strings) are replaced with `NUM`*
- *Function names (e.g. `SUM`, `AVG`, `MAX`) are replaced with `FUNC`*

All SQL keywords and structural syntax are retained. This abstraction allows the system to learn query structure patterns that generalize across different databases and schemas, without depending on specific table or column names.

For example, the query:

```
SELECT AVG(salary) FROM employees WHERE
department = 'sales'
```

is abstracted to the following template:

```
SELECT FUNC(ATT) FROM TAB WHERE ATT = NUM
```

Here, only the schema-specific tokens are replaced. SQL keywords such as SELECT, FROM, and WHERE remain unchanged.

Definition 2 (Query Fragment [21]) A *query fragment* refers to any original token in Q_i that was abstracted away in the template $T(Q_i)$. This includes:

- Table names (e.g. *employees*)
- Column names (e.g. *salary*, *department*)
- Literal values (e.g. *'sales'*)
- Function names (e.g. *AVG*)

Fragments are grouped into four categories: *TABLE*, *COLUMN*, *LITERAL*, and *FUNCTION*. During generation, the language model’s task is to predict these missing fragments based on the retrieved query template and generate the full SQL query.

Definition 3 (Query Transition) A *query transition* describes how the structure of a SQL query changes from one step to the next within the same session. We compare the templates of two consecutive queries, $T(Q_i)$ and $T(Q_{i+1})$, and classify transitions for each clause independently.

We focus on five common SQL clauses: *SELECT*, *FROM*, *WHERE*, *GROUP BY*, and *ORDER BY*. These were chosen because the vast majority of queries include only these clauses. They also tend to reflect the most meaningful structural edits, such as adding filters, reordering results, or changing grouped attributes.

For each clause, we assign one of the following transition labels:

- *none* — the clause remains unchanged
- *add* — the clause appears in $T(Q_{i+1})$ but not in $T(Q_i)$

- *remove* — the clause is dropped from $T(Q_i)$ in $T(Q_{i+1})$
- *refine* — the clause exists in both but its contents change

These transitions help the model understand how a user modifies their query structure over time.

5.2 Overview of the RAG Framework

Our full system follows a retrieval-augmented generation (RAG) framework, which combines the strengths of search-based retrieval and language model generation. At a high level, the model first retrieves a likely structure for the next SQL query, a *query template*, and then uses a language model to generate the full query based on this structure.

The overall architecture consists of three major components:

- A **transition-aware dual encoder** for retrieving the most likely next query template
- **Auxiliary transition classifiers** for predicting how specific SQL clauses (e.g. WHERE, GROUP BY) are likely to change in the next query. This is framed as a multi-class classification problem, where each clause can be labeled with a transition type such as “add,” “remove”, “refine”, or “none”.
- A **language model (LLM)** for generating the final SQL query

We assume that each session contains multiple SQL queries written by the same user while exploring or analyzing data. The goal is to predict what the user is likely to write next. To do this, our model considers the last two queries in the session, which allows it to learn patterns in how users build and refine their queries over time.

Due to computational limits, we do not use the full query history. Instead, we limit the input to the two most recent queries. Prior work has shown that the most recent query typically carries the most useful information for predicting what comes next [8, 21], and we leave the use of longer query histories as future work.

The rest of this chapter introduces each component of our method in more detail: query template retrieval using dual encoders, transition classification, and final SQL generation. Before we describe these individual modules, we provide a high-level summary of the full system in Algorithm 1.

Algorithm 1 Overview of Transition-Aware RAG for SQL Query Recommendation

Require: Session history $\{Q_{i-1}, Q_i\}$, trained encoders $\text{Enc}_q, \text{Enc}_a$, Clause-Level Transition Classifiers $\{f_c\}_{c \in \mathcal{C}}$, vector index \mathcal{I} , language model LLM

Ensure: Predicted next SQL query \hat{Q}_{i+1}

- 1: Convert Q_{i-1} and Q_i to templates: $T(Q_{i-1}), T(Q_i)$
 - 2: Encode combined input with query encoder: $\mathbf{q} \leftarrow \text{Enc}_q(T(Q_{i-1})[\text{SEP}]T(Q_i))$
 - 3: Normalize: $\tilde{\mathbf{q}} = \frac{\mathbf{q}}{\|\mathbf{q}\|}$
 - 4: Retrieve top- K next-template candidates $\{T_1, \dots, T_K\}$ from FAISS index \mathcal{I} using cosine similarity
 - 5: Predict transition probabilities for each clause: $\hat{y}_c = f_c(\mathbf{q})$ for all $c \in \mathcal{C}$
 - 6: **for** each retrieved candidate T_v **do**
 - 7: Extract clause-level transitions $y_c^{(v)}$ between $T(Q_i)$ and T_v
 - 8: Compute alignment score: $\delta_v = \alpha \cdot \frac{1}{|\mathcal{C}|} \sum_{c \in \mathcal{C}} \hat{y}_c^\top y_c^{(v)}$
 - 9: Update similarity: $\tilde{s}_v = s_v + \delta_v$
 - 10: **end for**
 - 11: Select top template: $T^* = \arg \max_v \tilde{s}_v$
 - 12: Generate full SQL query: $\hat{Q}_{i+1} = \text{LLM}(Q_{i-1}, Q_i, T^*)$
 - 13: **return** \hat{Q}_{i+1}
-

5.3 Query Template Retrieval

5.3.1 Template Abstraction

We represent each SQL query Q_i as a template $T(Q_i)$ by abstracting away schema-specific elements and replacing them with generic placeholders, following the formal definition provided in Section 5.1.

The resulting template $T(Q_i)$ captures the high-level structural intent of the

query, such as the presence of filtering, grouping, or aggregation, while removing identifiers tied to a specific database schema. This abstraction enables our system to learn structural patterns that generalize across domains and users, rather than overfitting to schema-specific details.

5.3.2 Dual Encoder Architecture

We employ a dual-encoder architecture to retrieve the most likely next query template based on recent query history. The model is trained on historical SQL query template pairs $\langle T(Q_{i-1}) \circ T(Q_i), T(Q_{i+1}) \rangle$, where the input consists of the last two templates from a user session, and the target is the next template. This setup allows the model to learn structural patterns in how users typically transition from one query to another.

To represent the input, we concatenate the two most recent templates using a special separator token:

$$\text{Input} = T(Q_{i-1}) [\text{SEP}] T(Q_i)$$

We define two transformer-based encoders:

- $\text{Enc}_q: \text{Input} \rightarrow \mathbb{R}^d$ — the query encoder maps the concatenated input templates into a d -dimensional embedding vector.
- $\text{Enc}_a: \text{Template} \rightarrow \mathbb{R}^d$ — the answer encoder maps the next query template $T(Q_{i+1})$ into an embedding in the same space.

Both encoders are initialized from pre-trained language models (e.g. CodeBERT [13], MiniLM [44]) and are fine-tuned jointly for the retrieval task.

Given a batch of N training examples, we compute the query and answer embeddings as:

$$\mathbf{q}_u = \text{Enc}_q(T(Q_{i-1}^{(u)}) [\text{SEP}] T(Q_i^{(u)})), \quad \mathbf{a}_u = \text{Enc}_a(T(Q_{i+1}^{(u)}))$$

To ensure that similarity comparisons are based on direction rather than mag-

nitude, all vectors are normalized to unit length:

$$\tilde{\mathbf{q}}_u = \frac{\mathbf{q}_u}{\|\mathbf{q}_u\|}, \quad \tilde{\mathbf{a}}_u = \frac{\mathbf{a}_u}{\|\mathbf{a}_u\|}$$

The similarity between each query and candidate answer in the batch is computed using a dot product, scaled by a temperature parameter τ :

$$s_{uv} = \frac{\tilde{\mathbf{q}}_u^\top \tilde{\mathbf{a}}_v}{\tau}$$

Here, τ is a positive constant that controls the sharpness of the similarity scores. A smaller τ produces a more peaked distribution, placing more emphasis on distinguishing between closely matched candidates.

We use the InfoNCE loss [39] to train the model. This loss encourages the similarity score between the correct query-answer pair $(\tilde{\mathbf{q}}_u, \tilde{\mathbf{a}}_u)$ to be higher than the similarity with all other answers in the batch:

$$\mathcal{L}_{\text{retrieval}}^{(u)} = -\log \frac{\exp(s_{uu})}{\sum_{v=1}^N \exp(s_{uv})} \quad \mathcal{L}_{\text{retrieval}} = \frac{1}{N} \sum_{u=1}^N \mathcal{L}_{\text{retrieval}}^{(u)}$$

This contrastive learning objective allows the model to construct a meaningful representation space in which query template pairs that often follow each other in real user sessions are close together. During inference, this embedding space enables efficient retrieval of the most likely next templates using simple nearest neighbor search.

5.3.3 Auxiliary Transition Classifiers

To help the encoder capture how SQL queries evolve structurally, we extend the architecture with a set of auxiliary classifiers. These classifiers are designed to predict the *query transitions* that occur between the last two query templates in a session and the next one. Specifically, they model how each SQL clause (e.g. WHERE, GROUP BY) is expected to change.

We focus on five common SQL clauses: SELECT, FROM, WHERE, GROUP BY, and ORDER BY. For each clause, the model predicts one of four possible transition types:

- `none` — the clause remains unchanged
- `add` — the clause is introduced in the next query
- `remove` — the clause is removed in the next query
- `refine` — the clause is modified but still present

To model these transitions, we attach a separate classifier head for each clause. Each classifier takes as input the query embedding \mathbf{q}_u , produced by the query encoder for the combined input $[T(Q_{i-1}^{(u)}), T(Q_i^{(u)})]$, and outputs a distribution over the four transition labels:

$$f_c : \mathbb{R}^d \rightarrow \mathbb{R}^4$$

Training labels. To generate training targets, we compare each clause in the current and next query templates. If the clause is newly introduced, it is labeled as `add`; if it is removed, `remove`; if it appears in both but with different content, `refine`; and if it appears unchanged, `none`. This yields a label $y_c^{(u)} \in \{0, 1, 2, 3\}$ for each clause c and training example u .

Each classifier is trained using the cross-entropy loss:

$$\mathcal{L}_c = \frac{1}{N} \sum_{u=1}^N \text{CE}(f_c(\mathbf{q}_u), y_c^{(u)})$$

The total transition loss is the average over all clauses:

$$\mathcal{L}_{\text{transition}} = \frac{1}{|\mathcal{C}|} \sum_{c \in \mathcal{C}} \mathcal{L}_c$$

Joint training with retrieval. These classifiers are trained jointly with the dual-encoder retrieval model. That is, the query encoder is shared across both tasks, and the gradients from both the retrieval and classification losses are used to update the model. The total training objective is:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{retrieval}} + \lambda \cdot \mathcal{L}_{\text{transition}}$$

where λ is a scalar hyperparameter that balances the two components.

This joint learning setup ensures that the encoder not only learns to retrieve relevant next query templates, but also becomes sensitive to how each part of the query is likely to change. This extra supervision helps the encoder learn more detailed, clause-aware representations, which lead to better SQL predictions that match how users write queries.

The training procedure jointly optimizes the dual encoder and transition classifiers using a multi-task objective. Once training is complete, we embed and index all training templates for fast retrieval at inference time. The full training and indexing workflow is summarized in Algorithm 2.

Algorithm 2 Training Procedure for Dual Encoder with Clause-Level Transition Classifiers

Require: Training triples $\{(T(Q_{i-1}^{(u)}), T(Q_i^{(u)}), T(Q_{i+1}^{(u)}))\}_{u=1}^N$, clause transition labels $\{y_c^{(u)}\}$ for $c \in \mathcal{C}$

Ensure: Trained encoders $\text{Enc}_q, \text{Enc}_a$, classifiers $\{f_c\}_{c \in \mathcal{C}}$, vector index \mathcal{I}

- 1: **for** each training step **do**
- 2: Encode input templates: $\mathbf{q}_u \leftarrow \text{Enc}_q(T(Q_{i-1}^{(u)})[\text{SEP}]T(Q_i^{(u)}))$
- 3: Encode next-query templates: $\mathbf{a}_u \leftarrow \text{Enc}_a(T(Q_{i+1}^{(u)}))$
- 4: Normalize: $\tilde{\mathbf{q}}_u = \mathbf{q}_u / \|\mathbf{q}_u\|, \tilde{\mathbf{a}}_u = \mathbf{a}_u / \|\mathbf{a}_u\|$
- 5: Compute pairwise similarity: $s_{uv} = \tilde{\mathbf{q}}_u^\top \tilde{\mathbf{a}}_v / \tau$
- 6: Compute retrieval loss:

$$\mathcal{L}_{\text{retrieval}} = -\frac{1}{N} \sum_{u=1}^N \log \frac{\exp(s_{uu})}{\sum_{v=1}^N \exp(s_{uv})}$$
- 7: Predict clause transitions: $\hat{y}_c^{(u)} \leftarrow f_c(\mathbf{q}_u)$ for all c
- 8: Compute classification loss:

$$\mathcal{L}_{\text{transition}} = \frac{1}{|\mathcal{C}|} \sum_{c \in \mathcal{C}} \text{CE}(\hat{y}_c^{(u)}, y_c^{(u)})$$
- 9: Compute total loss: $\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{retrieval}} + \lambda \cdot \mathcal{L}_{\text{transition}}$
- 10: Update all parameters via backpropagation
- 11: **end for**

Indexing Step:

- 12: Encode and normalize all unique training set templates $T(Q^{(v)})$ using Enc_a
 - 13: Store normalized embeddings $\tilde{\mathbf{a}}_v$ in FAISS vector index \mathcal{I}
-

5.3.4 Vector Database Indexing and Retrieval

Once the transition-aware dual encoder has been trained, we use it to embed all query templates from the training set and store them in a vector database for fast retrieval. Specifically, we use the answer encoder Enc_a to generate fixed-length vector representations for each query template $T(Q^{(v)})$:

$$\mathbf{a}_v = \text{Enc}_a(T(Q^{(v)})), \quad \tilde{\mathbf{a}}_v = \frac{\mathbf{a}_v}{\|\mathbf{a}_v\|}$$

The normalization step ensures that similarity comparisons between vectors are based on cosine similarity, which is more meaningful in high-dimensional semantic spaces.

To enable efficient retrieval, we store these normalized embeddings in FAISS [11], a high-performance vector search library.

Before indexing, we perform basic preprocessing:

- Duplicate query templates are removed to prevent retrieval redundancy.
- All embeddings are normalized to ensure consistent distance metrics.

At inference time, given the current session history $[T(Q_{i-1}), T(Q_i)]$, we compute its embedding using the query encoder Enc_q :

$$\mathbf{q} = \text{Enc}_q(T(Q_{i-1})[\text{SEP}]T(Q_i)), \quad \tilde{\mathbf{q}} = \frac{\mathbf{q}}{\|\mathbf{q}\|}$$

We then compute the cosine similarity between $\tilde{\mathbf{q}}$ and all stored candidate embeddings $\tilde{\mathbf{a}}_v$:

$$s_v = \tilde{\mathbf{q}}^\top \tilde{\mathbf{a}}_v$$

The top- K candidates with the highest similarity scores are selected:

$$\text{TopK}(T(Q_i)) = \text{argsort}_v(s_v)[1 : K]$$

These retrieved templates represent the most likely structural forms of the user’s next SQL query. To further narrow down the most suitable candidate, we apply a transition-based reranking step. This reranking uses predictions from the trained transition classifiers to select the template that best matches the expected

structural changes. The top-ranked template is then passed to the language model for the full SQL query generation.

5.3.5 Transition-Aware Reranking

Although the top- K query templates retrieved from the vector database are structurally similar to the current session context, they may not always align with how a user’s query is likely to evolve. To address this, we introduce a reranking step that leverages the transition classifiers trained jointly with the encoder.

This reranking process prioritizes candidates that not only have high embedding similarity but also match the expected structural changes predicted by the model. For a given session context $[T(Q_{i-1}), T(Q_i)]$, the query encoder produces an embedding \mathbf{q} , which is then passed to the clause-level transition classifiers $\{f_c\}_{c \in \mathcal{C}}$ to predict the transition type for each clause:

$$\hat{y}_c = f_c(\mathbf{q}) \quad \text{for each } c \in \mathcal{C}$$

where $\hat{y}_c \in \mathbb{R}^4$ is the predicted probability distribution over the four transition classes: none, add, remove, and refine.

For each retrieved candidate template $T(Q_{i+1}^*)$, we compute its true transition label for each clause by comparing its structure with that of $T(Q_i)$. Each clause c is assigned a one-hot label $y_c^* \in \{0, 1\}^4$ indicating the actual transition type.

We then compute a clause-level agreement score using the dot product between the predicted distribution and the true label:

$$\delta_c = \hat{y}_c^\top y_c^*$$

The overall transition alignment score for the candidate is the normalized average agreement across all clauses:

$$\delta_* = \alpha \cdot \frac{1}{|\mathcal{C}|} \sum_{c \in \mathcal{C}} \delta_c$$

This normalization ensures that the transition influence δ_* is bounded in $[0, \alpha]$, preventing it from overpowering the embedding-based similarity score regardless

of the number of clauses.

The final candidate score is computed by combining the transition alignment and embedding-based similarity:

$$\tilde{s}_* = s_* + \delta_*$$

Finally, the top- K candidates are re-ranked based on the updated scores \tilde{s}_* , and the highest scoring template is selected for generation. This reranking step helps guide the system toward next query templates that are not only structurally similar, but also consistent with the model’s understanding of realistic query evolution.

5.3.6 Full SQL Query Generation

After retrieving the most suitable query template candidate, the system proceeds to generate the final SQL query. This step is handled by a pretrained large language model (LLM), which fills in the schema-specific components, query fragments, of the selected query template based on the user’s last two SQL queries.

The LLM is provided with three key inputs:

- The previous SQL query Q_{i-1}
- The current SQL query Q_i
- The top-ranked next query template T^* , typically selected as T_1 after reranking

These components are formatted into a structured prompt that guides the generation process:

$$\hat{Q}_{i+1} = \text{LLM}(Q_{i-1}, Q_i, T^*)$$

Here, \hat{Q}_{i+1} is the fully generated SQL query, which is expected to follow the structure defined by the predicted template T^* . The language model uses Q_{i-1} and Q_i to understand the recent query context, and T^* to determine the structure of the next query.

The template T^* provides high-level guidance by specifying which SQL clauses should be included in the next query, such as filters, groupings, or sorting, without

dictating specific details. The LLM fills in those details by generating the appropriate table names, column names, functions, and literals, often by referring back to elements found in the recent queries.

The input prompt given to the LLM includes:

- The most recent two SQL queries (Q_i, Q_{i-1})
- The predicted template T^* , with placeholders for schema elements
- A simple instruction to complete the template and generate a valid next SQL query

Since many of the required fragments (e.g. tables, columns, or filters) often appear in recent queries [8, 21], the LLM can infer missing details by reusing or adapting elements from the prior context. This allows the model to stay consistent with the user’s ongoing analysis while producing a syntactically complete and executable SQL query.

Chapter 6

Experimental Analysis

In this chapter, we evaluate the effectiveness of our transition-aware retrieval-augmented generation (RAG) framework for next SQL query recommendation. We begin by outlining the experimental setup, which includes the SQLShare dataset [17], a set of representative baseline methods, and the hyperparameter configurations used for training and inference. We then introduce the evaluation metrics that measure both structural and content-level accuracy. Finally, we present empirical results and analysis, highlighting the benefits of modeling structural transitions and leveraging retrieved query templates in generating the next query within a session.

6.1 Experimental Setup

We evaluate our transition-aware SQL recommendation method on a large-scale, real-world dataset and compare it against several established baselines. This section introduces the dataset used in our experiments and outlines key implementation details for reproducibility.

6.1.1 Dataset: SQLShare Workload

We conduct our experiments on the SQLShare workload [17], a database platform that enabled users to upload their own datasets and write SQL queries directly through a web interface. As a result, SQLShare provides a rich and diverse collection of real user queries, each tied to a user-defined schema. The dataset spans 64

unique schemas and covers a wide range of domains, including biology, oceanography, and environmental science. Unlike traditional benchmarks built on a single fixed schema, SQLShare reflects the challenges of real-world, ad hoc analysis, requiring models to generalize across both structural and semantic variations, making it a compelling benchmark for SQL query recommendation tasks.

To support structured learning, we adopt a session-based formulation of the task. Each session consists of a sequence of SQL queries issued by a single user on a single dataset. From these sequences, we construct training instances of the form $(Q_{i-1}, Q_i) \rightarrow Q_{i+1}$, where the input consists of the two most recent queries and the target is the next query in the session. Sessions with fewer than three queries are discarded to ensure valid triplets. The dataset is split by session to prevent test leakage and to preserve session continuity during evaluation.

The SQLShare workload used in this study includes 1463 user sessions, each with more than two SQL queries. In total, we analyze around 17.6k SQL queries, of which 14k are unique, yielding an average session length of 12 queries. These figures reflect the exploratory nature of SQLShare, where users often iterate on previous queries within a session.

Although there are many queries, most of them are small changes to earlier ones, such as adjusting filter conditions or selected columns, rather than completely new query structures. When we group queries by structure using templates, the number of distinct patterns becomes much smaller. This shows that users often follow similar patterns and change only specific parts of a query. Because SQLShare includes both repeated query patterns and a wide variety of table and column names, it is a good dataset for testing systems like ours. It helps evaluate how well a model can handle both familiar query forms and different database schemas.

Even though SQLShare is a diverse and realistic benchmark, it is the only dataset on which we evaluate our approach, as it is the only one where we could run all selected baseline methods. This reflects a limitation of our current study rather than of the dataset itself. However, SQLShare effectively represents a collection of datasets, with 64 distinct and heterogeneous database schemas spanning multiple domains. This diversity makes it a strong benchmark for evaluating the generalization ability of SQL recommendation models. Evaluating our framework on additional datasets is left for future work.

6.1.2 Baseline Methods

To assess the effectiveness of our transition-aware RAG framework, we compare it against a set of representative baseline methods selected for their diversity in modeling strategies. These include a large language model without structural priors, a state-of-the-art sequence-aware deep learning, a collaborative filtering system, and a simple repetition-based heuristic.

Our choice of baselines reflects three main goals:

- To test whether large language models can generalize next query patterns without structural supervision
- To compare against a state-of-the-art sequence-aware deep learning method trained on SQLShare
- To evaluate simpler, schema-agnostic baselines as reference points

Generative LLM (No Guidance)

This baseline uses a pretrained large language model to generate the next SQL query directly from the last two queries, without retrieval, templating, or transition modeling. It tests whether a modern LLM can implicitly capture query evolution. We use GPT-4o mini [33], prompted with Q_{i-1} and Q_i , and apply greedy decoding. While such models are capable of producing fluent SQL syntax, they are not trained on SQLShare and lack inductive biases for structural consistency.

Sequence-Aware Deep Learning (Lai et al.)

We include a state-of-the-art neural baseline from Lai et al. [21], which splits the next SQL query recommendation task into two subproblems: (1) predicting the next query template via a classification model, and (2) predicting the likely query fragments (i.e. tables, columns, literals, and functions) using a sequence-to-sequence model. Both components are trained on SQLShare, and the model is designed to capture common patterns in how users modify their queries.

However, the method does not synthesize full SQL queries. As a result, although it is a strong baseline for the sub-tasks of template and fragment prediction, it lacks a mechanism for generating full SQL queries. In our evaluation, we compare against this baseline on its two native tasks, template classification and fragment prediction, rather than full SQL query generation.

QueRIE Framework (Collaborative Filtering)

This baseline adapts the QueRIE framework [12], a collaborative filtering method originally proposed for query recommendation. It constructs a feature vector summarizing the current session and retrieves past queries from similar sessions (excluding the current user). While it does not use SQL structure or transition modeling, it serves as a useful static baseline based on co-occurrence patterns. We include it to evaluate how well traditional recommendation techniques perform in the presence of diverse schemas.

Identity Baseline (Naive Q_i)

As a lower-bound reference, we include a naive identity baseline that simply returns Q_i as the prediction for Q_{i+1} . This reflects the observation that users often repeat queries or make minor edits. While incapable of modeling structural changes or user intent shifts, it can achieve surprisingly high token-level overlap due to schema reuse. Its inclusion helps highlight the added value of modeling transitions and structure explicitly.

6.1.3 Hyperparameters

The performance of our transition-aware RAG framework depends critically on a balance between retrieval accuracy and transition supervision. To achieve this, we selected and tuned a set of hyperparameters based on targeted experiments and prior design principles. Our goal was to ensure that the encoders produce semantically meaningful representations for SQL templates, while the transition classifiers offer additional clause-level guidance without overwhelming the retrieval objective.

For retrieval, we use CodeBERT (`microsoft/codebert-base`) [13], a transformer model pre-trained on natural language and programming languages, including SQL. Compared to general-purpose encoders like MiniLM, CodeBERT provides better inductive bias for parsing structured code-like syntax, making it particularly well-suited for SQL template representation. The temperature parameter $\tau = 0.1$ was selected to sharpen contrastive learning in the InfoNCE loss [39], encouraging more confident matching between consecutive query templates. The transition classification loss is weighted by $\lambda = 0.3$ to allow structural supervision to guide the encoder without dominating it.

At inference time, we retrieve the top- $K = 10$ candidates from the vector index, which we found to balance diversity and retrieval precision, while also giving the transition-aware reranking mechanism sufficient candidates to select a structurally suitable query template. The final SQL query is generated using GPT-4o mini [33], which fills in schema-specific fragments based on the retrieved template and recent query history.

Table 6.1 lists the set of hyperparameters used in training and inference.

Table 6.1: Key Hyperparameters for the Transition-Aware RAG Model

Hyperparameter	Value
Embedding model name	<code>microsoft/codebert-base</code>
Max sequence length	128
Batch size	32
Epochs	20
Learning rate	1e-5
Temperature (τ)	0.1
Transition loss weight (λ)	0.3
Retrieval top- K	10
LLM model	GPT-4o mini

6.2 Empirical Evaluation

We evaluate our model against several baselines using three complementary metrics that assess both structural and content-level quality in next-query recommen-

dition. These metrics include normalized edit distance, template match accuracy, and fragment-level F_1 scores. Together, they provide a comprehensive picture of how well each method approximates the next query in a session.

6.2.1 Evaluation Metrics

Edit Distance. This metric measures the normalized number of edits—insertions, deletions, or substitutions—required to transform the predicted query \hat{Q}_{i+1} into the ground truth Q_{i+1} . While computed at the character level, it offers a useful approximation of how closely a generated query aligns with the target overall. Lower values indicate higher fidelity.

Template Match Accuracy. This measures whether the predicted query has the same abstract structure (template) as the target query. It is a binary accuracy metric, reflecting how often the structural intent is correctly captured.

Fragment F-score. We extract and compare the sets of schema-specific fragments including, tables, columns, functions, and literals, between predicted and true queries. F_1 is computed for each type, providing insight into how well the model fills in database-specific details.

6.2.2 Results and Analysis

We now present the empirical results across all metrics. Tables 6.2, 6.3, and 6.4 report template match accuracy, fragment-level F_1 scores, and normalized edit distance, respectively. Together, these results show how different methods perform across both structural and content-level dimensions of next-query recommendation.

Template Match Accuracy. Table 6.2 shows that our method achieves the highest accuracy (0.57) in correctly predicting the structure of the next query. This reflects the effectiveness of our transition-aware retrieval stage in selecting plausible templates based on session context. The sequence-aware deep learning (0.51)

also performs competitively. The naive Q_i baseline achieves moderate performance (0.43), as users often repeat similar structures, while the generative LLM performs poorly (0.28) due to its lack of structural inductive bias.

Table 6.2: Template Match Accuracy

Method	Template Accuracy
Generative LLM (No Guidance)	0.28
Sequence-Aware Deep Learning	0.51
Collaborative Filtering (QueRIE)	–
Naive Q_i	0.43
Ours (RAG)	0.57

Fragment-Level F_1 Scores. Table 6.3 reports F_1 scores for table names, column names, functions, and literals. Our method achieves the highest scores for functions (0.74) and tables (0.66), highlighting its strength in capturing the query structure and user’s intent. This performance reflects the advantage of RAG in modeling both the overall query structure and session-specific elements, particularly for fragments like functions that are closely tied to query intent.

On column prediction, our model scores 0.61, slightly lower than the naive Q_i baseline (0.68), which benefits from the high likelihood of repeated column usage across consecutive queries. In contrast, our approach generalized beyond repetition by conditioning generation on session context, with introduces more variation but improves adaptability.

Our method also performs competitively on literals (0.52), which are harder to infer and often user-specific. The sequence-aware transformer achieves slightly better performance on columns (0.66) and literals (0.54), likely due to its explicit fragment-level training stage, which helps with frequently occurring but positionally variable elements like column names and literals. However, this comes at the cost of flexibility across more complex or less repetitive fragment types.

Collaborative filtering performs poorly across all categories, highlighting its limitations in modeling structured, intent-driven queries. Finally, the unguided

generative LLM struggles across the board, reinforcing the importance of structural and contextual guidance in accurate SQL generation.

Table 6.3: Fragment F₁ Scores by Fragment Type

Method	Table	Column	Function	Literal
Generative LLM (No Guidance)	0.60	0.45	0.42	0.35
Sequence-Aware Deep Learning	0.64	0.66	0.68	0.54
Collaborative Filtering (QueRIE)	0.16	0.24	0.25	0.06
Naive Q_i	0.61	0.68	0.54	0.40
Ours (RAG)	0.66	0.61	0.74	0.52

Edit Distance. Table 6.4 presents normalized edit distance between predicted and target queries. While the naive Q_i baseline achieves the lowest score (0.60), this is due to conservative predictions that closely resemble the input. Our model performs better than the generative LLM (0.63 vs. 0.73), demonstrating that retrieval-guided generation leads to more grounded and consistent outputs. Although our method makes more structural edits than naive repetition, it does so in a way that improves template and fragment accuracy, offering a better balance between change and correctness.

While the generative LLM baseline is capable of producing fluent and complete SQL queries, its lack of structural grounding often leads it to deviate significantly from the user’s original intent. This is reflected in its high normalized edit distance (0.73), indicating that the predicted queries differ substantially from the ground truth. In contrast, the naive Q_i baseline achieves the lowest edit distance (0.60), underscoring that many consecutive queries in SQLShare are only subtly modified. This highlights the nature of the task: effective recommendations often require minimal but meaningful changes to previous queries. Our method, which combines retrieval and generation, strikes a balance between these extremes. It avoids blindly copying prior queries while ensuring that generated outputs remain closely aligned with likely user intent. The result is a recommendation model that makes informed structural changes without drifting semantically from the session’s trajectory.

Table 6.4: Normalized Average Edit Distance (lower is better)

Method	Edit Distance
Generative LLM (No Guidance)	0.73
Naive Q_i	0.60
Ours (RAG)	0.63

Overall, our transition-aware retrieval-augmented framework outperforms all baselines in structural accuracy and fragment-level detail, while maintaining strong end-to-end fidelity. These results validate the value of combining template retrieval with LLM-based generation, especially in diverse and schema-rich environments like SQLShare.

Chapter 7

Conclusion and Future Work

In this thesis, we proposed a transition-aware retrieval-augmented generation (RAG) framework for recommending the next SQL query in a user session. By combining neural dual encoders for template retrieval, auxiliary transition classifiers, and large language models for full SQL query generation, our approach effectively bridges structural understanding with expressive query synthesis.

We demonstrated that modeling structural transitions as auxiliary tasks improves both retrieval accuracy and the consistency of generated queries. Rather than learning to directly output query fragments, our method defers fragment generation to a pretrained language model, leveraging the fact that most fragments used in the next query are already present in recent ones. Additionally, we showed that conditioning the system on only the last two queries of a session is sufficient to produce coherent and relevant recommendations, balancing performance with practical input size constraints.

Although our experiments were conducted solely on the SQLShare dataset, this choice enabled a fair comparison with existing baselines and offered a diverse setting with 2,697 sessions across 64 unique datasets. This diversity makes SQLShare a strong proxy for multi-schema environments, and our findings suggest that the proposed system generalizes well across various user and schema behaviors.

While the presented system addresses key challenges in SQL query recommendation, several opportunities remain for future exploration:

- **Incorporating natural language instructions during generation.** In our

current design, the language model generates the next SQL query conditioned on a structured query template retrieved from past examples. While this approach ensures structural consistency, it limits flexibility in how user intent is expressed. An alternative would be to augment the template with natural language instructions, such as "add a filter for recent dates" or "remove the GROUP BY clause". This would shift the generation step from structure-guided to instruction-guided SQL synthesis.

- **Advancing template retrieval.** Future versions may benefit from specialized encoders pretrained on SQL-specific structural patterns or multi-task training objectives that better align embedding space with query transitions.
- **Incorporating longer query histories.** Limiting the model to the last two queries is a reasonable design choice under computation constraints, but future iterations could integrate more context using session-aware encoders or memory-augmented architectures.
- **Expanding to multiple datasets.** Although SQLShare is multi-schema, incorporating additional benchmarks would enable direct performance comparisons in single-schema environments and improve the system's domain coverage.

In conclusion, this work highlights the potential of combining retrieval, structural supervision, and generative models for intelligent SQL query recommendation. While grounded in practical constraints, it opens the door for richer and more interactive data exploration systems built on structured query understanding.

Bibliography

- [1] Query auto-completion. In R. Alhajj and J. G. Rokne, editors, *Encyclopedia of Social Network Analysis and Mining, 2nd Edition*. Springer, 2018. doi:10.1007/978-1-4939-7131-2_100934. URL https://doi.org/10.1007/978-1-4939-7131-2_100934. → page 15
- [2] B. Ampel, C.-H. Yang, J. Hu, and H. Chen. Large language models for conducting advanced text analytics information systems research. *ACM Trans. Manage. Inf. Syst.*, 16(1), Feb. 2025. ISSN 2158-656X. doi:10.1145/3682069. URL <https://doi.org/10.1145/3682069>. → page 27
- [3] N. Arzamasova and K. Böhm. Scalable and data-aware SQL query recommendations. *Information Systems*, 96:101646, 2021. ISSN 0306-4379. doi:<https://doi.org/10.1016/j.is.2020.101646>. URL <https://www.sciencedirect.com/science/article/pii/S0306437920301101>. → page 21
- [4] L. Basyal and M. Sanghvi. Text summarization using large language models: A comparative study of mpt-7b-instruct, falcon-7b-instruct, and openai chat-gpt models. *ArXiv*, abs/2310.10449, 2023. URL <https://api.semanticscholar.org/CorpusID:264146201>. → page 27
- [5] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf. → pages 10, 11, 12, 28

- [6] G. Chatzopoulou, M. Eirinaki, and N. Polyzotis. Query recommendations for interactive database exploration. pages 3–18, 06 2009. ISBN 978-3-642-02278-4. [doi:10.1007/978-3-642-02279-1_2](https://doi.org/10.1007/978-3-642-02279-1_2). → page 2
- [7] F. Chiarello, V. Giordano, I. Spada, S. Barandoni, and G. Fantoni. Future applications of generative large language models: A data-driven case study on chatgpt. *Technovation*, 133:103002, 2024. ISSN 0166-4972. [doi:https://doi.org/10.1016/j.technovation.2024.103002](https://doi.org/10.1016/j.technovation.2024.103002). URL <https://www.sciencedirect.com/science/article/pii/S016649722400052X>. → page 27
- [8] M. Dehghani, S. Rothe, E. Alfonseca, and P. Fleury. Learning to attend, copy, and generate for session-based query suggestion. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, CIKM '17*, page 1747–1756, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450349185. [doi:10.1145/3132847.3133010](https://doi.org/10.1145/3132847.3133010). URL <https://doi.org/10.1145/3132847.3133010>. → pages 4, 15, 18, 28, 31, 40
- [9] J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In J. Burstein, C. Doran, and T. Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019. [doi:10.18653/v1/n19-1423](https://doi.org/10.18653/v1/n19-1423). URL <https://doi.org/10.18653/v1/n19-1423>. → page 10
- [10] J. Dodge, G. Ilharco, R. Schwartz, A. Farhadi, H. Hajishirzi, and N. A. Smith. Fine-tuning pretrained language models: Weight initializations, data orders, and early stopping. *ArXiv*, abs/2002.06305, 2020. URL <https://api.semanticscholar.org/CorpusID:211132951>. → page 11
- [11] M. Douze, A. Guzhva, C. Deng, J. Johnson, G. Szilvasy, P.-E. Mazaré, M. Lomeli, L. Hosseini, and H. Jégou. The faiss library. 2024. → page 37
- [12] M. Eirinaki, S. Abraham, N. Polyzotis, and N. Shaikh. Querie: Collaborative database exploration. *Knowledge and Data Engineering, IEEE Transactions on*, 26:1778–1790, 07 2014. [doi:10.1109/TKDE.2013.79](https://doi.org/10.1109/TKDE.2013.79). → pages 2, 22, 44
- [13] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, and M. Zhou. CodeBERT: A pre-trained model for

- programming and natural languages. In T. Cohn, Y. He, and Y. Liu, editors, *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1536–1547, Online, Nov. 2020. Association for Computational Linguistics. doi:10.18653/v1/2020.findings-emnlp.139. URL <https://aclanthology.org/2020.findings-emnlp.139/>. → pages 33, 45
- [14] C. Guo, Z. Tian, J. Tang, S. Li, Z. Wen, K. Wang, and T. Wang. Retrieval-augmented gpt-3.5-based text-to-sql framework with sample-aware prompting and dynamic revision chain. In B. Luo, L. Cheng, Z. Wu, H. Li, and C. Li, editors, *Neural Information Processing - 30th International Conference, ICONIP 2023, Changsha, China, November 20-23, 2023, Proceedings, Part VI*, volume 14452 of *Lecture Notes in Computer Science*, pages 341–356. Springer, 2023. doi:10.1007/978-981-99-8076-5_25. URL <https://doi.org/10.1007/978-981-99-8076-5.25>. → page 26
- [15] J. Guo, Z. Zhan, Y. Gao, Y. Xiao, J.-G. Lou, T. Liu, and D. Zhang. 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*, pages 4524–4535, Florence, Italy, July 2019. Association for Computational Linguistics. doi:10.18653/v1/P19-1444. URL <https://aclanthology.org/P19-1444>. → page 24
- [16] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, nov 1997. ISSN 0899-7667. doi:10.1162/neco.1997.9.8.1735. URL <https://doi.org/10.1162/neco.1997.9.8.1735>. → page 8
- [17] S. Jain, D. Moritz, D. Halperin, B. Howe, and E. Lazowska. Sqlshare: Results from a multi-year sql-as-a-service experiment. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD ’16*, page 281–293, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450335317. doi:10.1145/2882903.2882957. URL <https://doi.org/10.1145/2882903.2882957>. → page 41
- [18] J.-Y. Jiang and W. Wang. Rin: Reformulation inference network for context-aware query suggestion. CIKM ’18, page 197–206, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450360142. doi:10.1145/3269206.3271808. URL <https://doi.org/10.1145/3269206.3271808>. → page 15
- [19] M. I. Jordan. Serial order: a parallel distributed processing approach.

technical report, june 1985-march 1986. 5 1986. URL
<https://www.osti.gov/biblio/6910294>. → page 6

- [20] J. Kaddour, J. Harris, M. Mozes, H. Bradley, R. Raileanu, and R. McHardy. Challenges and applications of large language models. *ArXiv*, abs/2307.10169, 2023. URL
<https://api.semanticscholar.org/CorpusID:259982665>. → page 27
- [21] E. Y. Lai, Z. Zolaktaf, M. Milani, O. AlOmeir, J. Cao, and R. Pottinger. Workload-aware query recommendation using deep learning. In J. Stoyanovich, J. Teubner, N. Mamoulis, E. Pitoura, and J. Mühlig, editors, *Proceedings 26th International Conference on Extending Database Technology, EDBT 2023, Ioannina, Greece, March 28-31, 2023*, pages 53–65. OpenProceedings.org, 2023. doi:10.48786/edbt.2023.05. URL
<https://doi.org/10.48786/edbt.2023.05>. → pages 2, 4, 22, 28, 29, 30, 31, 40, 43
- [22] J. Lehmann, A. Meloni, E. Motta, F. Osborne, D. R. Recupero, A. Salatino, S. Vahdati, T. ScaDS.AI, Dresden, and De. Large language models for scientific question answering: An extensive analysis of the sciq benchmark. In *Extended Semantic Web Conference*, 2024. URL
<https://api.semanticscholar.org/CorpusID:269767509>. → page 27
- [23] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer. BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In D. Jurafsky, J. Chai, N. Schluter, and J. R. Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 7871–7880. Association for Computational Linguistics, 2020. doi:10.18653/v1/2020.acl-main.703. URL
<https://doi.org/10.18653/v1/2020.acl-main.703>. → page 10
- [24] P. S. H. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. Yih, T. Rocktäschel, S. Riedel, and D. Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL
<https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html>. → pages 2, 12, 25

- [25] M. Li, X. Lin, X. Chen, J. Chang, Q. Zhang, F. Wang, T. Wang, Z. Liu, W. Chu, D. Zhao, and R. Yan. Keywords and instances: A hierarchical contrastive learning framework unifying hybrid granularities for text generation. In S. Muresan, P. Nakov, and A. Villavicencio, editors, *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4432–4441, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi:10.18653/v1/2022.acl-long.304. URL <https://aclanthology.org/2022.acl-long.304>. → page 25
- [26] Q. Lyu, K. Chakrabarti, S. Hathi, S. Kundu, J. Zhang, and Z. Chen. Hybrid ranking network for text-to-SQL. *CoRR*, abs/2008.04759, 2020. URL <https://arxiv.org/abs/2008.04759>. → page 24
- [27] V. V. Meduri, K. Chowdhury, and M. Sarwat. Evaluation of machine learning algorithms in predicting the next SQL query from the future. *ACM Trans. Database Syst.*, 46(1), mar 2021. ISSN 0362-5915. doi:10.1145/3442338. URL <https://doi.org/10.1145/3442338>. → page 22
- [28] D. Mienye and N. Jere. Deep learning for credit card fraud detection: A review of algorithms, challenges, and solutions. *IEEE Access*, PP:1–1, 01 2024. doi:10.1109/ACCESS.2024.3426955. → pages x, 8
- [29] I. D. Mienye, T. G. Swart, and G. Obaido. Recurrent neural networks: A comprehensive review of architectures, variants, and applications. *Information*, 15(9), 2024. ISSN 2078-2489. doi:10.3390/info15090517. URL <https://www.mdpi.com/2078-2489/15/9/517>. → pages x, 7, 8
- [30] A. Mustar, S. Lamprier, and B. Piwowarski. Using BERT and BART for query suggestion. In *Joint Conference of the Information Retrieval Communities in Europe*, 2020. → pages 15, 19
- [31] A. Mustar, S. Lamprier, and B. Piwowarski. On the study of transformers for query suggestion. *ACM Trans. Inf. Syst.*, 40(1), oct 2021. ISSN 1046-8188. doi:10.1145/3470562. URL <https://doi.org/10.1145/3470562>. → page 15
- [32] OpenAI. Chatgpt. <https://chat.openai.com>, 2025. Accessed on April, 2025. → page 9
- [33] OpenAI, :, A. Hurst, A. Lerer, A. P. Goucher, A. Perelman, A. Ramesh, A. Clark, A. Ostrow, A. Welihinda, A. Hayes, A. Radford, A. Madry, A. Baker-Whitcomb, A. Beutel, A. Borzunov, A. Carney, A. Chow, A. Kirillov, A. Nichol, A. Paino, A. Renzin, A. T. Passos, A. Kirillov,

A. Christakis, A. Conneau, A. Kamali, A. Jabri, A. Moyer, A. Tam,
A. Crookes, A. Tootoochian, A. Tootoonchian, A. Kumar, A. Vallone,
A. Karpathy, A. Braunstein, A. Cann, A. Codispoti, A. Galu, A. Kondrich,
A. Tulloch, A. Mishchenko, A. Baek, A. Jiang, A. Pelisse, A. Woodford,
A. Gosalia, A. Dhar, A. Pantuliano, A. Nayak, A. Oliver, B. Zoph,
B. Ghorbani, B. Leimberger, B. Rossen, B. Sokolowsky, B. Wang, B. Zweig,
B. Hoover, B. Samic, B. McGrew, B. Spero, B. Giertler, B. Cheng,
B. Lightcap, B. Walkin, B. Quinn, B. Guarraci, B. Hsu, B. Kellogg,
B. Eastman, C. Lugaresi, C. Wainwright, C. Bassin, C. Hudson, C. Chu,
C. Nelson, C. Li, C. J. Shern, C. Conger, C. Barette, C. Voss, C. Ding, C. Lu,
C. Zhang, C. Beaumont, C. Hallacy, C. Koch, C. Gibson, C. Kim, C. Choi,
C. McLeavey, C. Hesse, C. Fischer, C. Winter, C. Czarnecki, C. Jarvis,
C. Wei, C. Koumouzelis, D. Sherburn, D. Kappler, D. Levin, D. Levy,
D. Carr, D. Farhi, D. Mely, D. Robinson, D. Sasaki, D. Jin, D. Valladares,
D. Tsipras, D. Li, D. P. Nguyen, D. Findlay, E. Oiwoh, E. Wong, E. Asdar,
E. Proehl, E. Yang, E. Antonow, E. Kramer, E. Peterson, E. Sigler,
E. Wallace, E. Brevdo, E. Mays, F. Khorasani, F. P. Such, F. Raso, F. Zhang,
F. von Lohmann, F. Sulit, G. Goh, G. Oden, G. Salmon, G. Starace,
G. Brockman, H. Salman, H. Bao, H. Hu, H. Wong, H. Wang, H. Schmidt,
H. Whitney, H. Jun, H. Kirchner, H. P. de Oliveira Pinto, H. Ren, H. Chang,
H. W. Chung, I. Kivlichan, I. O'Connell, I. O'Connell, I. Osband, I. Silber,
I. Sohl, I. Okuyucu, I. Lan, I. Kostrikov, I. Sutskever, I. Kanitscheider,
I. Gulrajani, J. Coxon, J. Menick, J. Pachocki, J. Aung, J. Betker, J. Crooks,
J. Lennon, J. Kiros, J. Leike, J. Park, J. Kwon, J. Phang, J. Teplitz, J. Wei,
J. Wolfe, J. Chen, J. Harris, J. Varavva, J. G. Lee, J. Shieh, J. Lin, J. Yu,
J. Weng, J. Tang, J. Yu, J. Jang, J. Q. Candela, J. Beutler, J. Landers,
J. Parish, J. Heidecke, J. Schulman, J. Lachman, J. McKay, J. Uesato,
J. Ward, J. W. Kim, J. Huizinga, J. Sitkin, J. Kraaijeveld, J. Gross, J. Kaplan,
J. Snyder, J. Achiam, J. Jiao, J. Lee, J. Zhuang, J. Harriman, K. Fricke,
K. Hayashi, K. Singhal, K. Shi, K. Karthik, K. Wood, K. Rimbach, K. Hsu,
K. Nguyen, K. Gu-Lemberg, K. Button, K. Liu, K. Howe, K. Muthukumar,
K. Luther, L. Ahmad, L. Kai, L. Itow, L. Workman, L. Pathak, L. Chen,
L. Jing, L. Guy, L. Fedus, L. Zhou, L. Mamitsuka, L. Weng, L. McCallum,
L. Held, L. Ouyang, L. Feuvrier, L. Zhang, L. Kondraciuk, L. Kaiser,
L. Hewitt, L. Metz, L. Doshi, M. Aflak, M. Simens, M. Boyd,
M. Thompson, M. Dukhan, M. Chen, M. Gray, M. Hudnall, M. Zhang,
M. Aljube, M. Litwin, M. Zeng, M. Johnson, M. Shetty, M. Gupta,
M. Shah, M. Yatbaz, M. J. Yang, M. Zhong, M. Glaese, M. Chen, M. Janner,
M. Lampe, M. Petrov, M. Wu, M. Wang, M. Fradin, M. Pokrass, M. Castro,
M. O. T. de Castro, M. Pavlov, M. Brundage, M. Wang, M. Khan,

M. Murati, M. Bavarian, M. Lin, M. Yesildal, N. Soto, N. Gimelshein, N. Cone, N. Staudacher, N. Summers, N. LaFontaine, N. Chowdhury, N. Ryder, N. Stathas, N. Turley, N. Tezak, N. Felix, N. Kudige, N. Keskar, N. Deutsch, N. Bundick, N. Puckett, O. Nachum, O. Okelola, O. Boiko, O. Murk, O. Jaffe, O. Watkins, O. Godement, O. Campbell-Moore, P. Chao, P. McMillan, P. Belov, P. Su, P. Bak, P. Bakkum, P. Deng, P. Dolan, P. Hoeschele, P. Welinder, P. Tillet, P. Pronin, P. Tillet, P. Dhariwal, Q. Yuan, R. Dias, R. Lim, R. Arora, R. Troll, R. Lin, R. G. Lopes, R. Puri, R. Miyara, R. Leike, R. Gaubert, R. Zamani, R. Wang, R. Donnelly, R. Honsby, R. Smith, R. Sahai, R. Ramchandani, R. Huet, R. Carmichael, R. Zellers, R. Chen, R. Chen, R. Nigmatullin, R. Cheu, S. Jain, S. Altman, S. Schoenholz, S. Toizer, S. Miserendino, S. Agarwal, S. Culver, S. Ethersmith, S. Gray, S. Grove, S. Metzger, S. Hermani, S. Jain, S. Zhao, S. Wu, S. Jomoto, S. Wu, Shuaiqi, Xia, S. Phene, S. Papay, S. Narayanan, S. Coffey, S. Lee, S. Hall, S. Balaji, T. Broda, T. Stramer, T. Xu, T. Gogineni, T. Christianson, T. Sanders, T. Patwardhan, T. Cunninghamman, T. Degry, T. Dimson, T. Raoux, T. Shadwell, T. Zheng, T. Underwood, T. Markov, T. Sherbakov, T. Rubin, T. Stasi, T. Kaftan, T. Heywood, T. Peterson, T. Walters, T. Eloundou, V. Qi, V. Moeller, V. Monaco, V. Kuo, V. Fomenko, W. Chang, W. Zheng, W. Zhou, W. Manassra, W. Sheu, W. Zaremba, Y. Patil, Y. Qian, Y. Kim, Y. Cheng, Y. Zhang, Y. He, Y. Zhang, Y. Jin, Y. Dai, and Y. Malkov. Gpt-4o system card, 2024. URL <https://arxiv.org/abs/2410.21276>. → pages 43, 45

- [34] D. H. Park and R. Chiba. A neural language model for query auto-completion. In N. Kando, T. Sakai, H. Joho, H. Li, A. P. de Vries, and R. W. White, editors, *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, Shinjuku, Tokyo, Japan, August 7-11, 2017*, pages 1189–1192. ACM, 2017. doi:10.1145/3077136.3080758. URL <https://doi.org/10.1145/3077136.3080758>. → page 15
- [35] S. A. Puthiya Parambath, C. Anagnostopoulos, and R. Murray-Smith. Improving sequential query recommendation with immediate user feedback, 05 2022. → pages 16, 20
- [36] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. 2019. → pages 10, 11, 27
- [37] G. Ren, X. Ni, M. Malik, and Q. Ke. Conversational query understanding

using sequence to sequence modeling. *Proceedings of the 2018 World Wide Web Conference*, 2018. → pages 16, 17

- [38] D. E. Rumelhart and J. L. McClelland. *Learning Internal Representations by Error Propagation*, pages 318–362. 1987. → page 6
- [39] E. Rusak, P. Reizinger, A. Juhos, O. Bringmann, R. S. Zimmermann, and W. Brendel. Infonce: Identifying the gap between theory and practice. *ArXiv*, abs/2407.00143, 2024. URL <https://api.semanticscholar.org/CorpusID:270869455>. → pages 34, 45
- [40] S. U. Singh and A. S. Namin. A survey on chatbots and large language models: Testing and evaluation techniques. *Natural Language Processing Journal*, 10:100128, 2025. ISSN 2949-7191. doi:<https://doi.org/10.1016/j.nlp.2025.100128>. URL <https://www.sciencedirect.com/science/article/pii/S2949719125000044>. → page 27
- [41] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf. → page 8
- [42] B. Wang, R. Shin, X. Liu, O. Polozov, and M. Richardson. 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*, pages 7567–7578, Online, July 2020. Association for Computational Linguistics. doi:[10.18653/v1/2020.acl-main.677](https://doi.org/10.18653/v1/2020.acl-main.677). URL <https://aclanthology.org/2020.acl-main.677>. → page 24
- [43] D. Wang and H. Fang. Exploring query reformulation for conversational information seeking. In E. M. Voorhees and A. Ellis, editors, *Proceedings of the Twenty-Eighth Text REtrieval Conference, TREC 2019, Gaithersburg, Maryland, USA, November 13-15, 2019*, volume 1250 of *NIST Special Publication*. National Institute of Standards and Technology (NIST), 2019. URL https://trec.nist.gov/pubs/trec28/papers/udel_fang.C.pdf. → page 15
- [44] W. Wang, F. Wei, L. Dong, H. Bao, N. Yang, and M. Zhou. Minilm: deep self-attention distillation for task-agnostic compression of pre-trained

transformers. In *Proceedings of the 34th International Conference on Neural Information Processing Systems, NIPS '20*, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546. → pages 10, 33

- [45] X. Wang, F. Cheng, Y. Wang, K. Xu, J. Long, H. Lu, and H. Qu. Interactive data analysis with next-step natural language query recommendation. *ArXiv*, abs/2201.04868, 2022. URL <https://api.semanticscholar.org/CorpusID:245906166>. → page 16
- [46] X. Xu, C. Liu, and D. X. Song. SQLNet: Generating structured queries from natural language without reinforcement learning. *ArXiv*, abs/1711.04436, 2017. URL <https://api.semanticscholar.org/CorpusID:10746949>. → page 24
- [47] K. Zhang, X. Lin, Y. Wang, X. Zhang, F. Sun, J. Cen, H. Tan, X. Jiang, and H. Shen. Refsql: A retrieval-augmentation framework for text-to-sql generation. In H. Bouamor, J. Pino, and K. Bali, editors, *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, pages 664–673. Association for Computational Linguistics, 2023. doi:10.18653/V1/2023.FINDINGS-EMNLP.48. URL <https://doi.org/10.18653/v1/2023.findings-emnlp.48>. → page 25
- [48] W. Zhang, Y. Deng, B. Liu, S. Pan, and L. Bing. Sentiment analysis in the era of large language models: A reality check. In K. Duh, H. Gomez, and S. Bethard, editors, *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 3881–3906, Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi:10.18653/v1/2024.findings-naacl.246. URL <https://aclanthology.org/2024.findings-naacl.246/>. → page 27
- [49] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong, Y. Du, C. Yang, Y. Chen, Z. Chen, J. Jiang, R. Ren, Y. Li, X. Tang, Z. Liu, P. Liu, J.-Y. Nie, and J.-R. Wen. A survey of large language models, 2023. → page 9
- [50] J. Zhong, W. Guo, H. Gao, and B. Long. Personalized query suggestions. *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2020. → page 15
- [51] V. Zhong, C. Xiong, and R. Socher. Seq2SQL: Generating structured queries from natural language using reinforcement learning. *ArXiv*, abs/1709.00103, 2018. URL <https://api.semanticscholar.org/CorpusID:25156106>. → page 24

- [52] W. Zhu, H. Liu, Q. Dong, J. Xu, S. Huang, L. Kong, J. Chen, and L. Li. Multilingual machine translation with large language models: Empirical results and analysis. In K. Duh, H. Gomez, and S. Bethard, editors, *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 2765–2781, Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi:10.18653/v1/2024.findings-naacl.176. URL <https://aclanthology.org/2024.findings-naacl.176/>. → page 27