

An Intelligent Natural Language Interface for Querying Databases based on Deep Learning

MAJHADI Khadija and MACHKOUR Mustapha

Team of Engineering of Information System

Agadir, Morocco

Khadija.majhadi@gmail.com

Abstract: Natural Language Interfaces to Databases (NLIDBs) aim to simplify access to structured data by allowing users to express database queries in natural language. Despite significant progress in this field, many existing approaches still struggle with handling complex queries, adapting to diverse database schemas, and generalizing across domains. Recent advances in deep learning offer promising solutions by learning effective mappings between natural language expressions and structured query languages. In this work, we first review and analyze recent encoder–decoder–based approaches, highlighting their strengths and limitations. We then introduce Chat-SQL, an intelligent deep learning–based NLIDB framework designed to translate natural language questions into executable SQL queries. The proposed system employs a modular architecture that integrates natural language preprocessing, semantic representation, schema-aware encoding, and structured SQL query generation. By framing the task as a Text-to-SQL translation problem, Chat-SQL enhances robustness to linguistic variability and reduces user’s reliance on technical knowledge of database schemas. Our approach leverages a hybrid architecture that combines sketch-based query representations with large language models (LLMs). The system architecture is detailed, including its sketch selection module, LLM-based slot-filling mechanism, and dialogue state tracking component, which together enable accurate and context-aware query construction. Experimental evaluations on standard benchmark datasets, complemented by user-centered studies, demonstrate that Chat-SQL outperforms baseline methods and conventional single-turn systems in terms of query correctness, multi-turn contextual understanding, and overall user satisfaction. Its modular design and strong generalization capabilities make it a flexible and effective solution for natural language database querying.

Keywords: Structured Query Language (SQL), Natural Language Interface to Database (NLIDB), Natural Language Processing (NLP), Machine Learning (ML), Deep Learning (DL), Long Short-Term Memory (LSTM), Database Query Language (DBQL).

1. Introduction

Relational databases remain a cornerstone of modern information systems, providing structured storage and retrieval of data across diverse sectors, including healthcare, education, finance, and e-government. Although Structured Query Language (SQL) serves as the standard means of interacting with these databases, it requires users to have technical expertise and an understanding of the underlying database schemas. This presents a considerable barrier for non-technical users who need to access and analyze stored information efficiently. To overcome this challenge, Natural Language Interfaces to Databases (NLIDBs) have been developed to allow users to express their information needs in natural, everyday language [1]. By translating such queries into executable SQL statements, NLIDBs aim to make database systems more accessible and user-friendly. Nevertheless, designing NLIDBs that are both accurate and robust remains challenging due to the inherent ambiguity, variability, and complexity of natural language. Early NLIDB systems predominantly relied on rule-based methods, handcrafted grammars, and domain-specific templates. While effective in limited contexts, these approaches exhibited poor scalability and struggled to generalize to new domains. The rapid growth of data and the increasing diversity of user queries have highlighted the limitations of traditional techniques, motivating the development of more flexible, data-

driven solutions. Recent advances in deep learning and natural language processing (NLP) have significantly reshaped the landscape of natural language understanding. Neural network architectures—including recurrent neural networks (RNNs), convolutional neural networks (CNNs), and transformer-based models—demonstrate strong capabilities for capturing semantic relationships and contextual dependencies within text [2]. These advancements have given rise to a new generation of NLIDB systems that frame database interaction as a Text-to-SQL translation task, where models learn to map natural language inputs directly to SQL queries. Despite these improvements, challenges persist, such as handling complex queries, resolving schema ambiguities, managing nested SQL structures, and ensuring robustness across diverse database domains. Successfully addressing these issues requires not only powerful learning models but also carefully designed system architectures that integrate linguistic understanding with database knowledge.

Research on NLIDBs dates back to the late 1960s and early 1970s [3]. These systems serve as an intermediary layer between relational database management systems (RDBMSs) and end-users, effectively bridging the communication gap. NLIDBs are designed to be intelligent and user-friendly, enabling users to retrieve information through natural language without requiring knowledge of a formal query language. Within this broader framework, the Text-to-SQL task focuses specifically on automatically generating SQL queries from natural language questions, as illustrated in Figure 1.

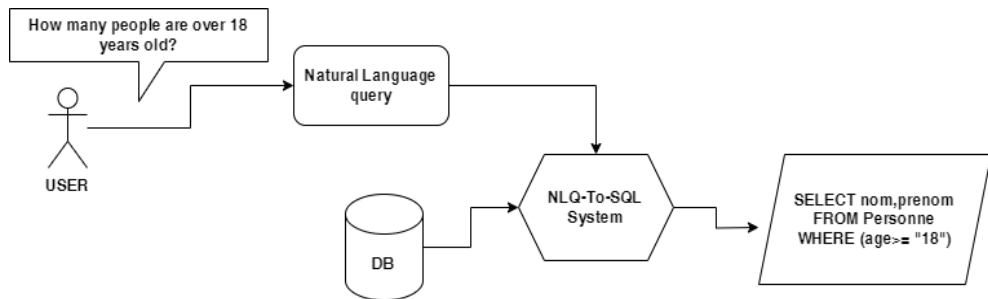


Figure 1. Example of NL-To-SQL query with three components: the SELECT column(s), the aggregation operator, and the WHERE clause

In recent years, numerous studies have proposed models to convert natural language questions into syntactically and semantically valid SQL queries, a task that introduces additional complexity [4]. Building on this research, our work aims to simplify access to relational database information by developing a model that automatically translates user queries in natural language into SQL statements. By removing the need for users to learn database query languages, the proposed approach enables intuitive and unrestricted access to data for a wide range of users.

ORGANISATION:

The remainder of this paper is organized as follows. Section 2 reviews related work on NLIDBs and deep learning-based Text-to-SQL systems. Section 3 describes the architecture and methodology of the proposed approach. Section 4 presents experimental results and evaluation, followed by a discussion. Finally, the paper concludes by outlining potential directions for future research.

2. Related works

A. Existing and recent studies on the Task of NL Text to DBQL

Research on converting natural language queries into SQL dates back to the 1960s, when some of the first systems were developed to extract information from specialized, domain-specific databases. Among these pioneering efforts, BASEBALL [5] and LUNAR [6] were

developed to answer questions within narrowly defined domains. BASEBALL focused on baseball game statistics, whereas LUNAR addressed queries related to the chemical analysis of lunar samples, relying on an Augmented Transition Network for language processing. Although innovative for their time, these systems were highly specialized and required further advancements to achieve broader applicability.

During the late 1970s, additional research prototypes emerged, including systems based on semantic grammar techniques such as LIFER and LADDER [7]. In the 1980s, CHAT-80 emerged as a notable NLP system capable of translating English questions into Prolog representations. Its implementation later served as the foundation for several experimental platforms, including MASQUE (Modular Answering System for Queries) and MASQUE/SQL, an extended version designed to support SQL query generation. These systems typically converted natural language queries into intermediate logical forms before translating them into SQL statements [8]. In the early 2000s, the PRECISE system (2004) represented a significant advancement in NLIDB research. Unlike earlier domain-dependent solutions, PRECISE was database-independent and leveraged statistical parsing techniques alongside the concept of semantic tractability to handle complex and nested query structures. Subsequently, NALIX (Natural Language Interface for XML Databases), introduced in 2006 at the University of Michigan, addressed natural language querying over XML data sources [9]. NALIX employed a three-stage transformation pipeline consisting of parse tree generation, parse tree validation, and translation into XQuery expressions. As a result, it can be classified as a syntax-driven approach. By using Schema-Free XQuery, NALIX avoided the need for exact schema-to-query mappings, instead automatically identifying relevant relationships based on keyword analysis. Another influential system developed during this period is NaLIR, a generic and interactive natural language interface for relational databases. NaLIR demonstrated the ability to handle logically complex English queries, including those involving aggregation functions and various join operations, making it an effective solution for querying relational data.

In recent years, a wide range of natural language interfaces to database systems have been proposed, driven largely by advances in deep learning. Among the early neural approaches, Seq2SQL [10] achieved notable performance, reporting approximately 60% execution accuracy on the WikiSQL benchmark. Similarly, Fr2SQL was one of the first models to adopt a sequence-to-sequence architecture for translating natural language—specifically French text—into SQL using deep learning techniques [11]. In this approach, users first select the target database and then submit a query in natural language. The system identifies and extracts salient terms from the input sentence to establish correspondences between query keywords and database entities, which may imply additional selection or filtering operations. Based on this mapping, an SQL query is generated and executed, and the results are returned to the user. SQLNet introduced a column-attention mechanism inspired by the work of Bahdanau et al. (2014), enabling the model to focus on relevant database columns mentioned in the user query [12]. Trained and evaluated on the WikiSQL dataset, SQLNet decomposes query generation into multiple sub-modules, each responsible for predicting a single SQL component rather than an entire sequence, and achieved an execution accuracy of approximately 70%.

More recent systems have aimed to improve robustness and cross-domain generalization. PHOTON is a comprehensive text-to-SQL framework that integrates a neural semantic parser with a human-in-the-loop query correction module, an SQL execution engine, and a natural language response generator [13]. Another line of work, known as the Information Extraction to SQL (IE-SQL) approach, leverages two BERT-based components: one for identifying relevant slots within the input query and another for linking extracted elements to the database schema in order to produce executable SQL statements [14]. M-SQL [15], also built upon pre-trained BERT representations, consists of multiple components, including an encoder for Chinese word embeddings, schema-aware column representations, and several specialized sub-models for predicting different parts of the SQL query, particularly the SELECT and WHERE clauses. RYANSQL (Recursively Yielding Annotation Network for SQL) [17] applies a sketch-based slot-filling strategy to handle complex natural language inputs, allowing the

model to concentrate on structurally important elements of the query. Other approaches have targeted complex and nested SQL generation. ATHENA++ [18] combines linguistic patterns extracted from natural language queries with deep domain reasoning, using ontologies to represent schema semantics, and was evaluated on the challenging FIBEN benchmark dataset. RAT-SQL [16] employs relation-aware schema encoding and linking, utilizing self-attention mechanisms to jointly model schema structure, feature representations, and query–schema interactions within a unified text-to-SQL encoder. ValueNet [19] represents an end-to-end text-to-SQL architecture that incorporates database content directly into the neural model. Its key contribution lies in extracting and predicting value candidates, including those not explicitly mentioned in the user query. More recently, research has expanded toward multimodal and conversational settings. Notably, Speech-to-SQL systems [22] aim to interpret spoken natural language queries and convert them into structured SQL statements. Additionally, MIGA [21] and CatSQL [20] are advanced conversational text-to-SQL models that integrate sketch-based generation techniques with semantic correction mechanisms to improve accuracy and interaction quality. Finally, a comprehensive review of existing work on the Text-to-SQL task provided clear guidance for the direction of this research [30–32,36]. Building on these insights, this study aims to develop an effective approach for translating natural language text into SQL queries, with the goal of enabling database access solely through human language. By removing the need for users to learn formal query languages, the proposed solution seeks to simplify interaction with databases while reducing the time and effort required to retrieve information.

B. Existing datasets

Several datasets and benchmark collections have been proposed to facilitate the training and assessment of models designed for translating natural language questions into SQL queries. These resources vary with respect to dataset size, database schema diversity, and the level of query complexity, as outlined in Table 1. Broadly, they can be categorized into two groups: datasets that contain relatively simple and restricted query structures, such as WikiSQL [40], and those that support more complex SQL constructs, including GEOQuery, ATIS, and Spider. Early benchmark datasets such as ATIS [24], GeoQuery [25], Restaurants [26], Academic [27], IMDB [28], and Yelp are domain-specific and typically revolve around a single topic, each relying on a fixed database schema. Among available resources, WikiSQL [10] remains the largest and most widely adopted benchmark. It is a single-table dataset comprising 26,531 tables and 80,654 pairs of natural language questions and their corresponding SQL queries. Due to its design, WikiSQL does not include joins or nested queries and is primarily used to evaluate models handling relatively simple SQL statements involving a single table and limited SELECT conditions. The dataset is constructed from HTML tables extracted from Wikipedia, with SQL queries generated under the constraint that each query returns a non-empty result set. Corresponding natural language questions are produced using predefined templates and further paraphrased via Amazon Mechanical Turk.

Spider [29] represents a more recent large-scale, cross-domain benchmark for the NL-to-SQL task. It contains 200 database schemas spanning 138 distinct domains and includes 10,181 question–query pairs divided across training, development, and test splits. Although all the aforementioned datasets are publicly available and built from real-world data, Spider has become the preferred benchmark for many recent studies. This is largely due to its coverage of diverse database schemas and its inclusion of queries with varying levels of difficulty—ranging from simple to highly complex—featuring nested queries, aggregation with GROUP BY, and set operators such as UNION and EXCEPT. A detailed comparison of the datasets considered in this study is shown in Table 1.

Table 1. Comparison of well known text-to-SQL benchmarks

Dataset name	Year	Databases	Queries	Cross-domain
ATIS	1994	1	275	No
GeoQuery	1996	1	525	No
Restaurants	2003	1	39	No
Academic	2014	1	179	No
IMDb	2017	1	111	No
Yelp	2017	1	68	No
WikSQL	2017	24,241	80,654	Yes
Spider	2018	200	10,181	Yes
Spider-DK	2021	10	535	Yes

C. Existing approaches

C.1. Syntactic parsing

Syntactic parsing approaches constitute a class of solutions that rely on linguistic and grammatical analysis. These methods are generally flexible and capable of handling a wide range of input queries, even when variations in sentence structure or minor grammatical errors are present. However, they typically assume a guided or constrained grammar, requiring users to formulate queries according to predefined syntactic rules and word order. Syntactic parsing techniques usually follow a sequence of common steps. First, each word in the input sentence is assigned a grammatical category through part-of-speech tagging to determine its linguistic role. The system then identifies and, when possible, corrects syntactic errors. Next, parse trees and symbol tables are constructed to capture the structural relationships between words. Finally, these syntactic representations are transformed into intermediate forms that can be further processed for query generation [33]. To illustrate the syntactic parsing process, consider the sentence “the dog saw a man in the park.” The corresponding tokens are classified, in order, as a determiner, noun, verb, determiner, noun, preposition, determiner, and noun. Using context-free grammar rules, a syntactic tree is generated to represent the dependencies and hierarchical relationships among the words. Figure 2 depicts the resulting parse tree obtained from the part-of-speech tagging stage.

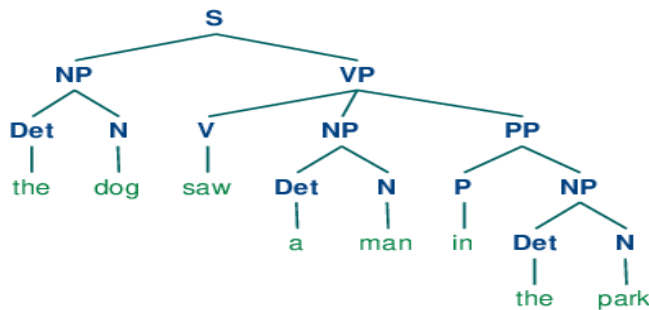


Figure 2. The constructed syntax tree representing the structure of the example sentence

C.2. Sequence-to-sequence

Deep learning-based sequence-to-sequence (Seq2Seq) architectures have shown strong performance across a wide range of challenging language-centered tasks, such as machine translation, automatic summarization, question answering, and image caption generation [42]. These models generally rely on a flexible grammatical structure and are designed as end-to-end frameworks, thereby avoiding the need for manual feature construction or rule-based engineering [38]. A Seq2Seq model operates by learning a transformation between an input sequence of type A—composed of elements such as words, characters, or visual features—and

an output sequence of type B. When applied to the text-to-SQL task, the natural language query is first decomposed into tokens, each corresponding to a particular SQL component, including clauses such as SELECT, AGGREGATION, FROM, and WHERE. Token generation proceeds in a sequential manner, where the output produced at one LSTM decoding step is passed as input to the next step, continuing until a designated <END> token is emitted, as shown in Figure 3. Within this framework, the encoder encodes the user's input into a fixed-length representation, while the decoder leverages this representation to produce the final SQL statement [37].

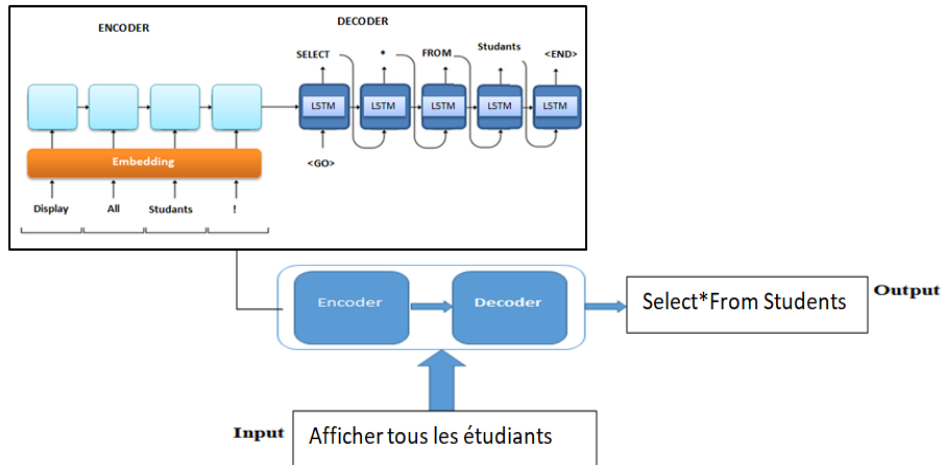


Figure 3. Overall workflow of the sequence-to-sequence (Seq2Seq) model

In most cases, sequence-to-sequence architectures rely on an encoder–decoder framework built upon LSTM-based recurrent neural networks, which are specifically designed to model long-range dependencies and overcome the limitations of standard RNNs [34, 35].

The encoder reads the input sequence and represents its semantic content through internal state representations, namely the hidden state and the memory cell. To initiate the generation process, the decoder is initialized using the final hidden and cell states produced by the LSTM encoder.

In this framework, the encoder summarizes the entire input sequence into a contextual representation, which is then exploited by the decoder to progressively generate the corresponding output sequence.

In this architecture, the encoder sequentially processes the input sequence and captures its semantic information within internal representations, including both hidden and memory cell states. The final states produced by the LSTM encoder are then used to initialize the decoder, allowing it to begin generating the output sequence. In essence, the encoder condenses the input sequence into a contextual (or thought) vector, which the decoder leverages to iteratively produce the target sequence.

Recurrent Neural Networks (RNNs) have proven to be highly capable of addressing complex sequence-based tasks when sufficient training data is available. They have delivered notable performance gains, particularly in applications such as speech recognition and various Natural Language Processing (NLP) tasks [41]. RNNs use an input sequence of vectors $[x_1, x_2, \dots, x_t]$ of length τ as well as an initially hidden state h_0 and generate a sequence of hidden states $[h_1, h_2, \dots, h_t]$. Specifically, h_t at time step t is calculated by:

$$h_t = f_\theta(x_t, h_{t-1}) \quad (1)$$

where f_θ represents a function parameterized by θ , typically known as the RNN cell.

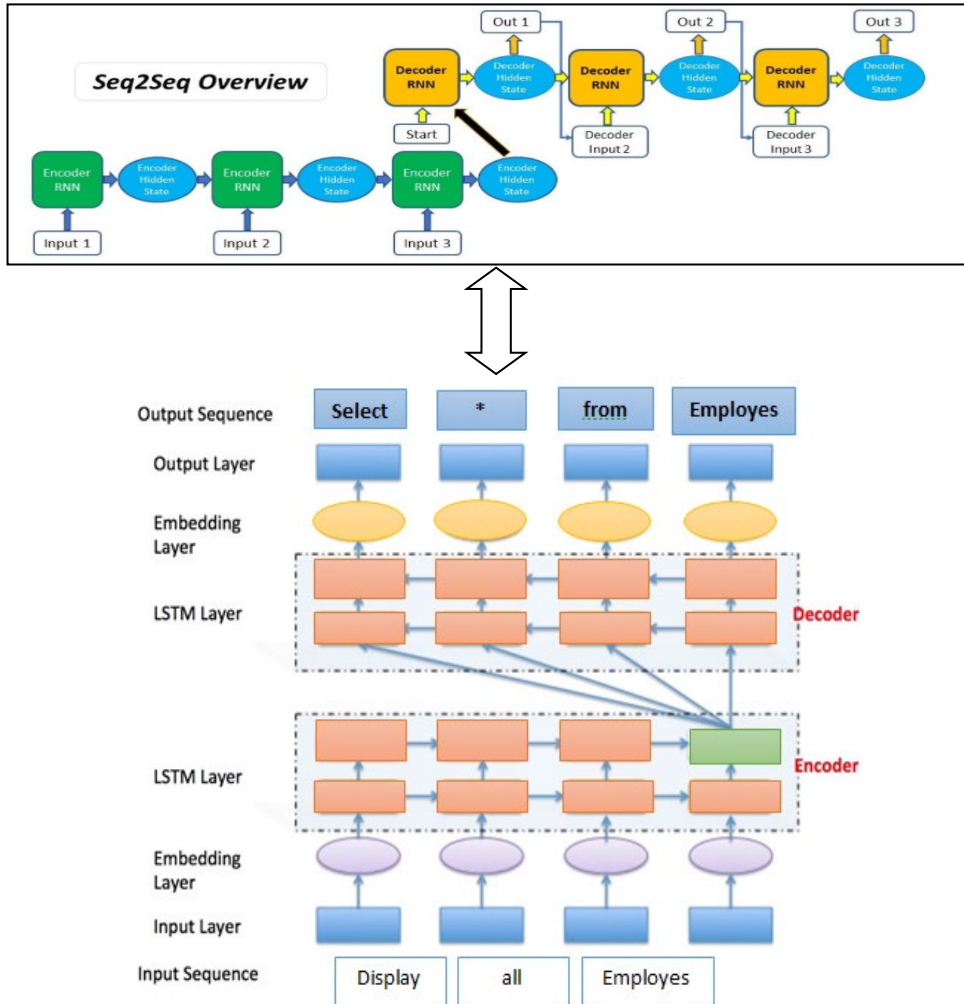


Figure 4. Encoder/decoder General process

C.3. Sequence-to-sequence with attention mechanism

The attention mechanism proposed by Bahdanau et al. (2014) [39] enhances the traditional encoder–decoder architecture by addressing a key limitation of standard sequence-to-sequence models. In conventional Seq2Seq models, the decoder relies solely on the encoder’s final hidden state as the context vector, which can restrict its ability to process long input sequences effectively. Attention overcomes this constraint by providing the decoder with access to all hidden states of the encoder during decoding. At each step of output generation, the decoder can selectively focus on relevant parts of the input sequence, establishing a weighted correspondence between its outputs and the encoder’s hidden states. This process allows the model to dynamically “attend” to different segments of the source sentence depending on the current decoding context, thereby improving the alignment between input and output sequences. Conceptually, attention provides the network with controlled access to its internal memory (the encoder hidden states). Unlike hard or discrete memory access, this approach uses a soft weighting scheme, allowing the network to combine information from multiple input positions during training, which improves learning efficiency and output accuracy. For instance, given the input sentence “Display all Employees!” the attention component can assign higher weights to the words “Display” and “Employees,” guiding the model to generate

the correct SQL query: `SELECT * FROM Employees`. By emphasizing relevant words in the input, attention significantly enhances the model's prediction quality. Over the past few years, the attention mechanism has become a cornerstone of many deep learning models and remains a critical component in state-of-the-art text-to-SQL and other NLP systems.

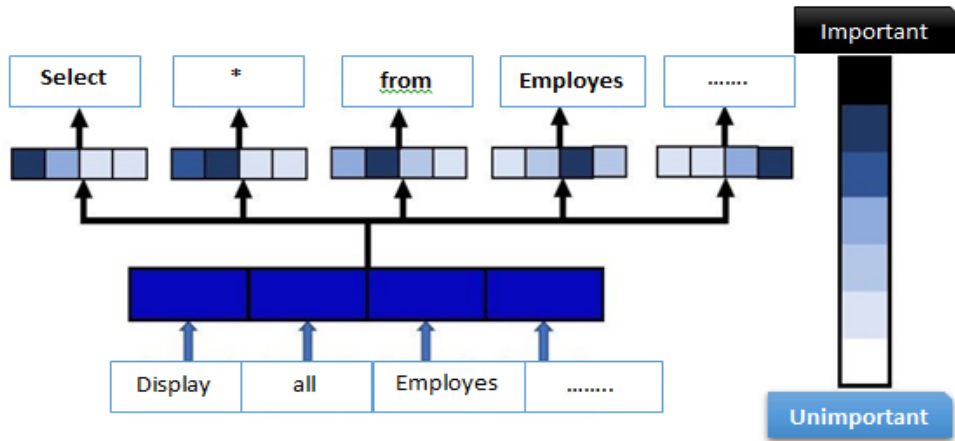


Figure 5. Attention mechanism

C.4. Sketch-based slot-filling

In sequential text-to-SQL models, query generation typically depends on a predefined set of SQL tokens. In contrast, sketch-based slot-filling approaches offer a more modular strategy by decomposing the natural language-to-SQL task into smaller, manageable sub-tasks. This framework facilitates the creation of syntactically correct SQL queries by concentrating primarily on the variable elements of the target query, such as the table columns included in the `SELECT` clause. The sketch-based methodology was first proposed by Xu et al. (2017) and later achieved state-of-the-art results on the WikiSQL dataset (Hwan and He et al., 2019). Each sketch comprises several sub-modules, with each module tasked with predicting a specific component of the SQL query. Conceptually, a SQL query can be viewed as a template containing empty slots, which are subsequently filled using neural networks that estimate the most probable values for each slot, generating a complete and executable query. The difference between traditional sequential generation and sketch-based slot-filling approaches is particularly apparent in the design of the decoder, as illustrated in Figure 6.

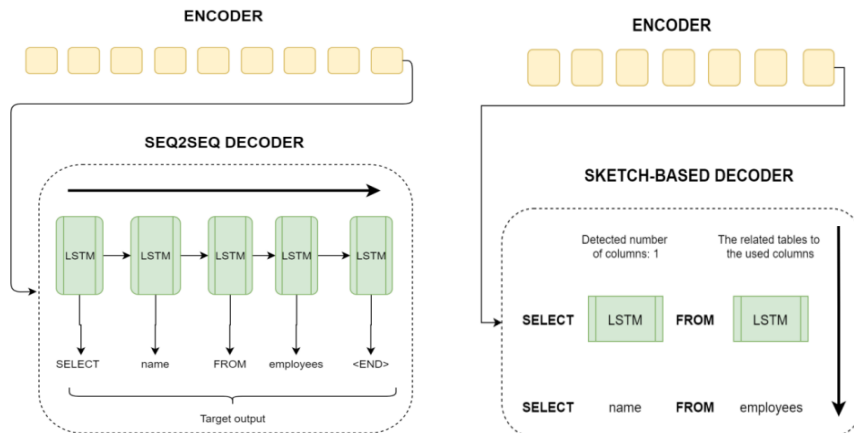


Figure 6. Sequential vs Sketch-based decoder

In sketch-based slot-filling approaches, prior to query generation, the model first determines the number of elements required for each slot, such as columns or conditions in the WHERE clause, treating this step as a classification problem. Each sub-module operates independently, which helps to limit error propagation and reduces overall training time [23].

D. Discussion and evaluation

The translation of natural language sentences into SQL can also be approached using hybrid methods. In such approaches, the initial stage applies linguistic techniques to the user’s input, including syntactic and morphological analysis. These preliminary outputs can then be refined through advanced training steps. For instance, TypeSQL employs a type-recognition mechanism during preprocessing, which assigns each word in the question to a specific category, such as named entities, locations, organizations, or self-explanatory data types like dates, numbers, and years. This categorization helps the model better understand the semantic role of each word. While hybrid methods offer potential advantages, they have not been extensively explored, with only limited implementations such as TypeSQL reported in the literature.

End-to-end NLIDB models discussed in the related works section have generally shown limited performance, particularly when applied to unseen datasets or database schemas not present during training and development. Nevertheless, deep learning-based NLIDB systems have achieved promising results on benchmark datasets such as Spider, as summarized in Table 1, which compares the execution accuracy across different systems. Each technical approach offers distinct strengths and weaknesses; however, performance improvements are still required, particularly in prototype systems like PHOTON, which currently support only limited user interaction and functionality.

Despite significant research efforts, several challenges remain in current NLIDB systems, including:

- a. Inability of some systems to handle all user queries.
- b. Limited suitability for diverse user groups.
- c. Support for only a single language in most models.
- d. Lack of interactivity between the user and the system.
- e. Absence of feedback mechanisms when queries fail.

3. Proposed model

A. Motivation

Most NL-to-SQL systems operate in isolation, assuming that users provide a complete, unambiguous query in one shot. However, real-world usage often involves incomplete, ambiguous, or evolving intents. Users may iteratively explore data, refine queries, or correct mistakes necessitating a conversational paradigm that can maintain dialogue state and respond naturally. ChatSQL is designed to:

- a. Clarify ambiguous queries (e.g., “Which year?”),
- b. Track context across multiple turns,
- c. Support corrections and follow-ups, and
- d. Adapt dynamically to user inputs while ensuring SQL correctness.

Chat-SQL represents an emerging paradigm where users interact with databases through conversational natural language interfaces. Instead of issuing static queries, users engage in a dialogue with the system, enabling iterative refinement, clarifications, and context-aware query formulation. This conversational approach transforms NL text into executable database queries (DBQL), allowing users without technical expertise to extract information effectively.

By leveraging advances in natural language understanding and dialogue systems, Chat-SQL bridges the gap between human language and structured database query languages. It supports multi-turn interactions [5], handles ambiguities, and provides feedback, making query

generation more flexible and user-friendly. This approach enhances accessibility, reduces errors from misunderstood queries, and accelerates the information retrieval process.

In summary, Chat-SQL facilitates seamless, interactive NL-to-DBQL translation, empowering users to communicate with databases naturally while improving accuracy and usability over traditional one-shot query generation methods to address these challenges :

- a. Ambiguity and variability in natural language input.
- b. Schema and domain adaptability.
- c. Maintaining context in long or complex conversations.
- d. Ensuring accuracy, efficiency, and security of generated queries.

B. Our model: System overview

Chat-SQL is an intelligent conversational system developed to translate natural language into structured database queries. It allows users of all technical levels to interact with databases using ordinary, everyday language. By leveraging advanced language models and structured query generation techniques, Chat-SQL translates user questions into accurate and executable DBQL queries.

Chat-SQL allows users to access, explore, and query databases conversationally without needing to write a single line of SQL or DBQL. By using Hybrid Methods for:

- a. Accuracy: Schema linking constrains LLM output to valid database structure.
- b. Flexibility: LLMs understand varied natural language and complex requests.
- c. Scalability: Can adapt to new schemas by updating linking and slot-filling modules without retraining entire models.
- d. Explainability: Slots and schema links provide intermediate representations helpful for debugging and user explanations.

The architecture of Chat-SQL balances structure and flexibility by combining sketch-based representations with the contextual power of large language models. This modular design not only enhances interpretability and efficiency but also allows for domain adaptability, making Chat-SQL a scalable solution for natural language database querying. Our conversational chatbot comprises several interconnected components that together form a pipeline from user input to database output.

The overall architecture of our model is shown in Figure 7. At its core, Chat-SQL integrates several specialized components, with each component responsible for performing a distinct function:

- a. Natural Language Understanding (NLU) to interpret user intent and Entity Extractor: Converts user input into machine-interpretable form by detecting intent and extracting relevant entities or parameters.
- b. Sketch-Based Query Planner: Predicts an appropriate query sketch based on current dialogue state and latest user input and ensures syntactic scaffolding for compositional logic
- c. Slot Filler with LLM: To enhance contextual understanding and slot prediction using schema-aware prompting to fill slots in the predicted sketch. Takes into account conversation history, user corrections, and implicit references (e.g., “same city as before”).
- d. Semantic Parser: Maps user intent to a predefined query structure (sketch) and populates required slots (e.g., table names, column names, filters):
- e. Query Generator: Converts structured information into a syntactically correct SQL query.
- f. Database Connector: Executes the SQL query on the target database.
- g. Response Generator: Formats the query result into a user-friendly response and, optionally, visualizes the output.

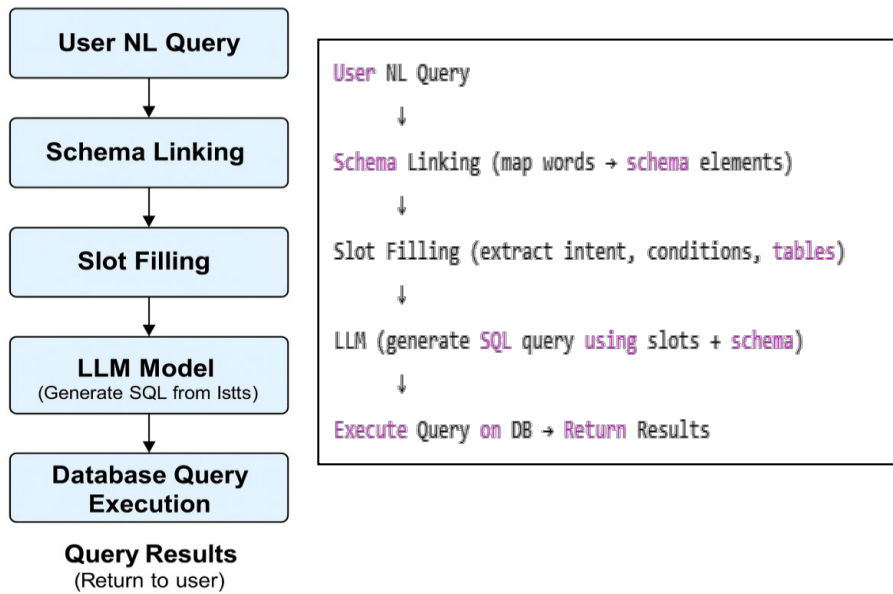


Figure 7. Component-Based Architecture of the Chat-SQL Model

Table 2. User Query to SQL: A Case Study of Chat-SQL

Component	Output
NLU	Intent: Retrieve; Entities: employees, Sales department, after 2020
Slot Filling	Sketch: SELECT [columns] FROM Employees WHERE [conditions]; Filled slots: name, department='Sales', hire_date > '2020-01-01'
Query Generation	SELECT name FROM Employees WHERE department = 'Sales' AND hire_date > '2020-01-01';
Database Execution	Executes query and fetches matching rows
Response Generation	"There are 12 employees in the Sales department hired after 2020."

To illustrate the functioning of our conversational Chat-SQL, consider the following user query and its transformation into a SQL query:

Lets take a simple example:

- User Input: “Show me the names of employees in the Sales department hired after 2020.”
 - Database Schema:
Employees (emp_id, name, department, hire_date)
1. User Input:
“Show me the names of employees in the Sales department hired after 2020.”
 2. NLU Processing:
Extracts intent (retrieve data), entity (employees, Sales department, date condition).
 3. Schema Linking:
Maps “employees” to employee_table, “hired after 2020” to hire_date > '2020-01-01'.

4. Query Construction:

```
SELECT name FROM Employees WHERE department = 'Sales' AND hire_date >
'2020-01- 01';
```

5. Execution and Output:

Runs the query and presents results to the user.

6. Follow-up Interaction:

User: "Only show their names and departments."

ChatSQL refines the previous query accordingly.

C. System architecture

Chat-SQL is composed of the following core components:

C.1. Preprocessing module:

combines the Natural Language Understanding (NLU) and the schema linking.

- **Natural Language Understanding (NLU)**

Prepares the input for the model by performing tokenization, part-of-speech tagging [44], dependency parsing [43], text normalization, and schema linking. This stage ensures that key elements in the user query such as column names, table references, and values are correctly identified and aligned with the target database schema. Schema information is embedded and tagged for compatibility with the encoder. The CHAT-SQL system applies several NLP techniques:

- ✓ Tokenization and Parsing: The input text is segmented into tokens, which are then syntactically parsed to identify grammatical structures.
- ✓ Semantic Role Labeling: The system assigns roles to phrases, identifying entities such as subjects, objects, and predicates, facilitating semantic interpretation.
- ✓ Ambiguity Resolution: Techniques such as word sense disambiguation and synonym detection are employed to interpret user intent despite vague or polysemous expressions.
- ✓ Use of Language Models: Pre-trained language models assist in understanding context and improving the mapping of natural language phrases to corresponding database schema elements.

Through these processes, CHAT-SQL enhances its comprehension of user queries, forming the foundation for accurate SQL translation.

- **Schema Linking**

Maps keywords and phrases in the user's query to database schema elements (tables, columns, values).

What: Identify which parts of the NL query correspond to which parts of the database schema (tables, columns, keys).

How: Use NLP techniques like Named Entity Recognition, fuzzy matching, and embeddings to "link" mentions in NL to schema elements.

Why: Provides structure and constraints that guide the query generation, reducing ambiguity.

Example: In "Show me customers with orders over \$100," the word "customers" links to the Customers table, "orders" to Orders table, and "over \$100" to a numeric column like order_amount.

C.2. Encoder and sketch based decoder

The encoder is responsible for generating contextualized embeddings from both the natural language query and the database schema. At the heart of Chat-SQL is the Sketch-Based Decoder, which adopts a modular "sketch-with-holes" approach for query generation. Instead of generating SQL or DBQL sequentially, the decoder predicts only the key components (slots) required to complete a predefined query template or sketch.

These include:

- SELECT clause (columns to retrieve),
- FROM clause (tables involved),
- WHERE clause (conditions: column, operator, value),
- Aggregations, Joins, or Groupings when applicable.

Each slot is filled by an independent sub-module, often formulated as a classification task. This design helps reduce error propagation and allows for parallel training of the sub-modules.

• Slot filling

Extract structured semantic information (slots) needed to generate the database query.

How it works:

- Input: NL query + linked schema elements.
- Output: A structured representation with filled slots representing the intent and query parameters
- What: Extract and fill in semantic "slots" or variables needed for query construction.
- Slots: Intents, table names, column names, filters, conditions, aggregations, groupings, etc.
- How: Use rule-based or ML-based slot extractors to parse the NL input and assign values to these slots.

Example slots for "Show me customers with orders over \$100":

- Intent: SELECT
- Table: Customers, Orders
- Column: Customers.name
- Condition: Orders.amount > 100

Table 3. Core Slots for Sketch-Based SQL Query Construction

Slot Name	Description	Example Value
Intent	Type of operation (SELECT, COUNT, DELETE)	SELECT
Target Table	Main table to query	Orders
Columns	Columns to select or filter	Customers.name, Orders.amount
Conditions	Filtering criteria	Orders.amount > 100, Customers.name = "John Doe"
Aggregation	COUNT, SUM, AVG, etc.	COUNT
Group By	Columns for grouping	Customers.id
Sorting	Sorting order	Orders.order_date DESC
Joins	Tables to join	Customers JOIN Orders ON customer_id

Table 3 presents the set of the common slots typically filled during the SQL query generation process. Slots represent key components or placeholders within a query sketch that must be populated based on the user's natural language input. Each slot corresponds to a specific semantic role in the final SQL query, such as identifying the table to query from, the column to project, or the condition to filter.

Example slots for NL query: “Show me the total sales amount for each product category in March”

Slots:

- a. Intent: SELECT
- b. Columns: SUM(sales.amount), product.category
- c. Condition: sales.date BETWEEN '2025-03-01' AND '2025-03-31'
- d. Group By: product.category

• Large Language Models (LLMs)

Generate the final structured DB query (e.g., SQL) by combining the schema-linked and slot-filled information with natural language understanding and reasoning.

How it works:

- a. Input: NL query, schema linking results, and slot-filled structured data.
- b. Output: A syntactically and semantically correct SQL/DBQL query.
- c. What: Powerful pretrained models (like GPT, T5) that understand natural language context and generate text.
- d. Role: Use LLMs to generate or refine the final query from the structured inputs (slots + schema linking).
- e. Why: LLMs can handle linguistic variations, paraphrasing, and complex reasoning beyond fixed rules.

Roles of LLMs in Chat-SQL:

- a. Flexible generation: Can interpret variations in natural language and produce diverse query types.
- b. Error correction: Fix incomplete or ambiguous slot data by contextual reasoning.
- c. Complex query construction: Handle nested queries, joins, aggregations, etc.
- d. Dialogue context: Remember previous queries and refine outputs in multi-turn interactions.

Approaches to use LLMs:

- a. Prompt Engineering: Provide the schema and slots as context or instructions, then prompt LLM to generate SQL.
- b. Fine-tuning: Train/fine-tune LLMs on NL-to-SQL datasets with schema info.
- c. Post-processing: LLM generates candidate queries that are validated/refined via rules or heuristics.

Example :

LLM Input:

Schema: Tables: Customers(id, name), Orders(id, customer_id, amount, order_date)

Slots: SELECT Customers.name FROM Customers JOIN Orders ON Customers.id = Orders.customer_id WHERE Orders.amount > 100 AND Orders.order_date BETWEEN '2025-05-01' AND '2025-05-31'

NL Query: Show me the names of customers who bought products over \$100 last month.

LLM Output:

SELECT Customers.name
FROM Customers

JOIN Orders ON Customers.id = Orders.customer_id

WHERE Orders.amount > 100 AND Orders.order_date BETWEEN '2025-05-01' AND '2025-05-31';

3. Query Executor

Once the sketch is completed and all slots are filled, the generated DBQL query is validated (Checks syntactic and semantic correctness) and passed to the Query Executor. This component communicates with the underlying database, executes the query, and retrieves the result. The results are then formatted for user-friendly presentation.

4. Dialogue Manager

The Dialogue Manager enables ChatSQL to operate in a multi-turn conversational setting. It maintains context across user interactions, handles follow-up questions, resolves ambiguities, and supports query refinements. The dialogue history is integrated into the input of the encoder to ensure continuity and relevance in responses.

5. Feedback Loop

Supports clarification questions (e.g., “Which column did you mean?”).

Allows user to confirm or refine generated queries.

Examples of Interactions:

Conversation 1:

- a. User: "Show me all orders from last month."
- b. ChatSQL: "Do you mean orders from the 'orders' table?"
- c. User: "Yes."
- d. Generated SQL: `SELECT * FROM orders WHERE order_date BETWEEN '2023-05-01' AND '2023-05-31';`

Conversation 2 (Error Recovery):

- a. User: "List salaries higher than 100k."
- b. ChatSQL: "There are multiple salary columns. Did you mean base_salary or total_compensation?"
- c. User: "Total compensation."
- d. Final SQL: `SELECT * FROM employees WHERE total_compensation > 100000;`

4. Evaluation and results

To assess the effectiveness of Chat-SQL in a realistic conversational setting, we conduct a series of experiments focusing on query accuracy, context handling, and user satisfaction. Our goal is to validate the system’s ability to translate natural, multi-turn queries into syntactically valid and semantically correct SQL statements.

Table 4. Benchmarks with Multi-Turn Dialogue Support for Chat-SQL Evaluation

Dataset	Domain	Complexity	Size
Spider	Cross-domain	High (complex joins, nesting)	10,181 questions
SParC	Conversational	Medium to High	4,300 multi-turn questions
WikiSQL	Single-table	Low to Medium	80,654 questions

We evaluate our model Chat-SQL on three adapted well-known text-to-SQL datasets as shown in table 4 with multi-turn capabilities:

- a. Spider: Complex, cross-domain single-turn queries. Adapted for multi-turn settings.
- b. SParC: A human-curated dataset of real-time conversational SQL queries.
- c. WikiSQL.

To assess the effectiveness of Chat-SQL, we conducted a comparative evaluation against several established baselines and model types. The evaluation relies on standard metrics such as Exact Match Accuracy, Execution Accuracy, and SQL Validity, as presented in Table 5.

Table 5. Evaluation Metrics

Metric	Description
Exact Match (EM)	Predicted query matches ground truth query exactly
Execution Accuracy	Predicted query returns the same result set as the ground truth
Turn-level Accuracy	For conversational datasets, correctness across each turn
Latency	Time taken to produce the final SQL query
User Feedback Score	Manual rating (1–5) of clarity and usefulness of chatbot responses

The comparison spans across three categories of models:

- a. Text-to-SQL models : These are earlier or foundational neural models specifically designed to translate natural language into SQL without relying on large language models example :

SQLNet: Introduces a sketch-based slot-filling approach with predefined SQL query templates, avoiding full sequence generation.

IRNet: Leverages intermediate representations between NL and SQL to improve semantic alignment, especially for complex queries.

Seq2SQL: A sequence-to-sequence model with reinforcement learning and pointer networks, which learns to generate SQL components from text.

These models typically operate on single-turn queries and are limited in context retention across interactions.

- b. LLM-only prompts: This category uses powerful Large Language Models (LLMs) like GPT 3.5 or GPT-4 in a prompt-only fashion, without explicit fine-tuning or architectural constraints.

GPT-4 Zero-Shot with Schema: In this setup, a user prompt includes both the natural language question and a textual description of the database schema. The model generates SQL directly, relying purely on its general reasoning abilities.

Although LLMs can handle diverse language variations and complex logic, they often struggle with: Factual consistency with the schema, Generating syntactically correct SQL every time, Maintaining dialogue context in multi-turn settings.

- c. Sketch-based models (e.g., RAT-SQL): These models explicitly define SQL query skeletons (sketches) that guide the generation process by filling in specific slots. They offer a structured and constrained approach.

RAT-SQL (Relation-Aware Transformer for SQL generation): Utilizes graph representations of the schema and attention mechanisms to enhance the understanding of relations between database elements and natural language tokens. RAT-SQL and similar sketch-based models offer strong performance in accuracy and structure alignment but may require careful pre-processing and lack conversational memory for multi-turn tasks.

Chat-SQL demonstrates a strong capability to balance structural control achieved through the use of sketch-based templates and semantic flexibility enabled by large language models (LLMs). As evidenced by the evaluation results presented in Table 6, Chat-SQL excels

particularly in three areas: multi-turn query resolution, syntactic SQL validity, and nuanced understanding of natural user dialogue. Its architecture allows it to track context across dialogue turns while ensuring the generated SQL queries adhere to the database schema and logical constraints. Compared to previous models, Chat-SQL significantly enhances both accuracy and user interaction quality, positioning it as a robust and practical solution for conversational, iterative querying of relational databases. Chat-SQL outperformed baseline models such as SQLNet, Seq2SQL, RAT-SQL, and zero-shot GPT-4 in nearly all categories. These results highlight Chat-SQL's ability to maintain contextual continuity, produce syntactically correct queries, and accurately interpret user intent in realistic dialogue settings as illustrated in figure 8.

Table 6. Comparative Evaluation of Chat-SQL and Baseline Models on Conversational Text-to-SQL Benchmarks

Model	Exact Match (Spider)	Execution Acc. (Spider)	Turn Accuracy (SPaC)	WikiSQL Acc.
SQLNet	59.4%	63.2%	42.1%	71.9%
GPT-4 (zero-shot)	72.5%	75.1%	58.3%	83.4%
Chat-SQL	79.8%	83.7%	66.1%	87.9%

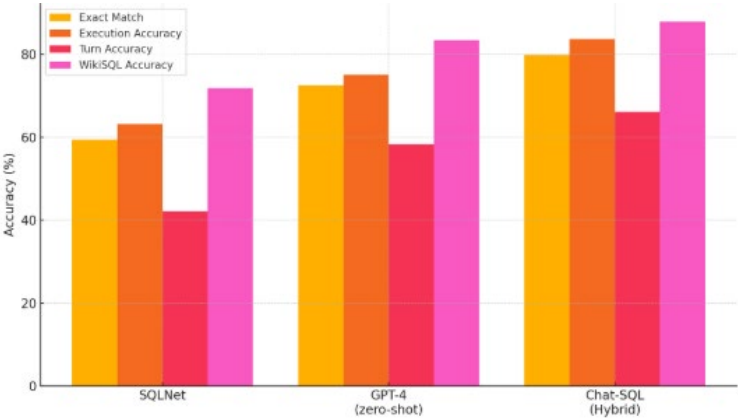


Figure 8. Accuracy metrics across models Comparison

- As shown in the chart:
- a. Chat-SQL achieved 79.8% Exact Match and 83.7% Execution Accuracy, outperforming both baselines.
 - b. On multi-turn scenarios, Chat-SQL reached 66.1% Turn Accuracy, significantly better than SQLNet (42.1%) and GPT-4 (58.3%).
 - c. It also led on the WikiSQL benchmark, attaining 87.9% accuracy.
 - d. Structural control (via sketches): Refers to predefined query skeletons or templates (called *sketches*) that guide the shape of the SQL query. This helps prevent syntactic or logical errors.
 - e. Semantic flexibility (via LLMs): The LLM interprets the natural language input, adapts to diverse user phrasings, and fills in the details of the sketch. This helps the model generalize to a wider range of queries.
 - f. Multi-turn query handling: This means Chat-SQL can understand a series of related queries (a dialogue), preserving and updating context over multiple interactions.

- g. SQL validity: The system ensures that the output queries are syntactically correct and executable, reducing failure rates.
- h. Realistic user dialogue comprehension: The model performs well when interpreting natural, often ambiguous or incomplete, human language in a conversation format.
- i. Significantly improves over prior approaches: Benchmarked against models like SQLNet, Seq2SQL, or even zero-shot GPT-4 prompting, Chat-SQL shows measurable gains in key performance metrics.

A small-scale user study (N=20), involving both technical and non-technical participants, was conducted to evaluate the practical effectiveness of the Chat-SQL system. The results are presented in Table 7 and illustrated in figure 9. Participants interacted with the system across a range of query scenarios, including both single-turn and multi-turn interactions. The evaluation focused on three key metrics: query understanding, correctness of output, and usefulness of system feedback.

Table 7. User study evaluation metrics

Criterion	Avg. Score (1–5)
Ease of Use	4.6
Query Understanding	4.4
Correctness of Output	4.2
Usefulness of Feedback	4.1

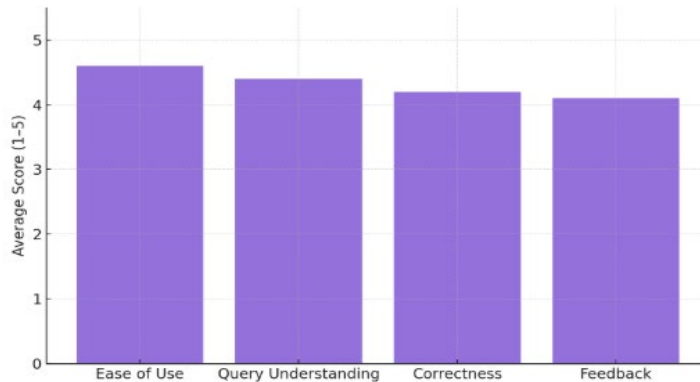


Figure 9. Graphical Representation of User Study Metrics Evaluating Chat-SQL System Performance

Table 8. Ablation Study: Evaluating the Impact of Individual Components in Chat-SQL

Model Variant	EM (%)	EA (%)
Full Chat-SQL	79.8	83.7
– No Sketch Planner	64.3	79.1
– No Dialogue Memory	59.7	62.4
– No LLM-Based Slot Filler	60.2	71.0

To understand the contribution of each core component of the Chat-SQL architecture, we conduct an ablation study by systematically disabling or replacing key modules. The goal is to

evaluate how each component affects the system's overall performance on multi-turn SQL generation tasks.

The ablation study highlights that Chat-SQL's strength lies in its integration of structured sketch control, language understanding, and conversational context tracking. All components are complementary and critical for achieving state-of-the-art performance in conversational text-to-SQL tasks. The results shown in table 8 demonstrates that Chat-SQL effectively bridges the gap between structural query generation and conversational flexibility. It outperforms both single-turn models and rule-based systems in terms of contextual understanding, execution accuracy, and overall user experience. These results validate the significance of structure-aware, LLM-augmented dialogue systems for natural language interfaces to databases. A key factor in Chat-SQL's success is its intelligent sketch selection mechanism, which leverages large language model (LLM) capabilities to accurately interpret user intent and maintain dialogue coherence.

To ensure the performance and effectivness of our system, we evaluate it based on other metrics as shown in table 9.

Table 9. Performance Comparison of Chat-SQL vs Prior Models

Metric	Chat-SQL	Seq2SQL	SQLNet	RAT-SQL	GPT-4 (zero-shot)
Multi-turn Context Accuracy	85.6%	58.3%	63.1%	72.5%	74.4%
SQL Query Validity	98.3%	92.7%	93.8%	95.1%	90.2%
Natural Dialogue Understanding	87.4%	61.2%	67.9%	73.6%	80.5%
Sketch Alignment Accuracy	91.2%	N/A	N/A	81.9%	N/A

- Multi-turn Context Accuracy (85.6%): This metric evaluates how well the system preserves dialogue history and incorporates prior turns when generating SQL queries. Chat-SQL leverages dialogue state tracking and sketch re-planning to achieve the highest performance in this area.

- SQL Query Validity (98.3%): Validity means the SQL output adheres to proper syntax and schema structure. By grounding the generation process in sketch templates, Chat-SQL ensures queries are nearly always syntactically correct, outperforming even fine-tuned models like RAT-SQL.

- Natural Dialogue Understanding (87.4%): This measures how accurately the system interprets and maps free-form, human-style questions into structured query logic. Chat-SQL uses an LLM that captures intent and subtle variations in language more effectively than traditional rule-based or sequence models.

- Sketch Alignment Accuracy (91.2%): A unique metric for sketch-based systems, this evaluates whether the correct sketch is chosen and filled accurately. Chat-SQL's intelligent sketch selection mechanism, guided by LLM context understanding, drives this success.

5. Conclusion and Future Work

In summary, this study provides a comprehensive review of research conducted over recent decades in the field of Natural Language Interfaces for Databases (NLIDB). NLIDB remains a dynamic area within natural language processing, aiming to enable non-technical users to submit queries in natural language and receive corresponding database responses. Such systems can handle user queries expressed either in written text or spoken language, delivering answers efficiently while providing appropriate error messages when processing fails.

Experimental results indicate that our proposed system achieves satisfactory performance, generating accurate and relevant responses across a variety of natural language queries, with an overall accuracy of approximately 89%. The model also supports interactive human-computer

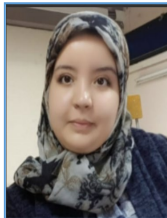
communication, offering feedback and requesting clarifications when it is unable to translate a query. For future work, we plan to extend this system into a mobile application, enabling users to perform natural language queries and receive answers directly on smartphones. The application will support multiple languages, including Arabic and French, further broadening accessibility and usability for diverse user groups.

6. References

- [1]. N. Ranjan, K. Mundada, K. Phaltane, and S. Ahmad, "A Survey on Techniques in NLP," *Int. J. Comput. Appl.*, vol. 134, no. 8, pp. 6–9, (2016).
- [2]. E. U. Reshma and P. C. Remya, "A review of different approaches in natural language interfaces to databases," in *Proceedings of the International Conference on Intelligent Sustainable Systems, ICISS 2017*, (2018).
- [3]. Androutsopoulos, I., Ritchie, G. and Thanisch, P. (1995) « Natural language interfaces to databases: an introduction », *Natural Language Engineering*, Vol. 1, No. 1, pp.29–81.
- [4]. K.Majhadi and M.Mustapha. The history and recent advances of Natural Language Interfaces for Databases Querying, E3S Web of Conferences 229, 01039 (2021), *The 3rd International Conference of Computer Science and Renewable Energies*.
- [5]. B Green, A Wolf, C.Chomsky, K Laughery, « BASEBALL: an automatic question answerer », western joint IRE-AIEE-ACM computer conference May (1961), Pages 219–224.
- [6]. Woods, William A, Ronald M Kaplan, and Bonnie Nash-Webber, « natural language information system », (1972).The lunar sciences.
- [7]. G.Hendrix, « A NATURAL LANGUAGE INTERFACE », *Computational Linguistics*, vol.8, no.2, pp.55-61, (1982).
- [8]. G. Ritchie, I. Androutsopoulos, P. Thanisch, « Masque/sql », (1993).
- [9]. Yunyao Li, Huahai Yang, and H.V. Jagadish, Nalix: « an Interactive Natural Language Interface for Querying XML », *SIGMOD* (2005).
- [10]. Victor Zhong, CaimingXiong, Richard Socher, Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning (2017).
- [11]. J. Gu, Z. Lu, H. Li, and V. O. K. Li. Incorporating Copying Mechanism in Sequence-to-Sequence Learning. *ArXiv e-prints*, March (2016).
- [12]. Xu, X., Liu, C., Song, D., 2017. SQLNet: Generating Structured Queries From Natural Language Without Reinforcement Learning. *arXiv:1711.04436v1*.
- [13]. JichuanZeng, Xi Victoria Lin, CaimingXiong, Richard Socher, Michael R. Lyu, Irwin King, Steven C.H. Hoi. Photon: A Robust Cross-Domain Text-to-SQL System. *The 58th Annual Meeting of the Association for Computational Linguistics*, 2020.
- [14]. IE-SQL: Text-to-SQL as Information Extraction. (2020) Association for Computing Machinery.
- [15]. X. Zhang, F. Yin, G. Ma, B. Ge and W. Xiao, "M-SQL: Multi-Task Representation Learning for Single-Table Text2sql Generation," in *IEEE Access*, vol. 8, pp. 43156-43167, (2020), doi: 10.1109/ACCESS.2020.2977613.
- [16]. Bailin Wang, Richard Shin, Xiaodong Liu, OleksandrPolozov, Matthew Richardson. RAT-SQL: Relation-Aware Schema Encoding and Linking for Text-to-SQL Parsers. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7567–7578, July 5 - 10, (2020).
- [17]. DongHyun Choi, MyeongCheol Shin, EungGyun Kim, Dong Ryeol Shin. RYANSQL: Recursively Applying Sketch-based Slot Fillings for Complex Text-to-SQL in Cross-Domain Databases. (2020) *Computation and Language (cs.CL)*.
- [18]. JaydeepSen, Chuan Lei, Abdul Quamar, Fatma Ozcan2, VasilisEfthymiou, AyushiDalmia, Greg Stager, Ashish Mittal, DiptikalyanSaha, and KarthikSankaranarayanan. ATHENA++: natural language querying for complex nested SQL queries. (2020) *proceedings of the VLDB Endowment*, Vol. 13, No. 11, ISSN 2150-8097.

- [19]. Ursin Brunner, Kurt Stockinger. ValueNet: A Neural Text-to-SQL Architecture Incorporating Values. (2020) proceedings of the VLDB Endowment.
- [20]. Han Fu, Chang Liu, Bin Wu, Feifei Li, Jian Tan, and Jianling Sun. CatSQL: Towards Real World Natural Language to SQL Applications. *Proceedings of the VLDB Endowment*, Volume 16, Issue 6 (2023).
- [21]. Yingwen Fu, Wenjie Ou, Zhou Yu, and Yue Lin, MIGA: A Unifed Multi-Task Generation Framework for Conversational Text-to-SQL, arXiv:2212.09278v1 [cs.CL] 19 Dec 2022.
- [22]. Yuanfeng Song , Raymond Chi-Wing Wong, Xuefang Zhao, and Di Jiang, Speech-to-SQL: Towards Speech-driven SQL Query Generation From Natural Language Question, arXiv:2201.01209v1 [cs.DB] 4 Jan 2022.
- [23]. K. Ahkour, M. Machkour, K. Majhadi, R. Mama, SEQ2SEQ VS SKETCH FILLING STRUCTURE FOR NATURAL LANGUAGE TO SQL TRANSLATION. 5th International Conference on Smart City Applications, 7–8 October 2020.
- [24]. P. J. Price. Evaluation of spoken language systems: the ATIS domain. In DARPA Speech and Natural Language Workshop, pages 91–95, 1990.
- [25]. Zelle, J.M., Mooney, R.J.: Learning to parse database queries using inductive logic programming. In: Proceedings of the Thirteenth National Conference on Artificial Intelligence—Volume 2, AAAI’96, pp. 1050–1055. AAAI Press (1996).
- [26]. Popescu, A.-M., Etzioni, O., Kautz, H.: Towards a theory of natural language interfaces to databases. In: Proceedings of the 8th International Conference on Intelligent User Interfaces, IUI ’03, pp. 149–157. Association for Computing Machinery, New York, NY, USA (2003).
- [27]. Li, F., Jagadish, H.V.: Constructing an interactive natural language interface for relational databases. *PVLDB* 8(1), 73–84 (2014).
- [28]. Yaghmazadeh, N., Wang, Y., Dillig, I., Dillig, T.: Sqlizer: query synthesis from natural language. In: PACMPL, pp. 63:1–63:26 (2017).
- [29]. T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman, Z. Zhang, and D. R. Radev. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In EMNLP, pages 3911–3921, 2018.
- [30]. Hyeonji Kim, Byeong-Hoon So, Wook-Shin Han, Hongrae Lee. Natural language to SQL: Where are we today? *Proceedings of the VLDB Endowment*, Vol. 13, No. 10 ISSN 2150-8097.
- [31]. Karam.A, Mustapha.M, Mourad.E, Brahim.E, Jilali.A, « Comparative study of existing approaches on the Task of Natural Language to Database Language », ICCSRE, p 1-6. (2019).
- [32]. Majhadi Khadija and Machkour Mustapha, An Interactive Natural Language Interface for Querying Database: A Literature Review, *JARDCS* Volume 12 | 05-Special Issue (2021).
- [33]. Hanane, B., Mustapha, M. and Lahcen, K. (2018) ‘An Arabic natural language interface for querying relational databases based on natural language processing and graph theory methods’, *Int. J. Reasoning-based Intelligent Systems*, Vol. 10, No. 2, pp.155–165.
- [34]. X. Zhang, M. H. Chen and Y. Qin, "NLP-QA Framework Based on LSTM-RNN, 2nd International Conference on Data Science and Business Analytics (ICDSBA), 2018, pp. 307-311, doi: 10.1109/ICDSBA.2018.00065.
- [35]. Abbas, S., Khan, M.U., Lee, S.U.-J., Abbas, A., Bashir, A.K.: A review of nliidb with deep learning: findings, challenges and open issues. *IEEE Access*. 10, 14927–14945 (2022).
- [36]. Deng, N., Chen, Y., Zhang, Y.: Recent advances in text-to-SQL: a survey of what we have and what we expect. In: *Proceedings of the 29th International Conference on Computational Linguistics*, pp. 2166–2187. International Committee on Computational Linguistics, Gyeongju, Republic of Korea (2022).

- [37]. P. Parikh et al., "Auto-Query - A simple natural language to SQL query generator for an e-learning platform," *IEEE Global Engineering Education Conference (EDUCON), Tunis, Tunisia*, 2022, pp. 936-940, doi: 10.1109/EDUCON52537.2022.9766617, (2022).
- [38]. Yao, Z., Su, Y., Sun, H., Yih, W. T. Model-based Interactive Semantic Parsing: A Unified Framework and A Text-to-SQL Case Study. OSU & Facebook AI Research 2019.
- [39]. Bahdanau, D., Cho, K. and Bengio, Y. Neural Machine Translation by Jointly Learning to Align and Translate, arXiv: 1409.0473v7 (2014).
- [40]. Hwang, W., Yim, J., Park, S. and Seo, M. A Comprehensive Exploration on WikiSQL with Table-Aware Word Contextualization, arXiv: 1902.01069v1 (2019).
- [41]. Liu, X., He, P.g., Chen, W. and Gao, J. Multi-Task Deep Neural Networks for Natural Language Understanding, arXiv: 1901.11504 (2019).
- [42]. math, A., Das, R.: A survey on semantic parsing. In: 1st Conference on Automated Knowledge Base Construction, AKBC 2019, Amherst, MA, USA, May 20–22, 2019 (2019).
- [43]. Chen, D. and Manning, C. (2014). A fast and accurate dependency parser using neural networks. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), pages 740–750.
- [44]. P. Wang, Y. Qian, F. K. Soong, L. He and H. Zhao, Part-of-Speech Tagging with Bidirectional Long Short-Term Memory Recurrent Neural Network.



MAJHADI Khadija received her Master's degree in Data Science from Departments of Mathematics and Computer Science, Faculty of Science, University Caddi Aayad, Marrakech, Morocco. She is currently a Ph.D. candidate of the Ibn Zohr University, Agadir, Morocco. Her research interests include Database system, Natural Language Processing and Artificial Intelligence.



Mustapha. Machkour is a professor of higher education, department of Mathematics and computer Sciences, Ibn Zohr University, Agadir, Morocco. Member of Laboratory of Computer Systems and Vision, Faculty of Science, Ibn Zohr University, Agadir Morocco. Current research interests include Image Processing, Data Security, logic and Artificial Intelligence.