Machine Learning with Applications 20 (2025) 100641

Contents lists available at ScienceDirect

Machine Learning with Applications

journal homepage: www.elsevier.com/locate/mlwa

ASKSQL: Enabling cost-effective natural language to SQL conversion for enhanced analytics and search

Arpit Bajgoti, Rishik Gupta *, Rinky Dwivedi

Department of Computer Science and Engineering, MSIT, Delhi, India

ARTICLE INFO

Keywords:
Natural language processing
SQL query generation
NL2SQL pipeline
Query recommendation
Semantic cache
Data analytics
Automation

ABSTRACT

Natural Language to SQL (NL2SQL) for database query and search has been a significant research focus in recent years. However, existing methods have predominantly concentrated on SQL query generation, overlooking critical aspects such as enterprise cost, latency, and the overall analytical search experience. This paper presents an end-to-end NL2SQL pipeline named ASKSQL that integrates optimized and adaptable query recommendation, entity-swapping module, and skeleton-based caching to enhance the search experience. The pipeline also incorporates an intelligent schema selector for efficiently handling large schema entity selection and a fast and scalable adapter-based query generator. The proposed pipeline emphasizes minimizing Large Language Model (LLM) costs by finding search patterns in previously requested or generated queries. The pipeline can also be tuned to adapt to trends and common patterns observed from the daily search analytics. Experimental results demonstrate an average increase in accuracy by 5.83% and an overall decrease in latency by 32.6% as the usage count of this search pipeline increases highlighting its effectiveness in improving the NL2SQL search experience.

1. Introduction

In the current landscape of generative AI trends, the automation of big data analysis and report generation takes center stage. Utilizing SQL database queries is pivotal to these tasks, and recent research endeavors aspire to seamlessly generate SQL queries from user prompts, introducing scalability and automation. This is particularly crucial for companies handling extensive data and daily analysis, where data analysts grapple with comprehending the schema of structured data. The NL2SQL method emerges as a pivotal solution, adept at automatically selecting pertinent schema items and generating queries based on user dialogues, primarily divided into two approaches: prompting Large Language Models (LLMs) and fine-tuning Pre-trained Language Models (PLMs). Each approach offers distinct advantages. Fine-tuned PLMs are especially effective in zero-shot scenarios with competitive databases and varied queries, whereas prompting techniques excel with complex queries involving inner joins and nested SQL structures.

The advent of domain-specific LLMs has significantly enhanced in-context learning outcomes, although their implementation in enterprise settings is often hindered by high costs and latency, rendering them impractical for daily analytics and frequent query repetition among data analysts. Conversely, fine-tuned PLMs, while unable to generalize well to unseen data or handle complex queries, are cost-effective and exhibit low latency due to a reduced parameter count. This efficiency, however, restricts their ability to manage extensive token usage necessary for multi-table interactions and high-dimensional querying, leading to diminished accuracy in generating joins and nested queries.

To address these challenges, we propose a hybrid approach that combines the strengths of both fine-tuned and prompted models. Our end-to-end pipeline enhances domain-specific accuracy and reduces latency, making it suitable for enterprise environments where query repetition and similarity are prevalent. This system also integrates caching and leverages semantic similarities to improve performance.

Contributions of our Work:

- Development of an end-to-end NL2SQL pipeline which incorporates Schema Selection, Query Generation, and Query Correction modules.
- A caching module that helps reduce Language Model calls by checking for semantically similar questions and providing similar history queries as recommendations which is connected to an entity swapping module.

* Corresponding author.
E-mail addresses: arpitbajgoti_cse_24@msit.in (A. Bajgoti), rishikgupta_cse_24@msit.in (R. Gupta), rinkydwivedi@msit.in (R. Dwivedi).

https://doi.org/10.1016/j.mlwa.2025.100641
Received 2 April 2024; Received in revised form 12 February 2025; Accepted 13 March 2025
Available online 9 April 2025

- Encoder models for schema selection and Adapter-based fine-tuning of low parameter PLM for multi-domain SQL generation, resulting in cost and time reduction.

The general framework for our prompting method follows a sequential three-step strategy: schema selection, SQL generation, and SQL correction with self-ranking. This research aims to develop a robust, enterprise-oriented system that efficiently handles the complexities and demands of modern data analytics. To support these goals, we leverage various techniques:

1.1. Pre-trained language models

Pretrained Language models are fundamental to our approach as they are trained on a large database of pre-written data optimized for specific language processing tasks. These models learn the social context and structure of the language during initial training, making them suitable for many low-level applications such as notes, answers, and translations. Pre-trained models can capture the nuances and meanings of words, making them highly adaptable to specific tasks. Pretrained language patterns have been incredibly successful in the field of natural language processing (NLP), leading to a shift from supervised learning to pre-learning and post-training. Large Language Models (LLMs) with SQL queries, enable the extraction and querying of information from unstructured text. By incorporating LLM-based operators into traditional query plans, the system unifies natural language and structured data querying. Results indicate that LLMs can return well-structured relations for a range of SQL queries, showcasing the potential for hybrid querying that leverages the precision of SQL with the expressive capabilities of LLMs. Challenges such as interpretability, biases, scalability, and security need consideration in further research and development (Saeed, De Cao, & Papotti, 2023). Building on the strengths of pre-trained language models, we further enhance our approach with the use of efficient search techniques achieved using ANN.

1.2. Approximate nearest neighbor

Approximate Nearest Neighbor (ANN) search is required in our approach to address the computational efficiency required in real-time recommendations. ANN search is a computational technique in computer science and machine learning that efficiently finds an approximation to the nearest neighbors of a given data point in a high-dimensional space. It sacrifices a degree of accuracy for faster query times, making it suitable for scenarios where computational efficiency is crucial. The nearest neighbor searches for items in the reference library that contain the fewest number of queries. Information storage plays an essential role in many fields, including computer vision, multimedia, machine learning, and recommendations. Although there has been extensive research on this problem, it is generally accepted that finding nearest neighbors in high Euclidean space is very expensive due to the curse of size. Experiments show that precision is less efficient than the brute force method (Li et al., 2019) and can be used when the number of dimensions is large (e.g. over 20). However, returning objects that are close enough (called approximate nearest neighbor search (ANNS) can be efficient and useful enough for many problems to support a lot of research. The integration of ANN search significantly boosts the performance of our NL2SQL pipeline by enhancing the speed and efficiency of data retrieval.

1.3. Quadratic assignment problem

Another critical component of our pipeline is solving the Quadratic Assignment Problem (QAP), which optimizes the distribution of two graphs by minimizing the cost of interconnecting them. This is key to subgraph identification and matching. In the area of subgraph mapping, QAP excels at finding isomorphic subgraphs; In subgraph matching, it strategically maps graph nodes to reduce the objective function encapsulated in a quadratic value matrix. This mathematical method is fundamental in optimization and works as a solution to various problems involving node functions and small relationships in subgraphs (Jafari & Sheykhan, 2021). Incorporating QAP into our pipeline helps us achieve more efficient subgraph identification and matching, further enhancing the performance and accuracy of our NL2SQL system.

By integrating these techniques into our NL2SQL pipeline, we address the challenges and limitations associated with transformer-based models and ensure the development of a scalable, efficient, and cost-effective solution for enterprise-level data analysis and reporting.

2. Related works

Significant progress has been made in prompt-based in-context learning, multi-task, and independent-level component development in the field of transformer-based language modeling. This body of work is systematically categorized at the component level, specifically addressing tasks such as schema Item classification, SQL query generation, query recommendation, and methods related to semantic caching.

2.1. Schema item classification

The objective of Schema Item Classification is to determine a table entity based on the given query. This approach involves encoding both the query and multiple table schemas into a unified textual representation. The goal is to extract probabilities associated with tables and their relevant columns within the schema. Typically, these tasks are interdependent, and the representations are learned within a multi-task model-based architecture. Shi et al. (2021) introduced a method that employs a column prediction and a column recovery layer, utilizing an encoder masking pre-training approach based on BERT-base (Devlin, Chang, Lee, & Toutanova, 2018) as the contextual encoder. Their work focuses on enhancing the encoder's capabilities through the joint consideration of multiple schema-related tasks. Similarly, Yu et al. (2020) proposed GRAPPA, a method that generates a synchronous context-free grammar for combined dialogue and schema understanding. This approach leverages a multi-task learning paradigm to improve the overall performance of schema item classification. Cao et al. (2021) took a unique direction by employing a line graphical relationship model-based approach. They encoded questions, columns, and table entities as separate nodes in the graph. This method, termed Line Graph Enhanced Text-to-SQL (LGESQL), focuses on mining relational features efficiently without relying on meta-paths. While the methods mentioned above enhance encoders through pre-training or the multi-task learning paradigm, independent efforts in table item classification have also been pursued. For instance, RESDSQL (Li, Zhang, Li, & Chen, 2023) utilizes bidirectional pooling on top of an encoder-based model like RoBERTa (Liu et al., 2019). This method calculates probabilities for each table and its respective columns independently. However, it is essential to note a limitation of RESDSQL, where the token limit of 512 poses a constraint, encompassing both user dialogue and the entire table schema. This encoding strategy may prove insufficient when handling extensive table schemas and multiple lengthy schema representations. The landscape of Schema Item Classification research exhibits a diverse range of methodologies, each addressing specific challenges and contributing to the ongoing evolution of techniques in this domain.

2.2. SQL query generation

The landscape of SQL query generation has witnessed significant advancements within the domain of data analytics, driven by the integration of language models. The core objective revolves around the generation of executable SQL queries based on given prompts, with two primary methodologies gaining prominence. The first approach revolves around prompting or instruction-tuning. Pourreza and Rafiei (2024) pioneered a chained prompting-based strategy, introducing a classification system that categorizes statements into easy, medium, and hard classifications. This method employs a few-shot prompting template selection technique. Gao et al. (2023) conducted an extensive exploration of various prompting techniques, consolidating their findings based on responses obtained using GPT-4 (Peng, Li, He, Galley, & Gao, 2023). This approach not only showcases the flexibility to customize logic through tailored prompts but also adapts effectively to diverse complexities in tasks. On the other end of the spectrum lies the approach based on fine-tuning pre-trained large language models (PLM). Arcadinho, Aparício, Veiga, and Alegria (2022), Li et al. (2023) employ the T5 model for query generation from dialogues. The former utilizes a constrained decoder of T5, while the latter integrates a NATSQL (Gan et al., 2021) layer, generating SQL skeletons and populating outputs with relevant columns, tables, and entities. Domínguez (2023) fine-tune the CodeLlama-13b (Roziere et al., 2023) decoder-based model for SQL query generation. Labs (2023) follows a similar fine-tuning approach by refining the llama-2-7b model with a combined dataset sourced from both custom and publicly available datasets. These fine-tuning approaches showcase adaptability to domain-specific contexts and contribute to the flexibility of SQL generation. While both prompting and fine-tuning approaches exhibit efficacy in SQL query generation, they carry distinct advantages and limitations. Prompting excels in scenarios with limited data, relying on a customized logic tailored to specific requirements. In contrast, fine-tuning approaches offer the flexibility needed for domain-specific SQL generation, providing adaptability across a spectrum of contexts and addressing varied requirements within the data analytics landscape.

2.3. Query recommendation

In commercial settings, language models trained on Natural Language to SQL (NL2SQL) tasks often exhibit fine-tuning specific to a particular vocabulary and syntax. Adapting these models for new customers or users who prefer specificity in their input queries poses a challenge. Consequently, history-based query recommendations become imperative from an interface perspective, allowing users to tailor SQL queries based on their unique query history. Achieving this task involves semantic-based history matching, where Fariha and Meliou (2019) introduced a query intent discovery approach utilizing a probabilistic abduction model. This model employs abduction-based reasoning to identify candidate queries, enhancing the user's ability to generate SQL queries with precision. Pawar and Mago (2018) take a corpus statistic approach, utilizing word, word order, and sentence-wise similarity analysis based on part-of-speech (POS) and synsets (Ercan & Haziyev, 2019). Their method contributes to refining query recommendations by incorporating semantic understanding. In a novel direction, Lin, Pradeep, Teofili, and Xian (2023) explored a vector database-based semantic similarity search approach. This trending method leverages vector databases to enhance the effectiveness of semantic similarity searches, providing efficient retrieval of relevant queries. Chandwani and Ahlawat (2023) investigate a fuzzy-based string matching score for sentence similarity, contributing to the diversity of methods aimed at retrieving relevant queries effectively. While vector database indexed sentence matching proves efficient, the ultimate goal is not only retrieval but also ensuring the adaptability of similar dialogues or questions for ordinal, company, or string and quantity-based searches. The methodologies discussed above collectively contribute to the development of efficient and adaptive query recommendation systems in NL2SQL tasks. As the research landscape evolves, these approaches play a vital role in enhancing user interaction and the overall usability of NL2SQL interfaces in diverse and dynamic commercial environments.

2.4. Semantic caching

Query-generating models, often hosted on large GPUs for expedited retrievals (Sheng et al., 2023), can become computationally expensive, leading to increased analytics time and higher expenses for companies, particularly in scenarios where users do not prioritize selecting semantic sentences. To address these challenges, a semantic caching-based layer is introduced, allowing for the quick and cost-efficient retrieval of queries. Chen et al. (2016) designed a sequential interference model based on chain LSTMs that could outperform previous models. This improvement was achieved by incorporating syntactic parsing information into the model, thus enhancing its performance. Liu, Wang, Lv, Zhen, and Fu (2020) contributed to this space by utilizing different syntactic tags as syntactic features, integrating them with word features to enhance similarity calculations. This approach aims to improve the efficiency of semantic caching by considering both syntactic and semantic elements. Xu, Xu, Dong, and Wen (2022) propose a Query Answer Pair (QAP)-based permutation matrix construction, leveraging both syntactic and semantic matrix aggregated affinity matrix. Building on this work, the same authors (Xu, Xu, Dong, & Wen, 2023) introduced a novel approach utilizing syntactic relationships between sentences for QAP solving. While these methods provide commendable results, their application to NL2SQL tasks may introduce false positives, as they lack specific filters to validate results.

ASKSQL addresses several critical gaps identified in related works within the field of transformer-based language modeling and NL2SQL applications. While previous approaches, such as those involving schema item classification, SQL query generation, query recommendation, and semantic caching, have demonstrated significant advancements, they often overlook key aspects like enterprise cost, latency, and the overall analytical search experience. Existing methods like GRAPPA and LGESQL focus on improving encoder capabilities but struggle with handling extensive table schemas due to token limits. Fine-tuning approaches for SQL query generation, while flexible, are computationally intensive and may not adapt well to domain-specific contexts without substantial data. Additionally, current query recommendation systems rely heavily on semantic matching, which can lead to inefficiencies in fast real-time environments.

Our method introduces a novel end-to-end NL2SQL pipeline that integrates compact models and innovative prompting practices, significantly reducing virtual memory consumption and inference times. By leveraging use case-specific data and optimizing for multi-user settings, we enhance both cost-effectiveness and operational efficiency. Our approach also incorporates advanced semantic caching techniques, ensuring rapid and accurate query retrievals. The pipeline includes an intelligent schema selector for efficiently handling large schema entity selection and a fast and scalable adapter-based query generator. This strategy not only minimizes Large Language Model (LLM) costs by identifying search patterns in previously requested or generated queries but also adapts to trends and common patterns observed from daily search analytics.

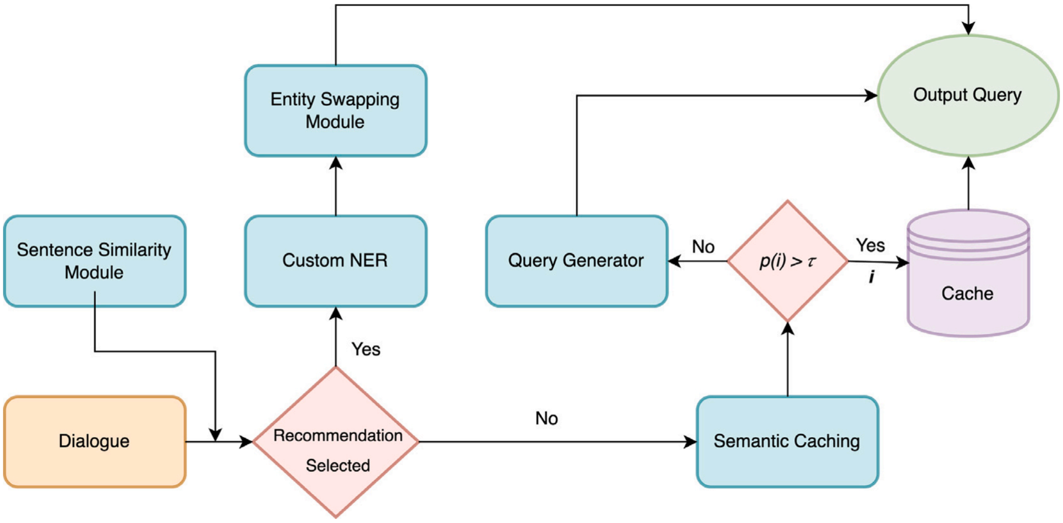


Fig. 1. Proposed architecture of end-to-end NL2SQL Pipeline.

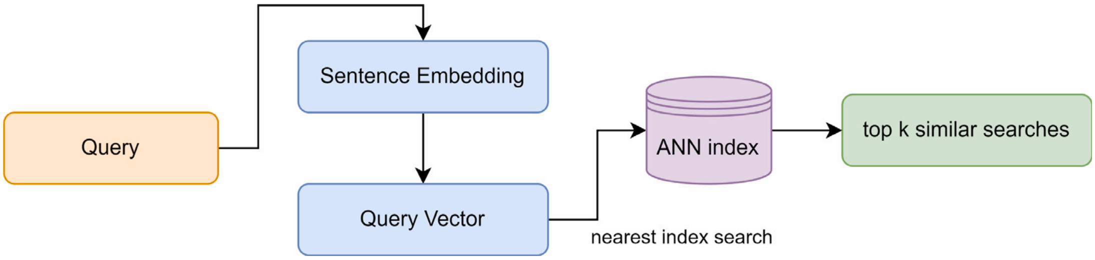


Fig. 2. ANN Based Sentence Similarity Module.

3. Proposed work

The proposed end-to-end ASKSQL pipeline is structured around several interconnected components. User input undergoes a similarity check against historical dialogues to generate recommendations. If a user selects a recommended dialogue, the associated SQL query is processed through an entity-swapping module, replacing tags with relevant entities from the chosen recommendation. In cases where users do not choose a recommendation, a semantic caching layer assesses if the query is similar to any cached or historical queries. This layer uses similarity probabilities with a high threshold to reduce false positives and connects to the entity-swapping module if the threshold is exceeded. If a new input lacks a historical match, the pipeline switches to the generative SQL module as shown in Fig. 1. This module identifies the relevant table based on the input text and generates a raw SQL query, which is then parsed into an extractable format suitable for either Big Query or PostgreSQL, depending on the specified database system.

3.1. Semantic similarity module

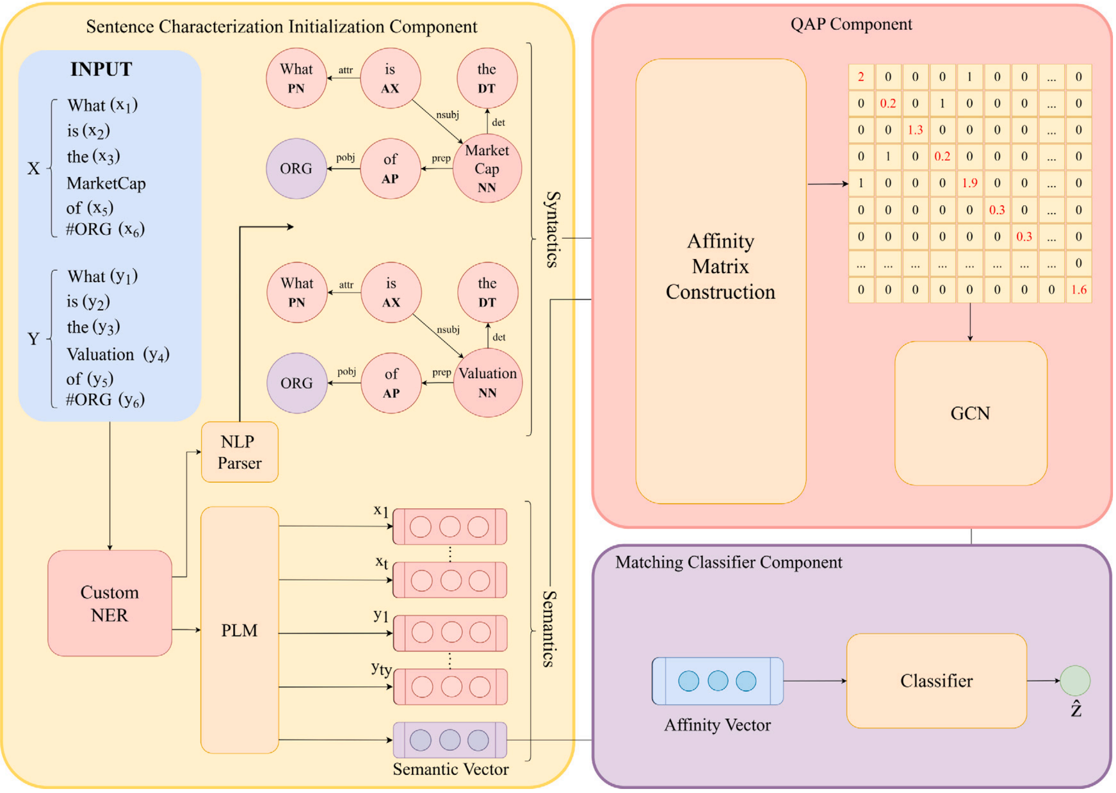
This process involves employing the Approximate Nearest Neighbors (ANN) search to calculate, for each query embedding ϕ_i, the set $\Psi(\phi_i, k')$ comprising the k' given query embeddings most closely related to ϕ_i based on an approximate distance metric. Subsequently, these query embeddings undergo a mapping process to link them back to their respective query $D(\phi_i, k')$ (Macdonald & Tonellotto, 2021):

$$D(\phi_i, k') = \{d \,\epsilon\, D : f_D(d) \cap \Psi(\phi_i, k')\} \tag{1}$$

The culmination of this process involves returning the union $D(k')$ of these sets. Specifically, after applying the Approximate Nearest Neighbors (ANN) search and mapping query embedding back to their respective given query, the final step is to aggregate these sets. The union $D(k')$ represents the combined set of queries from the identified nearest neighbors for each query embedding. This consolidated set serves as the output, providing a comprehensive representation of the relevant given queries based on the semantic similarity to the given natural language query.

$$D(k') = \bigcup_{i=1}^{|q|} D(\phi_i, k') \tag{2}$$

After identifying the approximate nearest queries $D(k')$, they are leveraged to calculate the ultimate list of top k queries for retrieval. This involves re-ranking the set of queries $D(k')$ by utilizing the query embeddings and the multiple embeddings associated with each query. This process aims to generate precise scores, refining the ranking to determine the final order of queries recommended. By incorporating both queries and given statements or question embeddings, this step enhances the accuracy of the ranking process, ensuring the returned results align closely with the semantic relevance to the given natural language query. as shown in Fig. 2. The similarity between query q and a given query or question



2	0	0	0	1	0	0	...	0
0	0.2	0	1	0	0	0	...	0
0	0	1.3	0	0	0	0	...	0
0	1	0	0.2	0	0	0	...	0
1	0	0	0	1.9	0	0	...	0
0	0	0	0	0	0.3	0	...	0
0	0	0	0	0	0	0.3	...	0
...	...	...	...	...	...	...	...	0
0	0	0	0	0	0	0	0	1.6

Fig. 3. Skeleton Aware-Association-Graph Network (SA-AGN).

d is determined by the dot product of their embeddings. The final similarity score $s(q, d)$ is then computed by summing up the maximum similarities between the respective query and statements or question embeddings.

$$s(q, d) = \sum_{i=1}^{|q|} \max_{j=1,\ldots,|d|} \phi_i^T \psi_j \tag{3}$$

To accomplish this, the precise, uncompressed embeddings are preserved in an additional index structure (Johnson, Douze, & Jégou, 2019). Given the extensive number of queries in the candidate set, it is imperative to store the entire index structure in memory for efficient processing. The above equation is pivotal in establishing the ultimate ranking of queries in response to the query.

3.2. Semantic caching module

This module is enabled whenever the user does not want to choose any recommendation provided by the sentence similarity module, the function of this module is to automatically detect a similar query if it matches the user query automatically without external assistance, such a function should be adaptable to changing queries and reduced false positive response to not generate wrong results. To address these constraints we propose Skeleton Aware-Association-Graph Network (SA-AGN), a robust Semantic Caching Module that solves a sub-graph matching problem using QAP. We use an entity-masked Parts-of-Speech tagger for word piece identity encoding, a semantic detector for inter-semantic connections between different words in a sentence, and a word embedding for extracting a context vector for each word in the sentence pair. The detailed formulation is discussed as follows:

3.2.1. Problem formulation

The formal description of matching a pair of natural language sentences can be expressed as follows: Let Z denote the set of labels defined by a specific matching task. In paraphrase identification (PI) tasks, $Z = 0, 1$, where '0' and '1' represent "dissimilar" and "similar" relationships, respectively. In natural language inference (NLI), $Z = 0, 1, 2$, where 0, 1, and 2 correspond to "contradiction", "neutral", and "entailment", respectively. We are given a set of training instances $D = (X_i, Y_i, z_i)_{i=1}^{N}$, where each sample $(X, Y, z) \in D$ consists of a sentence pair (X, Y) and its corresponding ground-truth matching label z. Additionally, X and Y represent two sequences of words: $X = [x_1, x_2, \ldots, x_{t_X}]$ and $Y = [y_1, y_2, \ldots, y_{t_Y}]$, where x_i and y_j denote the ith and jth words in X and Y respectively, and t_X and t_Y represent the lengths (number of words) of X and Y, respectively.

Fig. 3 shows an example of a semantically similar sentence pair where X = *"What is the marketcap of Apple Inc."* (token count $|X| = 7$) and Y = *"What is the valuation of Muthoot Finance Ltd."* (token count $|X| = 8$), using a custom-trained Finance NER, both the sentence's tokens are tagged based on the cardinal or organization value similar to the Entity Similarity Module, which converts the sentence into an Entity masked sentences where X_{masked} = *"What is the marketcap of ORG"* (token count $|X| = 7$) and Y_{masked} = *"What is the valuation of ORG"* (token count $|Y| = 6$). The masked sentences are passed through a POS tagger and a syntactic dependency parser to obtain intra-semantic dependencies, which are converted into a pair of directed graphs G_X, G_Y, the POS tags are converted into nodal features, and the syntactic dependency is represented as edge features

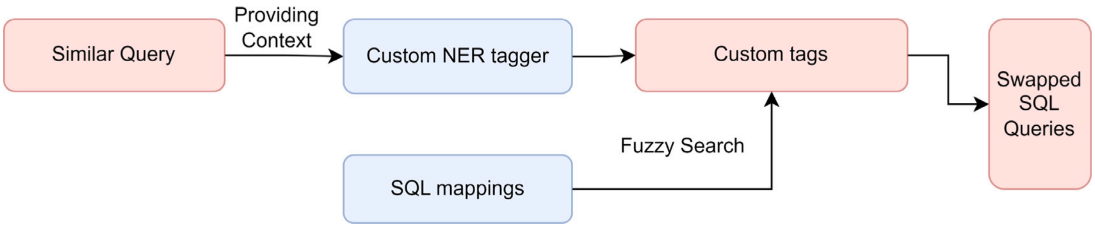


Fig. 4. Entity Swapping Module.

as shown in Fig. 3(a). To obtain a unified relationship between the pair, an association graph G is constructed utilizing an affinity matrix A derived from the sentence graphs G_X and G_Y. In this graph, the weights assigned to nodes and edges correspond to the diagonal and off-diagonal elements of the affinity matrix (refer to Fig. 2(b,c)). More precisely, node weights represent semantic similarities and Part-of-Speech (POS) attribute affinities of words, while edge weights indicate syntactic dependency (word-word relation) affinities.

For instance, the node (x_1, y_1) = (“What”, “Find”) could weigh, for example, $0.703 + 1.0 = 1.703$ where the 0.703 denotes the semantic similarity calculated as shown:

$$\cos(\theta) = \frac{x \cdot y}{\|x\|_2 \, \|y\|_2} \tag{4}$$

and 1.0 representing the similarity between the Part-of-Speech (POS) tags. For example, concerning the edges corresponding to the off-diagonal elements in the affinity matrix K, an edge could have a weight of 1.0 between the node (x_1, y_1) = (“What”,“Find”) and the node (x_2, y_2) = (“market cap”,“valuation”) due to the dependency relation between x_2 = “is” and x_4 = “valuation” being “nsubj”, while the dependency relation between y_2 = “is” and x_4 = “market cap” is also “nsubj”. Similar edges can be generated accordingly. Subsequently, the Quadratic Assignment Problem (QAP) is employed to acquire semantic and syntactic matching patterns in the affinity matrix K. Formally, a relaxed form of QAP can be expressed as:

$$max_S vec(S)_T K vec(S), \tag{5}$$

where the S matrix encodes the word-word correspondence, vec(S) represents the column-vectorized notation of S, and $K \in \mathbb{R}^{t_X t_Y \times t_X t_Y}$. The S matrix can be interpreted as an aggregated matching pattern of syntax and semantics.

3.3. Entity swapping module

Entity-swapping modules involve identifying placeholder(tags) with actual values present in a given natural language, it is done often after NER tagging, which involves tagging certain entities(organization, person, location). For instance, in the query *What is the revenue of Apple Inc.?*". NER identifies “*Apple Inc.*” as an organization, mapping it to the SQL query and resulting in a technically accurate statement like:

```
SELECT revenue FROM Financials WHERE Company = 'Apple Inc.';
```

Let us say, the next natural language comes as “*What is the revenue of Nokia Inc.?*”, as per NER tagging “*Nokia Inc.*” will be identified as an organization, since the earlier same kind of query was asked, simply “*Nokia Inc.*” will be swapped with “*Apple Inc.*”. After this task, it will be faster and easier to generate the new query, which will be:

```
SELECT revenue FROM Financials WHERE Company = 'Nokia Inc.';
```

We see that no new query is needed to be generated, by using the swapping module and NER tagging, the required query is generated as shown in Fig. 4

This meticulous approach ensures not only contextual accuracy but also conformity to the underlying database schema, optimizing the generated SQL query for precision and relevance. A custom-trained RoBERTa NER model is used to enhance the accuracy of entity recognition. RoBERTa, based on transformer architecture, excels in capturing contextual information, making it well-suited for NER tasks (Liu et al., 2019). The self-attention transformer model(like BERT) learns to identify patterns and relationships between words, improving the precision of entity classification.

$$Attention(Q, K, V) = SoftMax(\frac{QK^T}{\sqrt{d_k}})V \tag{6}$$

Conditional random field (CRF) layer. To leverage the interdependence among various tags, the Conditional Random Field (CRF) in the NER model is the last layer. This statistical modeling technique considers the relationships between different tags, enhancing the overall performance and coherence of our models. By incorporating CRF, we aimed to capture contextual dependencies, resulting in more robust and accurate predictions across various tasks and applications. In the context of a sequence of words denoted as $w = [w_1, w_2, \dots, w_T]$, where the corresponding golden label sequence is represented as $z = [z_1, z_2, \dots, z_T]$, and $Z(s)$ encompasses all valid label sequences, the probability of y is computed using the following equation (Yan, Deng, Li, & Qiu, 2019):

$$P(z|w) = \frac{\sum_{t=1}^{T} e^{f(z_{t-1}, z_t, w)}}{\sum_{z'}^{Z(w)} \sum_{t=1}^{T} e^{f(z'_{t-1}, z'_t, w)}} \tag{7}$$

In this context, the function $f(z_{t-1}, z_t, w)$ assesses both the transition score from z_{t-1} to z_t and the score for z_t. The main goal of optimization is to maximize the probability $P(z|w)$. In the decoding phase, the Viterbi Algorithm is utilized to determine the path that achieves the highest probability.

```
{
    "question": "What is the market cap of Apple Inc.?",
    "SQL": "SELECT "Market Cap" FROM Aggregate_Fundamentals_Data_Basic
        WHERE Company = ’Apple Inc.’;",
    "map": [
      {
        "que_word": "Apple Inc.",
        "sql_word": "Apple Inc.",
        "tag": "ORG",
        "idx_question": [26, 35],
        "idx_sql": [76, 85]
      }
    ]
  }
```

Listing 1 Example JSON file of SQL mapping data.

Let us take an example, here a JSON object is generated after custom NER tagging and Entity Swapping Module. The “question” attribute is the natural language input, and “SQL” represents the structured SQL output. The “map” attribute captures the mapping between language elements and their SQL counterparts, including details like the word, SQL equivalent, semantic indicator, and positional index. This aids in establishing relationships during the transformation process. In this example, The word “Apple Inc.” in the question is mapped to “Apple Inc.” in the SQL query. The tag “ORG” indicates that “Apple Inc.” is an organization. The indices specify the positions of “Apple Inc.” in both the question and the SQL query. This mapping allows the system to understand that the organization mentioned in the question corresponds to the “Company” column in the SQL query as shown in Listing 1. The swapping of ordinal values and selecting specific column values is essentially the process of identifying key elements in the question and mapping them to their counterparts in the SQL query to construct a meaningful and accurate query.

3.4. Query generator

In the case where a new input is encountered that is not detected by the semantic caching model, the query generation is handled by the query generator module. This module has different functions such as relevant schema selection based on input, SQL generation using auto-regressive transformer-based models, and DBEngine format parsing shown in Fig. 6. The details of the steps are provided below:

3.4.1. Schema selection

The relevant table schema selection based on the user input is an important aspect of any SQL query generation this can be handled loosely by the schema embedding and a vector database for CRUD operations for schema operations and management, the problem with such techniques is the ambiguous nature of the vector database which will retrieve the results irrespective of total relevance. This issue is resolved by the masked schema item classifier built on the encoder-based transformer model RoBERTa-large which takes the input in a format shown in Listing. 2 and returns the probability of the table as well as its columns which are referenced semantically directly or indirectly in the input.

```
{
    "input": "Which 5 users have the most visits and how much?",
    "schemas": [
    {
      "table_name": "clickstream",
      "columns": ["event_type","session_id","user_id","device","location"]
    },
    {
      "table_name":"search_metric",
      "columns": ["date","search_term","views","cartadds","orders","order_units"]
    },
    {
      "table_name":"product_metric",
      "columns":["product_id","order_units","gmv","avg_position","created_at"]
    }
  }
```

Listing 2 Structured query input: The input tables and query are extracted as the raw input to feed into the query generator module.

The pre-process involves hard schema filtering based on schema vector embedding to select top κ tables which have a count in the range 16–32 based on the batch size alerted based on the computational restrictions, then the schema item classifier assigns the probability for each table and its respective columns based on which a combined probability-based score is selected and the tables are ranked as shown in Fig. 5. The

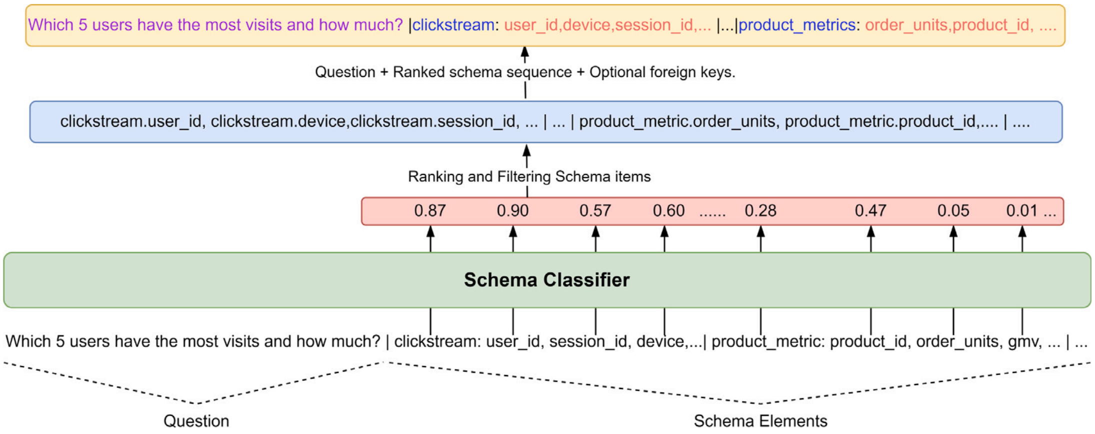

Which 5 users have the most visits and how much? |clickstream: user_id,device,session_id,... |...|product_metrics: order_units,product_id, ...
Question + Ranked schema sequence + Optional foreign keys.
clickstream.user_id, clickstream.device,clickstream.session_id, ... | ... | product_metric.order_units, product_metric.product_id,.... | ....
Ranking and Filtering Schema items
0.87
0.90
0.57
0.60
......
0.28
0.47
0.05
0.01 ...
Schema Classifier
Which 5 users have the most visits and how much? | clickstream: user_id, session_id, device,...| product_metric: product_id, order_units, gmv, ... | ...
Question
Schema Elements

Fig. 5. Schema Item Classifier inputs and output tokens.

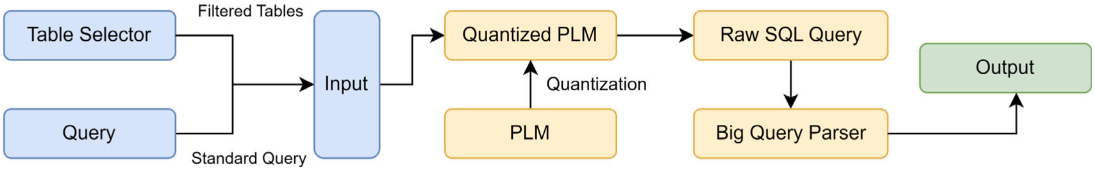


Fig. 6. Pre-trained LLM-based SQL query Generator.

classifier is based on a similar approach based on the Li et al. (2023) Ranking-enhanced Encoder which devices a cross-encoder that classifies the tables and columns collectively and ranks them based on the probabilities. The multi-token expanded column used in Li et al. (2023) is a pooling layer which is a two-layer BiLSTM layer, in our approach the pooling layer is removed by passing combining two or more word entities with a '-' to restrict excess token generation (e.g., the column "employee id" will be joined as 'employee-id"). Each table embedding can be denoted by $T_i \in R^{1\times d}(i \in 1,\dots,N)$ and each column embedding can be denoted by $C_k^i \in R^{1\times d}$ $(i \in 1,\dots,N, k \in 1,\dots,n_i)$, where d denotes the hidden size. After the probability prediction, the tables are selected based on a ranking equation:

$$R(s_i) = \frac{\frac{1}{k}\sum_{x=i}^{k} p(c_k) + p(t_i)}{2} \tag{8}$$

Here S_i is the ith schema, $p(t_i)$ is the probability that the table name is included in the input, and $p(c_i)$ represents the probability of the columns in table t_i by the Schema Classifier model.

Quantized pre-trained model. Quantization reduces the model size by using lower bit widths than floating-point precision, reducing the model size and computational requirements, while a Pre-trained Language Model(PLM) is a model that has been already trained on a large dataset to learn the patterns and structures of language based on data. Commence with a pre-trained language model (PLM) and employ quantitative methods to reduce precision intentionally. This yields a slightly smaller, compact, and memory-efficient PLM. Subsequently, fine-tune the model to optimize performance with a focus on parameter efficiency. The conclusion of this process results in the final model. A refined optimization that combines the benefits of precision reduction and fine-tuning to enhance performance in language-related tasks which can be called Quantized PLM. This process is called Quantization of PLM (Gong et al., 2023).

Uniform quantization can be utilized for both weights and activations. In this method, for a given tensor y in full precision, a rounding-to-nearest operation is employed to round y to the nearest unsigned integer grid values $y^{\mathbb{Z}}$. This process can be succinctly summarized as:

$$y^{\mathbb{Z}} = clip\left(\left\lfloor \frac{y}{\beta} \cdot 2^j \right\rceil + z; 0, 2^j - 1\right) \tag{9}$$

Here, $j \in \mathbb{N}$ denotes the bit-width, $\beta \in \mathbb{R}$ represents the scaling factor, and $z \in \mathbb{N}$ indicates the zero-point. After obtaining the quantized tensor $y^{\mathbb{Z}}$, one can approximate the full-precision version of the tensor $\hat{y}$ as:

$$\hat{y} = \frac{(y^{\mathbb{Z}} - z) \cdot \beta}{2^b} \tag{10}$$

This quantization approach is used to reduce the size of the PLM which is a minor accuracy loss that is thereby recovered during adapter-based finetuning as discussed later in Section 4.

4. Experiment

In this section, the proposed pipeline's programmatical construction, resources, libraries, datasets, and loss functions for training models are explored.

4.1. Datasets

The Quora Question Pairs (QQP) dataset, a large public resource for paraphrase identification, contains 404k labeled sentence pairs. We have utilized the same data split as in Radford, Narasimhan, Salimans, Sutskever, et al. (2018). The SNLI dataset, well-known for natural language inference (NLI), includes 570k labeled sentence pairs and follows the data splitting practices described in Conneau, Kiela, Schwenk, Barrault, and Bordes (2017). SciTail, another NLI dataset derived from science exams and the web, features 27k sentence pairs labeled as either "entailment" or "neutral" (Khot, Sabharwal, & Clark, 2018). These Datasets are used to train the Semantic Similarity model.

The NSText2SQL dataset, compiled from over 20 web sources with permissible licenses, features robust preprocessing methods and approximately 290,000 text-to-SQL pairs (Labs, 2023). WikiSQL provides 80,654 annotated questions and SQL queries from Wikipedia, crucial for natural language database interfaces (Zhong, Xiong, & Socher, 2017). The Stack, part of the BigCode Project, contains over 6TB of permissively licensed code across 358 languages and facilitates program synthesis with over 10 GB of data (Kocetkov et al., 2022). Spider, annotated by Yale students, offers a rich text-to-SQL dataset across 138 domains, containing 10,181 questions and 5,693 SQL queries, pushing systems towards better generalization (Yu et al., 2018). These datasets are used to for the pre-training of the PLM.

Lastly, the pipeline was fine-tuned on a domain specific custom enterprise dataset comprised of 4887 training and 1214 test samples, with 62% standard queries which consists of SELECT statements and grouping logic and complex queries which contains multiple JOINS and aggregations for a high dimension click-stream data.

4.2. Nearest neighbor indexes

The Semantic Similarity module should be programmed in a way such that it updates the recommendations every time the user changes the input in the text box and makes it responsive with little to no delay, this is achieved by ANN index libraries which provide fast similarity scores for the text. We employ Facebook AI Similarity Search (FAISS), a library designed for the rapid search of multimedia documents with similar content. FAISS comprises algorithms capable of searching through sets of vectors of varying sizes, including those that may exceed available memory capacity. Additionally, FAISS includes supporting code for evaluation and parameter optimization (Johnson et al., 2019).

4.3. Pre trained LLMs

In the experiments conducted, several pre-trained LLM models were used for query generation and getting a baseband model accuracy and latency. The various models used were: T5, Llama-7b-nl2sql, and CodeLlama-13b-Instruct-hf, trained for 60 epochs each. For further testing, the CodeLlama-13b-Instruct-hf model was quantized using QLoRA with an 8-bit quantization scheme and a temperature setting of 0.7. The experiments were conducted on a Tesla A100 40 GB virtual GPU. The average latency time recorded was 4.6 s for a 13b parameter quantized model which acts as the base latency for this experiment.

4.4. Hyperparameter settings

In the experiments conducted, the learning rate was set to 4×10^{-3}, and batch size was set to 8 allowing efficient processing while maintaining manageable memory usage. The model was trained over 16 epochs, using a weight decay of 0.001 to prevent overfitting and improve generalization. We also incorporated 100 warmup steps to introduce the learning rate gradually. The maximum sequence length was set to 700 tokens, accommodating complex queries and schemas within a single input sequence.

4.5. Accuracy measurement

Execution Accuracy (EX) was used for the measurement of accuracy. EX is defined as the proportion of examples in the evaluation set for which the executed results of both the predicted and ground-truth SQLs are identical, relative to the overall number of SQLs. Considering the result set as W_i executed by the i-th ground-truth SQL S_i, and the result set $\hat{W}_i$ executed by the predicted SQL $\hat{S}_i$ (Li, Hui, et al., 2024), EX can be computed by:

$$\mathrm{EX} = \frac{\sum_{i=1}^{M} \mathbb{1}(W_i, \hat{W}_i)}{M},$$

where $\mathbb{1}(\cdot)$ is an indicator function, which can be represented as:

$$\mathbb{1}(W_i, \hat{W}_i) = \begin{cases} 1, & \text{if } W_i = \hat{W}_i \\ 0, & \text{if } W_i \neq \hat{W}_i \end{cases}$$

5. Result

Table 1 compares the different semantic similarity models on QQP, SNLI, and SciTail datasets explained in Section 4.1, It is clear that the proposed method outperforms the standard sentence comparison methods, the ISG-GCN models show an increasing accuracy trend with improved semantic backbone encoder model, the proposed Skeleton Aware-Association-Graph Network (SA-AGN) outperforms the ISG and shows a similar behavior while increasing the complexity of encoder model.

Table 2 shows the comparison of different methods on Spider Dataset using key metrics: Average Tokens, which measures the complexity and verbosity of the generated queries; Latency, recorded in seconds to evaluate model response times crucial for real-time applications; Average cost per query in $, indicating the cost per query; Memory Usage, listed in gigabytes, which reflects computational resource demands crucial for scalability assessments; and Execution Accuracy. To calculate the metric of the proposed method for Table 3 the entity swapping and semantic caching modules are turned off due to their human dependency, so only Schema Selector and fine-tuned LLM modules are used.

We covered three categories of models: pre-trained, fine-tuned, and proposed. Pre-trained models such as C3SQL (Dong et al., 2023) and DINSQL (Pourreza & Rafiei, 2024) primarily employ GPT architectures, with token usage varying significantly—GPT-3.5-Turbo used 5702 tokens

Table 1
Comparison of different models Syntactic structural representation on Financial Sentence Semantic Similarity.

Models with syntactic structures	QQP:Acc (%)	SNLI:Acc (%)	SciTail:Acc (%)
HIM (Chen et al., 2016)	88.7	88.6	71.6
DDR – BERT$_{\text{LARGE}}$- (Yu, Xu, Xu, Pang, & Wen, 2022)	88.7	88.6	71.6
SemBERT$_{\text{BASE}}$ (Zhang et al., 2020)	89.8	90.2	92.1
SemBERT$_{\text{LARGE}}$ (Zhang et al., 2020)	90.7	91.0	92.1
SyntaxBERT$_{\text{BASE}}$ (Bai et al., 2021)	89.6	87.8	–
SyntaxBERT$_{\text{LARGE}}$ (Bai et al., 2021)	89.5	87.0	–
ISG – GCN – BERT$_{\text{BASE}}$ (Xu et al., 2022)	90.5	90.0	90.8
ISG – GCN – BERT$_{\text{LARGE}}$ (Xu et al., 2022)	90.8	90.4	91.9
ISG – GCN – RoBERTa$^{*}_{\text{LARGE}}$ (Xu et al., 2022)	91.4	91.2	93.3
Ours(SA – AGN – BERT$_{\text{BASE})}$	**90.8**	**90.4**	**91.5**
Ours(SA – AGN – BERT$_{\text{LARGE})}$	**90.8**	**90.7**	**92.3**
Ours(SA – AGN – RoBERTa$_{\text{LARGE})}$	**91.5**	**91.3**	93.1

Table 2
Comparison of different methods (PLM, fine-tuned, proposed) with low latency for SQL generation on tokens, latency, cost, memory and execution accuracy.

Method	Model	Avg Tokens	Latency(s)	Avg Cost ($)	Memory (GB)	EX (%)
C3SQL	GPT-3.5-Turbo	5702	199.57	0.0103	–	82
DINSQL	GPT-4	9571	899.674	0.2988	–	82.8
DAILSQL	GPT-4	930	87.42	0.0288	–	83.1
SUPERSQL	GPT-4	942	88.548	0.0377	–	87.01
RESDSQL-3B	T5	410	15.28	0.000123	6.71	81.8
RESDQL-3B+NatSQL	T5	410	15.76	0.000123	6.71	84.1
GraphixT5-3B+PICARD	T5	570	42.16	0.000171	6.71	81
ASKSQL(Proposed)	Llama-2-7b	300	8.23	0.00006	9.87	77.4
ASKSQL(Proposed)	CLlama-13b	300	14.41	0.000075	15.6	80.7
ASKSQL(Proposed)	CLlama-13b-quant	300	**4.6**	**0.000045**	**4.68**	79.2

Table 3
Comparison of Accuracy on Enterprise data for standard and complex queries.

Model	Standard Queries EX (%)	Complex Queries EX (%)	Average EX (%)
T5	77.02	63.74	70.38
Llama-7b-nl2sql	43.30	41.90	42.6
CodeLlama-13b	73.09	68.98	71.03
CodeLlama-13b-quantized	84.98	79.79	82.38

Table 4
Comparison of the effect of different component inclusion for latency and accuracy for CodeLlama-13b-quant model.

Component Inclusion	Inclusion Latency (seconds)	EX (%)	Average Cost ($)
Semantic Similarity	0.32	67.45	0.0000103
Semantic Similarity + Entity Swapping	0.66	72.20	0.0000202
Semantic Caching	0.43	84.56	0.0000139
Entity Swapping + Semantic Caching	1.5	88.21	0.0000243

on average, and GPT-4 used between 930 to 9571 tokens, which depicts generalization on large databases but is not practical for enterprise production due to cost and latency issues, DAILSQL (Gao et al., 2023) and SUPERSQL (Li, Luo, Chai, Li & Tang, 2024) made significant efforts towards reducing the average latency and average cost to 0.0288 s and 0.0377 s respectively. Fine-tuned models based on the T5 architecture, like RESDSQL-3B (Li et al., 2023) and its variants, consistently utilized fewer tokens (410 to 570) and showcased reduced latencies (15.28 to 42.16 s), demonstrating efficiency improvements through fine-tuning. Our proposed method, ASKSQL improves on the domain-specific fine-tuning by quantization and adapter-based fine-tuning including various Llama models achieving notable reductions in latency (4.6 to 14.41 s) and cost ($0.000045 to $0.000075), highlighting their optimization for economic and operational efficiency, with a slight reduction in accuracy.

Table 3 depicts the EX score for various PLM models tested in the pipeline for a custom enterprise data which has a specific domain and larger table dimensions as described in Section 4.1. Out of the four models, *CodeLlama-13b-quant* gave the best accuracy score of 84.98% on standard queries and 79.79% on JOINS and sub-queries, which is evident due to the large dimension of the tables and can be further reduced by adjusting the model attributes and dataset size.

Table 4 compares the average increase in latency per query, average accuracy score, and cost per query of different components in the proposed ASKSQL pipeline added to the vanilla query generator. Independently a simple Semantic Similarity (SS) based caching does not perform well with a poor accuracy of 67.45%, even when Entity Swapping (ES) is added on top of SS, the accuracy score still does not improve to a significant margin which is evident to users not selecting from the given history recommendations. In isolation, Semantic Caching emerges as a strong performer, demonstrating an increase to 84.56% (+2.175%) accuracy while also decreasing the response time by 0.43 s. Overall an average accuracy change from 84.98% to 88.21% is observed (+5.83%) and an average latency decrease from 4.6 s to 3.1 s (−32.6%) is observed when Entity Swapping and Caching Components are added on top of the query generator.

6. Discussion

6.1. Effect of semantic similarity and entity swapping

The utilization of the ANN index along with Vector Database ensures both speed and reliability in identifying semantic similarities, which is crucial for accurate natural language understanding in SQL query generation. Furthermore, the incorporation of caching in the pipeline enhances user experience by providing a dropdown-based interactive selection interface, while simultaneously reducing the computational overhead associated with repeated query calculations. Finally, entity swapping allows the pipeline to group similar queries together allowing access to direct database queries instead of generating the queries again. All these techniques help the pipeline to be more efficient when dealing with previously asked queries allowing for faster response times and reduced computational expenses.

6.2. Effect of semantic caching module

The Semantic Caching Module significantly reduces the number of operations performed by the Large Language Model (LLM) during query generation. However, one challenge is determining the optimal probability threshold for selecting cached results, as this can impact the balance between speed and accuracy. To address this, we employ a small yet robust model that can be fine-tuned based on the user's specific domain or schema-based data. This allows for efficient query generation while maintaining adaptability to different contexts, enhancing the overall effectiveness and usability of the pipeline.

6.3. Role of optimal schema selection

The Optimal Schema Selection mechanism plays a crucial role in enhancing the efficiency of schema selection by focusing on the most accurate schemas, rather than selecting all possible schemas. This approach allows the pipeline to be more efficient by reducing the overall computation cost. Moreover, the mechanism ensures data security by keeping the schema hidden, providing an abstraction layer that protects sensitive information. A unique feature of our approach is the ability to select individual columns from large tables, column selection is done by prioritizing the most relevant columns based on their probabilities, as outlined in Eq. (8). This comprehensive approach to schema and column selection enhances the effectiveness and security of the natural language to SQL pipeline, improving its usability and accuracy for users across various domains.

6.4. PLM selection for SQL generation

Selecting the right Language Model (LM) for generating SQL queries is a critical task. We employ a pre-trained LM that has been customized for specific use cases and further enhanced by adaptors. Instead of starting from scratch, we have found that utilizing a small LM trained specifically on large-scale SQL queries yields superior accuracy as evident from Table 1. When dealing with complex queries, a smaller LM with adaptors proves effective, however, the question and schema must be concise. We have also discovered that the accuracy of selecting the columns from the correct tables can be significantly improved by fine-tuning the adaptor with data containing the *table.column* notation. This approach significantly enhances the accuracy and efficiency of SQL query generation, particularly for complex questions and when selecting the correct columns.

7. Conclusion

This paper introduces an end-to-end Natural Language to SQL pipeline for faster and more efficient online searching on an enterprise database. The pipeline leverages an ANN index and vector similarity from historical searches for query recommendation, along with a custom-NER-based entity-swapping module, an association graph-based semantic caching network, an encoder-based model for entity item selection from the schema, and a QLoRA adapter-based SQL LLM fine-tuning for search preference optimization and accuracy. The observed average accuracy increases as the module adapts to new queries. Although the proposed ASKSQL pipeline is cost-efficient, accurate, and fast it suffers from large-scale domain analytics where the search queries are from different domains due to the usage of small language models and finance-tuned semantic caching component. Future work includes extending the caching and domain expansion from finance to the general domain as a plugin and introducing more resilient recommendations. These enhancements aim to further improve the pipeline's performance and applicability across various domains.

CRediT authorship contribution statement

Arpit Bajgoti: Conceptualization, Methodology, Software, Validation. **Rishik Gupta:** Data curation, Writing – original draft, Formal Analysis. **Rinky Dwivedi:** Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

References

Arcadinho, S., Aparício, D., Veiga, H., & Alegria, A. (2022). T5QL: Taming language models for SQL generation. arXiv preprint arXiv:2209.10254.

Bai, J., Wang, Y., Chen, Y., Yang, Y., Bai, J., Yu, J., et al. (2021). Syntax-BERT: Improving pre-trained transformers with syntax trees. arXiv preprint arXiv:2103.04350.

Cao, R., Chen, L., Chen, Z., Zhao, Y., Zhu, S., & Yu, K. (2021). LGESQL: line graph enhanced text-to-SQL model with mixed local and non-local relations. arXiv preprint arXiv:2106.01093.

Chandwani, G., & Ahlawat, A. (2023). Fuzzy local information C-means based clustering and fractional dwarf mongoose optimization enabled deep learning for relevant document retrieval. *Engineering Applications of Artificial Intelligence*, *126*, Article 106954.

Chen, Q., Zhu, X., Ling, Z., Wei, S., Jiang, H., & Inkpen, D. (2016). Enhanced LSTM for natural language inference. arXiv preprint arXiv:1609.06038.

Conneau, A., Kiela, D., Schwenk, H., Barrault, L., & Bordes, A. (2017). Supervised learning of universal sentence representations from natural language inference data. arXiv preprint arXiv:1705.02364.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.

Domínguez, J. m. (2023). Blar-SQL: Faster, stronger, smaller NL2sql.

Dong, X., Zhang, C., Ge, Y., Mao, Y., Gao, Y., Lin, J., et al. (2023). C3: Zero-shot text-to-sql with chatgpt. arXiv preprint arXiv:2307.07306.

Ercan, G., & Haziyev, F. (2019). Synset expansion on translation graph for automatic wordnet construction. *Information Processing & Management*, *56*(1), 130–150.

Fariha, A., & Meliou, A. (2019). Example-driven query intent discovery: Abductive reasoning using semantic similarity. arXiv preprint arXiv:1906.10322.

Gan, Y., Chen, X., Xie, J., Purver, M., Woodward, J. R., Drake, J., et al. (2021). Natural SQL: Making SQL easier to infer from natural language specifications. arXiv preprint arXiv:2109.05153.

Gao, D., Wang, H., Li, Y., Sun, X., Qian, Y., Ding, B., et al. (2023). Text-to-sql empowered by large language models: A benchmark evaluation. arXiv preprint arXiv:2308.15363.

Gong, Z., Liu, J., Wang, Q., Yang, Y., Wang, J., Wu, W., et al. (2023). PreQuant: A task-agnostic quantization approach for pre-trained language models. arXiv preprint arXiv:2306.00014.

Jafari, H., & Sheykhan, A. (2021). Using a new algorithm to improve the search answer in quadratic assignment problem (QAP). *International Journal of Research in Industrial Engineering*, *10*(2), 165–173.

Johnson, J., Douze, M., & Jégou, H. (2019). Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*, *7*(3), 535–547.

Khot, T., Sabharwal, A., & Clark, P. (2018). Scitail: A textual entailment dataset from science question answering. In *Proceedings of the AAAI conference on artificial intelligence*: *Vol. 32*, (No. 1).

Kocetkov, D., Li, R., Ben Allal, L., Li, J., Mou, C., Muñoz Ferrandis, C., et al. (2022). The stack: 3 TB of permissively licensed source code. *Preprint*.

Labs, N. S. (2023). NSText2SQL: An open source text-to-SQL dataset for foundation model training.

Li, J., Hui, B., Qu, G., Yang, J., Li, B., Li, B., et al. (2024). Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems*, *36*.

Li, B., Luo, Y., Chai, C., Li, G., & Tang, N. (2024). The dawn of natural language to SQL: Are we fully ready?. arXiv preprint arXiv:2406.01265.

Li, H., Zhang, J., Li, C., & Chen, H. (2023). Resdsql: Decoupling schema linking and skeleton parsing for text-to-sql. In *Proceedings of the AAAI conference on artificial intelligence*: *Vol. 37*, (No. 11), (pp. 13067–13075).

Li, W., Zhang, Y., Sun, Y., Wang, W., Li, M., Zhang, W., et al. (2019). Approximate nearest neighbor search on high dimensional data—experiments, analyses, and improvement. *IEEE Transactions on Knowledge and Data Engineering*, *32*(8), 1475–1488.

Lin, J., Pradeep, R., Teofili, T., & Xian, J. (2023). Vector search with openai embeddings: Lucene is all you need. arXiv preprint arXiv:2308.14963.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., et al. (2019). Roberta: A robustly optimized bert pretraining approach. arXiv preprint arXiv:1907.11692.

Liu, T., Wang, X., Lv, C., Zhen, R., & Fu, G. (2020). Sentence matching with syntax-and semantics-aware BERT. In *Proceedings of the 28th international conference on computational linguistics* (pp. 3302–3312).

Macdonald, C., & Tonellotto, N. (2021). On approximate nearest neighbour selection for multi-stage dense retrieval. In *Proceedings of the 30th ACM international conference on information & knowledge management* (pp. 3318–3322).

Pawar, A., & Mago, V. (2018). Calculating the similarity between words and sentences using a lexical database and corpus statistics. arXiv preprint arXiv:1802.05667.

Peng, B., Li, C., He, P., Galley, M., & Gao, J. (2023). Instruction tuning with gpt-4. arXiv preprint arXiv:2304.03277.

Pourreza, M., & Rafiei, D. (2024). Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *Advances in Neural Information Processing Systems*, *36*.

Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., et al. (2018). *Improving language understanding by generative pre-training*. San Francisco, CA, USA.

Roziere, B., Gehring, J., Gloeckle, F., Sootla, S., Gat, I., Tan, X. E., et al. (2023). Code llama: Open foundation models for code. arXiv preprint arXiv:2308.12950.

Saeed, M., De Cao, N., & Papotti, P. (2023). Querying large language models with SQL. arXiv preprint arXiv:2304.00472.

Sheng, Y., Zheng, L., Yuan, B., Li, Z., Ryabinin, M., Fu, D. Y., et al. (2023). High-throughput generative inference of large language models with a single gpu. arXiv preprint arXiv:2303.06865.

Shi, P., Ng, P., Wang, Z., Zhu, H., Li, A. H., Wang, J., et al. (2021). Learning contextual representations for semantic parsing with generation-augmented pre-training. In *Proceedings of the AAAI conference on artificial intelligence*: *Vol. 35*, (No. 15), (pp. 13806–13814).

Xu, C., Xu, J., Dong, Z., & Wen, J.-R. (2022). Semantic sentence matching via interacting syntax graphs. In *Proceedings of the 29th international conference on computational linguistics* (pp. 938–949).

Xu, C., Xu, J., Dong, Z., & Wen, J.-R. (2023). Syntactic-informed graph networks for sentence matching. *ACM Transactions on Information Systems*, *42*(2), 1–29.

Yan, H., Deng, B., Li, X., & Qiu, X. (2019). TENER: adapting transformer encoder for named entity recognition. arXiv preprint arXiv:1911.04474.

Yu, T., Wu, C.-S., Lin, X. V., Wang, B., Tan, Y. C., Yang, X., et al. (2020). Grappa: Grammar-augmented pre-training for table semantic parsing. arXiv preprint arXiv:2009.13845.

Yu, W., Xu, C., Xu, J., Pang, L., & Wen, J.-R. (2022). Distribution distance regularized sequence representation for text matching in asymmetrical domains. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, *30*, 721–733.

Yu, T., Zhang, R., Yang, K., Yasunaga, M., Wang, D., Li, Z., et al. (2018). Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2018 conference on empirical methods in natural language processing*. Brussels, Belgium: Association for Computational Linguistics.

Zhang, Z., Wu, Y., Zhao, H., Li, Z., Zhang, S., Zhou, X., et al. (2020). Semantics-aware BERT for language understanding. In *Proceedings of the AAAI conference on artificial intelligence*: *Vol. 34*, (No. 05), (pp. 9628–9635).

Zhong, V., Xiong, C., & Socher, R. (2017). Seq2SQL: Generating structured queries from natural language using reinforcement learning. CoRR, arXiv:1709.00103.