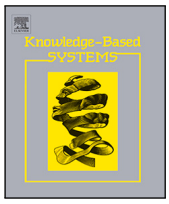
Knowledge-Based Systems 306 (2024) 112718

Contents lists available at ScienceDirect

Knowledge-Based Systems

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



A confidence-based knowledge integration framework for cross-domain table question answering

Yuankai Fan [a], Tonghui Ren [a], Can Huang [a], Beini Zheng [a], Yinan Jing [a], Zhenying He [a], Jinbao Li [c], Jianxin Li [b,*]

[a] School of Computer Science, Fudan University, Shanghai, China
[b] School of Business and Law, Edith Cowan University, Joondalup, Australia
[c] Shandong Artificial Intelligence Institute, School of Mathematics and Statistics, Qilu University of Technology (Shandong Academy of Science), Jinan, China

ARTICLE INFO

Keywords:
Table question answering
Natural language interfaces
Sequence-to-sequence model
Confidence estimation
Similarity ranking

ABSTRACT

Recent advancements in TableQA leverage sequence-to-sequence (Seq2seq) deep learning models to accurately respond to natural language queries. These models achieve this by converting the queries into SQL queries, using information drawn from one or more tables. However, Seq2seq models often produce uncertain (low-confidence) predictions when distributing probability mass across multiple outputs during a decoding step, frequently yielding translation errors. To tackle this problem, we present CKIF, a *confidence-based knowledge integration framework* that uses a two-stage deep-learning-based ranking technique to mitigate the low-confidence problem commonly associated with Seq2seq models for TableQA. The core idea of CKIF is to introduce a flexible framework that seamlessly integrates with any existing Seq2seq translation models to enhance their performance. Specifically, by inspecting the probability values in each decoding step, CKIF first masks out each low-confidence prediction from the predicted outcome of an underlying Seq2seq model. Subsequently, CKIF integrates prior knowledge of query language to generalize masked-out queries, enabling the generation of all possible queries and their corresponding NL expressions. Finally, a two-stage deep-learning ranking approach is developed to evaluate the semantic similarity of NL expressions to a given NL question, hence determining the best-matching result. Extensive experiments are conducted to investigate CKIF by applying it to five state-of-the-art Seq2seq models using a widely used public benchmark. The experimental results indicate that CKIF consistently enhances the performance of all the Seq2seq models, demonstrating its effectiveness for better supporting TableQA.

1. Introduction

Question-answering systems are designed to respond to a wide range of questions by utilizing evidence from structured knowledge bases such as tables [1] and knowledge graphs [2,3], as well as unstructured texts [4] or images [5,6]. In particular, table-based question answering (TableQA) is crucial for extracting and utilizing the vast amounts of structured data found in tables. It enables more efficient and accurate information retrieval across numerous fields by understanding tables and using evidence from relational databases to answer natural language (NL) questions. One line of research in TableQA involves translating NL questions into query languages such as SQL [7–11].

In light of the remarkable achievements of *encoder–decoder architecture* across a spectrum of natural language processing (NLP) tasks in recent years [12–14], existing works employing sequence-to-sequence (Seq2seq) deep-learning models have gained widespread recognition for addressing NL to SQL translation (NL2SQL) challenges [15–20]. That is, these models attempt to encode the input NL question (as well as the underlying data schema) and generate the target sequence token by token, proceeding from left to right while maximizing the posterior probability at each decoding step. Recently, large-scale language models have emerged as a new approach to address the TableQA problem [21–25], significantly enhancing their capabilities.[1]

However, due to the drastically different data and schema formats in TableQA tasks, the overall translation accuracy of existing models still remains far from practical use. For instance, the cutting-edge Seq2seq-based model [26], evaluated against the widely-used SPIDER

* Corresponding author.
E-mail addresses: fanyuankai@fudan.edu.cn (Y. Fan), thren22@m.fudan.edu.cn (T. Ren), huangcan22@m.fudan.edu.cn (C. Huang), bnzheng23@m.fudan.edu.cn (B. Zheng), jingyn@fudan.edu.cn (Y. Jing), zhenying@fudan.edu.cn (Z. He), lijinb@sdas.org (J. Li), jianxin.li@ecu.edu.au (J. Li).

[1] This paper primarily focuses on Seq2seq-based models, given that they are still widely adopted in this domain.
[2] https://yale-lily.github.io/spider

https://doi.org/10.1016/j.knosys.2024.112718
Received 25 July 2024; Received in revised form 15 October 2024; Accepted 6 November 2024
Available online 15 November 2024

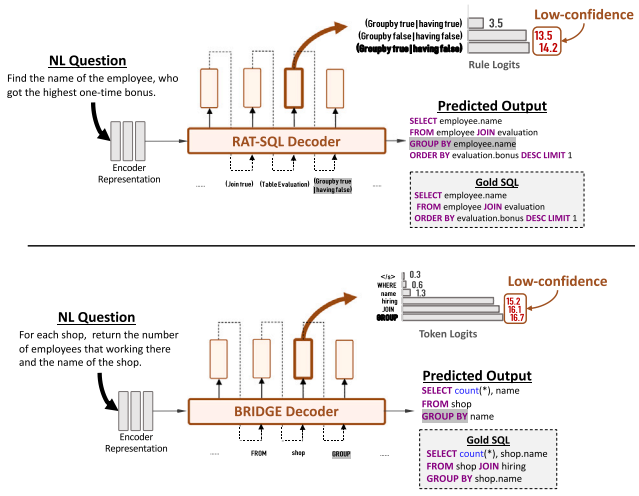


Fig. 1. Two low-confidence situations observed on existing Seq2seq deep-learning models, RAT-SQL [16] and BRIDGE [15], where RAT-SQL uses an abstract syntax tree-based decoder and BRIDGE uses a standard decoder.

benchmark [27], achieves a translation accuracy (defined as syntactic equivalence accuracy) of merely 74.0% on the test set according to the official leaderboard.[2]

The major problem we posit is that existing approaches primarily rely on the sequential decoding process derived from the Seq2seq paradigm to generate target outputs without any intervention. Unfortunately, the decoding process may produce erroneous outputs when models become "un-confident". This occurs when their prediction distribution spreads the probability mass across several outputs during specific decoding steps, which we refer to as *low-confidence situations*. Fig. 1 depicts low-confidence situations in existing models on TableQA, RAT-SQL [16] and BRIDGE [15]. While the models correctly predict the majority of the query sequences in both examples, each translation model encounters minor issues when low-confidence scenario comes: At the third decoding step of the models, RAT-SQL mistakenly selects the `GROUP BY` rule when the prediction distribution is concentrated on the first two rules, whereas BRIDGE fails to predict the join operation but straightforwardly select `GROUP BY` when the distribution is concentrated on the first three tokens. Moreover, our experiments reveal that RAT-SQL has about a 10% chance of encountering this low-confidence situation, while for BRIDGE, it even increases to 20% (See details in Section 4.). Hence, these low-confidence situations significantly undermine the models' reliability, highlighting the critical need for improved strategies to handle such scenarios and enhance overall accuracy.

In this paper, we present **CKIF**, a **C**onfidence-based **K**nowledge **I**ntegration **F**ramework to complement the traditional sequential decoding paradigm of existing models for the TableQA systems. CKIF operates directly on the outputs of Seq2seq models (i.e., SQL program in the case of NL2SQL translation), which can be optionally enabled at inference time. Notably, **CKIF is a unified framework that is compatible with any existing Seq2seq deep-learning translation models to improve their performance, which can be easily extended to apply to constrained decoding in other semantic parsing-related tasks in a similar way.**

Inspired by previous work on confidence estimation in machine translation [28–30], CKIF uses a token-level confidence measurement to find the decoded tokens with low confidence as potential errors and applies a three-stages generate-and-rank strategy for post-editing as follows. First, CKIF identifies and masks out the low-confidence predicted tokens from the Seq2seq translation model's output. It then generalizes the masked-out query to encompass all potential queries according to the underlying database schema and SQL grammar. CKIF employs a straightforward rule-based mapping from generalized queries to NL expressions. Finally, CKIF employs a two-stage similarity ranking pipeline to learn relevant patterns between the NL expressions and the provided NL question, leading to a more accurate answer using the best-matching query. *Here, by leveraging a generate-then-rank strategy in the post-editing process, we posit that* CKIF *is able to produce better translation than the conventional end-to-end generation approach used in Seq2seq translation models.*

To assess the effectiveness of CKIF, we perform experiments on the widely recognized TableQA benchmark named SPIDER, integrating CKIF with five state-of-the-art Seq2seq deep-learning models: BRIDGE [15], RAT-SQL [16], GAP [18], LGESQL [19] and RESDSQL [20]. The experimental results indicate a notable enhancement in the performance of all semantic parsers when CKIF is applied. Specifically, when incorporating CKIF, the translation accuracy of LGESQL reaches 77.4% on the validation set of the SPIDER benchmark, outperforming the majority of the most advanced deep-learning translation models.[3]

Our Contributions. We make the following contributions:

- Through the lens of confidence estimation, we introduce a low-confidence detection approach for TableQA to perceive potential errors in the traditional Seq2seq paradigm by observing the sequential decoding process.
- We propose a unified generate-and-rank strategy for Seq2seq models to tackle TableQA. By formulating the translation problem as a document retrieval task, we train a two-stage ranking pipeline to rank for optimal results.
- We apply CKIF to five state-of-the-art Seq2seq-based models on public TableQA benchmark. Our observations consistently demonstrate performance improvements in established Seq2seq-based models upon incorporating CKIF, thereby demonstrating its effectiveness.

The structure of the paper is as follows: Section 2 offers an overview of CKIF. Section 3 delves into the methodologies. The experiments on the public benchmark are conducted in Section 4, and Section 5 examines related literature. Finally, conclusions are drawn in Section 6.

2. Preliminaries

In this section, we formalize the problem of NL2SQL and introduce some preliminaries to understand CKIF.

2.1. Problem formulation

Formally, given a user question q expressed in NL and a database D with its associated schema S, the NL2SQL task aims to translate q into a SQL query l that can be executed on D to answer the user question q. This paper focuses on Seq2seq-based NL2SQL models. Here, we give the definition of NL2SQL with Seq2seq-based model solutions.

Given an NL question q and a structural database schema $S = (T, C, R)$ where $T = (t_1, t_2, \dots)$ presents multiple contained tables, $C = (c_1, c_2, \dots)$ indicates associated columns, and $R = (r_1, r_2, \dots)$ denotes foreign keys relations. A Seq2seq-based NL2SQL model aims to use its encoder to compute a contextual representation $\boldsymbol{c}$ by jointly embedding the NL query q with schema S, followed by the usage of its decoder to compute a distribution $P(Y \mid \boldsymbol{c})$ conditioned on $\boldsymbol{c}$ to generate a SQL query l corresponding to q.

[3] As the test set of SPIDER hosted on an evaluation server is inaccessible, we conducted our experiments using the validation set instead.

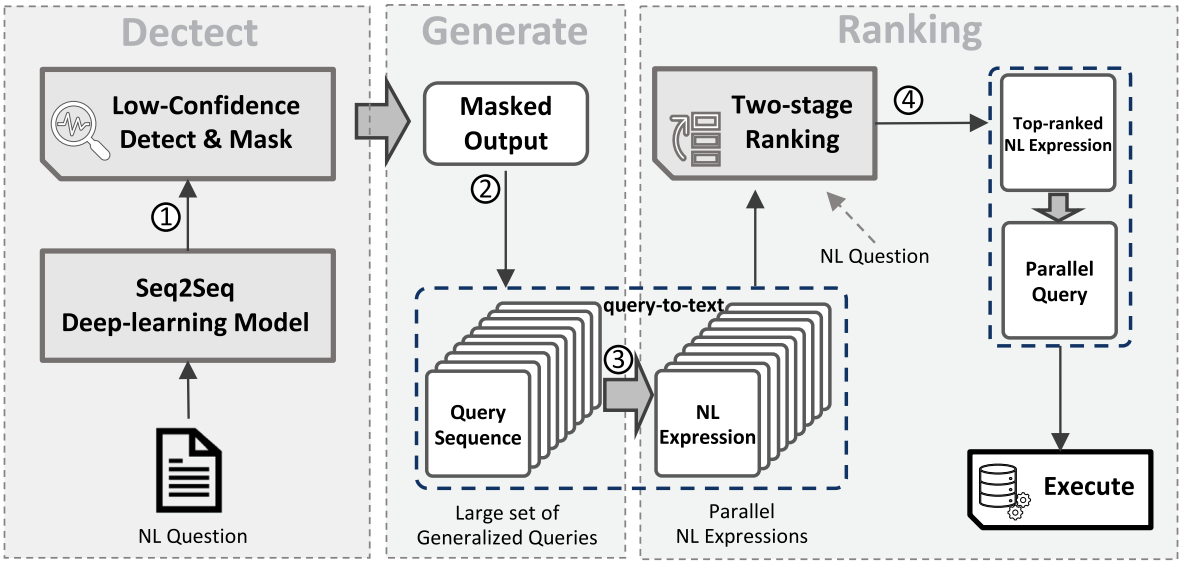


Fig. 2. The workflow of CKIF is as follows: (1) Detection: CKIF evaluates token-level confidence to identify and mask out uncertain predictions in the output sequence of a Seq2seq translation model; (2) Generation: CKIF generalizes the masked output to produce a large set of candidate queries based on the underlying database schema and query grammar; (3) Ranking: CKIF utilizes a two-stage ranking pipeline to determine the best-matching query (and hence the answer) for a given NL question.

2.2. Seq2seq-based NL2SQL models

Seq2seq models represent vital deep learning techniques designed to manage sequential data, facilitating the seamless conversion of sequences between distinct domains. These models can take the form of a fundamental encoder–decoder network built upon recurrent neural networks (RNNs), attention-based encoder–decoder RNNs, or transformer-based models.

We use the Seq2seq framework [31] for the conversion of NL questions into SQL representations. Given an NL question $\mathcal{Q} = \{q_1, q_2, \ldots, q_n\}$ and a database schema $\mathcal{S} = \langle \mathcal{C}, \mathcal{T}, \mathcal{F} \rangle$ that contains columns $\mathcal{C} = \{c_1, c_2, \ldots, c_{|\mathcal{C}|}\}$, tables $\mathcal{T} = \{t_1, t_2, \ldots, t_{|\mathcal{T}|}\}$, and a set of foreign-primary key pairs $\mathcal{F} = \{(c_{f_1}, c_{p_1}), (c_{f_2}, c_{p_2}), \ldots, (c_{f_{|\mathcal{C}|}}, c_{p_{|\mathcal{C}|}})\}$, a contextual representation, denoted as $\boldsymbol{c}$, is computed by the encoder, embedding the question $\mathcal{Q}$ together with schema $\mathcal{S}$. Subsequently, the model uses a decoder to compute a distribution $P(\mathcal{Y}|\boldsymbol{c})$ over the SQL programs $\mathcal{Y} = (y_1, \ldots, y_m)$. According to various model architectures, the decoder's learning target $\mathcal{Y}$ may encompass raw SQL tokens [15,32], intermediate representations [33,34], or abstract syntax trees [16,18,19].

2.3. Confidence estimation in Seq2seq translation

Confidence estimation in Seq2seq translation involves calculating a confidence score that represents the model's belief in the correctness of the generated translation.

Formally, given an NL question Q and a predicted meaning representation a produced by an underlying translation model, the confidence score of the predicted outcome is estimated as $s(Q, a) \in (0, 1)$. A higher score indicates greater confidence in the accuracy of the prediction. Depending on different confidence estimation methods, the confidence score s can be calculated using either probability-based [35] or dropout-based measures [36].

3. CONFIDENCE-BASED KNOWLEDGE INTEGRATION FRAMEWORK

In this section, we first give an overview of our proposed approach, CKIF. We then describe details about the related techniques used in CKIF, including low-confidence detection, confidence-aware generalization, query translation, and a two-stage ranking pipeline.

3.1. Overview

Fig. 2 provides an overview of CKIF. By plugging into a Seq2seq-based NL2SQL translation model,

① CKIF first measures the token-level confidence by during each decoding step, masking out the low-confidence predictions from the outputs.
② Secondly, CKIF generalizes the masked output to obtain a large set of candidate queries based on the underlying database schema and the query grammar.
③ Thirdly, a query translation model is used to synthesize the NL expression for each candidate query.
④ Lastly, we get the most suitable matching query result (and hence the answer) for a given NL question through a two-stage similarity ranking pipeline.

Confidence Detection. This step in Fig. 2-① examines the decoding states of the underlying model to determine if any uncertain predictions exist in the output sequence. This is accomplished by checking the probability distribution at each decoding step. Consider the bottom example shown in Fig. 1. when predicting the `GROUP BY` clause, it becomes evident that the majority of the probability mass is concentrated on the top-3 tokens, namely "*group*", "*join*" and "*hiring*". This concentration of probability mass suggests a low-confidence prediction for BRIDGE.

Note that if any low-confidence predictions are detected, CKIF masks all those predictions from the SQL output sequence. Following is the *masked-out query* that we obtained from the above example,

```
SELECT count(*), name FROM shop [MASK]
```

Candidate Generation. This step in Fig. 2-② is to generate all possible SQL queries based on the masked-out query obtained in the previous step. CKIF achieves this by using the masked-out query as a SQL "skeleton" and then generating a range of SQL queries by adhering to SQL grammar and the underlying database schema. In the BRIDGE example of Fig. 1, one possible SQL query that can be generated using the masked-out query above is the following,

```
SELECT count(*), shop.name FROM shop
JOIN hiring GROUP BY shop.name
```

Query Translation. This step in Fig. 2-③ is to utilize query translation techniques to generate an NL description for each generalized SQL query. For instance, here is the description of the above generalized query: "*Find the number of hiring and the name of shop for each name of the shop.*".

Semantic-Similarity Ranking. To find the correct generalized query corresponding to the given NL question, this step in Fig. 2-④ employs a two-stage ranking pipeline. In this pipeline, the first stage of ranking generates a collection of highly relevant candidates, subsequently assessed by the second-stage ranking model to ascertain the top-ranked query. At this point, we train the ranking models to rank the similarity in NL semantics. For example, consider the given NL question depicted in Fig. 1. The ranking pipeline identifies the above generalized (i.e., ground-truth) SQL query as the closest match to the given NL question, thereby producing the translation outcome.

3.2. Low-confidence detection

The sequential decoding process of a Seq2seq-based NL2SQL model primarily generates each output token at each decoding time step based on maximum likelihood. Previous approaches typically adopt a "passive" strategy, relying solely on this end-to-end decoding process to obtain the final output. An important question here is how to improve the naive decoding process and identify potential errors within it. Therefore, we introduce a token-level introspection mechanism to achieve this goal.

Intuitively, if a base translation model produces similar output values for multiple top predictions during the Seq2seq decoding process (as seen in Fig. 1), it suggests model uncertainty and indicates a low-confidence prediction. For simplicity, we assume that the decoder of the base translation model is constructed by a vanilla RNN, though the symptoms for other more complex models, such as Transformers, are analogous. A detailed explanation is provided below.

During the inference process, the decoder utilizes the RNN to unfold the target information. At step j, the target hidden state h_j is computed as

$$\boldsymbol{h}_j = \mathbf{RNN}(\boldsymbol{e}_{y_{j-1}}, \boldsymbol{h}_{j-1}, \boldsymbol{c}) \tag{1}$$

The probability distribution P_j, which encompasses all potential candidate symbols within the SQL vocabulary at step j, is generated by conditioning on the embedding of the symbol predicted at the previous step, the context representation from the source, and the hidden state.

$$\begin{aligned} \boldsymbol{t}_j &= g(\boldsymbol{e}_{y-1}, \boldsymbol{h}_j, \boldsymbol{c}) \\ \boldsymbol{l}_j &= \mathbf{W}_o(\boldsymbol{t}_j) \\ P_j &= \mathrm{softmax}(\boldsymbol{l}_j) \end{aligned} \tag{2}$$

where g stands for a linear transformation, W_o is applied to transform t_j into logits l_j that represents the non-normalized predictions for each target symbol. These logits are then normalized using a softmax function to derive output probabilities for each symbol. In the end, the symbol with the highest probability is selected as the predicted target symbol

$$y_j = \mathrm{argmax}(P_j) \tag{3}$$

Due to the fast-growing nature of the exponential function used in softmax computation, even small adjustments to the softmax inputs can result in significant alterations to the output distribution. Moreover, prediction probabilities derived from softmax distributions often lack a direct correlation with confidence [35]. Therefore, we employ an indicator function $\mathbb{L}$ to estimate the output at step j, utilizing the raw predictions, or logits, to assess confidence.

$$\mathbb{L}_{y_j} = \begin{cases} 0 & \text{if } | \, l_j^{top_1} - l_j^{top_2} \, | < \omega \\ 1 & \text{otherwise} \end{cases} \tag{4}$$

where $l_j^{top_1}$ and $l_j^{top_2}$ represent the top two logits at step j, while ω denotes a threshold value. That is, if the gap between the top two logits is smaller than the threshold ω, the prediction at step j is considered as low confidence.

Discussion. To balance the masking signal and the noise introduced during detection, a certain percentage of predictions should be identified as low confidence. If the threshold ω is set to too high (i.e., many tokens are masked), the resulting masked output can become excessively noisy, making it difficult to efficiently generate candidate queries in a controllable manner in the next stage. Conversely, if the threshold is too low (i.e., few tokens are masked), CKIF receives a weak correction signal for effective generation. We consider the threshold ω as a hyper-parameter for CKIF, chosen through careful experimentation to optimize the trade-off.

3.3. Confidence-aware candidate generation

Leveraging query knowledge is crucial for accurate generalization in CKIF. Given the complexity and strictness of the query language (i.e., SQL), a deep understanding of database schemas and syntax helps us better comprehend and address uncertainties present in the model. Therefore, we incorporate query knowledge into the generalization process to ensure that the generated queries align with the structure and syntax of the actual database.

By treating the model's predictions as masked query results and performing corresponding replacements based on database schemas and SQL syntax, we can generate a more precise set of possible queries. This query knowledge-based approach not only enhances the quality of generalization but also significantly improves the semantic alignment between the final results and the input NL expressions.

To view the masked query result as a template [37], we instantiate over the underlying database schema and generalize it to a collection of queries by randomly replacing each masked token with substitutions based on its associated type. In the context of CKIF, we classify the masked tokens into four types, each with specific substitutions derived from SQL grammar and schema understanding:

- *Masked Schema Token.* This kind of token represents low confidence in specific schema-related SQL tokens. In the example of Fig. 1, the column `employee.name` in **`SELECT`** clause or the table `employee` in **`FROM`** clause may be masked if low-confidence detects.
  Substitutions. By examining the **`FROM`** clauses, we first obtain all the valid tables[4] from the current SQL template, and randomly substitute a table or an associated column.
- *Masked Keyword Token.* This kind of token indicates low confidence in specific SQL clauses or keywords. For example, a masked-**`GROUP`** token shows the uncertainty about the whole **`GROUP`** clause, whereas a masked-`avg` token reflects the uncertainty about the aggregation.
  Substitutions. We follow the syntax of SQL grammar to replace the tokens. Notably, for substituting an SQL clause, we establish a series of constraints to restrict the syntactic complexity of each clause, preventing excessive search space exploration.
- *Masked Conjunction Token.* This token denotes the uncertainty of the number of clause elements. For example, a masked-**`AND`** token in the **`WHERE`** clause or a masked-`comma` token in the **`SELECT`** clause represent the uncertainty of the predicted number of predicates and the number of projections, respectively.
  Substitutions. Depend on the associated type of the token in SQL grammar, either adding or removing schema-related tokens (e.g., a column in **`SELECT`** clause), or a span of SQL elements (e.g., a predicate in **`WHERE`** clause).

[4] If any masked-table token exists, we may add other tables that have primary-foreign key relationships with the ones in the SQL template.

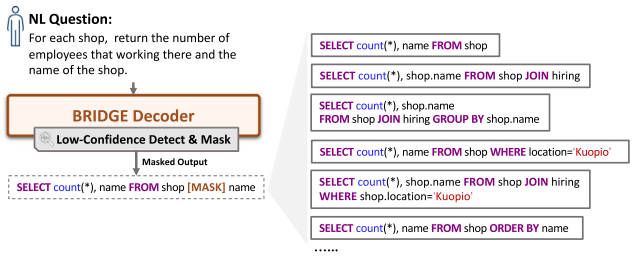


Fig. 3. Candidate query generation in CKIF.

- *Unpredicted Masked Token.* Even when a specific SQL clause has not been predicted, uncertainty may still be present. Therefore, we introduce masked tokens that represent unpredicted elements to account for this uncertainty. For example, an isolated masked-**WHERE** token may be added into a no-**WHERE** masked query output.

For the bottom example in Fig. 1, we present the candidate query generation results in Fig. 3. As the **GROUPBY** token is masked, the *Masked Keyword Token* rule is applied to generate all possible candidate queries in an attempt, for example, by removing **GROUPBY** clause, replacing it with a **JOIN** operation, or replacing with **WHERE** clause.

In this way, the generalization process generates all potential queries, significantly increasing the probability of correctly answering a given input NL question.

3.4. Query translation

For each generated query, a crucial task is to generate a high-quality NL expression that is logically consistent with the corresponding query to represent its semantics. In the field of NLP, translating queries into NL presents a unique challenge, particularly when dealing with complex or nested queries, where performance consistently degrades due to quality issues of the generation [38,39].

To overcome this challenge, utilizing query knowledge becomes imperative, which involves a comprehensive understanding of query syntax, semantics, and the functional relationships within queries. This ensures that the translation not only captures the literal query structure but also conveys the intent and context of the query, making the NL output more intuitive and relevant for users. Therefore, we opt to adopt and implement the rule-based method outlined in [40]. This approach enables us to construct a simple yet more reliable query translation generative model aimed at preserving overall translation quality.

Specifically, CKIF begins by treating a query as a text string and segments it into small segments, representing individual clauses akin to sub-trees in a parse tree. Subsequently, it constructs a directed graph of the SQL query on a clause-by-clause basis. The query graph, denoted as $G_q(V_q, E_q)$, is a graph where each node in V_q corresponds to an element within the query (e.g., the column `name` in the **SELECT** clause); Each edge in E_q has a specific type, signifying a particular relationship linking two query elements. For instance, in the SQL query shown in Fig. 1, the `employee` table node is associated with a "join" type edge that connects with the `evaluation` table node and a "select" type edge connects with the `name` column node and. Then, each element, whether a node or an edge, is assigned a label[5] to specify its semantics. Lastly, we traverse the graph using the traversal algorithm introduced in [40] and then concatenate the element-based labels encountered along the way to generate the description by incorporating descriptive expressions, such as *'find'*, *'return'*, among others.

[5] In our experiments' setup, we utilize the annotations of table/column names provided by SPIDER for its databases, employing them as node labels. Following [40], default labels are assigned for edge labeling.

Remarks. The need for creating the graph is essential for an efficient query translation process. That is, representing an SQL query as a graph allows for a better mapping between SQL elements (e.g., tables, columns, joins) and NL structures by capturing relationships and dependencies. In particular, this representation simplifies the translation of complex queries by breaking them down into simpler sub-graphs, thereby decomposing a complex query translation problem as a set of simpler tasks.

3.5. Two-stage similarity ranking

To select the optimal outcome from a vast array of NL expressions, we frame the selection problem as a *document retrieval task* and use the learning-to-rank techniques [41,42] to solve. This is achieved by leveraging existing powerful pre-trained language models as the foundational architecture and fine-tuning them with various learning objectives.

Fig. 4 illustrates the network architectures of the two ranking models employed in CKIF. The left network architecture represents the first-stage retrieval model, constructed upon the Siamese BERT-network [43]. The retrieval model adapts the pre-trained BERT model [44] utilizing Siamese and triplet network structures [45] to produce sentence embeddings. Conversely, the right network architecture corresponds to the re-ranking model, which employs a sentence pair classification structure based on BERT to determine the relevance (namely binary classification) between the provided NL question-description pairs.

3.5.1. First-phase retrieval model

Given an NL question $\mathcal{U}$ and a complete set of generated NL expressions $\mathcal{D} = \{d_1, d_2, \ldots, d_n\}$, the first stage retrieval model aims to extract top-k candidate NL expressions $\mathcal{D}' = \{d'_1, d'_2, \ldots, d'_k\}$ from the full set $\mathcal{D}$.

We utilize the Sentence-BERT model [43], which employs a siamese network architecture, to generate semantically meaningful embeddings for both the input NL question $\mathcal{U}$ and each NL expression d_i. Specifically, to rank these paired expressions, a similarity score is computed based on cosine similarity using the derived embeddings as below,

$$s_i = \mathrm{E}_a(\mathcal{U}) \cdot \mathrm{E}_b(d_i) \tag{5}$$

where E_a and E_b represent the two distinct encoders used to produce the respective semantic embeddings for a given question $\mathcal{U}$ and an NL expression d_i. We then use the binary cross-entropy loss to train the model,

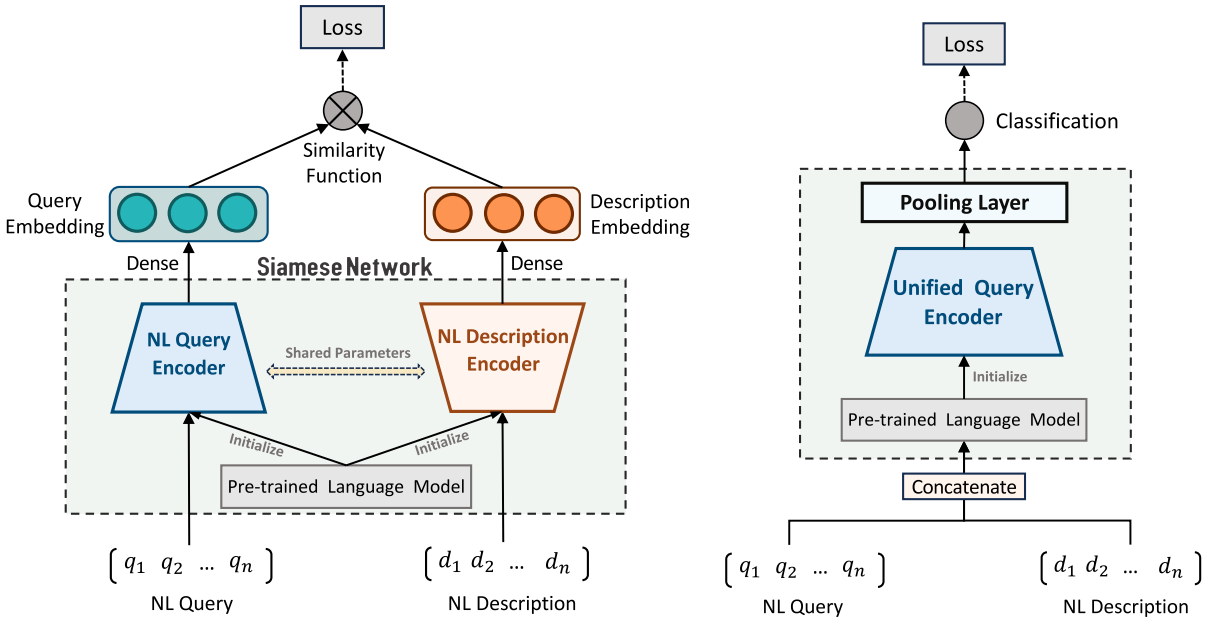
$$loss_{BCE} = -\frac{1}{n}\sum_{i=1}^{n}(y_i log(\sigma(s_i)) + (1 - y_i) log(1 - \sigma(s_i))) \tag{6}$$

in which $\sigma(\cdot)$ is the sigmoid function used to convert s_i into a probability, y_i is the binary label indicating whether the pair is similar (i.e., $y_i = 1$) or dissimilar (i.e., $y_i = 0$).

Fine-tune Data Generation. The fine-tune data for the sentence-BERT model includes a set of triples $\{(q_i, d_i, s_i)\}_{i=1}^{N}$, where q_i represents an NL question, d_i denotes an NL expression and s_i signifies the semantic similarity score between d_i and q_i. The calculation of the score s_i proceeds as follows. In the beginning, s_i is initialized to 1, and subsequently, each clause of the SQL query used to generate the NL expression d_i is compared with the "gold" query provided for q_i. Whenever a clause does not match, a penalty is imposed on the s_i value. This process iterates until all clauses have been compared or s_i reaches 0.

3.5.2. Second-phase re-ranking model

Given an input NL question $\mathcal{U}$ and a set of top-k candidate NL expressions $\mathcal{D}' = \{d'_1, d'_2, \ldots, d'_k\}$, the objective of the re-ranking model is to compute a relevance score s for each candidate NL expression $d'_i \in D'$ concerning the NL question $\mathcal{U}$. These scores are then ranked to identify the top-ranked NL expression.



(a) Retrieval model network (b) Re-ranking model network

Fig. 4. The overall network architectures of the two-stage ranking models.

The re-ranking model is constructed upon pre-trained BERT-based models and utilizes a sentence-pair classification architecture. In addition, as the primary objective of the second-phase ranking is to reduce errors in ranking NL expressions, we employ the listwise instead of pairwise paradigm [46,47] to enable the model to learn the features.

Specifically, sentence pairs are packed together into a single sequence, and the model encodes a sequence of vectors $\boldsymbol{x} = \{x_0, x_1, \ldots, x_{T_x}\}$ into a single vector c,

$$\begin{aligned} h_t &= \text{BERT}(x_t) \\ c &= q(\{h_0, \ldots, h_{T_x}\}) \end{aligned} \tag{7}$$

in which $q(\{h_0, \ldots, h_{T_x}\}) = h_0$, the hidden state corresponding to the `[CLS]` position. The vector c is then used to calculate the score s,

$$s = f(c) \tag{8}$$

where f is a linear dense layer. Then, the loss is computed,

$$loss = \sum_{i=1}^{n} L(S_i, Y_i) \tag{9}$$

where L denotes a listwise loss function that takes a list of sentence pairs (i.e., NL question-expression pairs) and their corresponding scores $S_i = \{s_0, s_1, \ldots, s_k\}$, Y_i represents the oracle relevant scores for those sentence pairs. A standard listwise function can be represented as follows:

$$L(S_i, Y_i) = -\frac{1}{k} \sum_{i=1}^{k} \left(\frac{e^{y_i}}{\sum_{j=1}^{k} e^{y_j}} log\left(\frac{e^{s_i}}{\sum_{j=1}^{k} e^{s_j}}\right)\right) \tag{10}$$

In this paper, we use the NeuralNDCG algorithm introduced in [48] to implement L.

Fine-tune Data Generation. The fine-tuned data of the re-ranking model is characterized by a collection of triples $\{(u_i, d_i, l_i)\}_{i=1}^{N}$. In each triple, u_i represents an NL question, d_i signifies an NL description of the corresponding SQL query, and l_i corresponds to binary scores, which indicates whether d_i is generated by the ground-truth SQL query of the NL question u_i or not.

Since we adopt a listwise approach for fine-tuning the model, we further organize the triples based on each NL question u_i. Consequently, we ultimately arrive at a collection of triples denoted as $\{(q_j, D_j, L_j)\}_{j=1}^{M}$. In this context, $D_j = \{d_{j1}, \ldots, d_{jn}\}$ represents the list of NL expressions associated with u_i, while $L_j = \{l_{j1}, l_{j2}, \ldots, l_{jn}\}$ comprises the corresponding boolean values corresponding to D_j.

4. Experiments

4.1. Experimental setup

We implement CKIF using the Pytorch framework and the AllenNLP library [49]. Our experiments were conducted on a system running Ubuntu 18.04, equipped with a 40-core 2.2 GHz CPU and 188 GB of RAM. All model training was performed utilizing the GeForce RTX 3090 GPU.

4.1.1. Dataset and evaluation criteria

We employ the SPIDER dataset [27] to assess the performance of CKIF. SPIDER stands as a comprehensive benchmark designed for tackling intricate cross-domain NL2SQL challenges. This dataset comprises 10,181 NL questions and encompasses 5693 unique complex SQL queries, covering 206 databases containing multiple tables within 138 domains. SPIDER categorizes SQL queries into four distinct difficulty levels: *Easy*, *Medium*, *Hard*, and *Extra Hard*. In our experiments, all models were trained using examples from the SPIDER training set and evaluated on the SPIDER validation set. It is noteworthy that SPIDER adheres to a zero-shot setting, ensuring that no database overlap exists between the training, validation (and test) sets.

Since SPIDER authors restrict access to the test set via an evaluation server, we conducted our experiments using the validation set, which comprises 1034 NL-SQL examples. Notably, the databases and schemas used in the validation set are distinct from those in the training set.

4.1.2. Evaluation metrics

The performance of the model is evaluated using two translation metrics: *translation accuracy* and *execution accuracy*. Additionally, two metrics, namely precision, and recall, are used to assess the quality of the low-confidence detection in CKIF.

Translation Accuracy. Given a top-1 generated query, if it matches the "gold" SQL syntactically, we deem the translation accurate. This metric serves as a minimum benchmark for translation accuracy, as a semantically correct query may exhibit syntactic differences from

Table 1
Overall translation results on the validation set of the Spider benchmark.

Translation model		Spider Validation Set			
		Exact Match (%)	Execution Match (%)	Precision (%)	Recall (%)
Bridge [15]	Base	68.7	68.0	–	–
	+ Ckif	70.5 (+1.8)	69.6 (+1.6)	**72.6**	54.5
Rat-sql [16]	Base	69.4	34.1	–	–
	+ Ckif	71.0 (+1.6)	35.2 (+1.1)	48.0	26.3
Rat-sql + Gap [18]	Base	71.8	34.9	–	–
	+ Ckif	74.0 (+2.2)	36.9 (+2.0)	66.0	**80.1**
Lgesql [19]	Base	75.1	36.3	–	–
	+ Ckif	**77.4** (+2.3)	**39.1** (+2.8)	65.9	42.0
Resdsql + T5-3b [20]	Base	76.0	79.4	–	–
	+ Ckif	**77.2** (+1.2)	**80.1** (+0.7)	63.2	51.3

the "gold" query. Note that this metric aligns with the *Exact Match Accuracy* metric introduced by Spider, which entails comparing sets for each query statement, with certain values excluded when calculating the accuracy of the two SQL queries.

Execution Accuracy. This metric assesses whether the execution result aligns with the ground truth query by executing the generated query against the underlying structured database. Similarly, it aligns closely with the *Execution Match Accuracy* metric introduced by Spider.

Precision. This metric measures the proportion of model outputs that are correctly identified as low-confidence out of all the outputs flagged as low-confidence. It indicates how accurate the detection is in Ckif.

Recall. This metric measures the proportion of actual low-confidence outputs that were correctly identified by the detection method. It reflects the model's ability to catch all true low-confidence outputs.

4.2. Implementation details

We present below the implementation details of Ckif used in our experiments.

Low-confidence Criteria. Due to variations in network designs among different models, it may not always be feasible to set a fixed threshold ω for all Seq2seq-based models. Therefore, we adopt a strategy where we limit low-confidence situations to no more than 20% of total queries and individually optimize the threshold ω for each Seq2seq model used in our experiments.

Retrieval Model. We initialize the model using the pre-trained *stsb-mpnet-base-v2*[6] sentence-transformer. We employ the Adam optimizer [50] with a learning rate of 2e-5, implementing a warm-up strategy for the initial 10% of total training steps. Additionally, we configure the batch size to 8.

Re-ranking Model. We initialize our model with the pre-trained RoBERTa [51] model. Throughout training, we also employ the Adam optimizer with a learning rate of 5e-6. To enhance training effectiveness, we implement a learning rate schedule that decreases the learning rate by half when training metrics plateau.

To better align with the listwise ranking paradigm, we organize the training triples based on the respective NL questions. More precisely, we set a threshold k to 50 and obtain a list of 50 NL expressions for each NL question. To accommodate GPU memory constraints, we limit the batch size to 2 for training. As a result, each training step incorporates a total of 100 NL question-expression pairs.

[6] https://huggingface.co/sentence-transformers/stsb-mpnet-base-v2

4.3. Baseline models

We use the following baselines for the experiments:

- Bridge [15] represents the schema and question as a tagged sequence, augmenting certain fields with cell values referenced in the question.
- Rat-sql [16] utilizes a relation-aware schema encoding approach, constructing the question-schema interaction graph based on n-gram patterns.
- Gap [18] is a pre-trained model that leverages Rat-sql as its foundation.
- Lgesql [19] is an NL2SQL model that utilizes line graph enhancements to capture underlying relational features without relying on metapath construction.
- Resdsql [20] represents the question and schema in a tagged sequence. Resdsql was implemented using three scales of the T5 model: Base, Large, and 3B. In our experiments, we use Resdsql with the 3B scale.

Remarks. We observed the *confidence calibration problem*[7] [52] in Lgesql. That is, Lgesql tends to exhibit a divergence between token accuracy and actual accuracy during training. This divergence becomes especially pronounced in TableQA, leading to inaccuracies in the internal token labeling positions and incorrect labeling of the correct queries when detecting instances of low confidence. Therefore, to align the model's accuracy with the ground-truth accuracy, we applied the *AvUc loss function* introduced in the computer vision field [53] (we made some improvements based on the SQL grammar) before applying Ckif on Lgesql.

4.4. Experimental results

Table 1 presents the overall translation accuracy of the models. It is evident that Ckif consistently improves the performance of all five models across both translation metrics. Remarkably, with the exception of the Resdsql model, none of the Seq2seq NL2SQL models explicitly manage specific values in SQL queries, often resulting in lower execution accuracy in comparison to their translation accuracy.

Notable achievements are witnessed (1) when employing Ckif with the Lgesql model, resulting in an impressive 2.3% improvement (from 75.1% to 77.4%) over the exact match metric, and (2) when applying to the Resdsql model, achieving an 80.1% execution match accuracy on the validation set of Spider.

On the other hand, the results under the precision and recall metrics illustrate the low-confidence detection outcomes of all baseline

[7] Calibration ensures the probability a model assigns to a prediction (i.e., confidence) matches the correctness of the prediction (i.e., accuracy).

Table 2
Translation accuracy (%) on the validation set of Spider by query difficulty levels.

Model	Easy	Medium	Hard	Extra Hard	Overall
Count	248	446	174	166	1034
Bridge	91.1	73.3	54.0	39.2	68.7
Bridge+Ckif	89.9(−1.2)	**76.0**(+2.7)	54.0	**44.0**(+0.8)	70.5
Ratsql	85.1	73.5	58.0	47.6	69.4
Ratsql+Ckif	**86.3**(+1.2)	**75.8**(+2.3)	**59.2**(+1.2)	47.6	71.0
Gap	91.5	74.2	64.4	44.2	71.8
Gap+Ckif	**91.9**(+0.4)	**76.7**(+2.5)	**66.1**(+1.7)	**48.2**(+4.0)	74.0
Lgesql	91.9	77.4	65.5	53.0	75.1
Lgesql+Ckif	**93.7**(+1.8)	**79.2**(+1.8)	**67.8**(+2.3)	49.1(−3.9)	76.7
Resdsql	94.0	85.7	65.5	55.4	76.0
Resdsql+Ckif	**94.4**(+0.4)	**88.0**(+2.7)	**67.5**(+2.0)	55.0(−0.4)	77.2

approaches augmented with Ckif on the validation set of Spider. While most baseline models with Ckif achieve strong precision and recall in the low-confidence detection process, Rat-sql performs poorly on the recall metric (26.3%). The reason is that we observed that Rat-sql tends to exhibit high "confidence" over its outputs, thereby leading to a relatively low rate of true low-confidence detection compared to other baseline models.

To delve deeper into the capabilities of Ckif, Table 2 presents a breakdown of the translation accuracy on the Spider benchmark. As can be seen, the performance of all translation models declines as the difficulty level increases. By applying Ckif, noteworthy enhancements are noted across all models in the categories of "Medium", "Hard", and "Extra Hard" queries. Regarding the variability noted in the "Easy" queries with Bridge, we have observed that Ckif occasionally prioritizes semantic-equivalent SQL queries, which presents challenges in accurately evaluating the translation accuracy metric. We posit that this instability can be mitigated if those semantic-equivalent query results are considered during the evaluation process.

4.5. Ablation study

We conduct an ablation study on Spider validation set to understand how various components of Ckif contributed to the outcomes. Our experiment involves comparing three different settings: (1) random low-confidence detection (and masking) for candidate query generation, namely without a low-confidence detection, (2) excluding the first-stage ranking model, and (3) the second-stage ranking model.

The results are presented in Table 3. The findings reveal that the generation process experiences a significant decline in performance when Ckif does not rely on low-confidence detection to recognize potential translation errors, which emphasizes the importance of confidence detection in our proposed methods. Additionally, the results also highlight the essential role of both ranking models, while compared to the first-stage ranking, the second-stage ranking model contributes much more significantly to Ckif, which emphasizes the importance of the fine-grain ranking.

4.6. Error analysis

To gain more insights into Ckif, we analyze the failed instances in relation to translation accuracy on the validation set of Spider benchmark for each step of Ckif.

In Table 4, the total count of NL questions with incorrect answers across the three stages (e.g., generalization, first-stage ranking process, and second-stage ranking process) is presented, highlighting the specific stages where errors occurred. We identify three primary causes for these failures.

- **Confidence Miscalibration Problem.** We identify the primary cause of failure in the top-1 category (i.e., generation miss count) stems primarily from inaccurate confidence detection that occurred in the initial phase of Ckif, which ultimately fails SQL generation. The root cause of this issue lies in the inherent nature of Seq2seq translation, where the token-level confidence estimation method used in Ckif may not accurately represent the true "confidence" of the model at specific decoding steps. For example, our observations indicate that instances of low confidence detected by Ckif tend to lag in occurrence, particularly when predicting complex SQL queries. To some extent, such failures can be mitigated by relaxing the generalization rules to encompass a wider range of potential SQL queries. Nevertheless, it may be crucial to explore a more effective and efficient method for confidence detection beyond the token-level approach.
- **Re-ranking Problem.** Another significant factor contributing to the failed cases is the second-stage ranking model's inability to discern the best-matching result. That is, the re-ranking model finds it challenging to identify the "gold" NL description among the top results, given that the sentences are largely similar but exhibit minor discrepancies in certain words. Consider the NL question, *"What is the name and capacity of the stadium with the most concerts after 2013?"* in Spider, the NL description of the corresponding "gold" SQL query is as follows:
  NL Description of Ground-truth SQL: *Find the name of the stadium, the capacity of the stadium about stadiums that had concerts. Return the top-1 answer only for the concert that its hosting year* ***is or after*** *2014 for each stadium in descending order of the number of concerts.*
  Top-ranked NL Description: *Find the name of the stadium, the capacity of the stadium about stadiums that had concerts. Return the answer only for the concert that its hosting year* ***is*** *2014 for each stadium in descending order of the number of concerts.*
  Such failure could potentially be mitigated if a method is discovered to avoid the re-ranking model from succumbing to the challenges posed by the confounding characteristics present in similar sentences.
- **NL Description Problem.** A minor fraction of failures can be attributed to the first-stage retrieval model, where Ckif struggles to capture the semantic relevance between a provided NL question and the corresponding NL description of the ground-truth query. This issue may become more pronounced when there is a notable difference in the length of the two sentences. We posit that such problems may be avoided by leveraging the machine learning approaches to learn a neural network model for summarizing lengthy sentences.

Based on the analysis above, we gain more understanding of the aspects related to Ckif and highlight potential improvements for future research.

5. Related work

Table Question Answering. TableQA has been a subject of investigation for many years within both the database management and NLP communities. Initial efforts relied on rule-based approaches, wherein manually crafted rules were used to translate NL questions into SQL queries tailored to individual databases. However, these early systems were susceptible to variations and paraphrasing in user queries due to their dependence on handcrafted rules.

The recent advancements in deep learning have spurred the development of new machine learning-based approaches for constructing TableQA systems [16,19,20,33,54–60]. Sqlnet [54] employs column attention and adopts a sketch-based approach to generate SQL queries by filling in slots. To augment the representation of database schema, [56] proposes to use a graph neural network to enhance the representation of the database schema. Rat-sql [16], on the other hand, utilizes a relational graph attention neural network to effectively manage predefined relations within databases. Irnet [33] introduces an intermediate representation that exhibits higher-level abstractions than SQL and

Table 3
Ablation study on Spider validation set. The "w/o Low-confidence Detection" denotes randomly selecting low-confidence outputs and masking for candidate generation.

Model	Generation Miss Count	Ranking Miss Count	Overall (%)
Base Model (Lgesql + Ckif)	174	60	77.4
w/o Low-confidence Detection	248	68	69.4 (−8.0)
w/o First-stage Ranking	174	72	76.2 (−1.2)
w/o Second-stage Ranking	174	353	49.0 (−28.4)

Table 4
Analysis of errors on the queries of Spider validation set.

Model	Generation Miss Count	First-stage Ranking Miss Count	Second-stage Ranking Miss Count
Bridge+Ckif	68	3	15
Rat-sql+Ckif	87	5	12
Gap+Ckif	114	9	44
Lgesql+Ckif	174	4	56

leverages custom type vectors to augment the representation of the NL questions and database schemas. Grappa [57] adopts the data augmentation technique and leverages pre-training approaches to bootstrap the learning process of the translation model. Bridge [15] integrates a Bert-based encoder with a sequential pointer-generator for end-to-end NL2SQL translation. The schema representation in Bridge is constructed as a tagged sequence appended to the NL question, with additional utilization of database content to enrich the sequence representations during the translation process. Smbop [17] revolutionizes the prevailing top-down decoding approach, adopting a reverse bottom-up parsing methodology to enhance the decoding process for Seq2seq models. In neural bottom-up parsing, the search procedure yields learned representations for the sub-trees within SQL queries, diverging from the top-down approach, where hidden states denote partial trees lacking precise semantics. Resdsql utilizes a ranking-enhanced encoder and skeleton-aware decoder to separate the schema linking and the skeleton parsing within the Seq2seq translation process. Given the remarkable performance of large language models (LLMs) across diverse NLP tasks, the second line of research has explored their utilization for TableQA [21,61–64]. [61,62] methodically assess the NL2SQL proficiency of current LLMs and highlight that while LLMs as NL2SQL models still have some gaps compared to fine-tuned translation models, they demonstrate robustness on new datasets. Recent research efforts [21,63] have focused on enhancing LLMs' prompts to improve query generation. Furthermore, a recent investigation [64] leverages the synergistic advantages of fine-tuned translation models and LLMs, aiming to support zero-shot TableQA using a hybrid approach.

Our work closely aligns with the first line of research, but unlike previous Seq2seq-based approaches that heavily rely on the naive sequential decoding paradigm, Ckif introduces a confidence-based method to detect potential errors during decoding and then employs a generate-then-rank approach to refine the decoding results.

Confidence Estimation. Confidence estimation has been a subject of investigation within several NLP tasks, including statistical machine translation [29,65,66] and semantic parsing [36]. A prevalent approach to modeling uncertainty in neural networks involves introducing distributions over the weights of network [67–69]. However, this often leads to models with a higher parameter count, necessitating corresponding adjustments in the training process, which can complicate their practical use. [70] establish a theoretical framework demonstrating that using dropout in neural networks can be viewed as an approximation of Bayesian inference in Gaussian processes.

In line with previous research, our work incorporates confidence estimation methods to identify potential low-confidence outputs of Seq2seq-based NL2SQL models, which effectively help improve their sequential decoding process, resulting in a performance gain.

6. Conclusion

This paper introduced Ckif, a practical confidence-based knowledge integration framework for cross-domain TableQA. Ckif was initiated by utilizing token-level confidence measurement to identify decoded tokens with low confidence, which are considered potential errors. Ckif then applied a three-stage generate-and-rank strategy for post-editing: Firstly, Ckif masked out the low-confidence predicted tokens from the output of an underlying Seq2seq model and generalized the masked-out output to capture all possible queries based on the underlying database schema. Secondly, Ckif employed a query translation model to translate the generalized queries into corresponding NL expressions. Finally, Ckif utilized a two-stage similarity ranking pipeline to learn relevant patterns between the NL expressions and the input NL question, enabling it to identify the nearest match and the query statement. We experimented on the public benchmark, Spider, with five existing Seq2seq-based translation models. The results demonstrated consistent enhancements in the performance of all baseline models when utilizing Ckif, demonstrating its effectiveness for better supporting TableQA systems.

CRediT authorship contribution statement

Yuankai Fan: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Tonghui Ren:** Software, Investigation, Data curation. **Can Huang:** Validation, Software, Investigation, Data curation. **Beini Zheng:** Writing – original draft. **Yinan Jing:** Supervision. **Zhenying He:** Supervision, Resources, Project administration. **Jinbao Li:** Supervision. **Jianxin Li:** Writing – review & editing, Supervision, Resources, Project administration.

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.

Acknowledgments

The authors would like to thank all the anonymous reviewers for their insightful comments and suggestions. This work was supported by the National Natural Science Foundation of China (Grant No. 62272106, 62072113, and 62172243).

Data availability

Data will be made available on request.

References

[1] Panupong Pasupat, Percy Liang, Compositional semantic parsing on semi-structured tables, in: Association for Computational Linguistics, ACL, 2015, pp. 1470–1480.

[2] Yunshi Lan, Gaole He, Jinhao Jiang, Jing Jiang, Wayne Xin Zhao, Ji-Rong Wen, A survey on complex knowledge base question answering: Methods, challenges and solutions, in: International Joint Conference on Artificial Intelligence, IJCAI, 2021, pp. 4483–4491.

[3] Haoyuan Chen, Fei Ye, Yuankai Fan, Zhenying He, Yinan Jing, Kai Zhang, X. Sean Wang, Staged query graph generation based on answer type for question answering over knowledge base, Knowl.-Based Syst. 253 (2022) 109576.

[4] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, Percy Liang, SQuAD: 100, 000+ questions for machine comprehension of text, in: Conference on Empirical Methods in Natural Language Processing, EMNLP, 2016, pp. 2383–2392.

[5] Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C. Lawrence Zitnick, Devi Parikh, VQA: visual question answering, in: International Conference on Computer Vision, ICCV, 2015, pp. 2425–2433.

[6] Jin Liu, GuoXiang Wang, Chongfeng Fan, Fengyu Zhou, Huijuan Xu, Question-conditioned debiasing with focal visual context fusion for visual question answering, Knowl.-Based Syst. 278 (2023) 110879.

[7] John M. Zelle, Raymond J. Mooney, Learning to parse database queries using inductive logic programming, in: AAAI, 1996, pp. 1050–1055.

[8] Luke S. Zettlemoyer, Michael Collins, Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars, in: UAI, 2005, pp. 658–666.

[9] Percy Liang, Michael I. Jordan, Dan Klein, Learning dependency-based compositional semantics, in: ACL, 2011, pp. 590–599.

[10] Jonathan Berant, Andrew Chou, Roy Frostig, Percy Liang, Semantic parsing on freebase from question-answer pairs, in: ACL, 2013, pp. 1533–1544.

[11] Yoav Artzi, Luke Zettlemoyer, Weakly supervised learning of semantic parsers for mapping instructions to actions, Trans. Assoc. Comput. Linguistics 1 (2013) 49–62.

[12] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, Yoshua Bengio, Learning phrase representations using RNN encoder-decoder for statistical machine translation, in: EMNLP, 2014, pp. 1724–1734.

[13] Ilya Sutskever, Oriol Vinyals, Quoc V. Le, Sequence to sequence learning with neural networks, in: NIPS, 2014, pp. 3104–3112.

[14] Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio, Neural machine translation by jointly learning to align and translate, in: ICLR, 2015.

[15] Xi Victoria Lin, Richard Socher, Caiming Xiong, Bridging textual and tabular data for cross-domain text-to-SQL semantic parsing, in: EMNLP, 2020, pp. 4870–4888.

[16] Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, Matthew Richardson, RAT-SQL: relation-aware schema encoding and linking for text-to-sql parsers, in: ACL, 2020, pp. 7567–7578.

[17] Ohad Rubin, Jonathan Berant, SmBoP: Semi-autoregressive bottom-up semantic parsing CoRRabs/2010.12412, 2020.

[18] Peng Shi, Patrick Ng, Zhiguo Wang, Henghui Zhu, Alexander Hanbo Li, Jun Wang, Cícero Nogueira dos Santos, Bing Xiang, Learning contextual representations for semantic parsing with generation-augmented pre-training, in: AAAI, 2021, pp. 13806–13814.

[19] Ruisheng Cao, Lu Chen, Zhi Chen, Yanbin Zhao, Su Zhu, Kai Yu, LGESQL: line graph enhanced text-to-SQL model with mixed local and non-local relations, in: ACL, 2021, pp. 2541–2555.

[20] Haoyang Li, Jing Zhang, Cuiping Li, Hong Chen, RESDSQL: decoupling schema linking and skeleton parsing for text-to-SQL, in: AAAI, 2023, pp. 13067–13075.

[21] Ruoxi Sun, Sercan Ö. Arik, Hootan Nakhost, Hanjun Dai, Rajarishi Sinha, Pengcheng Yin, Tomas Pfister, SQL-PaLM: Improved large language model adaptation for text-to-SQL CoRR abs/2306.00739, 2023.

[22] Xinyun Chen, Maxwell Lin, Nathanael Schärli, Denny Zhou, Teaching large language models to self-debug, in: ICLR, 2024.

[23] Tonghui Ren, Yuankai Fan, Zhenying He, Ren Huang, Jiaqi Dai, Can Huang, Yinan Jing, Kai Zhang, Yifan Yang, X. Sean Wang, PURPLE: Making a large language model a better SQL writer, in: ICDE, 2024, pp. 15–28.

[24] Yuankai Fan, Zhenying He, Tonghui Ren, Can Huang, Yinan Jing, Kai Zhang, X. Sean Wang, Metasql: A generate-then-rank framework for natural language to sql translation, in: ICDE, 2024, pp. 1765–1778.

[25] Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, Amin Saberi, CHESS: contextual harnessing for efficient SQL synthesis, CoRR abs/2405.16755, 2024.

[26] Jinyang Li, Binyuan Hui, Reynolds Cheng, Bowen Qin, Chenhao Ma, Nan Huo, Fei Huang, Wenyu Du, Luo Si, Yongbin Li, Graphix-T5: Mixing pre-trained transformers with graph-aware layers for text-to-SQL parsing, in: AAAI, 2023, pp. 13076–13084.

[27] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, Dragomir R. Radev, Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task, in: EMNLP, 2018, pp. 3911–3921.

[28] Simona Gandrabur, George F. Foster, Confidence estimation for translation prediction, in: HLT-NAACL, 2003, pp. 95–102.

[29] Nicola Ueffing, Hermann Ney, Word-level confidence estimation for machine translation, Comput. Linguist. 33 (1) (2007) 9–40.

[30] Jan Niehues, Ngoc-Quan Pham, Modeling confidence in sequence-to-sequence models, in: Kees van Deemter, Chenghua Lin, Hiroya Takamura (Eds.), INLG, 2019, pp. 575–583.

[31] Ilya Sutskever, Oriol Vinyals, Quoc V. Le, Sequence to sequence learning with neural networks, in: NeurIPS, 2014, pp. 3104–3112.

[32] Rui Zhang, Tao Yu, Heyang Er, Sungrok Shim, Eric Xue, Xi Victoria Lin, Tianze Shi, Caiming Xiong, Richard Socher, Dragomir R. Radev, Editing-based SQL query generation for cross-domain context-dependent questions, in: EMNLP, 2019, pp. 5337–5348.

[33] Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, Dongmei Zhang, Towards complex text-to-SQL in cross-domain database with intermediate representation, in: ACL, 2019, pp. 4524–4535.

[34] Yujian Gan, Xinyun Chen, Jinxia Xie, Matthew Purver, John R. Woodward, John H. Drake, Qiaofu Zhang, Natural SQL: making SQL easier to infer from natural language specifications, in: EMNLP, 2021, pp. 2030–2042.

[35] Dan Hendrycks, Kevin Gimpel, A baseline for detecting misclassified and out-of-distribution examples in neural networks, in: International Conference on Learning Representations, ICLR, 2017.

[36] Li Dong, Chris Quirk, Mirella Lapata, Confidence modeling for neural semantic parsing, in: ACL, 2018, pp. 743–753.

[37] Victor Zhong, Mike Lewis, Sida I. Wang, Luke Zettlemoyer, Grounded adaptation for zero-shot executable semantic parsing, in: EMNLP, 2020, pp. 6869–6882.

[38] Kun Xu, Lingfei Wu, Zhiguo Wang, Yansong Feng, Vadim Sheinin, SQL-to-text generation with graph-to-sequence model, in: EMNLP, 2018, pp. 931–936.

[39] Da Ma, Xingyu Chen, Ruisheng Cao, Zhi Chen, Lu Chen, Kai Yu, Relation-aware graph transformer for SQL-to-text generation, Appl. Sci. (2021).

[40] Georgia Koutrika, Alkis Simitsis, Yannis E. Ioannidis, Explaining structured queries in natural language, in: ICDE, 2010, pp. 333–344.

[41] Jinzhong Li, Huan Zeng, Lei Peng, Jingwen Zhu, Zhihong Liu, Learning to rank method combining multi-head self-attention with conditional generative adversarial nets, Array 15 (2022) 100205.

[42] Wael Etaiwi, Arafat Awajan, SemanticGraph2Vec: Semantic graph embedding for text representation, Array 17 (2023) 100276.

[43] Nils Reimers, Iryna Gurevych, Sentence-BERT: Sentence embeddings using siamese BERT-networks, in: EMNLP, 2019, pp. 3980–3990.

[44] Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova, BERT: pre-training of deep bidirectional transformers for language understanding, in: NAACL, 2019.

[45] Florian Schroff, Dmitry Kalenichenko, James Philbin, FaceNet: A unified embedding for face recognition and clustering, in: CVPR, 2015.

[46] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, Hang Li, Learning to rank: from pairwise approach to listwise approach, in: Zoubin Ghahramani (Ed.), ICML, 227, 2007, pp. 129–136.

[47] Razieh Rahimi, Ali Montazeralghaem, James Allan, Listwise neural ranking models, in: Yi Fang, Yi Zhang, James Allan, Krisztian Balog, Ben Carterette, Jiafeng Guo (Eds.), SIGIR, 2019, pp. 101–104.

[48] Przemyslaw Pobrotyn, Radoslaw Bialobrzeski, NeuralNDCG: Direct optimisation of a ranking metric via differentiable relaxation of sorting, CoRR abs/2102.07831, 2021.

[49] Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew E. Peters, Michael Schmitz, Luke Zettlemoyer, Allennlp: A deep semantic natural language processing platform, CoRR abs/1803.07640, 2018.

[50] Diederik P. Kingma, Jimmy Ba, Adam: A method for stochastic optimization, in: Yoshua Bengio, Yann LeCun (Eds.), ICLR, 2015.

[51] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, Veselin Stoyanov, RoBERTa: A robustly optimized BERT pretraining approach, CoRR abs/1907.11692, 2019.

[52] Shuo Wang, Zhaopeng Tu, Shuming Shi, Yang Liu, On the inference calibration of neural machine translation, in: ACL, 2020, pp. 3070–3079.

[53] Ranganath Krishnan, Omesh Tickoo, Improving model calibration with accuracy versus uncertainty optimization, in: NeurIPS, 2020.

[54] Xiaojun Xu, Chang Liu, Dawn Song, SQLNet: Generating structured queries from natural language without reinforcement learning, 2017, CoRR.

[55] Tao Yu, Zifan Li, Zilin Zhang, Rui Zhang, Dragomir R. Radev, TypeSQL: Knowledge-based type-aware neural text-to-SQL generation, in: North American Association for Computational Linguistics, NAACL, 2018.

[56] Ben Bogin, Jonathan Berant, Matt Gardner, Representing schema structure with graph neural networks for text-to-SQL parsing, in: Association for Computational Linguistics, ACL, 2019.

[57] Tao Yu, Chien-Sheng Wu, Xi Victoria Lin, Bailin Wang, Yi Chern Tan, Xinyi Yang, Dragomir R. Radev, Richard Socher, Caiming Xiong, Grappa: Grammar-augmented pre-training for table semantic parsing, in: International Conference on Learning Representations, ICLR, 2021.

[58] Torsten Scholak, Nathan Schucher, Dzmitry Bahdanau, PICARD: parsing incrementally for constrained auto-regressive decoding from language models, in: Empirical Methods in Natural Language Processing, EMNLP, 2021.

[59] Yuankai Fan, Zhenying He, Tonghui Ren, Dianjun Guo, Chen Lin, Ruisi Zhu, Guanduo Chen, Yinan Jing, Kai Zhang, X.Sean Wang, GAR: A generate-and-rank approach for natural language to sql translation, in: ICDE, 2023.
[60] Yuankai Fan, Tonghui Ren, Zhenying He, X.Sean Wang, Ye Zhang, Xingang Li, GenSQL: A generative natural language interface to database systems, in: ICDE, 2023.
[61] Nitarshan Rajkumar, Raymond Li, Dzmitry Bahdanau, Evaluating the text-to-SQL capabilities of large language models, CoRR abs/2204.00498, 2022.
[62] Aiwei Liu, Xuming Hu, Lijie Wen, Philip S. Yu, A comprehensive evaluation of ChatGPT's zero-shot text-to-sql capability, CoRR abs/2303.13547, 2023.
[63] Mohammadreza Pourreza, Davood Rafiei, DIN-SQL: decomposed in-context learning of text-to-SQL with self-correction, CoRR abs/2304.11015, 2023.
[64] Zihui Gu, Ju Fan, Nan Tang, Songyue Zhang, Yuxin Zhang, Zui Chen, Lei Cao, Guoliang Li, Sam Madden, Xiaoyong Du, Interleaving pre-trained language models and large language models for zero-shot NL2SQL generation, CoRR abs/2306.08891, 2023.
[65] John Blatz, Erin Fitzgerald, George F. Foster, Simona Gandrabur, Cyril Goutte, Alex Kulesza, Alberto Sanchís, Nicola Ueffing, Confidence estimation for machine translation, in: COLING, 2004.
[66] Radu Soricut, Abdessamad Echihabi, TrustRank: Inducing trust in automatic translations via ranking, in: ACL, 2010, pp. 612–621.
[67] John S. Denker, Yann LeCun, Transforming neural-net output levels to probability distributions, in: NeurIPS, 1990, pp. 853–859.
[68] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, Daan Wierstra, Weight uncertainty in neural networks, CoRR abs/1505.05424, 2015.
[69] Zhe Gan, Chunyuan Li, Changyou Chen, Yunchen Pu, Qinliang Su, Lawrence Carin, Scalable Bayesian learning of recurrent neural networks for language modeling, in: ACL, 2017, pp. 321–331.
[70] Yarin Gal, Zoubin Ghahramani, Dropout as a Bayesian approximation: Representing model uncertainty in deep learning, in: Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016, in: JMLR Workshop and Conference Proceedings, vol. 48, 2016, pp. 1050–1059.