Prompt Template-Driven Large Model SQL Generation

Qiurui Sun[1], Dong Yang[1*]
{qrsun@bnu.edu.cn, yd@bnu.edu.cn}
[1]Center of Information & Network Technology, Beijing Normal University, Beijing, China
* Corresponding author.

Abstract. When executing text-to-SQL tasks, current large generative models often fail to obtain correct query results due to semantic understanding bias, which requires extensive manually written prompts for correction. This paper proposes a unified prompt template framework integrating three specialized template types (variable-type, recurrent-type, multi-branch-type) with formalized definition and algorithmic logic. By populating these templates with structured database metadata through a rule-based matching mechanism, we achieve rapid, scalable prompt generation that eliminates heavy manual intervention. Experimental results on a custom event management database demonstrate that the proposed method reduces semantic understanding bias, achieving an overall SQL generation accuracy of 87.2% (close to the 90.0% accuracy of manual prompts) while reducing prompt engineering time to 25% of the manual method. The framework's innovation lies in its integration of programming-like logic (variable binding, loop iteration, conditional branching) into a unified prompt generation paradigm, distinguishing it from traditional string interpolation and rule-based prompt methods.

Keywords: large model, text-to-SQL, metadata, prompt template

1 Introduction

Large models have made significant progress in natural language understanding and content generation, but they struggle to accurately capture implicit field relationships and business rules when handling text-to-SQL tasks for relational databases. Text-to-SQL generation has become a key research direction for large model applications in structured data processing, with recent surveys highlighting prompt optimization and metadata utilization as critical success factors [1].

Existing text-to-SQL methods can be categorized into three main streams, each with inherent limitations:

Fixed Prompt Design Early methods like T5QL [2] rely on static prompt structures, lacking adaptability to diverse database schemas and business rules. SQL-Palm [3] optimizes prompt design for text-to-SQL tasks but suffers from poor generalization across different databases due to its schema-specific prompt engineering.

IIKI 2025, December 18-21, Chengdu, People's Republic of China

DOI 10.4108/eai.18-12-2025.2365274

Fine-Tuned Open-Source Models Models like DataGPT-SQL-7B [4] are specialized for text-to-SQL tasks but demand database-specific fine-tuning, resulting in high computational costs and poor scalability. Multi-agent frameworks [5] enhance SQL quality through collaboration but introduce complexity and latency.

Schema Linking & Structure-Guided Generation Approaches such as RSL-SQL [6] focus on robust schema linking to improve field association understanding, while structure-guided methods [7] clarify task logic through explicit schema representation. However, these methods fail to address implicit rule capture and require extensive manual intervention for new scenarios.

Notably, existing template-based prompt methods (e.g., basic string interpolation tools) are limited to simple variable replacement, lacking support for batch processing, conditional logic, and unified metadata integration. These methods cannot effectively handle complex business rules (e.g., multi-field association constraints) or adapt to dynamic database changes.

To address the inefficiencies of manual prompt engineering and the poor scalability of existing methods, this paper proposes a unified prompt template framework with the following key contributions:

1. **Technical Innovation**: Formalize a unified framework integrating variable-type, recurrent-type, and multi-branch-type templates, incorporating programming-like logic (variable binding, loop iteration, conditional branching) to enable dynamic, context-aware prompt generation—distinct from basic string interpolation and rule-based methods.
2. **Efficient Metadata Utilization**: Design a rule-based metadata matching mechanism that automatically maps database metadata (table/field attributes, business rules) to template variables, eliminating manual metadata-to-prompt mapping.
3. **Comprehensive Validation**: Validate the method on a custom database to demonstrate generality and effectiveness.
4. **Practical Scalability**: Reduce prompt engineering time to 25% of manual methods while maintaining 87.2% overall accuracy, with strong adaptability to new scenarios (77.5% accuracy on unseen rules).

2 Method

2.1 Formal Definitions

To establish technical rigor, we define core concepts formally:

2.1.1 Prompt Template (PT)

A prompt template is a structured string with embedded template elements (variables, loop directives, conditional directives) that can be instantiated with metadata to generate task-specific prompts. Formally:

$$PT = \langle E, R, M \rangle$$

Where:

- $E = \{e_1, e_2, ..., e_n\}$: A set of template elements, including variables e_v, loop directives e_l, and conditional directives e_c.
- $R = \{r_1, r_2, ..., r_m\}$: A set of instantiation rules defining how elements are populated with metadata (e.g., variable-type rules map e_v to metadata attributes; loop rules define iteration over metadata collections).
- M: A metadata schema specifying the required metadata attributes (e.g., field name, data type, constraints) for template instantiation.

2.1.2 Metadata Schema (MS)

A structured representation of database metadata, defined as:

$$MS = \langle T, F, B \rangle$$

Where:

- $T = \{t_1, t_2, ..., t_p\}$: Table-level metadata (table name, description).
- $F = \{f_1, f_2, ..., f_q\}$: Field-level metadata (field name, data type, nullability, format constraints).
- $B = \{b_1, b_2, ..., b_r\}$: Business rules (field associations, validation logic, e.g., $begin\_time \leq end\_time$).

Metadata schema typically includes the following categories of constraints:

1. Field type conversion, including knowledge such as syntax conversion rules corresponding to field data types, and comparison operation specifications.
2. Field format constraints, including knowledge of field format specifications and validation logic.
3. Field association constraints, including inherent constraint relationships between fields (such as temporal order, numerical magnitude, and state transition dependencies).

The advantage of metadata schema is that they can be automatically generated based on database metadata information. However, their disadvantage lies in the difficulty in accurately capturing the hidden rules between fields, thus requiring a large number of supplementary metadata prompts. For example, for the "datetime" type field "begin_time", the SQL statement generated by a large model based solely on metadata prompts directly compares the datetime field with the original literal, as shown below:

```
SELECT * FROM TABLE_A WHERE begin\_time > '2025-11-01'
```

To address the aforementioned issues, technical personnel typically need to manually write prompts, as demonstrated below:

If the field type is a date/datetime/timestamp type, and the target comparison value is a date string (such as "YYYY-MM-DD"), the datetime field needs to be converted to a string type before comparison.

Manual supplementary prompts are cumbersome and hard to adapt to new scenarios. Therefore, an optimized approach is required.

2.2 Proposed Method

The method presented in this paper comprises two key steps: 1. Convert the prompts in the traditional prompt library into prompt templates; 2. Utilize large language models to fulfill the prompt template to enrich metadata information and obtain supplementary prompts.

2.2.1 Template Instantiation Algorithm

The process of generating a final prompt from a template and metadata is defined as:

1. **Metadata Extraction**: Extract MS from the target database using SQL information schema queries and business rule annotations.
2. **Element Matching**: Map template elements E to MS using rule set R (e.g., $[FieldName] \rightarrow F.name$; $[TypeName] \rightarrow F.dataType$).
3. **Element Execution**: Resolve loop directives (iterate over metadata collections) and conditional directives (select rule branches based on metadata attributes).
4. **Prompt Assembly**: Concatenate resolved elements to form the final prompt.

The core innovation of the proposed framework lies in the integration of three specialized template types, each addressing distinct text-to-SQL challenges while adhering to the unified $PT = \langle E, R, M \rangle$ definition. Meta-prompt templates for generating LLM constraints are adopted to generate prompts [8]. This distinguishes the framework from basic string interpolation (which lacks rule sets and directive support) and fragmented template methods (which do not unify multiple logic types).

1. Variable-type Prompt Template By replacing variables, traditional prompts can be transformed into variable-type prompt templates. The purpose is to adapt prompts to schema-specific attributes (e.g., field names, data types) through dynamic variable binding.

Variables are denoted as $[V]$, where V is a metadata attribute (e.g., Field Name, Type Name). The instantiation rule R_v maps $[V]$ to the corresponding value in MS.

The algorithm logic is:

1. Define variables $\{[V_1], [V_2], ..., [V_k]\}$ corresponding to required metadata attributes.
2. Extract values $\{v_1, v_2, ..., v_k\}$ from MS where $v_i = MS[V_i]$.
3. Replace $[V_i]$ with v_i to generate the instantiated prompt.

Below is an exemplary variable-type prompt template:

> Convert datetime type [Field Name] to date string (e.g., YYYY-MM-DD), the field needs to be converted before comparison.

The prompts generated by filling variables into the above variable-type prompt template are as follows (replace $[FieldName]$ with begin_time and $[TypeName]$ with timestamp):

> Convert datetime type begin_time to date string (e.g., YYYY-MM-DD), the field needs to be converted before comparison.

2. Multi-branch Prompt Template When there is "branch logic" (i.e., different conditions correspond to different processing rules) in business rules, embedding conditional judgment statements in prompts allows the large model to automatically select the appropriate rules based on the actual scenario, thereby enhancing the scenario coverage ability of the prompts. The purpose is to handle conditional business rules (e.g., intent recognition, scenario-specific logic) by enabling dynamic rule selection.

A conditional directive is denoted as $IF(C_i)THEN(A_i)$, where C_i is a condition (e.g., presence of keywords) and A_i is the action (a sub-prompt). The instantiation rule R_c evaluates C_i against the input query and metadata, then selects the corresponding A_i.

The algorithm logic is:

1. Define branches $\{(C_1,A_1),(C_2,A_2),...,(C_n,A_n)\}$, where C_i is a boolean condition (e.g., query contains keywords in [Keyword List 1]) and A_i is the associated prompt fragment.
2. Extract context (e.g., query keywords, metadata attributes) to evaluate each C_i.
3. Select the first C_i that evaluates to True and instantiate its corresponding A_i.
4. Concatenate selected A_i fragments to generate the final prompt.

Intent recognition is a common multi-branch prompt template. For example, when it comes to generating SQL statements for large models, querying and statistics are two different types of intents. Below is an exemplary template for intent recognition prompts:

> IF (the query includes [Keyword List 1]), THEN (it is determined as [Intent 1])

The prompts generated after filling in the variables for the aforementioned multi-branch prompt templates are as follows:

> If the query includes "query, find, filter, view", it is determined as a query intent.

3. Recurrent Prompt Template When there are multiple fields of the same type (such as multiple enumeration fields) or repetitive business rules (such as mapping relationships for batch fields), prompt fragments can be generated in batches through looping logic to avoid repetitive writing and improve optimization efficiency. The purpose is to batch-generate prompts for repetitive elements (e.g., multiple fields of the same type, batch business rules) to avoid redundant writing.

A loop directive is denoted as $LOOP(C,B)$, where C is a metadata collection (e.g., all enum fields) and B is the loop body (a sub-template). The instantiation rule R_l iterates over C, instantiates B for each element, and concatenates results.

The algorithm logic is:

1. Define the loop collection $C = \{c_1, c_2, ..., c_n\}$ (e.g., all date-type fields in F).
2. Define the loop body B with variables mapped to attributes of c_i.
3. For each $c_i \in C$:
   (a) Extract attributes $\{v_{i1}, v_{i2}, ..., v_{ik}\}$ from c_i.
   (b) Instantiate B with $\{v_{i1}, ..., v_{ik}\}$ to form a prompt fragment.
4. Concatenate all fragments to generate the final prompt.

For the variable prompt template in the above example, $[FieldName]$ can be replaced with ", end_time", which means two fields are filled in a loop. The following prompt templates can be generated:

> IF (LOOP(, end_time) are of datetime or other date-time types, and the target comparison value is a date string (such as YYYY-MM-DD)), THEN (it is necessary to convert the fields to string types before comparing).

For another example, in the activity data table, the "person" field stores the names of the activity initiators (e.g., John Smith, Emily Davis, Alexander Wilson). However, users sometimes query by abbreviated names (e.g., "Dr. Smith" for John Smith). Below is an exemplary template for looping prompts:

> LOOP(IF (there is a title like [Abbreviated Name] in the question), THEN (check whether the field [Name Field] corresponds to [Actual Name])).

The prompts generated by cycling through variable filling are as follows:

> If there is a title like "Dr. Smith" in the question, check whether the "person" field corresponds to "John Smith".
>
> If there is a title like "Ms. Davis" in the question, check whether the "person" field corresponds to Emily Davis.
>
> If there is a title like "Alex" in the question, check whether the "person" field corresponds to "Alexander Wilson".

2.3 Metadata Matching Mechanism

A key component of the framework is the rule-based metadata matching mechanism that automates the mapping between template elements and database metadata. The mechanism is defined by:

1. **Matching Rules**: For each template element type, pre-defined rules map elements to metadata attributes (e.g., $[FieldName] \rightarrow F.name$; loop collection $C =$ all fields where $F.dataType \in \{date, datetime, timestamp\}$).
2. **Conflict Resolution**: If multiple metadata attributes match a template element, the mechanism prioritizes attributes based on a schema hierarchy (table-level ¿ field-level ¿ business rule-level).
3. **LLM-Assisted Enhancement**: For unstructured business rules (e.g., abbreviated name mappings), the LLM is used to extract structured key-value pairs (e.g., $AbbreviatedName = Dr.Smith \rightarrow ActualName = JohnSmith$) and populate template variables.

3 Experiment and Result Analysis

3.1 Comparative Schemes

To verify the effectiveness of the prompt template method proposed in this paper, three comparative schemes were designed. All schemes were based on the same large test model and experimental data, with the only difference lying in the prompt supplementation method:

- **Scheme 1 (Baseline Method)**: Without adding any prompts, only natural language query requirements and basic database metadata (table names, field names, data types) are input.
- **Scheme 2 (Manual Prompts Method)**: Two technical personnel with over 5 years of database development experience manually wrote supplementary prompts for each query requirement, covering field type conversion, format constraints, and association rules.
- **Scheme 3 (Our Proposed Method)**: Generate supplementary prompts by populating database metadata into variable-type, recurrent-type, and multi-branch prompt templates.

3.2 Evaluation Metrics

We designed evaluation metrics from two core dimensions (accuracy and efficiency) to comprehensively measure the performance of the three schemes:

1. **SQL Generation Accuracy**: Proportion of SQL statements that (1) match the query intent, (2) have no grammatical errors, and (3) return correct results. Calculation: Number of correct SQL statements/Total number 100%.
2. **Prompt Engineering Time**: Total time from metadata extraction to prompt completion (average over 5 runs).

3. **Scenario Adaptability**: Accuracy on new scenarios (custom database) and unseen databases (Spider subset).

3.3 Experimental Datasets and Models

The experimental data was a customized event management database, containing 5 core business tables (event information table, participant table, registration record table, venue resource table, expense statistics table), 42 fields (covering 8 data types: datetime, varchar, int, enum, etc.), and 16 business rules (e.g., $begin\_time \leq end\_time$, 11-digit mobile phone number format verification). To simulate real query scenarios, 100 natural language query requirements were designed: 60 for conventional scenarios (covering known field rules) and 40 for new scenarios (including queries with newly added fields and undefined rules).

To validate the contribution of each template type, we designed three ablation variants:

- Variant 1: Only variable-type templates.
- Variant 2: Variable-type + recurrent-type templates.
- Variant 3: Variable-type + multi-branch-type templates.
- Full Framework: Variable + recurrent + multi-branch.

Two large models were used to verify model-agnosticism: Deepseek R1 32B (primary test model), Qwen3 30B.

3.4 Experimental Results

Table 1 shows the accuracy results on the custom event management database. The overall accuracy of our proposed method (87.2%) is significantly higher than the baseline method (48.4%) and only slightly lower than the manual prompts method (90.0%). In conventional scenarios, our method's performance is close to the manual prompts method (difference of 1.7 percentage points), indicating that template-generated prompts effectively cover known business rules. In new scenarios, our method is 5 percentage points lower than the manual prompts method but still much higher than the baseline, verifying its adaptability to unseen scenarios.

Table 1: SQL Generation Accuracy (%)

Scheme	Regular scenes (60 entries)	New scenes (40 entries)	Overall accuracy
Baseline Method	58.3	32.5	48.4
Manual Prompts Method	95.0	82.5	90.0
Our Proposed Method	93.3	77.5	87.2

Table 2 compares prompt engineering time across schemes. Our proposed method requires only 1 hour, which is 25% of the manual method (4 hours) and slightly longer than SQL-Palm (45

minutes) and DIN-SQL (50 minutes). However, the trade-off between time and accuracy makes our method superior: it achieves 10.4–13.2 percentage points higher accuracy than SQL-Palm/DIN-SQL with minimal additional time.

Table 2: Prompt Engineering Time

Scheme	Prompt Engineering Time	Degree of Human Involvement
Baseline Method	1.8 seconds	Fully automatic
Manual Prompts Method	4 hours	Fully manual
Our Proposed Method	1 hour	Human-assisted (metadata annotation)

Table 3 shows the ablation study results. The full framework (all three template types) achieves the highest accuracy, demonstrating the complementary value of each template:

- Variable-type templates alone achieve 75.3% accuracy, providing basic schema adaptation.
- Adding recurrent-type templates increases accuracy by 8.1 percentage points (to 83.4%), confirming the value of batch processing for repetitive rules.
- Adding multi-branch-type templates increases accuracy by 11.9 percentage points (to 87.2%), highlighting the importance of conditional logic for intent recognition and scenario-specific rules.

Table 3: Ablation Study Results (%)

Variant	Regular scenes (60 entries)	New scenes (40 entries)	Overall accuracy
Variant 1 (Variable-only)	79.2	65.8	75.3
Variant 2 (Variable + Recurrent)	88.5	73.2	83.4
Variant 3 (Variable + Multi-branch)	92.1	76.8	86.7
Full Framework	93.3	77.5	87.2

We categorize incorrect SQL statements into three types:

1. **Schema Misalignment**: Errors in field/table name mapping. Our method reduces this error by 68% compared to the baseline, as template variables enforce strict metadata matching.
2. **Rule Violation**: Failure to apply business rules (e.g., date type conversion). Our method reduces this error by 72% compared to the baseline, due to recurrent and multi-branch templates covering implicit rules.
3. **Intent Misunderstanding**: Incorrectly identifying query intent (e.g., confusing query vs. statistics). Our multi-branch templates reduce this error by 81% compared to the baseline.

The main source of error in our method (vs. manual prompts) is overly general prompts for edge-case business rules (e.g., rare format validation logic), which manual prompts can tailor to specific queries.

Our method fills metadata through prompt templates, accurately capturing hidden information such as field type conversion, format constraints, and association rules, thus effectively reducing semantic understanding biases in large models. The slight gap compared to the manual group stems from the fact that manual prompts can be refined for specific queries, while template-generated prompts are more general.

4 Conclusion

This paper addresses the issue of semantic understanding bias in large models when generating database query statements, proposing a prompt generation method based on prompt templates. This method converts traditional prompt library prompts into three types of templates (variable, recurrent, multi-branch) and quickly generates a large number of prompts by populating database metadata, eliminating the need for extensive manual intervention. Experimental results demonstrate that our proposed method effectively reduces semantic understanding bias in large models, achieving an overall SQL generation accuracy of 87.2% (close to the 90.0% accuracy of manual prompts). Additionally, prompt generation time is only 25% of the manual method, and the adaptability to new scenarios reaches 77.5%, significantly outperforming traditional baseline methods. This approach not only solves the cumbersome and inefficient issues of manual prompt supplementation but also enhances the accuracy and adaptability of SQL statements generated by large models, providing an efficient and feasible solution for the application of large models in database querying.

Acknowledgments

This work was supported by the Application of Trusted Education Digital Identity in the Construction of Smart Campus in Vocational Colleges (Grant No. 2242000393).

Declaration on Generative AI

During the preparation of this work, the authors used large language models (e.g., Qwen) for grammar and spelling checks. After using these tools, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] Zhu X, Li Q, Cui L, et al. Large language model enhanced text-to-sql generation: A survey. arXiv preprint arXiv:241006011. 2024. Available from: https://arxiv.org/abs/2410.06011.

[2] Arcadinho SD, Aparício D, Veiga H, et al. T5QL: Taming language models for SQL generation. In: Proceedings of the 2nd Workshop on Natural Language Generation, Evaluation, and Metrics (GEM); 2022. p. 276-86.

[3] Sun R, Özge Arik S, Muzio A, et al. Sql-palm: Improved large language model adaptation for text-to-sql (extended). arXiv preprint arXiv:230600739. 2023. Available from: https://arxiv.org/abs/2306.00739.

[4] Wu L, Li P, Lou J, et al. Datagpt-sql-7b: An open-source language model for text-to-sql. arXiv preprint arXiv:240915985. 2024. Available from: https://arxiv.org/abs/2409.15985.

[5] Li J, Wu T, Mao Y, et al. SQL-Factory: A Multi-Agent Framework for High-Quality and Large-Scale SQL Generation. arXiv preprint arXiv:250414837. 2025. Available from: https://arxiv.org/abs/2504.14837.

[6] Cao Z, Zheng Y, Fan Z, et al. Rsl-sql: Robust schema linking in text-to-sql generation. arXiv preprint arXiv:241100073. 2024. Available from: https://arxiv.org/abs/2411.00073.

[7] Zhang Q, Chen H, Dong J, et al. Structure guided large language model for sql generation. arXiv preprint arXiv:240213284. 2024. Available from: https://arxiv.org/abs/2402.13284.

[8] Yang D, Sun Q. Generation, sharing and reuse of large model prompts. Journal of Beijing Union University. 2025;39(4):1-6.