Composable Prompt Modularization in Text-to-SQL Agents: A Practical Approach

Daniele Carnaúba Gonçalves[1], Bárbara S. Neves[1], Paulina Irene Velasquez Ferrufino[1], Davi Uchoa Cavalcante Costa[3], Amanda Venceslau[2], Wellington Franco[1], José Antônio Macêdo[1], Raimir Holanda[3]

[1]Computer Science Department, Federal University of Ceara,
Av. Mister Hull, s/n - Pici, Fortaleza, 60455-760, Ceará, Brasil.

[2]Virtual University Institute, Federal University of Ceará,
Av. Mister Hull, s/n - Pici, Fortaleza, 60455-760, Ceará, Brasil.

[3]Computer Science Department, University of Fortaleza,
Avenida Washington Soares, 1321, Fortaleza, 60811-905, Ceará, Brasil.

carnaubadani@alu.ufc.br, barbara.steph@alu.ufc.br,

paulinavelasquez@alu.ufc.br, daviuch02@gmail.com,

amanda@virtual.ufc.br, wellington@crateus.ufc.br

jose.macedo@insightlab.ufc.br, raimir@unifor.br

Abstract. *Text-to-SQL agents often rely on monolithic prompts that combine instructions and examples into a single block, making them difficult to maintain, less interpretable, and more prone to structural errors. To address these limitations, this work proposes a Composable Prompt Modularization approach that decomposes prompts into semantic components for question classification and Text-to-SQL generation, leveraging large language models within a modular prompting pipeline. We conduct a case study involving an intelligent assistant for the public sector. The results show that the modular architecture is a competitive alternative to monolithic prompting, achieving up to 95.39% execution accuracy and strong structural alignment, with average component-matching scores of 0.954. The findings reveal a trade-off between prompting strategies: while the monolithic prompt is slightly more efficient when fewer examples are available (Top-k = 3), the modular approach makes better use of additional context, improving execution accuracy by 0.66 percentage points at Top-k = 5 (94.74% vs. 94.08%). In the context of public transparency, where the availability of information enables citizens to interpret, validate, and audit data, the modular approach appears particularly well-suited.*

1. Introduction

The advance of Large Language Models (LLMs) has driven the application of natural language translation to structured queries, *Text-to-SQL*, offering new avenues for direct integration between language processing and databases. However, the effectiveness of these models heavily depends on prompt design, and single-block (monolithic) approaches have limitations in stability, maintainability, and adaptability across different

contexts [Schnabel and Neville 2024, Zhuo et al. 2024]. This observation has led to the development of more structured methods, such as *template-based prompt engineering* [Mao et al. 2025] and modular frameworks like LangGPT [Wang et al. 2024], which apply principles of composition and reuse to improve prompt organization and consistency.

Despite these advances, applying prompt modularization to optimize models for SQL generation tasks remains underexplored, especially in settings that require a clear separation between stages such as classification and query generation. For example, [Tan et al. 2024] introduces specialized prompts for schema linking and clause-wise SQL generation, while [Petrola et al. 2025] proposes a heuristic-guided LLM framework for SQL generation in real-world relational databases. These efforts highlight the importance of structured prompt design but also raise the question of how to build composable prompt architectures that balance accuracy and interpretability while maintaining maintainability in *Text-to-SQL* agents.

As a case study, we propose *Composable Prompt Modularization*, which decomposes prompts into independent semantic modules for classification and SQL generation, enabling targeted adjustments. The approach is evaluated using Component Matching (CM) and Execution Accuracy (EA) in a public-sector external control institution, where non-technical users query complex transparency data on public works and personnel. This setting involves heterogeneous and evolving systems, complex relational schemas, administrative and legal terminology, data-versioning rules, and indicators dependent on institutional and accounting contexts. Moreover, because transparency databases may include personal, confidential, or access-controlled information, such systems require traceability, auditable query generation, controlled access, and identification of official data sources. Although motivated by these requirements, this study does not fully address the associated privacy, legal, access-control, and data-governance challenges.

The experimental results show that the modular approach achieves up to 95.39% execution accuracy and average component matching scores of up to 0.954, while also revealing a structured trade-off with monolithic prompting depending on the amount of retrieved context. In particular, the modular strategy improves execution accuracy by 0.66 percentage points at Top-k = 5 compared to the monolithic baseline, whereas the monolithic approach remains slightly more effective in low-context scenarios (Top-k = 3). These findings suggest that modular prompt design is especially effective in settings where richer contextual information is available.

The main contributions of this work are: (i) a modular architecture for composable prompt design in *Text-to-SQL* tasks; (ii) an empirical evaluation on a real-world public-sector dataset; and (iii) a comparative analysis with monolithic prompting that highlights trade-offs in accuracy and contextual efficiency.

The remainder of this article is organized as follows. Section 2 reviews related work on prompt engineering, Text-to-SQL, and LLM-based agent architectures. Section 3 presents the proposed modularization approach and evaluation metrics. Section 4 describes the institutional case study, experimental protocol, and results. Section 5 discusses the findings, limitations, and directions for future work.

2. Related Work

Prompt engineering has evolved from manually assembled instructions toward reusable and structured formats. LLMs are sensitive to variations in prompt components, motivating more systematic design practices [Zhuo et al. 2024]. Studies of real-world applications identify recurring elements such as instructions, examples, constraints, and output specifications [Mao et al. 2025], while LangGPT organizes prompts into declarative semantic modules [Wang et al. 2024]. Promptware engineering further emphasizes versioning, traceability, reuse, and controlled modification [Chen et al. 2025].

Recent surveys organize Text-to-SQL research according to schema representation, query generation, prompting, fine-tuning, benchmarks, and evaluation methods [Shi et al. 2025, Liu et al. 2026]. Prompt-based approaches increasingly decompose generation into intermediate stages. DIN-SQL, for example, separates schema linking, query classification and decomposition, SQL generation, and self-correction [Pourreza and Rafiei 2023].

Agent-based systems extend this decomposition through specialized components. MAC-SQL uses selector, decomposer, and refiner agents to reduce schema context, generate queries, and correct errors [Wang et al. 2025]. [Santos et al. 2026] combine multiple agents, retrieval-augmented generation, and metadata to construct question-specific contexts. Recent methods also investigate vector-based schema retrieval, execution-guided correction, and bidirectional table and column selection [Piao et al. 2026, Nahid et al. 2026].

The present work combines structured prompt engineering with agent-based Text-to-SQL at a different design level. It focuses on the composable organization of prompt headers, instructions, retrieved examples, schema descriptions, and output constraints, as well as the separation between institutional-domain classification and SQL generation. Unlike methods that introduce explicit schema-linking or execution-guided refinement algorithms, the proposed architecture compares modular and monolithic prompt formulations within the same institutional pipeline.

3. Composable Prompt Modularization

The proposed methodology adopts the concept of *Composable Prompt Modularization* as a systematic, reusable approach for designing agents based on Large Language Models (LLMs) for the task of translating natural language into SQL (*Text-to-SQL*).

The methodology organizes the agent's pipeline into two main layers: (i) intent classification and (ii) SQL query generation. Figure 1 illustrates the operational flow of the proposed API, from the initial user message to the activation of the corresponding SQL generation path.

As shown in Figure 1, the user message is first converted into embeddings and then classified into institutional domains such as people, public works, or others. This decision activates the corresponding SQL module, which translates the intent into an executable query on the institutional database [Gao et al. 2024, Lan et al. 2023]. After intent classification, the SQL-generation prompt is enriched with a domain-specific schema context. Based on the identified domain, the relevant schema descriptions are retrieved from application-level constants and dynamically inserted into the prompt, reducing ambiguity

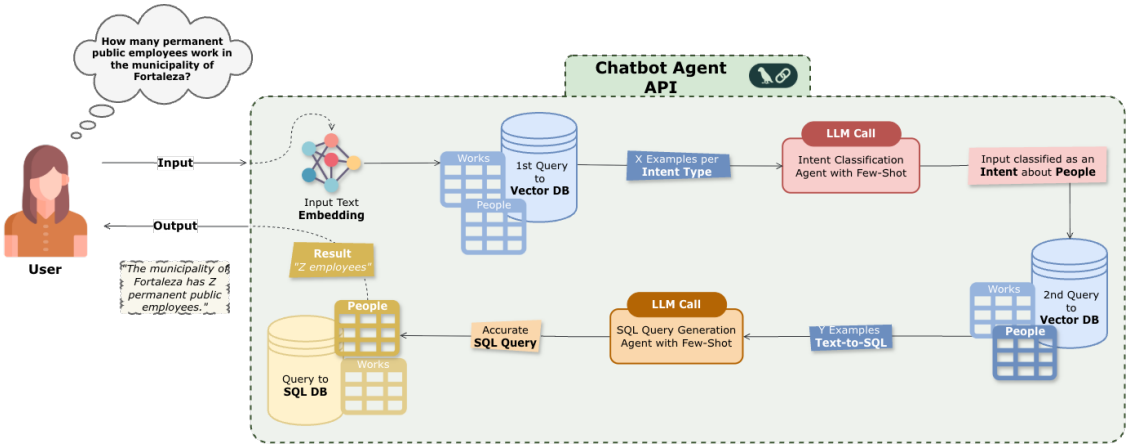

Figura 1. General flow of the Text-to-SQL agent API with modular classification and query generation.

and promoting consistent and controlled query generation [Luo et al. 2024].

Unlike monolithic prompts that condense extensive instructions into a single block, modularization allows prompts to be decomposed into independent semantic units, thereby promoting stability, traceability, and scalability in real-world applications [Wang et al. 2024, Mao et al. 2025, Chen et al. 2025]. Figure 2 presents the conceptual structure of the *prompt* modules and the functional role assigned to each component.

Each module has a distinct responsibility: the **System Prompt** defines the assistant's institutional role and domain boundaries; the **Prompt Header** establishes exclusive thematic categories and output restrictions; the **Prompt Instruction** provides contextualized few-shot examples; and the **Prompt Footer** standardizes the output format for seamless integration with subsequent modules. The **SQL Prompt** sequence (Header, Instruction, Schema, and Footer) extends this process to the generation of structured SQL queries, ensuring semantic fidelity and syntactic adherence to the relational schema [Zhuo et al. 2024, Schnabel and Neville 2024, Liu et al. 2025].

The user's message is processed by the classification prompt modules, denoted by the prefix **CLF** (*classification*): **CLF_PROMPT_HEADER**, **CLF_PROMPT_INSTRUCTION**, and **CLF_PROMPT_FOOTER**. After classification, the system selects the corresponding **SQL_PROMPT** module set for query generation. After categorization, the system selects the appropriate **SQL_PROMPT** module set, which generates the final query. If access to the information system (SIM) is unavailable, the agent applies a standard fallback response defined in the System Prompt. This modular design ensures component reuse and facilitates maintainability [Shinn et al. 2023].

Each module is treated as an autonomous yet interoperable functional unit. The use of *runtime injection* enables the substitution of examples and database schemas without altering the main prompt structure. This compositional principle promotes adaptability, version control, and traceability, core aspects of *promptware* engineering [Liang et al. 2025, Chen et al. 2025]. Consequently, the proposed approach allows new domains to be integrated without requiring the entire logical workflow to be rewritten.

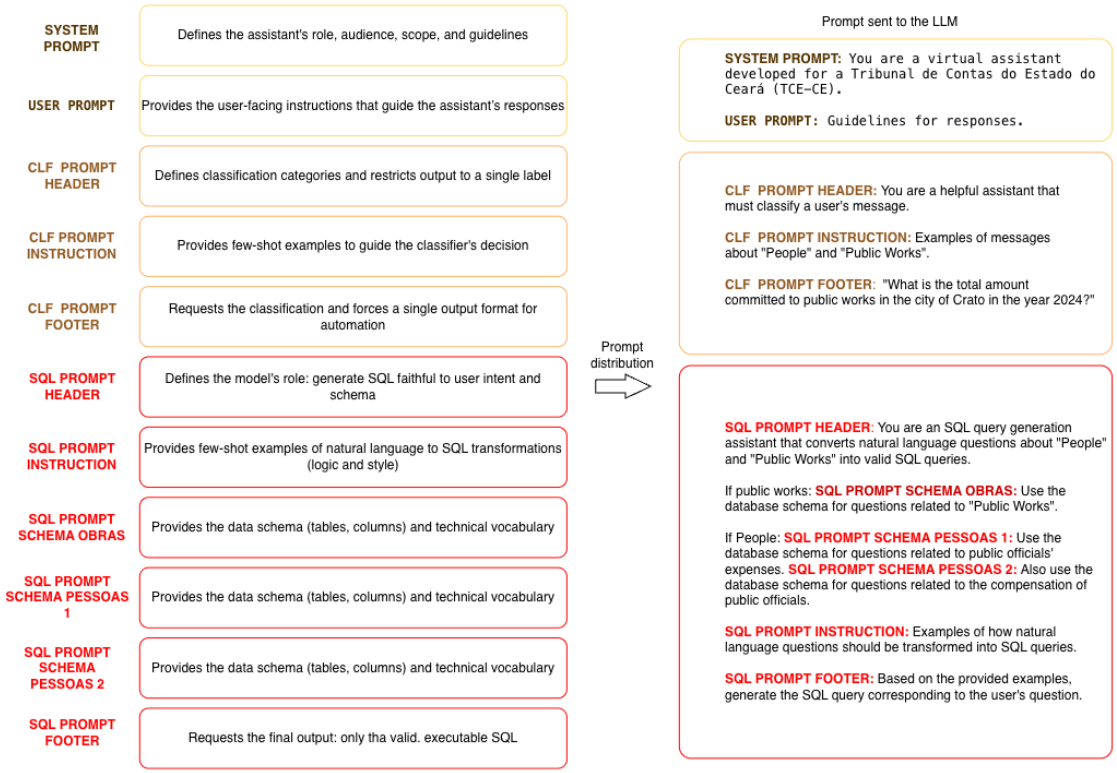

Figura 2. Structure of composable prompt modules for classification and SQL generation.

3.1. Evaluation Metrics

To assess the performance of the *Text-to-SQL* agents, two complementary metrics are employed: **Component Matching (CM)** and **Execution Accuracy (EA)**. The CM metric evaluates the structural correspondence between the generated SQL query and the reference query, verifying the alignment of key syntactic blocks such as `SELECT`, `WHERE`, `GROUP BY`, and `ORDER BY`. This process is based on the *Partial Component Match F1 (PCM-F1)* metric proposed by [Yavuz et al. 2021], which computes partial overlap between predicted and reference components, reflecting both syntactic and semantic adherence. As discussed in prior Text-to-SQL evaluations [Lan et al. 2023], structural matching provides fine-grained insight into query composition, complementing execution-based metrics. Thus, CM helps identify discrepancies that may not affect execution outcomes but still affect structural correctness [Gao et al. 2024]. The EA metric, in turn, evaluates whether the generated query produces the correct execution result when compared with the reference answer. Widely adopted in Text-to-SQL benchmarks [Gao et al. 2024], it captures end-to-end task performance and complements the structural analysis provided by CM. EA is defined in Equation 1.

$$EA = \frac{\text{Number of correct answers}}{\text{Total number of executed tests}} \times 100\% \quad (1)$$

Table 1 summarizes both metrics and their complementary roles in evaluating query generation quality.

Tabela 1. Summary of Evaluation Metrics for Text-to-SQL Tasks

Metric	Description	Key References	Remarks
Component Matching (CM)	Compares generated and reference SQL queries at the component level using the *Partial Component Match F1 (PCM-F1)* variant to assess partial overlap among clauses such as `SELECT`, `WHERE`, and `GROUP BY`.	[Yavuz et al. 2021], [Lan et al. 2023]	Provides interpretable, fine-grained evidence of structural correctness.
Execution Accuracy (EA)	It compares the obtained database outputs with the expected results, defined as the proportion of queries executed correctly out of the total number of tests.	[Gao et al. 2024]	Captures end-to-end correctness, but may overlook structural differences.

4. Case Study

The approach was applied within a public-sector external control institution to improve public access to information on public works and personnel data. The modular agent was integrated into the institution's transparency portal, allowing citizens to query complex datasets in natural language without requiring SQL knowledge. The database used was refined to preserve confidentiality and semantic consistency by removing sensitive and ambiguous fields. This implementation validates the potential of modularization for governmental domains with strict auditability and quality control requirements [Wang et al. 2024, Mao et al. 2025, Luo et al. 2024].

4.1. Data Preparation and Constraints

To enable database use with a large language model, the database was restricted to views containing public works and personnel data from the institution's information system. Columns with confidential information or misleading names were removed, since these inconsistencies often caused the model to generate incorrect queries.

The evaluation dataset comprises 152 natural language questions paired with reference SQL queries: 51 on Public Works, 30 on public-agent expenses, and 71 on public-agent remuneration. The schema context provided to the SQL-generation agents includes three views and 103 listed attributes. This dataset is distinct from the contextual demonstration repository, which contains 59 candidate examples used by the retrieval component.

4.2. Application Scenario

The case study covers two transparency domains: *Public Works* and *Personnel*. The former includes contracts, execution, payments, and project progress, while the latter addresses public servants, compensation, benefits, and related expenditures. These domains were selected for their public relevance and involve structured data that is often difficult for non-technical users to explore directly.

Users submit natural language questions that are translated into SQL and answered using official institutional records. For example, the question "What was the total amount committed to public agent expenses in Crateús in 2024?" is represented by the following query:

```
SELECT SUM(vl_empenhado_ne) AS total_empenhado
FROM temp.vw_gastos_agentes_publicos
WHERE nm_municipio = 'CRATEUS'
      AND dt_ref_ne BETWEEN 202401 AND 202412;
```

This example illustrates aggregation and temporal and municipal filtering. It represents the application workflow but not the complete range of operations in the 152-query evaluation set.

4.3. System Architecture in the Deployed Institutional Setting

The deployed solution follows a modular multi-agent architecture centered on a pipeline orchestrator. The chatbot API serves as the main entry point, receiving user requests and forwarding them to the orchestration layer. The pipeline then coordinates a sequence of specialized components responsible for classification, SQL generation, structured retrieval, optional semantic recovery, and final answer composition.

A first specialized agent performs intent classification. Its role is to determine whether the question belongs to one of the supported institutional domains, namely *Public Works* or *Personnel*. If the query does not match either category, the system does not attempt unrestricted access to the database; instead, it produces a fallback response that guides the user and preserves operational safety. This design reduces unnecessary executions and prevents the model from generating unsupported answers outside the intended scope.

For in-domain requests, a second agent is responsible for generating SQL queries based on the user's question and the detected intent. This agent receives both the natural-language input and the constrained schema context, generating a structured query to be executed against the relational database. The retrieved results are then post-processed by the pipeline and converted into a clear, citizen-facing response. In addition to SQL-based retrieval, the architecture also supports semantic search via a vector database, enabling complementary contextual recovery when strictly structured data is insufficient.

4.4. Operational Flow

The operational flow, illustrated in Figure 1, can be summarized as a sequence of coordinated steps within a modular pipeline. First, the user submits a natural-language query, which is transformed into an embedding. This representation is used to retrieve relevant demonstrations from a contextual repository containing 59 Text-to-SQL examples. Each record combines a natural language question, a reference SQL query, and schema related metadata. The retrieved examples support intent classification and provide domain specific demonstrations that are incorporated into the SQL-generation prompt.

The relational schema is supplied through a separate mechanism, in which view names, column names, and descriptions are obtained from predefined structures and dynamically inserted into the corresponding prompt. Thus, semantic retrieval provides contextual demonstrations, while schema injection defines the database elements available to

the generator. The resulting query is executed against the institutional database, and the retrieved data are returned to the user in a concise and interpretable form.

This flow reflects an important design principle of the system: the language model does not serve as a direct source of factual truth, but rather as an interface layer that translates user intent into controlled access to official data. In other words, the model is used primarily for interpretation and generation, while factual grounding is delegated to institutional databases and official services. This distinction is especially relevant in the governmental domain, where traceability and consistency of answers are more important than generative flexibility.

4.5. Implementation Environment

The prototype was implemented in Python and relies on external services for language processing, vector retrieval, and deployment. Large language model capabilities were provided through Azure OpenAI/Foundry services, which were used for both classification and response generation. Vector-based semantic retrieval was supported by Pinecone, while structured data access relied on the institution's relational database.

During development, the system could be executed locally in two complementary ways. A Gradio-based interface was used for rapid testing and end-to-end behavior inspection, allowing the team to validate prompts, routes, and agent interactions. In parallel, the application could also be run through Modal in local-serving mode, reproducing the same API logic that will later be used in deployment. This dual setup simplified debugging and reduced the gap between development and production environments.

For production, the chatbot API was deployed on Modal as a serverless HTTP service. This choice reduced infrastructure overhead and enabled the team to package dependencies, expose stable endpoints, and maintain alignment with the project's existing Python-based architecture.

4.6. Design Rationale

Several design decisions became particularly relevant in this case study. First, a modular agent architecture was preferred over a monolithic prompting strategy. This separation allowed the system to assign well-defined responsibilities to each module, making debugging, prompt tuning, and future extension easier. In particular, separating intent classification from SQL generation reduced the risk of unnecessary database queries and improved controllability.

Second, the project adopted a hybrid strategy combining SQL-based access with retrieval-augmented generation. For numerical and transactional questions, SQL remained the primary source of truth, ensuring fidelity to structured institutional records. Semantic retrieval served only as a complementary mechanism when additional explanatory context was useful. This balance was essential for preserving accuracy while still allowing natural and helpful responses.

Third, managed services were prioritized in the system's initial version. Solutions such as Modal and Pinecone were selected to reduce DevOps complexity and accelerate experimentation, even at the cost of increased dependence on external services. In the context of an institutional prototype focused on practical deployment, this trade-off was considered acceptable.

4.7. Evaluation in the Governmental Setting

Table 2 presents the overall Execution Accuracy (EA) obtained by each model under the modular prompting strategy across different Top-k settings. These results provide an initial comparative view of how the evaluated models respond as contextual retrieval depth increases.

Tabela 2. Execution accuracy (%) obtained with the modular prompt for different models under distinct Top-k settings.

Model	Top-k = 3	Top-k = 5	Top-k = 10
ChatGPT 5	**94.08**	**94.74**	**95.39**
Gemini 2.5 Pro	**94.08**	94.08	94.08
DeepSeek-v3.1	89.47	90.13	92.76
GPT-4o mini	86.84	83.55	81.58

The best overall EA was achieved by ChatGPT 5 with Top-k = 10, reaching 95.39%. Close results were also observed for Top-k = 5 (94.74%) and Top-k = 3 (94.08%). Gemini 2.5 Pro presented stable performance across all configurations (94.08%), while DeepSeek-v3.1 improved as more context was provided, reaching 92.76% at Top-k = 10. In contrast, GPT-4o mini showed a consistent degradation as Top-k increased, dropping to 81.58% at Top-k = 10. These results suggest that stronger models tend to benefit more from a larger number of contextual examples, whereas smaller models may be negatively affected by the noise introduced when additional examples are included in the prompt.

In addition to execution-based performance, structural fidelity was evaluated through Component Matching. Table 3 summarizes the average CM obtained by each model, including the monolithic baseline for ChatGPT 5. This table provides an overall structural comparison before moving to the clause-level analysis.

Tabela 3. Overall component match average for each model and prompt strategy across different Top-k settings.

Model	Prompt Strategy	Top-k = 3	Top-k = 5	Top-k = 10
ChatGPT 5	Modular	0.951784	0.952859	0.949323
DeepSeek-v3.1	Modular	0.952420	0.950280	0.953426
Gemini-2.5-Pro	Modular	0.954139	0.954323	0.953118
GPT-4o mini	Modular	0.949567	0.944037	0.942552
ChatGPT 5	Monolithic	**0.955835**	**0.955984**	**0.956284**

The reported CM averages are based on 152 generated queries across all configurations. Clause-level averages are conditional on the occurrence of each clause in the corresponding reference SQL. All models achieved high structural similarity, with average CM values typically ranging from 0.94 to 0.95. The highest modular mean CM was obtained by Gemini 2.5 Pro at Top-k = 5 (0.954323), followed closely by DeepSeek-v3.1 at Top-k = 10 (0.953426). The monolithic version of ChatGPT 5 achieved slightly higher overall CM values across all Top-k settings, indicating that structural agreement does not always evolve in the same way as execution accuracy.

To better understand which SQL components remain more stable and which ones are more challenging, Tables 4, 5, and 6 report clause-level Component Matching results for the modular prompting strategy under each Top-k setting. Table 4 shows that, under Top-k = 3, the models already preserve the main structural backbone of the SQL queries. In particular, `ORDER BY` reached perfect agreement in all cases, while `FROM` and `GROUP BY` remained close to perfect values. By contrast, `HAVING` and `SELECT` exhibited lower scores, indicating that these clauses require more precise logical composition.

Tabela 4. Clause-level component match for modular prompting with Top-k = 3.

Clause	ChatGPT 5	DeepSeek-v3.1	Gemini-2.5-Pro	GPT-4o mini
FROM	0.996897	**0.999370**	0.996849	0.997498
GROUP BY	**0.991245**	**0.991245**	0.989109	**0.991245**
HAVING	**0.808846**	**0.808846**	**0.808846**	**0.808846**
LIMIT	0.984127	**1.000000**	0.984127	**1.000000**
ORDER BY	**1.000000**	**1.000000**	**1.000000**	**1.000000**
SELECT	**0.891423**	0.883158	0.887298	**0.891423**
WHERE	0.955679	0.967385	**0.968161**	0.950533
Average	0.951784	0.952420	**0.954139**	0.949567

As shown in Table 5, the Top-k = 5 setting preserves the same general pattern while slightly improving the average structural alignment for some models. Gemini 2.5 Pro achieved the highest modular mean CM in this configuration. Once again, `ORDER BY` remained perfectly matched, and `FROM` and `GROUP BY` stayed highly stable, whereas `SELECT` and `HAVING` continued to be the most challenging components.

Tabela 5. Clause-level component match for modular prompting with Top-k = 5.

Clause	ChatGPT 5	DeepSeek-v3.1	Gemini-2.5-Pro	GPT-4o mini
FROM	0.996137	**0.999370**	0.994087	0.998152
GROUP BY	**0.991245**	**0.991245**	0.984843	**0.991245**
HAVING	**0.808846**	**0.808846**	**0.808846**	**0.808846**
LIMIT	0.984127	**1.000000**	**1.000000**	0.984127
ORDER BY	**1.000000**	**1.000000**	**1.000000**	**1.000000**
SELECT	**0.891423**	0.880223	0.888433	0.889111
WHERE	0.959842	0.965083	**0.968472**	0.935189
Average	0.952859	0.950280	**0.954323**	0.944037

Table 6 indicates that increasing the retrieval depth to Top-k = 10 does not uniformly improve structural performance. DeepSeek-v3.1 reached the best modular mean CM in this setting, but the overall pattern remains consistent: `ORDER BY`, `FROM`, and `GROUP BY` were the most stable clauses, while `SELECT` and `HAVING` remained comparatively weaker. This reinforces the idea that additional context may help some models, but its impact is neither linear nor universal.

To assess the effect of prompt design more directly, Table 7 compares the overall

Tabela 6. Clause-level component match for modular prompting with Top-k = 10.

Clause	ChatGPT 5	DeepSeek-v3.1	Gemini-2.5-Pro	GPT-4o mini
FROM	0.996112	**0.999370**	0.993994	0.998164
GROUP BY	**0.991245**	**0.991245**	**0.991245**	**0.991245**
HAVING	**0.808846**	**0.808846**	0.805374	**0.808846**
LIMIT	0.984127	0.984127	**1.000000**	0.984127
ORDER BY	**1.000000**	**1.000000**	**1.000000**	**1.000000**
SELECT	**0.891423**	0.887687	0.884941	0.888223
WHERE	0.947885	0.966938	**0.967468**	0.931354
Average	0.949323	**0.953426**	0.953118	0.942552

execution accuracy of ChatGPT 5 under modular and monolithic prompting, while Table 8 details the corresponding clause-level structural comparison. Table 7 shows a clear trade-off between prompting strategies. The monolithic prompt slightly outperforms the modular one at Top-k = 3, indicating better efficiency when only a few contextual examples are available. However, at Top-k = 5, the modular approach achieves higher performance, suggesting a better capacity to exploit additional retrieved context. At Top-k = 10, both strategies converge to the same execution accuracy.

Tabela 7. Execution accuracy (%) of ChatGPT 5 under modular and monolithic prompting strategies.

Prompt Strategy	Top-k = 3	Top-k = 5	Top-k = 10
Modular	94.08	**94.74**	**95.39**
Monolithic	**94.74**	94.08	**95.39**

Monolithic prompting obtains slightly higher overall CM values for ChatGPT 5 across all Top-k settings, especially due to stronger `WHERE` clause matching. Even so, this structural advantage does not translate into a uniform superiority in execution accuracy, reinforcing that structural alignment and end-to-end performance capture related but distinct aspects of Text-to-SQL behavior.

Tabela 8. Clause-level comparison between modular and monolithic prompting strategies for ChatGPT 5 across Top-k settings.

Clause	Mod. k=3	Mod. k=5	Mod. k=10	Mono. k=3	Mono. k=5	Mono. k=10
FROM	**0.996897**	0.996137	0.996112	0.996665	0.996536	0.996576
GROUP BY	**0.991245**	**0.991245**	**0.991245**	**0.991245**	**0.991245**	**0.991245**
HAVING	**0.808846**	**0.808846**	**0.808846**	**0.808846**	**0.808846**	**0.808846**
LIMIT	0.984127	0.984127	0.984127	**1.000000**	0.984127	0.984127
ORDER BY	**1.000000**	**1.000000**	**1.000000**	**1.000000**	**1.000000**	**1.000000**
SELECT	**0.891423**	**0.891423**	**0.891423**	**0.891423**	**0.891423**	0.891041
WHERE	0.955679	0.959842	0.947885	0.967947	0.969542	**0.970884**
Average	0.951784	0.952859	0.949323	0.955835	0.955984	**0.956284**

Overall, the results indicate that the proposed modular pipeline is effective for natural language access to governmental transparency data, achieving high execution accuracy and strong structural alignment with reference queries. More importantly, the findings highlight a trade-off between prompting strategies: while the monolithic prompt is more efficient in low-context scenarios (Top-k = 3), the modular approach better exploits richer contextual information, achieving superior performance when more examples are available (Top-k = 5). In this sense, the main advantage of the modular approach lies not in universal superiority across all settings, but in its better organization, clearer separation of responsibilities, and stronger potential for controlled evolution in institutional systems.

5. Discussion and Final Considerations

The results show that modular prompt design is a viable and competitive strategy for *Text-to-SQL* systems in governmental settings, although not universally superior to monolithic prompting. The experiments reveal a trade-off between strategies: monolithic prompting performs slightly better in low-context scenarios, while the modular approach benefits more from additional retrieved context. Clause-level results indicate that core SQL components such as `FROM`, `GROUP BY`, and `ORDER BY` remain stable, whereas `SELECT` and `HAVING` are more challenging, suggesting that complex logical composition remains a key bottleneck. By emphasizing modularization, reusability, traceability, and component-oriented evaluation, the proposed approach connects software engineering principles with natural language processing and promptware engineering [Zhuo et al. 2024, Chen et al. 2025]. Thus, *Composable Prompt Modularization* emerges as a promising paradigm for building more accurate, auditable, and adaptable *Text-to-SQL* agents in institutional contexts.

The current prototype depends on managed external services for language model inference, vector retrieval, and deployment. Although these services supported rapid development, they raise concerns related to cost, availability, reproducibility, data governance, and institutional control. Future work will investigate self-hosted open-source language models, small language models, and locally deployable vector stores, evaluating them in terms of execution accuracy, latency, computational requirements, privacy, and maintenance. We also intend to compare the proposed approach with related *Text-to-SQL* methods using EA and CM under a common evaluation protocol, either on a public benchmark or on the institutional dataset under equivalent experimental conditions.

Acknowledgment‘s

The authors would like to thank Zairo Bastos for his support in the implementation of the agents, and Ricardo Wagner for preparing and providing a subset of data derived from the official Municipal Information System (SIM) database, which enabled the development and testing of the proposed solution. The authors also acknowledge the financial support provided by TCE and Funcap under Project Process No. 31052.002953/2024-14.

Declaration on the use of generative AI. The authors used OpenAI's ChatGPT (GPT-5) for language revision and text organization. All content was reviewed and approved by the authors, who assume full responsibility for the manuscript.

Referências

Chen, Z., Wang, C., Sun, W., Yang, G., Liu, X., Zhang, J. M., and Liu, Y. (2025). Promptware engineering: Software engineering for llm prompt development. *arXiv preprint arXiv:2503.02400*.

Gao, D., Wang, H., Li, Y., Sun, X., Qian, Y., Ding, B., and Zhou, J. (2024). Text-to-sql empowered by large language models: A benchmark evaluation. *Proceedings of the Very Large Data Bases Endowment (PVLDB)*, 17(5):1132–1145.

Lan, W., Wang, Z., Chauhan, A., Zhu, H., Li, A., Guo, J., Zhang, S., Hang, C.-W., Lilien, J., Hu, Y., Pan, L., Dong, M., Wang, J., Jiang, J., Ash, S., Castelli, V., Ng, P., and Xiang, B. (2023). UNITE: A Unified Benchmark for Text-to-SQL Evaluation. `https://arxiv.org/abs/2305.16265`. arXiv:2305.16265.

Liang, J. T., Lin, M., Rao, N., and Myers, B. A. (2025). Prompts are programs too! understanding how developers build software containing prompts. *Proceedings of the ACM on Software Engineering*, 2(FSE):1591–1614.

Liu, M., Wang, X., Xu, J., Yi, W., and Wolfson, O. (2026). A systematic review of natural language interfaces for databases. *Frontiers of Computer Science*, 20.

Liu, Y., Xu, J., Zhang, L. L., Chen, Q., Feng, X., Chen, Y., Guo, Z., Yang, Y., and Cheng, P. (2025). Beyond prompt content: Enhancing llm performance via content-format integrated prompt optimization. *arXiv preprint arXiv:2502.04295*.

Luo, Y., Zhang, Z., Zhang, Y., Xie, Z., Wang, J., Jatowt, A., and Yang, Z. (2024). Data-scarce event argument extraction: A dynamic modular prompt tuning model based on slot transfer. In *2024 7th International Conference on Machine Learning and Natural Language Processing (MLNLP)*, pages 1–6. IEEE.

Mao, Y., He, J., and Chen, C. (2025). From prompts to templates: A systematic prompt template analysis for real-world llmapps. *arXiv preprint arXiv:2504.02052*.

Nahid, M. M. H., Rafiei, D., Zhang, W., and Zhang, Y. (2026). Rethinking schema linking: A context-aware bidirectional retrieval approach for text-to-SQL. In *Findings of the Association for Computational Linguistics: EACL 2026*, pages 4516–4546, Rabat, Morocco. Association for Computational Linguistics.

Petrola, L., Brayner, A., and Franco, W. (2025). Heuristic-guided text-to-sql translation with llms: Optimizing natural language interfaces for relational databases. In *Simpósio Brasileiro de Banco de Dados (SBBD)*, pages 126–139. SBC.

Piao, S., Lee, J., and Park, S. (2026). LitE-SQL: A lightweight and efficient text-to-SQL framework with vector-based schema linking and execution-guided self-correction. In *Findings of the Association for Computational Linguistics: EACL 2026*, pages 3593–3608, Rabat, Morocco. Association for Computational Linguistics.

Pourreza, M. and Rafiei, D. (2023). DIN-SQL: Decomposed in-context learning of text-to-SQL with self-correction. In *Advances in Neural Information Processing Systems*, volume 36, pages 36339–36348. Curran Associates, Inc.

Santos, W. F., Santos, P. V. d., Martins, M. S. R., Lekakis, L. F., Rosa, F. L., Costa, B. M., Filho, M. A. P., and Montalvão, I. A. (2026). Multi-agent architecture with RAG and

dynamic context windows for text-to-SQL optimization. In *Proceedings of the 17th International Conference on Computational Processing of Portuguese (PROPOR 2026) – Vol. 1*, pages 988–993, Salvador, Brazil. Association for Computational Linguistics.

Schnabel, T. and Neville, J. (2024). Symbolic prompt program search: A structure-aware approach to efficient compile-time prompt optimization. *arXiv preprint arXiv:2404.02319*.

Shi, L., Tang, Z., Zhang, N., Zhang, X., and Yang, Z. (2025). A survey on employing large language models for text-to-SQL tasks. *ACM Computing Surveys*, 58(2):1–37.

Shinn, N., Cassano, F., Labash, B., Gopinath, A., Narasimhan, K., and Yao, S. (2023). Reflexion: Language agents with verbal reinforcement learning, 2023. *URL https://arxiv.org/abs/2303.11366*, 1.

Tan, Z., Liu, X., Shu, Q., Li, X., Wan, C., Liu, D., Wan, Q., and Liao, G. (2024). Enhancing text-to-sql capabilities of large language models through tailored promptings. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 6091–6109.

Wang, B., Ren, C., Yang, J., Liang, X., Bai, J., Chai, L., Yan, Z., Zhang, Q.-W., Yin, D., Sun, X., and Li, Z. (2025). MAC-SQL: A multi-agent collaborative framework for text-to-SQL. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 540–557, Abu Dhabi, UAE. Association for Computational Linguistics.

Wang, M., Liu, Y., Liang, X., Li, S., Huang, Y., Zhang, X., Shen, S., Guan, C., Wang, D., Feng, S., et al. (2024). Langgpt: Rethinking structured reusable prompt design framework for llms from the programming language. *arXiv preprint arXiv:2402.16929*.

Yavuz, S., Gur, I., Su, Y., and Yan, X. (2021). Text-to-sql in the wild: A naturally-occurring dataset based on stack exchange data. In *Proceedings of the 3rd Workshop on Natural Language Processing for Programming (NLP4Prog)*, pages 77–87, Online. Association for Computational Linguistics.

Zhuo, J., Zhang, S., Fang, X., Duan, H., Lin, D., and Chen, K. (2024). Prosa: Assessing and understanding the prompt sensitivity of llms. *arXiv preprint arXiv:2410.12405*.