Beyond Natural Language Plans: Structure-Aware Agentic Planning for Query-Focused Table Summarization

Weijia Zhang[1] **Yifei Yuan**[2] **Songgaojun Deng**[3] **Evangelos Kanoulas**[1]
[1]University of Amsterdam [2]ETH Zurich [3]Eindhoven University of Technology
weijia.zhangplus@gmail.com

Abstract

Query-focused table summarization requires complex reasoning, often approached through step-by-step natural language (NL) plans. However, NL plans are inherently ambiguous and lack structure, making their conversion into executable programs like SQL brittle, especially for multi-table tasks. To address this, we introduce TaSoF, a schema-constrained, dependency-aware plan representation inspired by formalisms used in traditional multi-agent systems, and an agentic framework, SPaGe, that organizes the reasoning process into three phases: 1) *Structured Planning* to generate TaSoF from a query, 2) *Graph-based Execution* to convert plan steps into SQL and model dependencies via a directed acyclic graph for parallel execution, and 3) *Summary Generation* to produce query-focused summaries. Our method explicitly captures complex dependencies and supports dependency-aware execution. Experiments on two datasets show that SPaGe consistently outperforms prior models in the evaluated single- and multi-table settings and achieves higher per-step execution success than the evaluated NL plan formats.

1 Introduction

Query-focused table summarization aims to generate concise textual summaries from tabular data conditioned on user queries and has recently emerged as a core challenge in both natural language generation and table understanding (Zhao et al., 2023, 2024; Zhang et al., 2024a). A central difficulty is accurately extracting and aggregating query-relevant information from complex tables. Early work primarily relies on end-to-end models that perform reasoning implicitly (Liu et al., 2022b,a; Jiang et al., 2022), which require large-scale training data and offer limited interpretability (Nguyen et al., 2025). With the advent of large language models (LLMs), recent approaches have shifted toward explicit multi-step table reasoning, leveraging query decomposition or predefined workflows to generate and execute structured reasoning plans (Ye et al., 2023; Zhao et al., 2024; Wang et al., 2024b; Zhang et al., 2024c; Li et al., 2025).

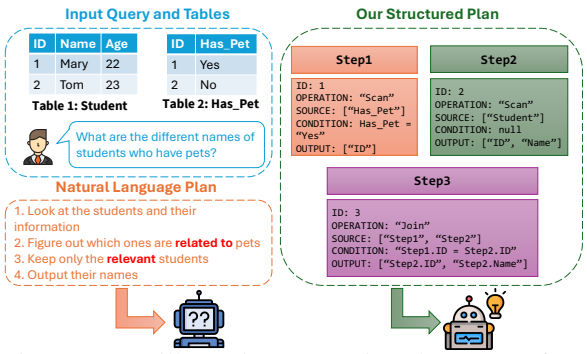


Figure 1: An illustrative comparison between a free-form NL plan and our structured plan, TaSoF. The NL plan contains vague, underspecified steps that omit key execution details, whereas TaSoF makes the operations explicit.

Despite recent progress in plan-based reasoning for table understanding, most existing methods still generate reasoning plans as free-form natural language (NL). While intuitive, NL plans lack explicit structure and executable semantics (Cao et al., 2025), making their conversion into executable table operations (e.g., SQL) brittle and error-prone. Underspecified or vague expressions can easily lead to incorrect operator selection, faulty joins, or execution failures (Zhao et al., 2024; Nguyen et al., 2025). Figure 1 illustrates this issue. The NL plan contains ambiguous phrases such as "*figure out which ones are related to pets*," which omit essential execution details like filtering conditions, join keys, and intermediate outputs. As a result, execution modules must implicitly infer semantics, leading to inconsistent or incorrect programs, particularly in multi-table settings.

To address these limitations, we move from ambiguous NL plans to a schema-constrained rep-

Proceedings of the 19th International Natural Language Generation Conference: Long Papers, pages 828–841
October 17-21, 2026 ©2026 Association for Computational Linguistics

resentation (Cao et al., 2025). Inspired by prior work on formal reasoning systems (d'Inverno et al., 1997; Hilaire et al., 2000; Gruer et al., 2002), we define the reasoning plan as a Table State and operation Flow (TaSoF), which explicitly represents table operations and their dependencies. As shown in Figure 1, TaSoF specifies operation types, data sources, conditions, and intermediate outputs. These explicit fields reduce the interpretation required during LLM-based step-to-SQL translation and expose dependencies for graph-based scheduling. In the example, this process yields the query-relevant summary, "*There is only one student who has pets, and the name of this student is Mary.*"

Building on this formulation, we introduce SPaGe (Structured Planning and Graph-based execution), an agentic framework that supports both single- and multi-table summarization. The framework consists of three stages: 1) *Structured Planning*: an LLM generates a multi-step plan in the schema-constrained TaSoF format; 2) *Graph-based Execution*: each step is converted into executable SQL and executed via an external interpreter, with the overall plan modeled as a Directed Acyclic Graph (DAG) to enable parallel execution of independent steps; 3) *Summary Generation*: the final summary is produced from the execution results and conditioned on the user query.

We evaluate SPaGe on two public query-focused table summarization datasets, FeTaQA and QFMTS. Across both datasets, it consistently outperforms all compared baselines. In the plan-format ablation, TaSoF also obtains higher summary scores and per-step execution success than the evaluated NL formats.

To our knowledge, TaSoF is the first schema-constrained plan representation evaluated for query-focused table summarization, distinguishing it from structured query planning in Text-to-SQL (Section 6). Our framework builds on established components, including LLM-based planning, SQL execution, and DAG scheduling. Its contribution lies in adapting and integrating them through a dependency-aware intermediate representation for this task. Unlike Text-to-SQL systems that produce a single query as the final answer, or NL-plan agents whose free-form plans require implicit interpretation, TaSoF provides executor-agnostic steps with named fields and a fixed operation vocabulary. The fields make step-to-SQL translation more explicit, support dependency-based scheduling, and enable field-level error diagnosis. In our plan-format ablation, this design yields a per-step execution success rate of 98.2% versus 94.4–95.1% for the evaluated NL formats. Our key contributions are summarized as follows:

- We define TaSoF, a schema-constrained, dependency-aware representation of table states and operations for query-focused table summarization.
- We propose SPaGe, a structure-aware agentic framework that combines TaSoF with graph-based execution and query-focused summary generation.
- We evaluate the framework on two benchmarks and analyze summary quality, per-step executability, component contributions, and reductions in DAG execution depth.

2 Problem Formulation

The task of *query-focused table summarization* aims to produce a concise, natural language summary s that directly addresses the information need expressed in a user query q, given a collection of relational tables $\mathcal{T} = \{T_1, T_2, \ldots, T_n\}$. Formally, we define this process as a mapping:

$$f : (q, \mathcal{T}) \mapsto s,$$

where f is the summarization function, and s is tailored to capture only the information relevant to q. Each table $T_i \in \mathcal{T}$ consists of rows organized according to a schema:

table name: $name$ **col:** $h_1 \mid \ldots \mid h_n$
row 1: $c_{1,1} \mid \ldots \mid c_{1,n}$ $\ldots$ **row m:** $c_{m,1} \mid \ldots \mid c_{m,n}$.

Here, n is the number of columns and m is the number of rows in T_i. Unlike Text-to-SQL and short-form table question answering, which typically return a single value or set of cells, query-focused table summarization produces a multi-sentence output s that aggregates and contextualizes multiple query-relevant facts (e.g., reference summaries average 58.5 words on QFMTS; Table 1). The task therefore demands both accurate table reasoning and faithful text generation.

3 Methodology

3.1 Framework Overview

Our framework follows an agentic "plan-then-execute" paradigm (Wang et al., 2023) comprising

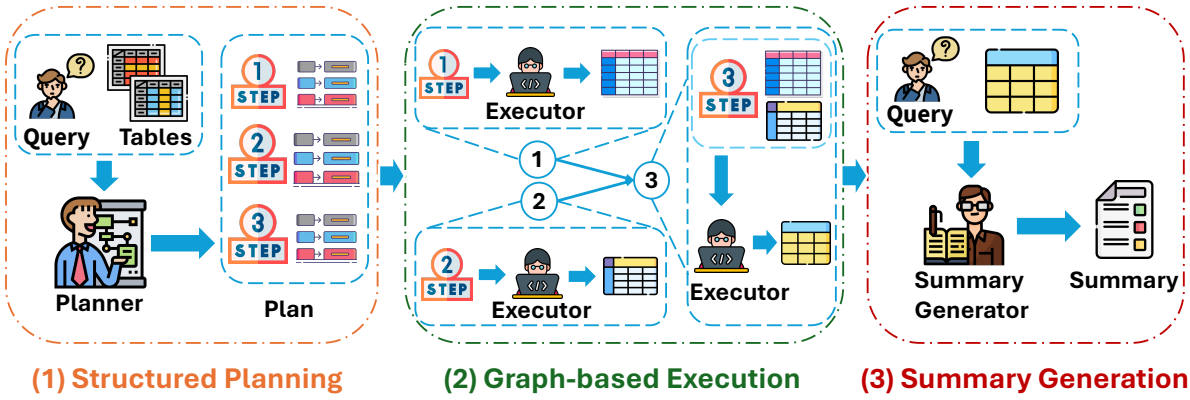


Figure 2: Overview of our framework, SPaGe. First, the planner generates a structured plan. The executor then constructs a directed acyclic graph from step dependencies and executes independent planning steps in parallel according to the graph structure. Finally, the summary generator synthesizes the final execution output into an informative summary tailored to the query.

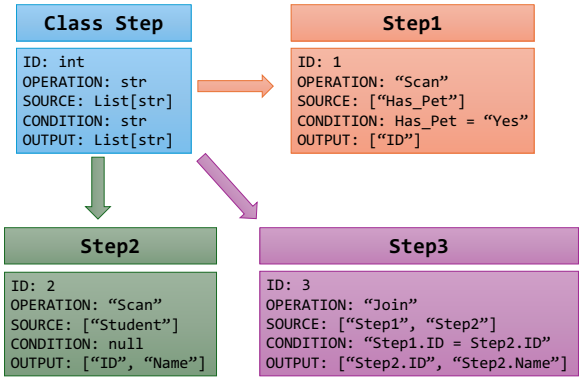


Figure 3: Class definition of our structured plan, TaSoF, alongside an example plan instance.

three phases: *Structured Planning*, *Graph-based Execution*, and *Summary Generation*. Each phase is handled by a dedicated component (Figure 2). The planner generates multiple atomic, structured steps from the given query q and tables $\mathcal{T}$. The executor identifies dependencies among these steps, constructs a Directed Acyclic Graph (DAG) to determine the execution order, and executes a corresponding SQL query for each step, using the results of previous steps to produce intermediate execution tables. Finally, the summary generator synthesizes these execution tables into a coherent and concise textual summary.

3.2 Structured Planning

We first describe TaSoF, a structured planning strategy that represents a plan P as a sequence of schema-constrained, machine-readable steps $P = \{s_1, s_2, \ldots, s_n\}$. This structure makes the information passed from planning to execution explicit.

Plan Definition. Each step within the plan defines a table operation that transforms one or more source tables into a result table. The class definition for the step is shown in Figure 3. Specifically, we formulate a **step** as a conventional software-engineering class with five attributes: $S_i = \{\texttt{ID}, \texttt{OPERATION}, \texttt{SOURCE}, \texttt{CONDITION}, \texttt{OUTPUT}\}$. Here, `ID` denotes the step identifier, and `OPERATION` specifies the table operation to be performed. Inspired by the query plan language (QPL) (Eyal et al., 2023), we consider nine table operations that cover the most common scenarios. Definitions of these operations are provided in Appendix A.1. The `SOURCE` field specifies the source tables involved in the operation. These may include the original input tables or the intermediate result tables returned by previous steps. The `CONDITION` field defines any constraints or filters applied during the operation, such as row or column selections and keys used for table joins. Finally, `OUTPUT` denotes the output columns of the resulting table. For example, in the three-step plan shown in Figure 3, the step whose `ID` is 1 extracts the *ID* column from the *Has_Pet* table under the condition *Has_Pet = "Yes"*.

Plan Generation. To generate structured plans, we leverage the structured-output capabilities of modern LLMs. Given a user query q and a set of input tables $\mathcal{T}$, we prompt the LLM to produce a sequence of structured planning steps that conform to a predefined schema. Since LLMs operate on textual inputs, we follow prior work (Liu et al., 2022b; Pal et al., 2023) and linearize each table into a text representation. Specifically, a table with n columns and m rows is converted using the format described

in Section 2. Unlike previous methods (Wang et al., 2024b; Zhao et al., 2024) that linearize entire tables and incur substantial token overhead for large tables, our planner operates on only the top k rows, where $k \ll m$. This design preserves planning quality while significantly reducing input length. We provide a detailed analysis of the impact of k in Section 5.2.

3.3 Graph-based Execution

The execution phase aims to obtain an intermediate execution table for each planning step s_i. We observe that some steps can be scheduled concurrently because they depend on distinct input tables or separate preceding steps (as shown by the example plan in Figure 3). The first and second steps operate on different source tables and are therefore independent. Thus, they can occupy the same execution cycle, reducing the dependency depth relative to sequential execution.

Graph Construction. In general, each plan can be formulated as a DAG, denoted by $G = (V, E)$, where $V = \{s_1, \ldots, s_n\}$ denotes the set of planning steps, and $E = \{e_{ij}\}$ represents the set of step dependencies. Specifically, an edge e_{ij} exists if the output table from step s_i is used as input to step s_j. This graph-based formulation explicitly captures inter-step dependencies and exposes independent steps that can be scheduled in the same execution cycle.

Program-Augmented Execution. Given a planning step, LLMs may struggle to generate well-formatted intermediate tables directly. To address this limitation, we enhance the LLM-based executor by incorporating an external code interpreter. Specifically, the executor leverages the code generation capabilities of the LLM to produce executable code. Following prior work (Cheng et al., 2023; Nguyen et al., 2025), we prompt the LLM to generate a SQL query q_i^{sql} for each step s_i. We include several demonstrations in the prompt to support in-context learning for SQL generation (Brown et al., 2020). Specifically, we adapt the demonstrations from Eyal et al. (2023) to suit our task setting. The code interpreter executes the generated SQL query q_i^{sql} to produce the corresponding execution table. Per-step execution success is analyzed in Section 5.2.

Property	FeTaQA	QFMTS
Table Source	Wiki	Database
Avg. # Tables	1.0	1.8
Avg. # Rows	15.2	3118.0
Avg. # Words in Summary	23.3	58.5
Test Size	2,003	608

Table 1: Data statistics for the two benchmarks. FeTaQA is a *single-table* dataset, while QFMTS is a *multi-table* dataset.

3.4 Summary Generation

Once the execution process is complete, we use an LLM-based summary generator to produce a comprehensive summary from the original query q and the final table produced by the last step s_n. This summary is expected to integrate all query-relevant information derived from the input tables.

4 Experimental Setup

4.1 Datasets

We use two popular query-focused table summarization datasets, FeTaQA (Nan et al., 2022) and QFMTS (Zhang et al., 2024a). FeTaQA is designed for *single-table* scenarios and comprises tables from Wikipedia. It provides manually annotated query-summary pairs aligned with the corresponding tables. In contrast, QFMTS is designed for *multi-table* scenarios. It contains queries and relational tables from databases, along with annotated summaries tailored to the associated queries. Key dataset statistics are presented in Table 1.

4.2 Baselines

We evaluate our framework against two categories of baselines: prompting-based methods and program-augmented models. For prompting-based baselines, we include **Chain-of-Thoughts (CoT)** (Wei et al., 2022), **ReFactor** (Zhao et al., 2023), **DirectSumm** (Zhang et al., 2024a), and **Reason-then-Summ** (Zhang et al., 2024a). For program-augmented baselines, we consider **Binder** (Cheng et al., 2023), **Dater** (Ye et al., 2023), and **TaPERA** (Zhao et al., 2024). Additional details on all baseline models are provided in Appendix B.1.

4.3 Evaluation Protocols

Following Zhao et al. (2024), we evaluate the quality of generated summaries using both automatic metrics and human judgments. For automatic evalu-

Method	FeTaQA			QFMTS		
	BLEU	ROUGE-L	METEOR	BLEU	ROUGE-L	METEOR
Prompt-based						
CoT	28.2	51.0	56.9	31.5	54.3	58.1
ReFactor	26.2	53.6	57.2	N/A	N/A	N/A
DirectSumm	29.8	51.7	58.2	33.6	57.0	62.8
Reason-then-Summ	<u>31.7</u>	52.6	<u>60.7</u>	40.8	62.7	66.2
Program-Augmented						
Binder	25.5	47.9	51.1	<u>42.5</u>	<u>65.3</u>	<u>70.7</u>
Dater	29.8	<u>54.0</u>	59.4	N/A	N/A	N/A
TaPERA	29.5	53.4	58.2	N/A	N/A	N/A
Our Structured-Aware						
SPaGe	**33.8**	**55.7**	**62.3**	**45.7**	**68.3**	**73.4**
- *w/o Planner*	29.2↓ 4.6	51.4↓ 4.3	59.3↓ 3.0	40.6↓ 5.1	63.2↓ 5.1	67.7↓ 5.7
- *w/o Executor*	32.2↓ 1.6	53.6↓ 2.1	60.9↓ 1.4	42.3↓ 3.4	65.8↓ 2.5	70.8↓ 2.6
- *w/o Planner and Executor*	28.6↓ 5.2	50.3↓ 5.4	57.8↓ 4.5	36.3↓ 9.4	59.3↓ 9.0	65.2↓ 8.2

Table 2: Automated evaluation results on the benchmarks. The best and second-best scores are highlighted in **bold** and <u>underlined</u>, respectively. ReFactor, Dater, and TaPERA are designed for single-table inputs and are therefore not applicable to QFMTS, a multi-table benchmark.

Plan Format	BL	R-L	ME	ESR
QDMR	42.3	65.1	69.4	94.4
QPL	43.4	66.2	71.5	95.1
TaSoF (Ours)	**45.7**	**68.3**	**73.4**	**98.2**

Table 3: Comparison of TaSoF with NL plan formats on QFMTS. BL, R-L, ME, and ESR denote BLEU, ROUGE-L, METEOR, and execution success rate, respectively. ESR refers to the proportion of plan steps that can be successfully executed. The best results are highlighted in **bold**.

ation, we report three standard metrics: **BLEU** (Papineni et al., 2002), **ROUGE** (Lin and Hovy, 2003), and **METEOR** (Banerjee and Lavie, 2005). Since such n-gram metrics capture surface-level similarity but correlate poorly with factual and reasoning quality in table-grounded generation (Liu et al., 2023; Song et al., 2024; Wang et al., 2024a), we complement them at two levels. At the execution level, we report the per-step execution success rate (ESR), defined as the proportion of plan steps whose generated SQL can be executed (Section 5.2); ESR measures syntactic executability rather than semantic plan correctness. At the fact level, we conduct a human evaluation along three dimensions: 1) **Correctness:** consistency with the final execution table; 2) **Faithfulness:** factual support from the original input tables; and 3) **Coverage:** completeness with respect to the user query and execution results. The human evaluation is conducted by three of the authors, who independently rate 100 sampled outputs per system (details in Appendix B.3).

4.4 Implementation Details

We adopt `gpt-4o-mini` as the backbone LLM for the planner, executor, and summary generator, motivated by its strong performance on table understanding tasks (Nguyen et al., 2025). To reduce backbone-related confounding, we rerun *all* baseline models with the same `gpt-4o-mini` backbone, following their original papers only to configure their prompts and hyperparameters. All automated results are averaged over three independent runs. Further implementation details are provided in Appendix B.2.

5 Results and Analyses

5.1 Main Results

As shown in Table 2, SPaGe outperforms all baseline models on both the FeTaQA and QFMTS datasets. The improvement is especially pronounced on QFMTS, where its ROUGE-L score is 68.3, compared with 65.3 for the second-best model. We find that program-augmented models such as Binder outperform the strongest prompt-based baseline, Reason-then-Summ, on QFMTS but underperform on FeTaQA. This discrepancy likely stems from dataset differences: QFMTS primarily contains standard relational tables, whereas

FeTaQA includes many free-form tables collected from Wikipedia. Binder relies on NL plans, which can lead to incorrect SQL generation on such tables and consequently degraded performance. In contrast, SPaGe achieves consistently strong results across the two evaluated datasets. These results show that the framework can handle both the relational tables in QFMTS and the free-form Wikipedia tables in FeTaQA, although broader generalization remains untested.

5.2 Ablation Study

Framework Ablations. Table 2 presents an ablation study of three SPaGe variants. In *w/o Planner*, the executor generates SQL directly from the user query; *w/o Executor* relies solely on the LLM to produce intermediate tables; and *w/o Planner and Executor* generates the summary directly from the query. The results show that the structured planner is a critical component of the framework, as its removal leads to the largest performance degradation across both datasets. In addition, the executor contributes more substantially on QFMTS than on FeTaQA, as evidenced by the larger performance drop. A plausible explanation is that FeTaQA primarily consists of Wikipedia-based tables, which are less amenable to SQL-style execution, thereby reducing the executor's relative impact.

Ablation on Plan Formats. To evaluate the effectiveness of our structured plan TaSoF, we compare it with NL plan formats, including QPL (Eyal et al., 2023) and QDMR (Wolfson et al., 2022), adapting our planner to generate each format while keeping the backbone and downstream execution and generation components unchanged. This comparison provides our closest controlled evidence for the plan representation, although the planner prompts necessarily differ to elicit each format and therefore remain a possible confound. As shown in Table 3, TaSoF consistently outperforms the NL plan formats across all text-overlap metrics and achieves a higher per-step ESR of 98.2% (vs. 95.1% for QPL).

ESR measures *syntactic executability*, defined as the proportion of plan steps whose generated SQL can be executed. It does not measure semantic plan correctness or whole-plan success. Higher ESR co-occurs with higher summary scores across the three formats, and the human evaluation reports a correctness score of 4.08/5 and faithfulness of 3.65/5 for SPaGe (Table 6); these are indirect, output-level signals rather than direct validation of intermediate plans. We do not evaluate plans against gold reasoning traces, label semantic planning errors, or measure how incorrect intermediate steps propagate to summaries. These analyses remain important future work.

# Top-k Rows	Avg. # Tokens	BL	R-L	ME	ESR
5	217.9	**45.7**	**68.3**	**73.4**	**98.2**
10	381.6	44.9	67.8	72.7	97.5
20	591.0	44.3	67.2	72.5	96.7
50	933.4	43.2	66.5	71.3	92.6

Table 4: The effect of varying the number of top-k rows used for plan generation on QFMTS. Avg. # Tokens denotes the average number of tokens from the input tables provided to the planner. BL, R-L, ME, and ESR are the same as in Table 3. The best results are highlighted in **bold**.

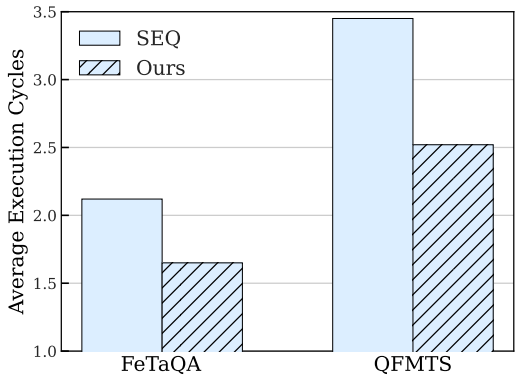


Figure 4: Comparison of the average number of execution cycles under sequential execution (SEQ) and our graph-based execution (Ours).

Ablation on Top-k Rows. To study the effect of row truncation on plan generation, we ablate the number of retained top-k rows (Table 4). Retaining the top five rows achieves the best performance across all metrics on both benchmarks, suggesting that column headers and schema information carry much of the signal needed for plan generation. Retaining only the top-k rows ($k \ll m$) preserves performance and execution success while significantly reducing token usage. In contrast, prior methods such as DirectSumm and TaPERA linearize entire tables, incurring substantial token overhead, especially on multi-table datasets like QFMTS, which contain more than 3,000 rows per table on average.

5.3 Parallelism of Graph-based Execution

We quantify the dependency depth of graph-based execution using *execution cycles*; this is a structural measure, not observed latency. Under sequential execution, the number of cycles equals the number of plan steps. Graph-based execution instead

Error Mode	Representative Example

Semantic Mismatch between Query and Table Schema	**Query:** What is the average duration of projects?
	Tables:
	Table 1: Projects
	Table Schema: *[ProjectID, ProjectName, StartDate, EndDate, Cost]*
	Natural Language Plan:
	1. *Find duration of each project.*
	2. *Calculate the average of these durations.*
	3. *Return the average duration for all projects.*
	Our TaSoF {`ID; OPERATION; SOURCE; CONDITION; OUTPUT`}:
	`Step1 = {1;Scan;[Projects];null;[StartDate, EndDate]}`
	`Step2 = {2;Aggregate;[Step1];null;`
	`[AVG(EndDate - StartDate) AS AvgDuration]}`

Complex Step Dependencies in Multi-Table Scenarios	**Query:** List the names and department locations of employees who are over 40 and work in departments located in "London".
	Tables:
	Table 1: Employees
	Table Schema: *[EmpID, EmpName, Age, DeptID]*
	Table 2: Departments
	Table Schema: *[DeptCode, DeptName, Location]*
	Natural Language Plan:
	1. *Get employees over 40 from the employees table.*
	2. *Find department locations in "London" from the departments table.*
	3. *Combine employee names and department locations.*
	Our TaSoF {`ID; OPERATION; SOURCE; CONDITION; OUTPUT`}:
	`Step1 = {1;Scan;[Employees];Age > 40;[EmpID, EmpName, DeptID]}`
	`Step2 = {2;Scan;[Departments];Location = "London";[DeptCode, Location]}`
	`Step3 = {3;Join;[Step1, Step2];Step1.DeptID = Step2.DeptCode;`
	`[Step1.EmpName, Step2.Location]}`

Table 5: Representative examples of two illustrative NL-plan failure modes discussed in our qualitative analysis (Section 5.5). We do not estimate the frequency of these modes. For readability, we flatten our class-based structured plan. Red phrases highlight underspecified NL steps, while green phrases show how our plan makes the corresponding operations explicit.

Method	Correctness	Faithfulness	Coverage
Reason-then-Summ	3.62	3.08	3.35
Binder	3.74	3.22	3.55
SPaGe	**4.08**	**3.65**	**3.90**

Table 6: Human evaluation results on QFMTS. The best results are highlighted in **bold**. Scores are based on a 5-point Likert scale (1–5) and averaged over three annotators on 100 randomly sampled examples per model. Evaluation details are provided in Appendix B.3.

places independent steps in the same cycle, so the number of cycles equals the length of the longest path in the DAG. For example, although the plan in Figure 3 contains three steps, Steps 1 and 2 can occupy one cycle, followed by Step 3, resulting in two cycles. This measure does not capture wall-clock latency, total LLM calls, API rate limits, or database overheads (see Limitations).

Figure 4 compares the average execution cycles of sequential and graph-based execution. Graph-based execution reduces the number of cycles by approximately 28% on QFMTS. The multi-table structure of QFMTS creates more independent branches than FeTaQA, making the difference in dependency depth more pronounced; the figure does not establish a corresponding wall-clock reduction.

5.4 Human Evaluation

Table 6 reports the human evaluation results for three representative methods: the prompt-based Reason-then-Summ, the program-based Binder, and our approach, SPaGe. Our method achieves the highest score across all three dimensions (correctness, faithfulness, and coverage). Fleiss' kappa indicates substantial consistency among the three author annotators on all dimensions, but it does not establish annotator independence or eliminate potential author bias (see Appendix B.3). In particular, SPaGe outperforms Reason-then-Summ on faithfulness (3.65 vs. 3.08). One plausible explanation is that SQL execution extracts query-relevant information before summary generation, whereas Reason-then-Summ relies solely on LLM generation over the table context. SPaGe also outperforms Binder under the same evaluation protocol, showing that the complete structured-planning pipeline is associated with higher summary scores; the hu-

man evaluation does not directly establish the semantic correctness of its generated SQL.

5.5 Qualitative Error Analysis

To illustrate *how* NL plans can fail where TaSoF succeeds, we qualitatively examine failed NL-plan executions and present two illustrative error modes in Table 5. We do not quantify the frequency of these modes or claim that they exhaust the sources of failure. The per-step execution success rates in Table 3 (98.2% for TaSoF vs. 94.4–95.1% for NL formats) provide only an aggregate measure of syntactic executability.

The first error mode is a **semantic mismatch** between the query and the table schema. In the corresponding example, the query uses the term *duration* even though it is not present in the table schema. The NL plan also includes the underspecified step "*Find duration of each project*," which leads to execution failure. In contrast, TaSoF grounds the calculation in the existing *StartDate* and *EndDate* source columns and expresses the requested average as an explicit aggregate expression.

The second error mode involves **complex step dependencies**, which are especially problematic when multiple input tables are involved. In the corresponding example, the NL plan fails to connect the first two steps properly. As a result, the final step wrongly performs a union of their outputs, even though the query requires their intersection. TaSoF addresses this by using the `SOURCE` field to explicitly link steps. For instance, Step 3 cites Steps 1 and 2 as its sources, ensuring the correct interpretation of the query.

6 Related Work

Program-Augmented Table Reasoning. Table reasoning requires operations such as arithmetic calculations and the filtering of irrelevant rows or columns. Early methods fine-tune end-to-end models that reason implicitly (Liu et al., 2022b; Jiang et al., 2022), offering limited interpretability. Recent program-augmented methods (Cheng et al., 2023; Ye et al., 2023; Zhang et al., 2024b) instead generate executable code (e.g., SQL or Python) to derive intermediate results, making reasoning traceable and verifiable, but they primarily target single-table scenarios (Chen et al., 2025). Our use of SQL execution also relates to LLM-based Text-to-SQL, where structured intermediate representations and agentic decomposition have been explored, including prompt organization in DAIL-SQL (Gao et al., 2024), multi-agent collaboration in MAC-SQL (Wang et al., 2025a), retrieval-augmented synthesis in CHESS (Talaei et al., 2024), schema routing in DBCopilot (Wang et al., 2025b), and logical operator plans in AgenticData (Sun et al., 2025). However, Text-to-SQL synthesizes a single query whose execution result *is* the final answer, whereas we generate a faithful, discourse-level summary for which SQL execution serves only as intermediate grounding. Moreover, TaSoF is an executor-agnostic representation of table states and operation flow rather than a SQL dialect, coordinating multi-step, multi-table reasoning via a dependency DAG that conditions summary generation. Advances in Text-to-SQL are therefore complementary, as stronger step-to-SQL translation can be directly plugged into our executor.

Planning for Table Understanding. LLM-based planning has been widely explored for table understanding tasks, including table summarization (Zhao et al., 2023; Sui et al., 2024; Zhang et al., 2024a), question answering (Pasupat and Liang, 2015; Nan et al., 2022), and fact verification (Chen et al., 2020). Existing approaches either decompose a complex query into subqueries that are addressed sequentially (Ye et al., 2023; Zhao et al., 2024) or follow predefined workflows that iteratively select and execute operations until a stop condition is met (Wang et al., 2024b; Zhang et al., 2024c; Li et al., 2025; Mao et al., 2024). These methods predominantly rely on natural language (NL) plans, whose linguistic ambiguity can hinder consistent execution. Beyond the table domain, PGPO (Cao et al., 2025) shows that pseudocode-style plans benefit LLM agents, yet its plans remain semi-structured text improved via preference optimization rather than a schema-constrained representation with explicit dependencies. To our knowledge, TaSoF is the first such plan representation evaluated specifically for query-focused table summarization in both single- and multi-table settings.

7 Conclusion

Query-focused table summarization aims to create concise, query-specific summaries from tabular data. While recent advancements use LLMs for multi-step reasoning, NL reasoning plans can leave operations and dependencies underspecified, complicating their translation into executable pro-

grams. To address this, we propose TaSoF, a schema-constrained, dependency-aware plan representation, and SPaGe, an agentic framework that integrates it with graph-based execution and summary generation. Across the two evaluated benchmarks, SPaGe outperforms the compared baselines, and its plans achieve higher per-step execution success than the evaluated NL formats. Broader claims about model robustness, semantic plan correctness, and real-world latency require further evaluation.

Limitations

We address ambiguity in natural-language planning through a schema-constrained representation, but the present evaluation has several limitations: 1) All systems use one closed-source backbone (`gpt-4o-mini`). The planner, executor, and summary generator are architecturally model-agnostic, but this design property is not evidence of robustness across backbones. Sensitivity to alternative and open-weight models remains untested. 2) We evaluate two benchmarks rather than larger databases or noisy real-world settings. Robustness to noisy schemas, missing values, ambiguous queries, schema perturbations, and multi-level or nested headers is unknown. Future work could also combine Python- and SQL-based execution for table structures that are poorly matched to SQL. 3) Execution cycles measure DAG dependency depth, not end-to-end efficiency. We do not measure wall-clock latency, total LLM calls, retry overhead, end-to-end token use, monetary cost, API rate limits, or database overheads. 4) Automatic metrics primarily capture surface-level similarity. Our human evaluation assesses output correctness and faithfulness, but it was conducted by three authors rather than independent annotators. We also lack gold reasoning traces, semantic plan error labels, and an analysis of how incorrect intermediate steps affect summaries. 5) The top-k analysis varies only k on the studied benchmarks. Generalization across query types and schema complexity, including cases where key evidence lies outside the retained top rows, remains unexplored.

Ethical Considerations

The initial draft was written by the authors, and AI tools were used primarily to correct grammatical errors. We have designed this work to minimize ethical risks while considering potential societal impacts. All datasets are publicly available, and the tabular inputs come from open databases and Wikipedia. To the best of our knowledge, they contain no personally identifiable information. We primarily use closed-source LLMs from the GPT family. Generated summaries are manually checked for factual accuracy and for biased, discriminatory, or harmful content. A possible risk is the generation and execution of SQL queries as intermediate steps, which could lead to incorrect outputs or inefficiencies. Although significant negative impacts are unlikely, we remain aware of possible misuse and advocate for transparency, reproducibility, and human oversight in high-stakes applications.

References

Satanjeev Banerjee and Alon Lavie. 2005. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pages 65–72.

Tom B Brown, Benjamin Mann, et al. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*.

Zouying Cao, Runze Wang, Yifei Yang, Xinbei Ma, Xiaoyong Zhu, Bo Zheng, and Hai Zhao. 2025. PGPO: Enhancing agent reasoning via pseudocode-style planning guided preference optimization. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 14966–14985, Vienna, Austria. Association for Computational Linguistics.

Wenhu Chen, Hongmin Wang, et al. 2020. TabFact: A large-scale dataset for table-based fact verification. In *ICLR*.

Yuyan Chen, Qingpei Guo, Shuangjie You, and Zhixu Li. 2025. MedTransTab: Advancing medical cross-table tabular data generation. In *Proceedings of the Eighteenth ACM International Conference on Web Search and Data Mining, WSDM 2025, Hannover, Germany, March 10-14, 2025*, pages 764–772.

Zhoujun Cheng, Tianbao Xie, Peng Shi, Chengzu Li, Rahul Nadkarni, Yushi Hu, Caiming Xiong, Dragomir Radev, Mari Ostendorf, Luke Zettlemoyer, Noah A. Smith, and Tao Yu. 2023. Binding language models in symbolic languages. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Mark d'Inverno, Michael Fisher, Alessio Lomuscio, Michael Luck, Maarten de Rijke, Mark Ryan, and Michael J. Wooldridge. 1997. Formalisms for multi-agent systems. *Knowledge Engineering Review*, 12(3):315–321.

Ben Eyal, Moran Mahabi, Ophir Haroche, Amir Bachar, and Michael Elhadad. 2023. Semantic decomposition of question and SQL for text-to-SQL parsing. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 13629–13645.

Joseph L Fleiss. 1971. Measuring nominal scale agreement among many raters. *Psychological bulletin*, 76(5):378.

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2024. Text-to-SQL empowered by large language models: A benchmark evaluation. *Proceedings of the VLDB Endowment*, 17(5):1132–1145.

Pablo Gruer, Vincent Hilaire, Abder Koukam, and Krzysztof Cetnarowicz. 2002. A formal framework for multi-agent systems analysis and design. *Expert Systems With Applications*, 23(4):349–355.

Vincent Hilaire, Abder Koukam, Pablo Gruer, and Jean-Pierre Müller. 2000. Formal specification and prototyping of multi-agent systems. In *Engineering Societies in the Agent World, First International Workshop, ESAW 2000, Berlin, Germany, August 21, 2000, Revised Papers*, volume 1972 of *Lecture Notes in Computer Science*, pages 114–127.

Zhengbao Jiang, Yi Mao, Pengcheng He, Graham Neubig, and Weizhu Chen. 2022. OmniTab: Pretraining with natural and synthetic data for few-shot table-based question answering. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 932–942. Association for Computational Linguistics.

J. Richard Landis and Gary G. Koch. 1977. The measurement of observer agreement for categorical data. *Biometrics*, 33(1):159–174.

Qianlong Li, Chen Huang, Shuai Li, Yuanxin Xiang, Deng Xiong, and Wenqiang Lei. 2025. GraphOTTER: Evolving LLM-based Graph Reasoning for Complex Table Question Answering. In *Proceedings of the 31st International Conference on Computational Linguistics (COLING)*, pages 5486–5506.

Chin-Yew Lin and Eduard Hovy. 2003. Automatic evaluation of summaries using N-gram co-occurrence statistics. In *NAACL*, pages 150–157.

Ao Liu, Haoyu Dong, Naoaki Okazaki, Shi Han, and Dongmei Zhang. 2022a. PLOG: Table-to-logic pretraining for logical table-to-text generation. In *EMNLP*.

Qian Liu, Bei Chen, Jiaqi Guo, Morteza Ziyadi, Zeqi Lin, Weizhu Chen, and Jian-Guang Lou. 2022b. TAPEX: Table pre-training via learning a neural SQL executor. In *ICLR*.

Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. 2023. G-Eval: NLG evaluation using GPT-4 with better human alignment. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Qingyang Mao, Qi Liu, Zhi Li, Mingyue Cheng, Zheng Zhang, and Rui Li. 2024. PoTable: Towards systematic thinking via plan-then-execute stage reasoning on tables. *Preprint*, arXiv:2412.04272.

Linyong Nan, Chiachun Hsieh, et al. 2022. FeTaQA: Free-form table question answering. *Transactions of the Association for Computational Linguistics*, 10:35–49.

Giang Nguyen, Ivan Brugere, Shubham Sharma, Sanjay Kariyappa, Anh Totti Nguyen, and Freddy Lecue. 2025. Interpretable LLM-based Table Question Answering. *Transactions on Machine Learning Research*.

Vaishali Pal, Andrew Yates, Evangelos Kanoulas, and Maarten de Rijke. 2023. MultiTabQA: Generating tabular answers for multi-table question answering. In *ACL*, pages 6322–6334.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: A method for automatic evaluation of machine translation. In *ACL*, pages 311–318.

Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables. In *ACL*, pages 1470–1480.

Hwanjun Song, Hang Su, Igor Shalyminov, Jason Cai, and Saab Mansour. 2024. FineSurE: Fine-grained summarization evaluation using LLMs. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*.

Yuan Sui, Mengyu Zhou, Mingjie Zhou, Shi Han, and Dongmei Zhang. 2024. Table meets LLM: Can large language models understand structured table data? A benchmark and empirical study. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining, WSDM 2024, Merida, Mexico, March 4-8, 2024*, pages 645–654.

Ji Sun, Guoliang Li, Peiyao Zhou, Yihui Ma, Jingzhe Xu, and Yuan Li. 2025. AgenticData: An agentic data analytics system for heterogeneous data. *Preprint*, arXiv:2508.05002.

Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. 2024. CHESS: Contextual harnessing for efficient SQL synthesis. *Preprint*, arXiv:2405.16755.

Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Linzheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, and Zhoujun Li. 2025a. MAC-SQL: A multi-agent collaborative framework for text-to-SQL. In *Proceedings of the 31st International Conference on Computational Linguistics (COLING)*.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. 2023. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2609–2634.

Tianshu Wang, Xiaoyang Chen, Hongyu Lin, Xianpei Han, Le Sun, Hao Wang, and Zhenyu Zeng. 2025b. DBCopilot: Natural language querying over massive databases via schema routing. In *Proceedings of the 28th International Conference on Extending Database Technology (EDBT)*.

Yuqi Wang, Lyuhao Chen, Songcheng Cai, Zhijian Xu, and Yilun Zhao. 2024a. Revisiting automated evaluation for long-form table question answering. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 14696–14706.

Zilong Wang, Hao Zhang, Chun-Liang Li, Julian Martin Eisenschlos, Vincent Perot, Zifeng Wang, Lesly Miculicich, Yasuhisa Fujii, Jingbo Shang, Chen-Yu Lee, and Tomas Pfister. 2024b. Chain-of-table: Evolving tables in the reasoning chain for table understanding. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*.

Tomer Wolfson, Daniel Deutch, and Jonathan Berant. 2022. Weakly supervised text-to-SQL parsing through question decomposition. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pages 2528–2542.

Yunhu Ye, Binyuan Hui, Min Yang, Binhua Li, Fei Huang, and Yongbin Li. 2023. Large language models are versatile decomposers: Decomposing evidence and questions for table-based reasoning. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2023, Taipei, Taiwan, July 23-27, 2023*, pages 174–184. ACM.

Weijia Zhang, Vaishali Pal, Jia-Hong Huang, Evangelos Kanoulas, and Maarten de Rijke. 2024a. QFMTS: Generating query-focused summaries over multi-table inputs. In *Proceedings of the 27th European Conference on Artificial Intelligence (ECAI)*, pages 3875–3882.

Yunjia Zhang, Jordan Henkel, Avrilia Floratou, Joyce Cahoon, Shaleen Deep, and Jignesh M. Patel. 2024b. ReAcTable: Enhancing ReAct for table question answering. *Proc. VLDB Endow.*, 17(8):1981–1994.

Zhehao Zhang, Yan Gao, and Jian-Guang Lou. 2024c. E^5: Zero-shot hierarchical table analysis using augmented LLMs via explain, extract, execute, exhibit and extrapolate. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 1244–1258.

Yilun Zhao, Lyuhao Chen, Arman Cohan, and Chen Zhao. 2024. TaPERA: Enhancing faithfulness and interpretability in long-form table QA by content planning and execution-based reasoning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12824–12840.

Yilun Zhao, Zhenting Qi, Linyong Nan, Boyu Mi, Yixin Liu, Weijin Zou, Simeng Han, Ruizhe Chen, Xiangru Tang, Yumo Xu, Dragomir Radev, and Arman Cohan. 2023. QTSumm: Query-focused summarization over tabular data. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1157–1172.

A Details on Methodology

A.1 Definitions of Table Operations

Definitions of the nine operations used in TaSoF are provided in Table 7.

Operation	Functionality
Scan	Read a base table with optional predicate and projection.
Aggregate	Compute grouped or global aggregate expressions.
Filter	Remove non-matching tuples from an intermediate stream.
Sort	Sort a stream by an expression.
TopSort	Select the top-k tuples.
Join	Perform a logical join between two streams.
Except	Compute set difference.
Intersect	Compute set intersection.
Union	Compute set union.

Table 7: Definitions of the nine operations used in the structured planning steps.

B Details on Experimental Setup

B.1 Details on Baselines

We compare our approach against two categories of baselines: prompting-based and program-augmented models.

Prompt-based Models. These models perform table reasoning by relying only on the internal reasoning capabilities of LLMs.

- **CoT** (Wei et al., 2022) uses chain-of-thought (CoT) prompting to guide LLMs in generating a summary from the query and relational tables.
- **ReFactor** (Zhao et al., 2023) adapts heuristic methods to extract query-relevant facts from the given tables. It then concatenates these facts with the query as context for summary generation by an LLM.
- **DirectSumm** (Zhang et al., 2024a) addresses the task in a single phase by jointly extracting query-relevant facts from tables and synthesizing a comprehensive query-focused summary from those facts.
- **Reason-then-Summ** (Zhang et al., 2024a) addresses the task in two sequential phases: extracting query-relevant facts from tables and synthesizing a comprehensive query-focused summary based on the retrieved facts.

Program-Augmented Models. Unlike prompt-based models, program-augmented models use the code-generation capabilities of LLMs to produce SQL queries, execute them with an external database engine, and use the results to support reasoning.

- **Binder** (Cheng et al., 2023) converts the given NL query into either a Python program or a SQL query and executes it to obtain the final answer. In our experiments, we use the version of Binder that generates SQL queries for a fair comparison.
- **Dater** (Ye et al., 2023) decomposes a given query into subqueries, converts each subquery into a SQL query, and aggregates the execution results into a final summary.
- **TaPERA** (Zhao et al., 2024) generates natural language plans through query decomposition and uses an executable Python program to obtain the intermediate answer for each planning subquery. It then aggregates all intermediate query-answer pairs into a final summary.

B.2 Further Implementation Details

For the planner, executor, and summary generator in our framework, we set the temperature, top-p, and maximum output tokens to 0.1, 0.95, and 400, respectively. We prepend the input prompt with three manually selected in-context demonstrations to enable effective in-context learning (Brown et al., 2020). For all baseline models, we use the same `gpt-4o-mini` backbone as our framework and follow their original papers to configure the remaining settings, such as prompts and hyperparameters.

B.3 Details on Evaluation Protocols

Automated Evaluation. Following Zhao et al. (2024), we evaluate the quality of the generated summaries against reference summaries. We consider the following metrics:

- **BLEU** (Papineni et al., 2002) is a precision-oriented metric that evaluates n-gram overlap between generated and reference summaries. We use SacreBLEU to compute BLEU scores;
- **ROUGE** (Lin and Hovy, 2003) measures recall-oriented word overlap between generated and reference summaries. We report the F1 variant of ROUGE-L;

- **METEOR** (Banerjee and Lavie, 2005) measures unigram matches between generated and reference summaries.

We report the mean performance over three independent runs for all automated evaluation metrics.

Human Evaluation. We conduct human evaluation to assess the quality of generated summaries. Following prior work, we evaluate each summary along three dimensions:

- **Correctness:** The summary is consistent with the facts in the final execution table.
- **Faithfulness:** All statements in the summary are factually supported by the original input tables.
- **Coverage:** The summary addresses all information needs expressed in the user query and reflects all relevant facts from the execution table.

Following Zhao et al. (2023), each dimension is rated on a 5-point Likert scale ranging from 1 (worst) to 5 (best). For each model, we randomly sample 100 test examples from the evaluated dataset. Three annotators, all of whom are authors of the paper and proficient in SQL and English, carry out the annotations. They independently assess each summary with access to the corresponding SQL query, input tables, and execution tables. Each summary receives three annotations per dimension, and we report the average score across annotators.

To assess the consistency of the annotations, we measure inter-annotator agreement using Fleiss' kappa (Fleiss, 1971) across the three annotators for each dimension, obtaining $\kappa = 0.78$ for correctness, $\kappa = 0.63$ for faithfulness, and $\kappa = 0.71$ for coverage. All three values fall within the 0.61–0.80 range, indicating *substantial agreement* according to the interpretation of Landis and Koch (1977). This agreement reflects consistency among the author annotators; it does not establish annotator independence or eliminate potential author bias.

C Detailed Prompts

To facilitate reproducibility, we provide detailed prompts for our proposed framework. The planner prompt is shown in Figure C.1, and the executor prompt used for step-to-SQL generation is shown in Figure C.2.

Figure C.1: Planner Prompt for Structured Planning

Goal

Produce a **structured plan** expressed as a JSON document that conforms to the following Pydantic models:

```
class Step(BaseModel):
   id:  int                                                                    # 1, 2, 3, ...
   op:  str                                                  # "Scan", "Join", "Aggregate", ...
   src:  List[str]                          # table names or step IDs (prefix step IDs with "step")
   pred:  Optional[str] = None                                              # WHERE / ON clause
   out:  List[str]                                                            # projection list

class Plan(BaseModel):
   steps:  List[Step]                                # ordered list; execution order = list order
```

Instruction

Given
1) a natural-language *question*, and
2) a schema specified as a list of `<table>` specifications in the following format:

```
<table>:  <comma-separated list of columns>
   col:  <column header 1> | <column header 2> | ...| <column header n>
   row 1:  <value 1,1> | <value 1,2> | ...| <value 1,n>
   row 2:  <value 2,1> | <value 2,2> | ...| <value 2,n>
   ...
   row m:  <value m,1> | <value m,2> | ...| <value m,n>
```

Input

Demonstrations: ⟨*few-shot demonstrations here*⟩
Query: ⟨*input query here*⟩
Tables: ⟨*input tables here*⟩

Figure C.2: Executor Prompt for Step-to-SQL Generation

Goal

Produce a SQL query based on the given structured step. A plan contains a sequence of steps, and each step specifies one operation. All operations take either tables or previous results as input. The structure of a step is as follows:

```
class Step(BaseModel):
   id:  int                                                                    # 1, 2, 3, ...
   op:  str                                                  # "Scan", "Join", "Aggregate", ...
   src:  List[str]                          # table names or step IDs (prefix step IDs with "step")
   pred:  Optional[str] = None                                              # WHERE / ON clause
   out:  List[str]                                                            # projection list
```

Instruction

Let's think step by step to convert a structured step to a SQL query. Constraints for the SQL query:
(1) The SQL query MUST be compatible and executable by `pandasql`.
(2) Always perform case-insensitive string comparisons using the `LOWER()` function.

Input

Demonstrations: ⟨*few-shot demonstrations here*⟩
Step: ⟨*input structured step here*⟩