EASE: Entity-Aware Sub-table Generation for Real-world Multi-table QA

Myunghoon Kang[1], **Dahyun Jung**[1*], **Suhyune Son**[1*], **Seonmin Koo**[1*], **Changwoo Chun**[1,3], **Daniel Rim**[3], **Haeyoung Kwon**[4], **Yuna Hur**[2†], **Heuiseok Lim**[1,2†]
[1]Korea University, [2]Human-inspired AI Research
[3]Hyundai Motor Company, [4]Hyundai AutoEver
[1,2]{chaos8527, dhaabb55, ssh5131, fhdahd, yj72722, limhseok}@korea.ac.kr
[3]{cwchun, drim}@hyundai.com, [4]kwon@hyundai-autoever.com

Abstract

Recent advancements in table-based question answering (table QA) have been driven by the development of table-specific reasoning strategies for leveraging large language models. Previous works employ sub-table-based reasoning, which involves matching query-relevant table values and aggregating them into sub-tables for precise reasoning. However, these approaches are limited to scenarios with query-relevant single tables, failing to handle real-world table QA settings that involve noisy multi-table sets. To address the challenges of real-world table QA, we propose **EASE**: **E**ntity-**A**ware **S**ub-table Generation for R**E**al-world Multi-table QA framework. Given a noisy multi-table set, EASE first extracts key entities from the question to construct a sub-table schema. It then populates this schema by utilizing a selected set of column values from the noisy multi-table set, thereby facilitating efficient and effective sub-table-based reasoning. We introduce a Noisy Multi-table QA dataset and conduct extensive experiments to evaluate EASE's effectiveness on real-world table QA. Our results demonstrate that EASE effectively filters out irrelevant information while incorporating pertinent table values, leading to efficient and effective performance on real-world table QA. Our dataset can be found https://github.com/Metalchaos8527/ease_noisy_multi-table_qa.git

1 Introduction

Tables provide a structured representation of data by organizing information into rows and columns, making them a valuable resource for solving diverse real-world problems (Zhu et al., 2021; Moosavi et al., 2021; Hernandez et al., 2022). Consequently, table-based reasoning necessitates systems not only to retrieve relevant values but also to aggregate and process them effectively. Table-based question answering (table QA) addresses this challenge by enabling systems to process, interpret, and manipulate structured information to generate accurate answers. Specifically, the system is required to (1) understand the table's structural schema, and (2) comprehend the semantic meaning of its values (Cheng et al., 2023; Min et al., 2024; Fang et al., 2024).

To fulfill these requirements, previous works employ specialized table QA training strategy (Herzig et al., 2020; Liu et al., 2022; Jiang et al., 2022; Gu et al., 2022) for enhancing the structural comprehension capabilities of transformer-based models (Vaswani et al., 2017; Devlin et al., 2019; Lewis et al., 2020). Meanwhile, recent studies have leveraged the emergent abilities of large language models (LLMs) to improve reasoning capabilities in table QA (Wei et al., 2022a; Yang et al., 2024). These approaches involve sub-table-based reasoning, where models extract and aggregate relevant table values from a single-table, thereby enhancing the accuracy and interpretability of table-based reasoning (Cheng et al., 2023; Ye et al., 2023; Wang et al., 2024b). While these methods demonstrate efficacy in single-table settings, they do not consider the complexities of real-world table QA.

In real-world table QA, the system needs to reference multiple tables to generate accurate answers (Pal et al., 2023). Additionally, the multi-table input may contain noisy tables unrelated to answering the question (Maamari et al., 2024; Wang et al., 2025). Consequently, a real-world table QA system is required to filter out irrelevant information and effectively aggregate and process multiple tables to generate an accurate answer for complex questions. However, existing approaches do not address the complexities of real-world table QA, leading to erroneous reasoning. As shown in Figure 1(a) and Figure 1(b), existing approaches generate incorrect answers due to their reliance on single-table

[*]These authors contributed equally to this work.
[†]Corresponding authors

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 277–302

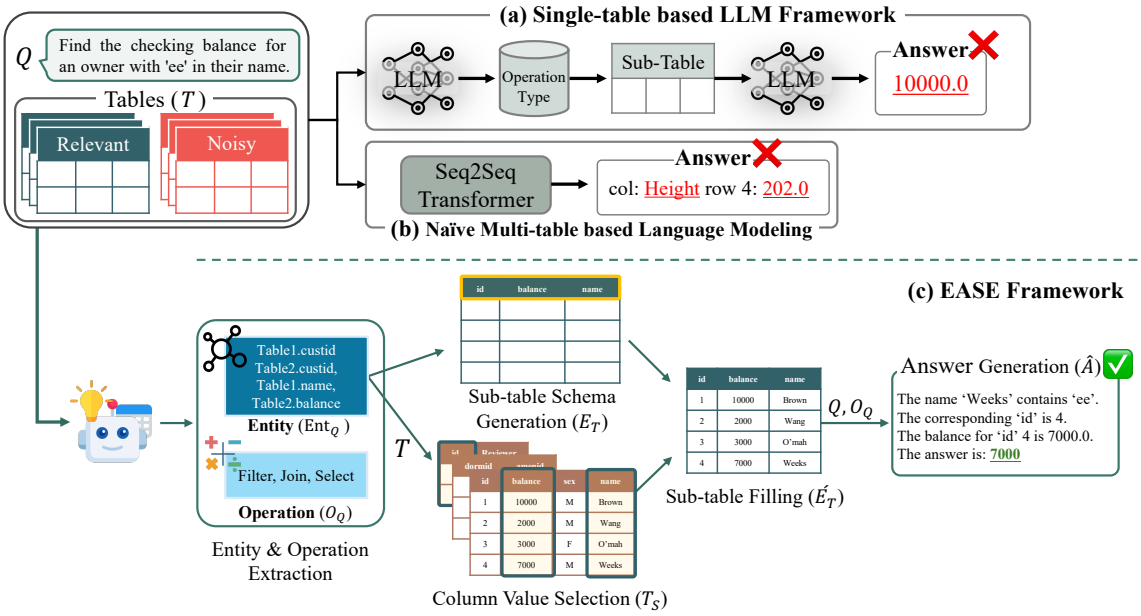

Figure 1: Illustration of the comparison between (a) Single-table-based LLM Framework, (b) Naïve Multi-table-based Language Modeling and (c) EASE Framework in real-world table QA. Given a noisy input table set T, (a) fails to generate precise answers due to its lack of multi-table-based reasoning capabilities. Even (b) refers wrong value in noisy table despite its multi-table specific training. In contrast, (c) EASE generates correct answers through entity-aware sub-table generation, successfully filtering out noisy tables and aggregating relevant table values from the noisy input table set.

operations and oversight for noisy multi-table settings. Therefore, it is crucial to develop systems capable of not only processing multiple tables but also identifying and filtering out noisy tables to generate the accurate answer.

To address the complexities of real-world table QA, we propose **EASE**: **E**ntity-**A**ware **S**ub-table Generation for **RE**al-world Multi-table QA. EASE is an LLM-based framework that consists of five modules, by extracting query-aware entities from the tables and subsequently creating and filling the table schema. As illustrated in Figure 1(c), given a question and a noisy input table set, EASE first extracts entities and operations to distinguish relevant table values from the noisy input table set. Subsequently, EASE utilizes the extracted entities to select the relevant column value set from the noisy input table set. Finally, EASE constructs a sub-table schema and fills its cell based on the selected column value set, ultimately generating the final answer through the required operations.

To validate EASE's effectiveness, we construct a Noisy Multi-table QA dataset by injecting noisy tables into the MultiTabQA dataset (Pal et al., 2023) based on realistic settings. We conduct extensive experiments on the Noisy Multi-table QA dataset to compare EASE with general reasoning and table-specific reasoning methods. We summarize our contributions as follows:

- We introduce the Noisy Multi-table QA dataset to reflect real-world table QA settings.
- We propose the EASE framework, which constructs the entity-aware sub-table to aggregate relevant table values and generate an accurate answer among noisy input table sets.
- We demonstrate the effectiveness and efficiency of the EASE framework through comprehensive experiments on the Noisy Multi-table QA dataset.

2 Related Work

2.1 Leveraging Language Model for Table QA

Existing works on table QA are categorized into training-based and prompting-based approaches. TaPas (Herzig et al., 2020) extends BERT (Devlin et al., 2019) architecture to jointly train on textual and tabular inputs, enabling effective table-based QA on transformer-based models. TableLLaMA (Zhang et al., 2024) focuses on handling diverse table-based tasks through instruction-tuning on LLMs. The prompting-based approaches mitigate the limitations of training-based methods,

such as lack of interpretability and robustness to diverse table types, through reasoning with LLMs. Dater (Ye et al., 2023) introduces evidence and question decomposition to process large tables and complex queries efficiently. Chain-of-Table (Wang et al., 2024b) extends the standard Chain-of-Thought (Wei et al., 2022b) to table-based reasoning, enabling LLMs to iteratively generate operations and update the sub-tables. Although these approaches advance table-based reasoning without additional training, they rely on single-table setting and fall short in realistic settings that require multi-table information.

2.2 Multi-table QA

While existing table QA systems are limited to single-table settings, real-world table QA often necessitates complex processing and reasoning across multiple tables to answer questions. MultiTabQA (Pal et al., 2023) addresses this challenge by proposing a dataset and training methods designed to handle multiple tables. Wu et al. (2025) introduce the MMQA benchmark to tackle multi-hop reasoning capability in multi-table question answering. Additionally, Wang et al. (2024a) present a preprocessing strategy for removing irrelevant columns to overcome input sequence length limitations and numerical computation constraints in multi-table QA. However, these studies leverage direct end-to-end methods, resulting in a lack of intermediate reasoning steps.

2.3 Ensuring Reliability in Table QA

Recent works have introduced several approaches to guarantee robust and consistent structural comprehension in Table QA. ITR (Lin et al., 2023) seeks to address the sequence length limitations, where key information is often lost due to truncation caused by length constraints, by extracting only relevant subsets of tables. To enhance the robustness of table structural comprehension, Liu et al. (2024) propose a normalization strategy to mitigate performance degradation under table perturbation when utilizing LLMs for table QA. RoBUT (Zhao et al., 2023) evaluates the robustness of existing table QA systems by leveraging human-annotated adversarial perturbations. Despite efforts to build reliable and robust table QA systems, existing approaches have solely focused on settings involving structural noise. However, they have largely overlooked situations where a table, although contextually similar, is irrelevant to the question.

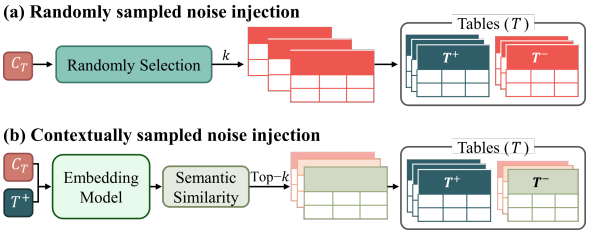

Figure 2: Illustration of the process of constructing a Noisy Multi-table QA dataset through noise injection under two settings: (a) randomly sampled noise injection and (b) contextually sampled noise injection.

3 Noisy Multi-table QA Dataset

In this section, we describe the problem formulation and construction process of the Noisy Multi-table QA dataset. To reflect the complexities of real-world table QA settings, we introduce the Noisy Multi-table QA dataset, which is constructed by injecting noisy tables unrelated to the input question.

3.1 Problem Formulation

In real-world table QA, each instance comprises a set of tables T, a natural language question Q, and its corresponding ground truth answer A. The set of tables T is composed of relevant tables T^+ and noisy tables T^-, formalized as $T = T^+ \cup T^-$, where $T^+ = \{t_1^+, \ldots, t_i^+\}_{i=1}^{m-k}$ and $T^- = \{t_1^-, \ldots, t_j^-\}_{j=1}^{k}$. Here, m denotes the total number of input tables, and k represents the number of noisy tables that are irrelevant to generate the answer. Given the question Q and noisy input table set T, the system should generate a precise answer by effectively handling the noise within the data.

3.2 Dataset Construction

Motivation Since we aim to evaluate the system's ability to generate the answer in the presence of a noisy input table set, the input table set is constructed to include both query-relevant and irrelevant tables. We utilize the MultiTabQA dataset (Pal et al., 2023) as a source dataset for constructing the Noisy Multi-table QA dataset. Each instance in the source dataset comprises a natural language question paired with a set of query-relevant input tables. Consequently, the system is required to perform multi-table processing to generate the answer. However, since the MultiTabQA dataset only provides the necessary input tables for each query, it does not fully reflect real-world settings where the model is required to distinguish relevant informa-

tion from noise. Therefore, we inject noisy tables into the dataset to reflect a more challenging and realistic setting of real-world table QA.

Candidate Table Pool Construction To inject noisy tables into the MultiTabQA dataset, we formulate the overall dataset construction as a search problem. We construct a candidate table pool C_T for retrieving noisy tables. The C_T encompasses all tables from the dataset. We eliminate duplicate tables that share identical table names and column names. Additionally, we exclude any tables present in T^+ for each instance.

Noise Injection Following the construction of the candidate table pool C_T, we sample the noisy table T^- in two distinct settings: (1) **randomly sampled noise injection** and (2) **contextually sampled noise injection**. Figure 2 provides an overview of each noise injection setting. In the **randomly sampled noise injection** setting, we randomly retrieve k noisy tables from C_T. This setting is designed to simulate real-world table QA where irrelevant data may inadvertently be included. In contrast, the **contextually sampled noise injection** setting reflects a more realistic scenario where contextually similar yet irrelevant tables exist. We inject contextual noise through semantic similarity to inject noisy tables. Specifically, we calculate similarity scores by using embeddings of column names for each pairwise combination between a relevant table instance t_i^+ and the tables in the candidate pool C_T. We utilize a pre-trained text embedder[1], to compute embeddings and calculate cosine similarity scores. The tables with the top-k highest similarity scores are then selected as the noisy tables T^-. With the Noisy Multi-table QA dataset, we can evaluate the system's ability to generate accurate answers under real-world table QA settings.

4 EASE Framework

To address the complexities of real-world table QA, we propose the EASE framework. As illustrated in Algorithm 1, EASE comprises five modules that primarily leverage the advanced reasoning capability of LLMs. First, EASE extracts key entities and operations (line 2) necessary for answering the question. Next, it initializes a sub-table schema as a structural foundation (line 3), which is subsequently filled with contextually relevant selected column value set from a noisy input table set (line 4 and line 5). Finally, EASE synthesizes the filled sub-table and extracted operations to generate a step-by-step reasoning process for answer generation (line 6). We now detail each module of EASE. All prompts and examples used in EASE are provided in Appendix A.3.

Algorithm 1: EASE Framework

Input: T: Noisy input table set, $\acute{T}$: Compact subset of T with top-3 rows retained per table, Q: Natural language question,
LM: LLM-based module, SM: Symbolic language module
Output: $\hat{A}$ is the predicted answer to the question.

1 **Function** EASE(T, $\acute{T}$, Q):
2 $\quad \mathrm{Ent}_Q, O_Q \leftarrow \mathtt{LM}_{\mathtt{Extract}}(\acute{T}, Q)$
$\quad\quad \triangleright$ Extract key entities Ent_Q and operations O_Q
3 $\quad E_T \leftarrow \mathtt{LM}_{\mathtt{SchemaGen}}(\mathrm{Ent}_Q)$
$\quad\quad \triangleright$ Generate sub-table schema E_T for subsequent reasoning
4 $\quad T_S \leftarrow \mathtt{SM}_{\mathtt{ColSelect}}(\mathrm{Ent}_Q, T)$
$\quad\quad \triangleright$ Select relevant column value set T_S from T
5 $\quad \acute{E_T} \leftarrow \mathtt{LM}_{\mathtt{SchemaFill}}(E_T, T_S)$
$\quad\quad \triangleright$ Fill the cell of E_T from T_S
6 $\quad \hat{A} \leftarrow \mathtt{LM}_{\mathtt{AnswerGen}}(Q, \acute{E_T}, O_Q)$
$\quad\quad \triangleright$ Generate answer of Q based on $\acute{E_T}$ and O_Q
7 **return** $\hat{A}$

Entity & Operation Extraction In real-world table QA, the key challenge lies in identifying relevant data from a noisy input table set. To address this, EASE identifies key information and required operations by capturing the relevance between the question and tables. Specifically, to ensure efficiency, $\mathtt{LM}_{\mathtt{Extract}}$ leverages an LLM to extract key entities Ent_Q and operations O_Q from the question Q and a compact subset $\acute{T}$. Formally, we define this subset as $\acute{T} = \{\mathrm{head}(t, 3) | t \in T\}$, where $\mathrm{head}(t, k)$ is an operator that extracts the top k rows of a table t (line 2). This method enables EASE to distinguish pertinent values and plan required operations while minimizing the initial computational load.

Sub-table Schema Generation In the Sub-table Schema Generation module, we construct a table schema to serve as a structural foundation for sub-table-based reasoning. Given Ent_Q, $\mathtt{LM}_{\mathtt{SchemaGen}}$ utilizes LLMs to generate a markdown-style tabular format, where each column name corresponds to an entity from Ent_Q (line 3). By generating E_T, we guide an LLM for subsequent sub-table filling and answer generation.

Column Value Selection Filling cells in sub-table schema E_T necessitates obtaining relevant table values. Therefore, we employ the symbolic language Python (using the pandas package) to select these values for filling cells in E_T. Given

[1]We use E5-large-v2 (Wang et al., 2022) as the text embedding model.

Ent_Q and T, $\texttt{SM}_{\texttt{ColSelect}}$ outputs column value set T_S by selecting corresponding column value from T based on Ent_Q (line 4). This results in a smaller yet concise set T_S that preserves pertinent information for efficient reasoning.

Sub-table Filling Once the sub-table schema E_T and column value set T_S are established, EASE enriches the content of E_T in Sub-table Filling. In detail, $\texttt{LM}_{\texttt{SchemaFill}}$ fills the cell of E_T by aggregating values from T_S, generating filled sub-table $\acute{E}_T$ (line 5). This process ensures structural coherence while providing the necessary information for accurate answer generation.

Answer Generation Finally, EASE employs table-based reasoning to address the complex question. With the query-relevant sub-table $\acute{E}_T$ prepared, $\texttt{LM}_{\texttt{AnswerGen}}$ generates step-by-step reasoning guided by the required operations O_Q, ultimately producing the prediction $\hat{A}$.

5 Experimental Setup

Datasets We utilize the Noisy Multi-table QA dataset for the experiments. From the MultiTabQA dataset (Pal et al., 2023), we extract samples under the 75th percentile of table length while excluding instances with excessively long tables. We also convert the format of the ground truth answer from a tabular format to free-form using GPT-4o (Hurst et al., 2024), thereby enabling generation-based evaluation and ensuring the interpretability of the systems' output. For candidate table pool construction and noise injection, we set $m = 4$ and $k \in \{1, 2, 3\}$, reflecting the varying number of noisy tables across instances. The Noisy Multi-table QA dataset comprises a total of 600 samples. Detailed dataset statistics are presented in Table 6.

Baseline Methods We compare EASE with LLM-based reasoning methods such as (1) Generic Reasoning and (2) Table-Specific Reasoning.

(1) Generic Reasoning. These methods represent generic reasoning prompting applicable in task-agnostic scenarios. The Zero-Shot method prompts LLMs to directly answer the given question Q using the whole noisy input table set T. Similarly, the Few-Shot method provides several examples (Q, T, A) to guide LLMs in generating an answer. Meanwhile, Chain-of-Thought prompting (Wei et al., 2022b) instructs LLMs to produce a step-by-step reasoning chain before deriving the final answer.

(2) Table-specific Reasoning. These methods are specifically developed for table-based reasoning with LLMs, encompassing training and prompting-based approaches. TableLLaMA (Zhang et al., 2024) is an instruction-tuned model specialized for handling table-based tasks. MultiTabQA (Pal et al., 2023) is a TAPEX (Liu et al., 2022)-based model fine-tuned on the MultiTabQA dataset for standard multi-table QA. Chain-of-Table (Wang et al., 2024b) is a prompting-based method that instructs the LLMs to find table manipulation operations and corresponding arguments for each round of reasoning. Throughout the reasoning process, the Chain-of-Table executes these operations and iteratively generates sub-tables until the endpoint is reached.

We use GPT-3.5 (turbo-0125)[2], GPT-4o (2024-08-06) (Hurst et al., 2024) and Claude-3.5 (haiku-20241022)[3] as the backbone LLMs for both generic reasoning and table-specific reasoning methods, with the exception of TableLLaMA and MultiTabQA. We use the public checkpoint of TableLLaMA[4] based on the LLaMA-2-7b (Touvron et al., 2023) and MultiTabQA[5] based on BART (Lewis et al., 2020). The implementation details including generation hyperparameters and prompt templates are provided in Appendix A.2 and Appendix A.3.

Evaluation Metrics We adopt two generation-based evaluation metrics for real-world table QA: the ChrF score (Popović, 2015) and Substring Exact Match (EM) score. ChrF score is a character-level evaluation metric that calculates F1-score based on the precision and recall of N-grams between the generated output and the ground truth. Substring EM score evaluates whether the ground truth appears as an exact substring within the model-generated answer, providing a straightforward measure of response quality.

In addition, we propose a Table-F1 score to better assess the factuality of the model-generated answer by checking the alignment between table values generated in the prediction and the corresponding ground truth. This metric first filters out any irrelevant substrings in the generated output that do not originate from the table values. Subsequently, it computes the F1-score based on the remaining table-relevant substrings of the model-

[2]https://openai.com/index/chatgpt/
[3]https://www.anthropic.com/claude/haiku
[4]https://github.com/OSU-NLP-Group/TableLlama
[5]https://github.com/kolk/MultiTabQA

Methods	GPT-3.5			Claude-3.5			GPT-4o		
	ChrF	**Substring EM**	**Table-F1**	**ChrF**	**Substring EM**	**Table-F1**	**ChrF**	**Substring EM**	**Table-F1**
Generic Reasoning									
Zero-Shot	**44.41**	31.10	43.28	43.21	37.87	49.07	54.77	43.15	57.20
Few-Shot	41.71	37.53	43.09	37.25	52.89	51.20	56.38	55.12	57.80
Chain-of-Thought	12.62	11.87	14.85	**54.16**	42.87	**58.50**	60.12	48.33	64.73
Table-specific Reasoning									
TableLLaMA†	-	-	-	18.41	12.28	17.20	-	-	-
Chain-of-Table	32.37	**44.40**	43.79	14.41	43.56	38.48	35.54	39.90	39.76
EASE	30.15	43.28	**44.85**	38.27	**55.81**	57.27	**63.80**	**66.02**	**69.65**

Table 1: Experimental results on a randomly sampled noise injection setting. Note that TableLLaMA† is a table task-specific fine-tuned model, where the backbone architecture is fixed and cannot be altered.

Methods	GPT-3.5			Claude-3.5			GPT-4o		
	ChrF	**Substring EM**	**Table-F1**	**ChrF**	**Substring EM**	**Table-F1**	**ChrF**	**Substring EM**	**Table-F1**
Generic Reasoning									
Zero-Shot	**42.97**	29.87	42.38	43.34	39.20	49.68	52.87	41.64	55.86
Few-Shot	40.33	35.86	41.45	35.87	58.40	53.03	54.20	53.97	56.54
Chain-of-Thought	12.93	13.97	15.22	**53.26**	42.43	58.68	56.52	44.85	60.66
Table-specific Reasoning									
TableLLaMA†	-	-	-	18.00	15.32	20.26	-	-	-
Chain-of-Table	32.01	39.58	40.93	14.65	46.17	40.82	32.80	48.06	46.83
EASE	29.63	**41.67**	**43.05**	48.29	**58.53**	**61.92**	**59.43**	**62.06**	**65.68**

Table 2: Experimental results on a contextually sampled noise injection setting. Note that TableLLaMA† is a table task-specific fine-tuned model, where the backbone architecture is fixed and cannot be altered.

generated answer and the ground truth, leading to fine-grained evaluation for table-based factuality of the model-generated answer. Given a generated answer $\hat{A}$, we first extract the generated table values $\hat{A}_{\text{table}}$ by filtering out any table irrelevant substrings. Table-F1 is then computed based on the precision and recall of $\hat{A}_{\text{table}}$ with respect to the ground truth A. Precision, recall, and Table-F1 score are defined as follows:

$$\text{Table-Precision} = \frac{|\hat{A}_{\text{table}} \cap A|}{|\hat{A}_{\text{table}}|} \quad (1)$$

$$\text{Table-Recall} = \frac{|\hat{A}_{\text{table}} \cap A|}{|A|} \quad (2)$$

$$\text{Table-F1} = \frac{2 \times \text{Table-Precision} \times \text{Table-Recall}}{\text{Table-Precision} + \text{Table-Recall}} \quad (3)$$

A high Table-F1 score indicates that the generated table values closely match the ground truth, A, while a lower score suggests either missing or extraneous information in $\hat{A}_{\text{table}}$.

6 Main Results

In this section, we present comparative experimental results of EASE and baselines. We also provide additional analysis on the efficiency of EASE and robustness of EASE with varying numbers of noisy tables. Further supplementary comparisons with sub-table-based table QA baselines and additional analyses on EASE are presented in Appendix C.

6.1 Effectiveness of EASE Framework

We compare EASE with both generic and table-specific reasoning methods on the Noisy Multi-table QA dataset. The results for randomly sampled and contextually sampled noise injection settings are presented in Table 1 and Table 2, respectively. EASE consistently achieves the highest Substring EM and competitive Table-F1 scores except for Claude-3.5 variants, demonstrating its effectiveness in filtering noise and extracting relevant information. Notably, with GPT-4o, EASE achieves the best performance in both settings. EASE also performs competitively with Claude-3.5, demonstrating strong generalization capabilities. In particular, when using GPT-3.5 and Claude-3.5 as the backbone, EASE shows comparable performance with baselines in Substring EM and Table-F1. These results highlight EASE's ability to extract and validate relevant information from noisy tables more effectively.

Despite being a general framework without task-specific fine-tuning, EASE outperforms TableLLaMA, a table-specific fine-tuned model, across both noise settings. Meanwhile, the Chain-of-Thought method shows the worst performance

Methods	ChrF	Substring EM	Table-F1
Conventional setting			
MultiTabQA	34.12	50.35	51.90
CoT*	58.04	47.46	63.40
Chain-of-Table*	30.18	48.39	44.27
EASE*	**61.82**	**62.55**	**66.73**
Randomly sampled noise injection			
MultiTabQA	29.57	41.96	43.44
CoT*	60.12	48.33	64.73
Chain-of-Table*	35.54	39.90	39.76
EASE*	**63.80**	**66.02**	**69.65**
Contextually sampled noise injection			
MultiTabQA	29.30	41.08	42.49
CoT*	56.52	44.85	60.66
Chain-of-Table*	32.80	48.06	46.83
EASE*	**59.43**	**62.06**	**65.68**

Table 3: Performance comparison of MultiTabQA and EASE under the conventional MultiTabQA setting and the proposed noisy multi-table QA settings. The * indicates the result of LLM-based reasoning methods with GPT-4o as the backbone LLM.

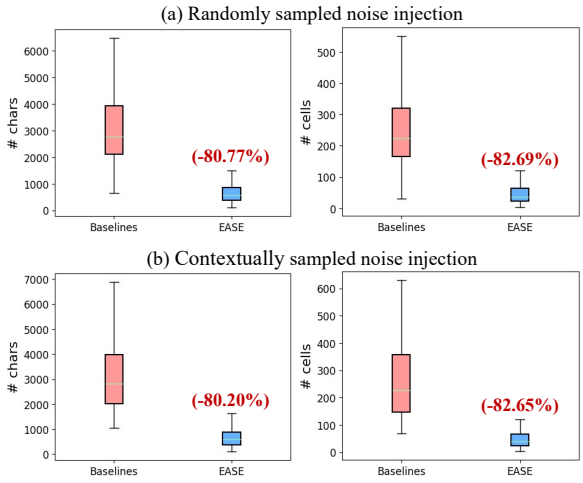

Figure 3: Box plots illustrating the input table length distribution between the baselines and EASE. The term "# chars" represents the number of characters, and "# cells" denotes the number of cells in the input tables. Baselines encompass all the generic reasoning and table-specific reasoning methods, excluding EASE.

among other baselines when using GPT-3.5 as the backbone. This is attributed to the GPT-3.5's input length constraint, leading to incomplete and erroneous answer generation. In contrast, EASE achieves the highest Table-F1 score in both settings when using GPT-3.5 as the backbone compared to all baselines, demonstrating the effectiveness of its column value selection across various LLMs.

To further examine whether task-specific multi-table QA training alone is sufficient for noisy multi-table QA, we compare EASE with MultiTabQA (Pal et al., 2023), a model tailored for standard multi-table QA, under both conventional and noisy settings. As shown in Table 3, MultiTabQA drops from 51.90 Table-F1 on the conventional MultiTabQA dataset to 43.44 and 42.49 under randomly and contextually sampled noise injection, respectively. In contrast, EASE maintains strong performance across settings, achieving 66.73 on the conventional settings and 69.65 and 65.68 under noisy settings. These results suggest that reasoning in noisy multi-table QA depends not merely on multi-table QA-tailored training, but on employing appropriate methods for explicitly filtering irrelevant information and effectively aggregating pertinent values from noisy multi-table inputs.

While EASE shows robust performance compared to baselines, we observe an inconsistent performance of EASE on the ChrF metric. Since ChrF evaluates the overlap between the reference answer and the prediction based on a character-level n-gram F1-score, the longer predictions generated by EASE due to its step-by-step reasoning tend to be penalized, resulting in lower ChrF scores compared to other baselines. In contrast, Substring EM and Table-F1 directly assess whether the gold answer appears in the generated answer, making them more reliable for factual evaluation. EASE's superior performance on these metrics highlights its strong factual accuracy and answer relevance in real-world table QA tasks.

6.2 Efficiency of EASE Framework

We evaluate the efficiency of EASE by comparing input table lengths with baseline methods, including generic and table-specific reasoning approaches. As shown in Figure 3, EASE significantly reduces both the number of characters (# chars) and the number of cell values (# cells) compared to baselines. This reduction is attributed to EASE's sub-table generation method, which selectively fills the sub-table schema with only relevant table values through its Column Value Selection module. EASE minimizes input size by filtering out irrelevant information from noisy input tables while maintaining reasoning quality. These findings indicate that EASE not only optimizes input length but also reduces computational overhead, enhancing its scalability for real-world table QA applications.

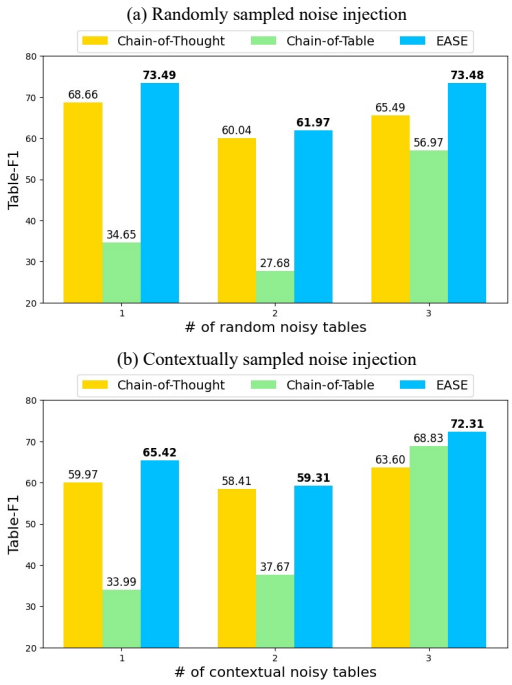

Figure 4: Experimental results by number of tables for random and contextual noisy tables.

6.3 Robustness of EASE under Different Numbers of Noisy Tables

We evaluate the robustness of EASE by comparing its performance against Chain-of-Thought and Chain-of-Table across various noisy table ratios. Figure 4 illustrates the Table-F1 scores of each method using GPT-4o as the backbone. Overall, EASE outperforms both Chain-of-Thought and Chain-of-Table across all settings, regardless of the number of noisy tables. These results demonstrate that EASE effectively filters out irrelevant information while preserving essential content for reasoning, ensuring robust performance in real-world table QA settings. Notably, when generating an answer from a noisy input table set comprising three noisy tables and one relevant table, EASE outperforms the baselines in all settings. By leveraging Entity & Operation Extraction, EASE effectively discards irrelevant values and focuses on generating answers based on pertinent table content. In cases where the table set includes two noisy and two relevant tables, all methods experience performance degradation. This result arises from the equal contribution of noisy and relevant information, which places a burden on the system to balance filtering out irrelevant data with aggregating the pertinent values. In contrast, EASE still outperforms the baselines by constructing a sub-table schema that utilizes concise and relevant column values.

Methods	ChrF	Substring EM	Table-F1
Randomly sampled noise injection			
EASE	**63.80**	**66.02**	**69.65**
w/o $\texttt{LM}_{\texttt{Extract}}$	48.51	51.32	54.59
w/o $\texttt{LM}_{\texttt{SchemaGen}}$	52.16	63.17	68.17
w/o $\texttt{SM}_{\texttt{ColSelect}}$	61.66	61.47	64.56
Contextually sampled noise injection			
EASE	**59.43**	**62.06**	**65.68**
w/o $\texttt{LM}_{\texttt{Extract}}$	47.83	48.55	51.91
w/o $\texttt{LM}_{\texttt{SchemaGen}}$	49.96	59.28	64.92
w/o $\texttt{SM}_{\texttt{ColSelect}}$	59.06	58.10	62.59

Table 4: Results of ablating the Entity & Operation Extraction module (*w/o* $\texttt{LM}_{\texttt{Extract}}$), Sub-table Schema Generation (*w/o* $\texttt{LM}_{\texttt{SchemaGen}}$) and Column Value Selection (*w/o* $\texttt{SM}_{\texttt{ColSelect}}$).

7 Ablation Study

To assess the impact of each module in EASE on real-world table QA, we conduct an ablation study by discarding the Entity & Operation Extraction module (*w/o* $\texttt{LM}_{\texttt{Extract}}$), Sub-table Schema Generation (*w/o* $\texttt{LM}_{\texttt{SchemaGen}}$) and Column Value Selection (*w/o* $\texttt{SM}_{\texttt{ColSelect}}$). The results are shown in Table 4. Since $\texttt{LM}_{\texttt{Extract}}$ is crucial for identifying relevant table values among noisy multi-tables, its absence leads to a significant performance decline, with an average decrease of 14.415 in Table-F1 score. When $\texttt{LM}_{\texttt{SchemaGen}}$ is ablated, the performance also degrades, indicating that constructing a structural foundation followed by contextual sub-table filling is more effective than employing a generic sub-table generation approach. Lastly, the absence of $\texttt{SM}_{\texttt{ColSelect}}$ leads to both inefficient and ineffective real-world table QA performance. These findings highlight that $\texttt{SM}_{\texttt{ColSelect}}$ improves performance by filtering out irrelevant tables, thereby providing a more concise source for subsequent Sub-table Filling. Overall, the core modules in EASE contribute to effective table-based reasoning.

8 Case Study

We qualitatively verify the reasoning capability of EASE using two examples from the Noisy Multi-table QA dataset. Table 5 presents the responses generated by EASE alongside those from Chain-of-Thought and Chain-of-Table. The results demonstrate that in noisy multi-table QA settings, EASE effectively extracts relevant table information in a

QUESTION 1	Find the total balance across checking accounts.
Input Tables	{'CHECKING': {'columns': ['custid', 'balance'], 'index': [0, 1, 2, 3, 4, 5, 6], 'data': [[1, 10000.0], [2, 2000.0], [3, 3000.0], [4, 7000.0], [5, 10000.0], [6, 77.0], [7, 7.0]], 'table_name': 'CHECKING'}, 'Tryout': {'columns': ['pID', 'cName', 'pPos', 'decision'], 'index': [0, 1, 2, 3, 4, 5], 'data': [[10001, 'LSU', 'goalie', 'no'], [10001, 'ASU', 'goalie', 'yes'], [20002, 'FSU', 'striker', 'yes'], ...
Gold Answer	32084.0
Chain-of-Thought	Thus, the total balance across checking accounts is 32107.0. ✗
Chain-of-Table	Therefore, the answer is: 32184.0 ✗
EASE	The total balance across checking accounts is 32,084.0. ✓
QUESTION 2	**For each position, what is the minimum time students spent practicing?**
Input Tables	{'Player': {'columns': ['pID', 'pName', 'yCard', 'HS'], 'index': [0, 1, 2, 3, 4, 5], 'data': [[10001, 'Andrew', 'no', 1200], [20002, 'Blake', 'no', 1600], [30003, 'Charles', 'no', 300], ..., 'College': {'columns': ['cName', 'state', 'enr'], 'index': [0, 1, 2, 3], 'data': [['LSU', 'LA', 18000], ['ASU', 'AZ', 12000], ['OU', 'OK', 22000], ['FSU', 'FL', 19000]], 'table_name': 'College'}, 'Projects': {'columns': ...
Gold Answer	1200 for goalie, 300 for mid, 1600 for striker.
Chain-of-Thought	Therefore, the answer to the question, based on the provided data, is that the information is not available. ✗
Chain-of-Table	therefore, the answer is: [Empty Cell] - 89, goalie - 98, mid - 789, striker - 256 ✗
EASE	For each position, the minimum time students spent practicing is: - Goalie: 1200 - Striker: 1600 - Mid: 300 ✓

Table 5: Comparison of generation results for each method in Noisy Multi-table QA dataset. The red components are the noisy table in the input table. Detailed reasoning steps for EASE are presented in Appendix B.

step-by-step manner, leveraging its structured reasoning framework. As seen in Question 1, EASE exhibits strong numerical reasoning capabilities in complex arithmetic scenarios that require structural awareness of tables. Additionally, for Question 2, which necessitates integrating multi-table information, EASE accurately extracts and utilizes the relevant data to generate correct answers. In contrast, Chain-of-Thought and Chain-of-Table struggle with table-based reasoning, failing to recognize key information required for answering questions.

9 Error Analysis

To understand the structural bottlenecks of EASE, we analyze failure cases under noisy multi-table QA settings. As shown in Figure 5, the failure patterns vary depending on the type of injected noise. Across both noise injections, the most prevalent error type is Answer Generation Error. These errors typically stem from two distinct sources: semantic errors, such as misinterpreting the user intent or table schema, and arithmetic/logical errors, including miscalculating averages or ignoring top-k constraints during the final generation step.

Under contextual noise, EASE specifically struggles with Entity Extraction, frequently over-extracting entities from noise tables. Since this phenomenon is absent under random noise injection, it is directly attributed to the contextual similarity of column and table names between the ground truth schemas and the sampled noisy tables. Furthermore, EASE exhibits structural Table Filling Errors across both settings. These errors generally occur when the model fails to enumerate all matching values for complex joins or neglects underlying join constraints.

In conclusion, the error analysis reveals remaining challenges in sub-table filling and answer generation across both settings, with contextual noise further complicating entity extraction. Addressing these limitations necessitates more granular, phase-aware noise filtration and dynamic sub-table-based reasoning strategies in future research.

10 Conclusion

In this paper, we tackled real-world table QA by introducing the Noisy Multi-table QA dataset and novel EASE framework. The Noisy Multi-table QA dataset is constructed by injecting both randomly and contextually sampled noise, thereby reflecting the real-world table QA settings in which the system should reference multiple tables to generate accurate answers. To address the complexities of real-world table QA, we have proposed an EASE framework, which constructs entity-aware sub-table schema and then fills its cell with a selected column value set. Our extensive experiments showcase EASE's outstanding performance, demonstrating effective sub-table-based reasoning and efficient sub-table filling by column value selection. This entity-aware sub-table generation design sheds light on the effective utilization of LLMs' table-based reasoning capabilities in real-world table QA settings.

Limitations

Here, we outline two key limitations: (1) the restricted scope of our work and (2) The inference cost of proprietary LLMs. **(1) The restricted scope of our work.** We focus on real-world table QA, where answering a question requires reasoning over both relevant and noisy tables. To implement this setting, we conduct experiments on the Noisy Multi-table QA dataset, built from MultiTabQA (Pal et al., 2023) with injected noisy tables. Although this setup is appropriate for evaluating EASE under noisy multi-table conditions, our experiments remain limited to this benchmark construction. For that reason, we do not expect EASE to work transparently on other types of tables that are different from the MultiTabQA dataset, such as a spreadsheet or database format. Moreover, focusing on real-world table QA, we did not evaluate our approach on conventional table QA datasets such as WikiTQ (Pasupat and Liang, 2015) and FeTaQA (Nan et al., 2022).

(2) The inference cost of proprietary LLMs. To address real-world table QA, we primarily rely on proprietary LLMs to leverage state-of-the-art reasoning capabilities. This leads to high inference costs, compared to the open-source LLMs such as LLaMA (Dubey et al., 2024) or Mixtral (Jiang et al., 2024). We conducted preliminary experiments on utilizing instruction-tuned LLaMA-3.1-8B[6] on both generic and table-specific reasoning methods. However, LLaMA-3.1-8B fails to generate an answer due to limited table understanding capability. We believe future work could extend the approaches for enhancing table-based reasoning capabilities of open-source LLMs in real-world tables.

Ethics Statement

In this section, we discuss the ethical considerations of our work.

(1) Privacy and Languages. The Noisy Multi-table QA dataset was constructed upon the MultiTabQA dataset, which encompasses Spider (Yu et al., 2018), GeoQuery (Zelle and Mooney, 1996) and Atis (Dahl et al., 1994) in English. The Spider dataset consists of web-crawled data annotated with human-generated SQL queries. GeoQuery addresses geographical question answering using Geobase data (e.g., identifying the location of a specific city), while ATIS encompasses scenarios where users request flight information through automated airline travel inquiry systems. Since each dataset is based on human annotations or domain-specific table-based question answering, there is minimal risk of including personal information or offensive content. **(2) Licenses and terms.** The source dataset, MultiTabQA, is developed under the MIT license. Similarly, TableLLaMA (Zhang et al., 2024), which we use as a baseline, is also distributed under the MIT license. We primarily employ LangChain[7], available under the MIT license, for implementing EASE. **(3) Intended use.** Our proposed Noisy Multi-table QA datasets and EASE framework are intended to be used on table-based reasoning on real-world table QA. This dataset can serve as a basis for generating synthetic training data for multi-table QA, as it emphasizes generating accurate answers under noisy input table sets.

Acknowledgements

This work was supported by Hyundai Motor Company and Kia. This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education(NRF-2021R1A6A1A03045425). This work was supported by the Commercialization Promotion Agency for R&D Outcomes(COMPA) grant funded by the Korea government(Ministry of Science and ICT)(2710086166). This work was supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIT) (RS-2024-00398115, Research on the reliability and coherence of outcomes produced by Generative AI). This work was supported by Institute for Information & communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) (No. RS-2022-II220369, (Part 4) Development of AI Technology to support Expert Decision-making that can Explain the Reasons/Grounds for Judgment Results based on Expert Knowledge).

References

Nikhil Abhyankar, Vivek Gupta, Dan Roth, and Chandan K. Reddy. 2025. H-STAR: LLM-driven hybrid SQL-text adaptive reasoning on tables. In *Proceedings of the 2025 Conference of the Nations of the*

[6] meta-llama/Llama-3.1-8B-Instruct

[7] https://python.langchain.com/

Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), pages 8841–8863, Albuquerque, New Mexico. Association for Computational Linguistics.

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*.

Deborah A Dahl, Madeleine Bates, Michael K Brown, William M Fisher, Kate Hunicke-Smith, David S Pallett, Christine Pao, Alexander Rudnicky, and Elizabeth Shriberg. 1994. Expanding the scope of the atis task: The atis-3 corpus. In *Human Language Technology: Proceedings of a Workshop held at Plainsboro, New Jersey, March 8-11, 1994*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Xi Fang, Weijie Xu, Fiona Anting Tan, Ziqing Hu, Jiani Zhang, Yanjun Qi, Srinivasan H. Sengamedu, and Christos Faloutsos. 2024. Large language models (LLMs) on tabular data: Prediction, generation, and understanding - a survey. *Transactions on Machine Learning Research*.

Zihui Gu, Ju Fan, Nan Tang, Preslav Nakov, Xiaoman Zhao, and Xiaoyong Du. 2022. PASTA: Table-operations aware fact verification via sentence-table cloze pre-training. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 4971–4983, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Mikel Hernandez, Gorka Epelde, Ane Alberdi, Rodrigo Cilla, and Debbie Rankin. 2022. Synthetic data generation for tabular health records: A systematic review. *Neurocomput.*, 493(C):28–45.

Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Eisenschlos. 2020. TaPas: Weakly supervised table parsing via pre-training. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4320–4333, Online. Association for Computational Linguistics.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.

Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. 2024. Mixtral of experts. *arXiv preprint arXiv:2401.04088*.

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, Seattle, United States. Association for Computational Linguistics.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online. Association for Computational Linguistics.

Weizhe Lin, Rexhina Blloshmi, Bill Byrne, Adria de Gispert, and Gonzalo Iglesias. 2023. An inner table retriever for robust table question answering. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9909–9926, Toronto, Canada. Association for Computational Linguistics.

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

Tianyang Liu, Fei Wang, and Muhao Chen. 2024. Rethinking tabular data understanding with large language models. 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 450–482, Mexico City, Mexico. Association for Computational Linguistics.

Karime Maamari, Fadhil Abubaker, Daniel Jaroslawicz, and Amine Mhedhbi. 2024. The death of schema linking? text-to-SQL in the age of well-reasoned language models. In *NeurIPS 2024 Third Table Representation Learning Workshop*.

Dehai Min, Nan Hu, Rihui Jin, Nuo Lin, Jiaoyan Chen, Yongrui Chen, Yu Li, Guilin Qi, Yun Li, Nijun Li, and Qianren Wang. 2024. Exploring the impact of table-to-text methods on augmenting LLM-based question

answering with domain hybrid data. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 6: Industry Track)*, pages 464–482, Mexico City, Mexico. Association for Computational Linguistics.

Nafise Sadat Moosavi, Andreas Rücklé, Dan Roth, and Iryna Gurevych. 2021. Scigen: a dataset for reasoning-aware text generation from scientific tables. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

Linyong Nan, Chiachun Hsieh, Ziming Mao, Xi Victoria Lin, Neha Verma, Rui Zhang, Wojciech Kryściński, Hailey Schoelkopf, Riley Kong, Xiangru Tang, Mutethia Mutuma, Ben Rosand, Isabel Trindade, Renusree Bandaru, Jacob Cunningham, Caiming Xiong, Dragomir Radev, and Dragomir Radev. 2022. FeTaQA: Free-form table question answering. *Transactions of the Association for Computational Linguistics*, 10:35–49.

Vaishali Pal, Andrew Yates, Evangelos Kanoulas, and Maarten de Rijke. 2023. MultiTabQA: Generating tabular answers for multi-table question answering. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6322–6334, Toronto, Canada. Association for Computational Linguistics.

Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1470–1480, Beijing, China. Association for Computational Linguistics.

Maja Popović. 2015. chrF: character n-gram F-score for automatic MT evaluation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 392–395, Lisbon, Portugal. Association for Computational Linguistics.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Bing Wang, Chunhao Wang, Xingpeng Zhang, Chunlan Zhao, Xiaoling Yang, and Kuan Guo. 2024a. Multitable question answering method based on correlation evaluation and precomputed cube. In *International Conference on Knowledge Science, Engineering and Management*, pages 393–405. Springer.

Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. 2022. Text embeddings by weakly-supervised contrastive pre-training. *arXiv preprint arXiv:2212.03533*.

Yihan Wang, Peiyu Liu, and Xin Yang. 2025. Linkalign: Scalable schema linking for real-world large-scale multi-database text-to-sql. *arXiv preprint arXiv:2503.18596*.

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*.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. 2022a. Emergent abilities of large language models. *Transactions on Machine Learning Research*. Survey Certification.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. 2022b. Chain of thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*.

Jian Wu, Linyi Yang, Dongyuan Li, Yuliang Ji, Manabu Okumura, and Yue Zhang. 2025. MMQA: Evaluating LLMs with multi-table multi-hop complex questions. In *The Thirteenth International Conference on Learning Representations*.

Jingfeng Yang, Hongye Jin, Ruixiang Tang, Xiaotian Han, Qizhang Feng, Haoming Jiang, Shaochen Zhong, Bing Yin, and Xia Hu. 2024. Harnessing the power of llms in practice: A survey on chatgpt and beyond. *ACM Trans. Knowl. Discov. Data*, 18(6).

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 '23, page 174–184, New York, NY, USA. Association for Computing Machinery.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921, Brussels, Belgium. Association for Computational Linguistics.

John M Zelle and Raymond J Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the thirteenth national conference on Artificial intelligence-Volume 2*, pages 1050–1055.

Tianshu Zhang, Xiang Yue, Yifei Li, and Huan Sun. 2024. TableLlama: Towards open large generalist models for tables. 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 6024–6044, Mexico City, Mexico. Association for Computational Linguistics.

Yilun Zhao, Chen Zhao, Linyong Nan, Zhenting Qi, Wenlin Zhang, Xiangru Tang, Boyu Mi, and Dragomir Radev. 2023. RobuT: A systematic study of table QA robustness against human-annotated adversarial perturbations. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6064–6081, Toronto, Canada. Association for Computational Linguistics.

Fengbin Zhu, Wenqiang Lei, Youcheng Huang, Chao Wang, Shuo Zhang, Jiancheng Lv, Fuli Feng, and Tat-Seng Chua. 2021. TAT-QA: A question answering benchmark on a hybrid of tabular and textual content in finance. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3277–3287, Online. Association for Computational Linguistics.

A Detailed Experimental Setup

A.1 Data Statistics

We construct the Noisy Multi-table QA dataset by applying two noise injection methodologies to the MultiTabQA (Pal et al., 2023) dataset. MultiTabQA dataset is developed under MIT license. The samples are randomly selected for the dataset construction. Table 6 shows statistics of the Noisy Multi-table QA dataset.

	# of noisy tables		
	1	2	3
Randomly sampled noise injection			
# of examples	100	100	100
Min. table length	23.00	40.00	72.00
Max. table length	3,799.00	11,045.00	11,775.00
Avg. table length	745.73	909.36	1,298.20
Min. noisy table length	32.00	41.00	72.00
Max. noisy table length	3,799.00	11,045.00	11,775.00
Avg. noisy table length	795.27	1,078.98	968.88
Contextually sampled noise injection			
# of examples	100	100	100
Min. table length	23.00	40.00	72.00
Max. table length	11,107.00	3,737.00	11,775.00
Avg. table length	843.62	811.47	1,298.20
Min. noisy table length	67.00	49.00	72.00
Max. noisy table length	11,107.00	3,107.00	11,775.00
Avg. noisy table length	1,731.84	756.66	1,133.57

Table 6: Statistics of the Noisy Multi-table QA dataset.

A.2 Implementation Details

We provide the implementation details, including supplementary experimental setups, inference hyperparameters for each module in EASE and baselines.

Supplementary Experimental Setups For the experiments, we implement a generic reasoning baseline and EASE using the langchain package. For Chain-of-Table (Wang et al., 2024b), we employ its public code base[8], and adopt the WikiTQ (Pasupat and Liang, 2015) version prompt for the DynamicPlan module. Additionally, we implement TableLLaMA (Zhang et al., 2024)[9] and MultiTabQA (Pal et al., 2023)[10] using their public codebases and checkpoints, and conduct evaluations on a single NVIDIA RTX A6000 48GB GPU.

[8]https://github.com/google-research/chain-of-table
[9]https://github.com/OSU-NLP-Group/TableLlama
[10]https://github.com/kolk/MultiTabQA

Module	temperature	top_p	output_tokens	examples
GPT-3.5				
$\text{LM}_{\texttt{Extract}}$	0.3	1.0	4096	6
$\text{LM}_{\texttt{SchemaGen}}$	0.3	1.0	4096	0
$\text{LM}_{\texttt{SchemaFill}}$	0.3	1.0	4096	0
$\text{LM}_{\texttt{AnswerGen}}$	0.3	1.0	4096	0
Claude-3.5				
$\text{LM}_{\texttt{Extract}}$	0.3	1.0	8192	6
$\text{LM}_{\texttt{SchemaGen}}$	0.3	1.0	8192	0
$\text{LM}_{\texttt{SchemaFill}}$	0.3	1.0	8192	0
$\text{LM}_{\texttt{AnswerGen}}$	0.3	1.0	8192	0
GPT-4o				
$\text{LM}_{\texttt{Extract}}$	0.3	1.0	16383	6
$\text{LM}_{\texttt{SchemaGen}}$	0.3	1.0	16383	0
$\text{LM}_{\texttt{SchemaFill}}$	0.3	1.0	16383	0
$\text{LM}_{\texttt{AnswerGen}}$	0.3	1.0	16383	0

Table 7: Generation hyperparameters for EASE

Module	temperature	top_p	output_tokens	examples
GPT-3.5				
Zero-Shot	0	1.0	4096	0
Few-Shot	0	1.0	4096	3
Chain-of-Thought	0	1.0	4096	0
Claude-3.5				
Zero-Shot	0	1.0	8192	0
Few-Shot	0	1.0	8192	3
Chain-of-Thought	0	1.0	8192	0
GPT-4o				
Zero-Shot	0	1.0	16383	0
Few-Shot	0	1.0	16383	3
Chain-of-Thought	0	1.0	16383	0

Table 8: Generation hyperparameters for generic reasoning baselines

Generation Hyperparameters The generation hyperparameters for EASE is shown in Table 7. For the generic reasoning baselines (Table 8), we set hyperparmeters based on the preliminary experiments shown in Table 9 and Table 10. For Chain-of-Table, we adhere to the original configuration: $\texttt{temperature} \in \{0, 1\}$, $\texttt{top\_p} = 1$, and $\texttt{decode\_steps} = 200$. For TableLLaMA we set $\texttt{temperature} = 0.6$, $\texttt{top\_p} = 0.9$, and $\texttt{max\_gen\_len} = 512$.

A.3 Prompt Template

We present the prompt templates and shot examples for each module in EASE and generic reasoning baselines.

EASE Framework

- Entity & Operation Extraction: See Table 20 for prompt template and Table 21 shot examples.

Methods	ChrF	Substring EM	Table-F1
zero-shot	54.77	43.15	57.20
⌞ temp 0.3	55.14	42.66	57.31
few-shot	56.38	55.12	57.80
⌞ 6-shot, temp 0.3	56.54	54.65	57.48
CoT	60.12	48.33	64.73
⌞ temp 0.3	58.46	46.22	62.98

Table 9: Preliminary experimental results for selecting generation hyperparameters for generic reasoning baselines under randomly sampled noise injection setting.

Methods	ChrF	Substring EM	Table-F1
zero-shot	52.87	41.64	55.86
⌞ temp 0.3	53.88	42.46	56.39
few-shot	54.20	53.97	56.54
⌞ 6-shot, temp 0.3	56.11	53.76	56.29
CoT	56.52	44.85	60.66
⌞ temp 0.3	57.23	45.28	61.76

Table 10: Preliminary experimental results for selecting generation hyperparameters for generic reasoning baselines under contextually sampled noise injection setting.

- Sub-table Schema Generation: See Table 22.
- Sub-table Filling: See Table 23.
- Answer Generation: See Table 24.

Generic Reasoning Baselines

- Zero-shot & Few-shot: See Table 25 for prompt template and Table 26 shot examples for Few-shot.
- Chain-of-Thought: See Table 27.

B Outputs of Each Module in EASE

The outputs of each module in the EASE for the given question and tables are presented in Table 28 and Table 29.

C Additional Analysis

We present additional analysis to validate EASE's generalizability, effectiveness, and robustness.

C.1 Supplementary Comparisons with Sub-table-based Table QA Baselines

To evaluate the effectiveness of EASE's multi-stage sub-table generation, we conduct comparative experiments with DATER (Ye et al., 2023) and H-STAR (Abhyankar et al., 2025), baselines that also aim to filter irrelevant table content. While these methods seek to extract pertinent information from the input table set, they differ fundamentally in their utilization of the resulting sub-table. DATER employs an extractive approach, prompting an LLM to look up the full table and select indices of relevant rows and columns. This sub-table then serves as 'sub-evidence' for a final reasoner that aggregates results from multiple SQL queries. Similarly, H-STAR constructs a sub-table through a multi-view extraction process, and generates the final answer with the generated sub-table. In contrast, EASE generates a consolidated sub-table, containing the query-relevant partial information, that serves as the primary context directly grounding the final reasoning process. Although H-STAR utilizes the sub-table as a main source, it employs additional SQL executions for final answer generation, leading to computational overhead. In the following, we present a comparative empirical analysis of EASE against DATER and H-STAR.

Methods	Backbone	ChrF	Substring EM	Table-F1
Randomly sampled noise injection				
CoT	GPT-3.5	12.62	11.87	14.85
Chain-of-Table	GPT-3.5	**32.37**	**44.40**	43.79
DATER	GPT-3.5	31.69	29.32	30.09
DATER	GPT-4o	9.85	8.05	7.93
EASE	GPT-3.5	30.15	43.28	**44.85**
Contextually sampled noise injection				
CoT	GPT-3.5	12.93	13.97	15.22
Chain-of-Table	GPT-3.5	32.01	39.58	40.93
DATER	GPT-3.5	**33.05**	29.83	31.40
DATER	GPT-4o	9.64	7.22	7.64
EASE	GPT-3.5	29.63	**41.67**	**43.05**

Table 11: Experimental results of EASE and DATER under the Noisy Multi-table QA dataset with GPT-3.5 as the default backbone LLM. For DATER, we additionally report results using GPT-4o.

Comparison with DATER We conduct comparative experiments by implementing DATER and EASE with GPT-3.5 as the backbone LLM. As shown in Table 11, EASE consistently outperforms both DATER and Chain-of-Table across both noise settings. We attribute this performance gap to a critical limitation in DATER's design: its failure to perform multi-table aggregation during sub-table generation. By merely selecting relevant table content, DATER produces a fragmented collection of evidence, forcing the subsequent SQL generation step to reason over disjointed information. This leads to erroneous queries that cannot capture complex inter-table relationships. In addition, DATER performs even worse when implemented with GPT-4o, indicating that it does not consistently benefit from a stronger backbone LLM and generalizes less

Methods	ChrF	Substring EM	Table-F1
Randomly sampled noise injection			
CoT*	60.12	48.33	64.73
Chain-of-Table*	35.54	39.90	39.76
H-STAR	9.56	6.07	7.75
EASE*	**63.80**	**66.02**	**69.65**
Contextually sampled noise injection			
CoT*	56.52	44.85	60.66
Chain-of-Table*	32.80	48.06	46.83
H-STAR	9.67	4.72	6.54
EASE*	**59.43**	**62.06**	**65.68**

Table 12: Experimental results of EASE and H-STAR under the Noisy Multi-table QA dataset with GPT-4o as the backbone LLM.

effectively to noisy multi-table QA settings. Conversely, EASE's sub-table generation utilizes both extractive and generative manner by first extracting candidate subset, then generating sub-table by aggregating those pertinent values, thereby serving key information for the reasoning in Noisy Multi-table QA settings. Overall, these results support the effectiveness of EASE's consolidated sub-table generation for robust reasoning under noisy multi-table settings.

Comparison with H-STAR We additionally compare EASE with H-STAR (Abhyankar et al., 2025) to examine how it transfers to our noisy multi-table QA setting. As shown in Table 12, H-STAR performs substantially worse than all other baselines under both randomly and contextually sampled noise injection, achieving Table-F1 scores of 7.75 and 6.54, respectively, while EASE attains 69.65 and 65.68. These results suggest that H-STAR does not transfer effectively to noisy multi-table QA settings, where the model must distinguish relevant evidence from noisy tables and consolidate information across multiple tables. In contrast, EASE explicitly performs query-aware filtering and multi-table aggregation before final answer generation, which enables more robust reasoning under noisy multi-table inputs.

C.2 Informativeness of the sub-table generated by EASE

As demonstrated in the ablation study (Section 7), eliminating the sub-table generation degrades the performance of EASE for real-world table QA. To further assess the importance of the sub-table generation for real-world table QA, we conduct an additional experiment. We extract the EASE generated sub-table for each case then utilize them as inputs for TableLLaMA (Zhang et al., 2024), a model fine-tuned for single-table QA task. The experimental results, presented in Table 13, reveal that utilizing EASE-generated sub-tables as input for TableLLaMA leads to performance improvements across all metrics and settings. This suggests that the sub-table encapsulate relevant and aggregated table information essential for question answering, highlighting the rich informativeness of EASE-generated sub-table. In conclusion, our findings provide further empirical evidence for the effectiveness of sub-table generation as a crucial intermediate representation for real-world table QA.

Methods	ChrF	Substring EM	Table-F1
Randomly sampled noise injection			
TableLLaMA	18.41	12.28	17.20
EASE* + TableLLaMA	**25.94**	**19.60**	**30.67**
Contextually sampled noise injection			
TableLLaMA	18.00	15.32	20.26
EASE* + TableLLaMA	**23.07**	**17.05**	**24.65**

Table 13: Experimental results of using the EASE-generated sub-table as input on TableLLaMA. The * indicates the result of LLM-based reasoning methods with GPT-4o as the backbone LLM.

Methods	GPT-4o accuracy
Randomly sampled noise injection	
MultiTabQA	0.003
Chain-of-Table*	0.56
EASE*	**0.78**
Contextually sampled noise injection	
MultiTabQA	0.01
Chain-of-Table*	0.49
EASE*	**0.64**

Table 14: Experimental results of EASE and baselines under unanswerable cases. The * indicates the result of LLM-based reasoning methods with GPT-4o as the backbone LLM.

C.3 Robustness of EASE under unanswerable cases

Real-world table QA often involves scenarios with partial or completely missing information necessary for answering a question. To address this, we conduct additional experiments to evaluate the handling of faulty table cases. Specifically, we design a noise-only multi-table QA setting, where each data instance consists exclusively of query-irrelevant tables, creating inherently unanswerable scenarios. In such cases, the system should ideally generate

Methods	Randomly sampled noise injection	Contextually sampled noise injection
	GPT-4o	
Zero-Shot	57.20	55.86
CoT	64.73	60.66
Chain-of-Table	39.76	46.83
Direct sub-table extraction	58.41	55.43
EASE	**69.65**	**65.68**
	Claude-3.5	
Methods	Randomly sampled noise injection	Contextually sampled noise injection
Zero-Shot	49.07	49.68
CoT	**58.50**	58.68
Chain-of-Table	38.48	40.82
Direct sub-table extraction	50.16	51.13
EASE	57.27	**61.92**

Table 15: Experimental results of Direct sub-table extraction methods, EASE and baselines on Noisy Multi-table QA dataset with GPT-4o, Claude-3.5 as a backbone LLM. All results are presented in Table-F1 metric.

outputs such as "unanswerable" or "None." To verify the framework's robustness under a faulty input table set scenario, we conduct experiments on the MultiTabQA model, Chain-of-Table, and EASE. To evaluate the framework's performance in unanswerable cases, we introdcue an LLM-as-a-judge metric utilizing GPT-4o to judge each framework's output's semantic similarity between each framework's output and a reference set containing responses to unanswerable situations (e.g., "unanswerable", "none", "not provided", "unavailable", "not contain").

As illustrated in Table 14, EASE surpasses all baselines, showing comparable robustness in unanswerable situations. This can be attributed to EASE's Entity Extraction module, which effectively discriminates between query-relevant and irrelevant tables within the input table sets. In contrast, MultiTabQA, fails to handle unanswerable cases by always generating output based on the given faulty input table sets. In summary, EASE demonstrates superior robustness compared to baselines, showcasing its generalizability in handling faulty table cases.

C.4 Necessity of multi-step reasoning pipeline of EASE under Noisy Multi-table QA settings

In Noisy Multi-table QA settings, the system is required to conduct multi-step reasoning by filtering out irrelevant information and effectively aggregating multiple table information in to generate an accurate answer. Therefore, we conducted additional experiments for validating the multi-stage design choice of EASE by adding the 'Direct sub-table extraction' method. We use the prompt shown in Table 16, by instructing the LLM to first extract the sub-table and generate the answer based on the extracted sub-table. For the experiment, we set the GPT-4o and Claud-3.5 as a backbone model and conducted experiments on CoT, Chain-of-Table, and EASE.

«SYSTEM»
Answer the following question based on the provided table in the JSON format. Please identify relevant sub-tables that are needed for answering the question. When generating the final answer, generate after the format 'Final Answer:'.

«USER»
Table Data: `{tables}`
Question: `{question}`

Table 16: Prompt for direct sub-table generation method.

The experimental results for validating the necessity of multi-step reasoning pipeline of EASE are shown in Table 15. These results show that the direct sub-table extraction method shows a marginal gain over zero-shot prompting, but lags behind the EASE and other baselines. Morevoer, Direct sub-table extraction even shows lower performance on contextually sampled noise injection with GPT-4o as a backbone. This indicates that instructing the model to extract the sub-table and answer simultaneously is an inappropriate design choice under Noisy Multi-table QA settings, as the system fails to extract and aggregate pertinent table values for sub-table generation. In contrast, EASE's explicit multi-stage pipeline ensure a structured decomposition of the task, leading to more robust performance under varying noise conditions. These findings highlight the importance of multi-step reasoning over direct sub-table extraction approaches, reinforcing the design motivation behind EASE.

Methods	GPT-3.5			Claude-3.5			GPT-4o		
	ChrF	**Substring EM**	**Table-F1**	**ChrF**	**Substring EM**	**Table-F1**	**ChrF**	**Substring EM**	**Table-F1**
Chain-of-Thought	12.62	11.87	14.85	54.16	42.87	**58.50**	60.12	48.33	64.73
Chain-of-Table	32.37	**44.40**	43.79	14.41	43.56	38.48	35.54	39.90	39.76
EASE w programming	**41.99**	34.81	37.03	**59.99**	53.92	57.43	**71.09**	64.99	67.98
EASE	30.15	43.28	**44.85**	38.27	**55.81**	57.27	63.80	**66.02**	**69.65**

Table 17: Performance of EASE with different implementations for the sub-table filling module in a randomly sampled noise injection setting : LLM-based vs. Programming-based.

Methods	GPT-3.5			Claude-3.5			GPT-4o		
	ChrF	**Substring EM**	**Table-F1**	**ChrF**	**Substring EM**	**Table-F1**	**ChrF**	**Substring EM**	**Table-F1**
Chain-of-Thought	12.93	13.97	15.22	53.26	42.43	58.68	56.52	44.85	60.66
Chain-of-Table	32.01	39.58	40.93	14.65	46.17	40.82	32.80	48.06	46.83
EASE w programming	**40.71**	31.81	35.16	**57.11**	51.04	54.51	**67.63**	**62.29**	64.65
EASE	29.63	**41.67**	**43.05**	48.29	**58.53**	**61.92**	59.43	62.06	**65.68**

Table 18: Performance of EASE with different implementations for the sub-table filling module in a contextually sampled noise injection setting : LLM-based vs. Programming-based.

C.5 Validity of LLM-based sub-table filling compared to programming-based approach

To tackle the challenges posed by real-world table QA, we proposed EASE framework which generates precise sub-tables through the Sub-table Schema Generation and Sub-table Filling modules. This design is inspired by the findings of Min et al. (2024), which demonstrated that LLMs exhibit improved performance on table QA tasks when tables are presented in a markdown format within a Retrieval-Augmented Generation (RAG) context. Accordingly, our Sub-table Filling module employs an LLM to generate sub-tables in this markdown style, thereby improving downstream QA accuracy. However, despite its potential for higher accuracy, this LLM-based generation process introduces computational overhead compared to programming-based approaches. To analyze this trade-off between accuracy and efficiency, we conduct a comparative experiment by substituting the Sub-table Filling module with a code generation-based implementation. In this setup, we prompted the LLM with the curated pandas code generation instruction, question Q and selected relevant column value set T_S to generate a pandas script that simultaneously performs the sub-table filling and derives the final answer.

Table 17 and Table 18 summarize our comparative experiments, which indicate that the code generation-based implementation (EASE w programming) yields mixed and often unfavorable results in Noisy Multi-table QA settings. While EASE w programming improved the syntactic similarity of answers, it struggled to enhance overall QA accuracy. The performance varied significantly depending on the backbone LLMs. With a highly capable model like GPT-4o, the EASE w programming still outperformed the Chain-of-Thought baseline, though it fall short of the original EASE. However, the performance degradation was more prominent for Claude-3.5 and GPT-3.5, where EASE w programming not only underperformed the original EASE but also lagged behind other baselines. This reveals a key findings: leveraging LLMs for code generation is less robust and highly dependent on the model's coding generation capability, whereas the LLM-based sub-table generation in EASE offers a higher end-to-end task performance and maintains generalizability. Consequently, these experiments empirically validate our design choice in EASE, highlighting the effectiveness of direct sub-table generation for achieving reliable end-to-end performance across a range of LLMs.

C.6 Discussion on computational overhead of EASE

To quantify the inference cost discussed in Limitations, we conduct a theoretical analysis of inference cost with respect to the token consumption complexity. We analyze the token consumption of CoT, Chain-of-Table, and EASE. For Chain-of-Table, which operates iteratively, we measure token consumption by tracking the mean and maximum number of reasoning iterations. We define the following variables for our analysis: N (rows per table), M (columns per table), x (rows in a sub-table), y

Methods	**Token consumption complexity**
CoT	$O(TNM)$
Chain-of-Table	$O(TNM + kxy)$
EASE	$O(TrM + T_{\text{sub}}sc)$

Table 19: Theoretical analysis of token consumption for each method represented in Big O notation.

(columns in a sub-table), T (number of tables), k (number of sub-table generation iterations), and r (rows for entity extraction). In our experiments, we set $T = 4$ for all methods and $r = 3$ for EASE.

As shown in Table 19, both CoT and Chain-of-Table initially receive the full tables as input. However, Chain-of-Table's token consumption further increases as it iteratively processes sub-tables of size $x \times y$ for k iterations ($x \leq N, y \leq M$). We empirically observe that k can reach up to 6, leading to substantial token consumption. In contrast, EASE significantly reduces input size by first processing only the top three rows ($r = 3$) of each table for Entity & Operation Extraction. Subsequently, its sub-table filling module operates on a much smaller and more concise input T_s, which consists of fewer number of tables ($T_{\text{sub}} \leq T$) containing only query-relevant rows ($s \ll N$) and columns ($c \ll M$).

In terms of generation time, CoT exhibited the lowest latency as it produces the answer in a single inference step. Conversely, Chain-of-Table recorded the longest generation time due to its iterative reasoning process, which required an average of four and up to a maximum of six steps in our experiments. EASE requires a fixed number of four sequential LLM calls to execute its modular pipeline for filtering and aggregation. Consequently, while EASE is highly efficient on token consumption, this efficiency comes at the cost of higher generation latency compared to the direct CoT baseline.

C.7 Module-wise Error Distribution

Figure 5 shows the module-wise error distribution of EASE under different noise settings.

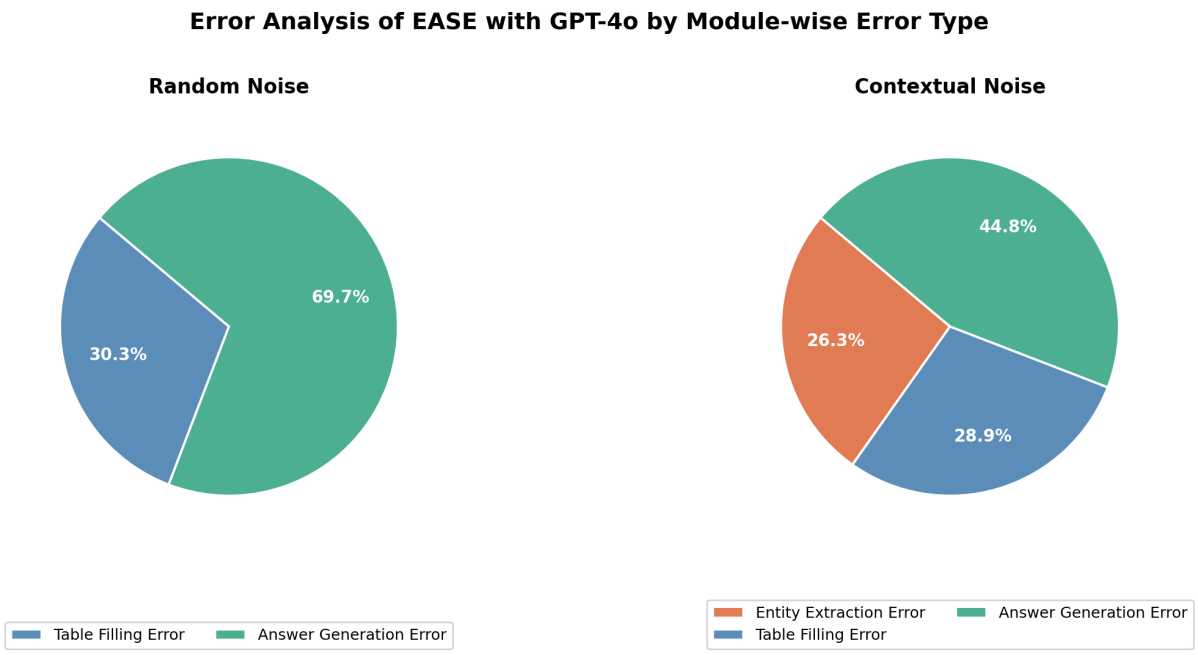

Figure 5: Error distribution of EASE with GPT-4o by module when randomly and contextually sampled noise are injected. While Answer Generation accounts for most errors in both noisy settings, contextual noise also increases upstream failures in Entity Extraction, suggesting that semantically similar noise makes early-stage filtering more difficult.

«SYSTEM»
Task description:
You are an expert data scientist. You will be given a question and the schemas of the tables. Your job is to extract important entities and desired operations from a given question. The extracted entities and operations will be used as groundwork for constructing a reasoning table to write a faithful and precise answer to the given question. Below is the procedure for extracting entities and operations from the question.

Procedure.
1. Find the relevancy between the question and tables: Read the question first, then look at the schemas of the tables to find the columns that might be useful sources for answering it.
2. Extract operations that should be used when answering the question. The operation might be about counting, comparison, sorting, arithmetic operations, grouping, etc.
- If the question merely requires a selection of the value within the table, the operation should be 'select'
3. Extract the important entities in the question. The extraction must follow these criteria:
1) Discard the entity that is irrelevant to any columns of the given tables.
2) Do not discard the entity that can be reasoned by using the values from the given tables' columns.
3) If the entity's name is the same as the table's name, extract the relevant column name from that corresponding table.
4) If the extracted entity emerges from more than two tables, extract all the co-occurred entities, and the table name should delimit the entities.
5) Extract entity as 'Table_name.Column_name' format.

Note
- Please be aware that the provided example tables are just subsets of the original, complete tables. Therefore, the entities you wish to extract may not be present in the row of the tables.
- This should not lead to extraction and operation failure: not extracting any entities or operations because of the absence of the value in the given example tables.
- Do not generate "None" or "Unanswerable" for the **Operations** and **Extracted Entities**. You must extract at least one entity or operation.

Output format:
- **Explanation**: Describe step by step reasoning chain of the extraction
- **Operations**: #o1: Operation name,
#o2: Operation name, ...
#oj: Operation name.
- **Extracted Entities**: #e1: Entity name,
#e2: Entity name, ...
#ei: Entity name.

{few_shot}

«USER»
Question: {question}
Example Tables: <table_name> : {table_name_1} col: {column_names of table_name_1}
row1: {row1 values of table_name_1} row 2 : {row2 values of table_name_1} row 3 : {row3 values of table_name_1}
...
<table_name> : {table_name_n} col: {column_names of table_name_n}
row1: {row1 values of table_name_n} row 2 : {row2 values of table_name_n} row 3 : {row3 values of table_name_n}

Table 20: Prompt for Entity & Operation Extraction. The input is constructed as a pair of Question and Example Tables. For Example Tables, it consist of {table_name_n} which represents the names of the table, {column_names of table_name_n} that specifies the list of column names and the corresponding values in the first three rows.

Input	Question: Show all card type codes and the number of cards in each type. Example Tables: <table_name> : Customers_Cards col : card_id | customer_id | card_type_code | card_number | date_valid_from | date_valid_to | other_card_details row 1 : 1 | 14 | Credit | 4560596484842 | 2011-04-17 09:05:28 | 2018-03-07 17:06:19 | 5567915676420343 row 2 : 2 | 9 | Credit | 4859448397570735 | 2012-05-22 02:05:41 | 2018-02-25 15:43:32 | 4539333582760 row 3 : 3 | 6 | Debit | 348237249146948 | 2014-08-01 14:26:45 | 2018-03-24 15:29:00 | 4916210554814
Output	**Explanation**: 1. **Relevancy between the question and table**: - The question is asking for "all card type codes and the number of cards in each type," which directly relates to the 'Customers_Cards' table. - The 'card_type_code' column in the 'Customers_Cards' table is relevant for identifying the different types of card codes. - The question also requires counting the number of cards for each type, which involves the 'card_id' column in the 'Customers_Cards' table. 2. **Extract operations**: - **Grouping**: We need to group the data by 'card_type_code' to categorize the cards into their respective types. - **Counting**: After grouping, we need to count the number of 'card_id' entries within each group to determine the number of cards for each type. 3. **Extract important entities**: - 'Customers_Cards.card_type_code' is extracted because it represents the card type codes that need to be listed. - 'Customers_Cards.card_id' is extracted to perform the counting operation to find out how many cards exist for each type. **Operations**: - #o1: Group (by 'card_type_code' to categorize cards) - #o2: Count (to count 'card_id' entries within each type) **Extracted Entities**: - #e1: Customers_Cards.card_type_code - #e2: Customers_Cards.card_id

Table 21: Example for Few-Shot setting in Entity & Operation Extraction.

«USER»
Generate a small empty table format to answer queries with the extracted entities as columns. The empty table should be represented in standard tabular format.
The output format should be like: ```| entity1| entity 2 | entity3 | ... | entity 4|
|———|———————|——-| ... |———————|
| | | | ... | |```
Just give me the output.

Input: {entities}

Table 22: Prompt for Sub-table Schema Generation. {entities} is the entities extracted in the Entity & Operation Extraction.

«SYSTEM»
Fill the table using the given data. Extract and organize information from the Source Data and fill the Empty table.

Input:
* Source Data (Provided Tables)
* Empty Table Structure (Columns)

Output Format:
1. Filled Table: Provide the completed table in this markdown format:
```| Column 1 | Column 2 | ... | Column N | |—————|—————|—|—————| | Value 1 | Value 2 | ... | Value N | | ... | ... | ... | ... |```
2. Reasoning Steps: Explain how you filled the table step-by-step for each column and any rules applied to handle missing or conflicting data.

«USER»
Fill the table using the given data. Extract and organize information from the Source Data and fill the Empty table.
Input:
* Source Data (Provided Tables): {tables}
* Empty Table Structure (Columns): {empty_table}

Output Format:
1. Filled Table: Provide the completed table in this markdown format:
```| Column 1 | Column 2 | ... | Column N | |—————|—————|—|—————| | Value 1 | Value 2 | ... | Value N | | ... | ... | ... | ... |```
2. Reasoning Steps: Explain how you filled the table step-by-step for each column and any rules applied to handle missing or conflicting data.

Table 23: Prompt for Sub-table Filling. {tables} contains the tables fitted in the Column Value Selection step, and {empty_table} contains the entities extracted in the Entity & Operation Extraction.

«SYSTEM»
You are a data analysis expert. Please answer the question using step-by-step reasoning based on the provided table and the list of potentially required operations.

Instructions:
1. Parse the structured data carefully.
2. Identify relationships between tables and question.
3. Provide a step-by-step reasoning process based on the provided operations.
4. Answer the question using data from all tables. When generating the Final Answer, write conciesly and precisely.

The output should be like
Reasoning step:
Final Answer:

«USER»
Table Data: {table}
Question: {question}
Operations: {operations}

Table 24: Prompt for Answer Generation. {table} is the table obtained in the Sub-table Filling, and {operations} are the operations extracted in the Entity & Operation Extraction.

«SYSTEM»
You are an expert data scientist. You will be given a Question and Table as input.
The Table is a text that is formed as a Python dictionary. The dictionary is a nested dictionary with the table's name as the main key and columns, index, and data as subkeys.
Your job is to answer the question with the given table.

For example, when given below input:
Table: {{ "department": {{"columns": ["Department_ID", "Name", "Creation", "Ranking", "Budget_in_Billions", "Num_Employees"], "index": ["row 1", "row 2", "row 3", "row 4", "row 5", "row 6", "row 7", "row 8", "row 9", "row 10", "row 11", "row 12", "row 13", "row 14", "row 15"], "data": [[1, "State", 1789, 1, 9.96, 30266], [2, "Treasury", 1789, 2, 11.1, 115897], [3, "Defense", 1947, 3, 439.3, 3000000], [4, "Justice", 1870, 4, 23.4, 112557], [5, "Interior", 1849, 5, 10.7, 71436], [6, "Agriculture", 1889, 6, 77.6, 109832], [7, "Commerce", 1903, 7, 6.2, 36000], [8, "Labor", 1913, 8, 59.7, 17347], [9, "Health and Human Services", 1953, 9, 543.2, 67000], [10, "Housing and Urban Development", 1965, 10, 46.2, 10600], [11, "Transportation", 1966, 11, 58.0, 58622], [12, "Energy", 1977, 12, 21.5, 116100], [13, "Education", 1979, 13, 62.8, 4487], [14, "Veterans Affairs", 1989, 14, 73.2, 235000], [15, "Homeland Security", 2002, 15, 44.6, 208000]] }}, {{ "management": {{"columns": ["department_ID", "head_ID", "temporary_acting"], "index": ["row 1", "row 2", "row 3", "row 4", "row 5"], "data": [[2, 5, "Yes"], [15, 4, "Yes"], [2, 6, "Yes"], [7, 3, "No"], [11, 10, "No"]]}} }}
Question: Show the name and number of employees for the departments managed by heads whose temporary acting value is 'Yes'?

The output should be:
```json
{{"answer": "...",}}
```

Just give me the output.

{few_shot}

«USER»
Table Data: {tables}
Question: {question}

Table 25: Prompt for Zero-Shot and Few-Shot in generic reasoning baseline. The system prompt includes a detailed description of the model's role and tasks. Additionally, the model takes in noisy multi-table information as {tables} and a query as {question} as inputs to generate a response. In a Few-shot setting, examples relevant to the task are provided in {few_shot}.

Input	Table Data: {'Customer_Orders': '{"columns": ["order_id", "customer_id", "order_status_code", "order_date"], "index":[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14], "data": [[1,12,"Completed","2018-02-10 15:44:48"], [2,4,"New","2018-01-31 17:49:18"], [3,1,"PartFilled","2018-02-26 12:39:33"], [4,4,"Pending","2018-03-07 16:55:17"], [5,4,"New","2018-02-12 19:34:12"], [6,11,"PartFilled","2018-03-06 16:35:51"], [7,1,"Cancelled","2018-02-15 02:25:32"], [8,13,"Pending","2018-03-05 23:17:54"], [9,7,"Pending","2018-02-09 11:16:46"], [10,11,"Cancelled","2018-03-22 10:49:25"], [11,8,"Cancelled","2018-02-16 19:42:39"], [12,4,"New","2018-02-02 23:42:01"], [13,15,"PartFilled","2018-02-26 05:34:18"], [14,6,"Completed","2018-03-18 15:12:39"], [15,10,"Pending","2018-03-16 22:42:46"]]}', Question: 'List the order id, customer id for orders in Cancelled status, ordered by their order dates.'
Output	Order IDs 7, 11, and 10 with Customer IDs 1, 8, and 11 respectively.

Table 26: Example for Few-Shot setting in the generic reasoning baseline.

«SYSTEM»
You are an expert data scientist. You will be given a Question and Table as input.
The Table is a text that is formed as a Python dictionary. The dictionary is a nested dictionary with the table's name as the main key and columns, index, and data as subkeys.
Your job is to answer the question with the given table.

For example, when given below input: Table: {{ "department": {{"columns": ["Department_ID", "Name", "Creation", "Ranking", "Budget_in_Billions", "Num_Employees"], "index": ["row 1", "row 2", "row 3", "row 4", "row 5", "row 6", "row 7", "row 8", "row 9", "row 10", "row 11", "row 12", "row 13", "row 14", "row 15"], "data": [[1, "State", 1789, 1, 9.96, 30266], [2, "Treasury", 1789, 2, 11.1, 115897], [3, "Defense", 1947, 3, 439.3, 3000000], [4, "Justice", 1870, 4, 23.4, 112557], [5, "Interior", 1849, 5, 10.7, 71436], [6, "Agriculture", 1889, 6, 77.6, 109832], [7, "Commerce", 1903, 7, 6.2, 36000], [8, "Labor", 1913, 8, 59.7, 17347], [9, "Health and Human Services", 1953, 9, 543.2, 67000], [10, "Housing and Urban Development", 1965, 10, 46.2, 10600], [11, "Transportation", 1966, 11, 58.0, 58622], [12, "Energy", 1977, 12, 21.5, 116100], [13, "Education", 1979, 13, 62.8, 4487], [14, "Veterans Affairs", 1989, 14, 73.2, 235000], [15, "Homeland Security", 2002, 15, 44.6, 208000]] }}, {{ "management": {{"columns": ["department_ID", "head_ID", "temporary_acting"], "index": ["row 1", "row 2", "row 3", "row 4", "row 5"], "data": [[2, 5, "Yes"], [15, 4, "Yes"], [2, 6, "Yes"], [7, 3, "No"], [11, 10, "No"]]}} }}
Question: Show the name and number of employees for the departments managed by heads whose temporary acting value is 'Yes'?

The final answer should be:
```json
{{"answer": "...",}}
```

«USER»
Table Data: `{tables}`
Question: `{question}`
Let's think step by step.

Table 27: Prompt for Chain-of-Thought in generic reasoning baseline.

Question: Find the total balance across checking accounts.
Input Tables: {'CHECKING': {'columns': ['custid', 'balance'], 'index': [0, 1, 2, 3, 4, 5, 6], 'data': [[1, 10000.0], [2, 2000.0], [3, 3000.0], [4, 7000.0], [5, 10000.0], [6, 77.0], [7, 7.0]], 'table_name': 'CHECKING'}, 'Tryout': {'columns': ['pID', 'cName','pPos', 'decision'], 'index': [0, 1, 2, 3, 4, 5], 'data': [[10001, 'LSU', 'goalie', 'no'], [10001, 'ASU', 'goalie', 'yes'], [20002, 'FSU', ...

Entity & Operation Extraction	**Explanation**: 1. **Relevancy between the question and tables**: - The question asks for the total balance across checking accounts. This directly relates to the 'CHECKING' table. - The 'balance' column in the 'CHECKING' table is relevant for calculating the total balance. 2. **Extract operations**: - **Summation**: We need to sum up all the values in the 'balance' column to find the total balance across all checking accounts. 3. **Extract important entities**: - 'CHECKING.balance' is the key entity needed to perform the summation operation to find the total balance. **Operations**: - #o1: Sum (to calculate the total of all values in the 'balance' column) **Extracted Entities**: - #e1: CHECKING.balance
Sub-table Schema Generation	``` | CHECKING | balance | |———————|———————| | | | ```
Column Value Selection	{'CHECKING': {'columns': ['balance'], 'index': [0, 1, 2, 3, 4, 5, 6], 'data': [[10000.0], [2000.0], [3000.0], [7000.0], [10000.0], [77.0], [7.0]], 'table_name': 'CHECKING'})

Table 28: Results generated by the Entity & Operation Extraction, Sub-table Schema Generation, and Column Value Selection steps for the given question and tables.

Question: Find the total balance across checking accounts.
Input Tables: {'CHECKING': {'columns': ['custid', 'balance'], 'index': [0, 1, 2, 3, 4, 5, 6], 'data': [[1, 10000.0], [2, 2000.0], [3, 3000.0], [4, 7000.0], [5, 10000.0], [6, 77.0], [7, 7.0]], 'table_name': 'CHECKING'}, 'Tryout': {'columns': ['pID', 'cName' ,'pPos', 'decision'], 'index': [0, 1, 2, 3, 4, 5], 'data': [[10001, 'LSU', 'goalie', 'no'], [10001, 'ASU', 'goalie', 'yes'], [20002, 'FSU', ...

Sub-table Filling	To fill the empty table using the provided source data, we need to extract and organize the information from the given data structure. Here's how we can do it step-by-step: ### Reasoning Steps: 1. **Understanding the Source Data Structure**: - The source data is a dictionary with a key "CHECKING" that maps to another dictionary. - This inner dictionary contains: - "index": A list of indices '[0, 1, 2, 3, 4, 5, 6]'. - "columns": A list of column names '['balance']'. - "data": A list of lists, where each sub-list contains a single float value representing the balance. 2. **Mapping Source Data to Table Structure**: - The empty table structure has columns '['CHECKING', 'balance']'. - The "CHECKING" column will be filled with the indices from the source data. - The "balance" column will be filled with the corresponding balance values from the source data. 3. **Filling the Table**: - For each index in the source data, create a row in the table. - The first column "CHECKING" will contain the index value. - The second column "balance" will contain the corresponding balance value from the "data" list. 4. **Handling Missing or Conflicting Data**: - There is no missing or conflicting data in this case. Each index has a corresponding balance value. ### Filled Table: `` | CHECKING | balance | |————-|———| | 0 | 10000.0 | | 1 | 2000.0 | | 2 | 3000.0 | | 3 | 7000.0 | | 4 | 10000.0 | | 5 | 77.0 | | 6 | 7.0 | ``` Each row corresponds to an index from the source data, with the balance value filled in the respective column. This approach ensures that all data from the source is accurately represented in the filled table.
Answer Generation	Reasoning step: 1. Identify the relevant column for the operation, which is the 'balance' column in the 'CHECKING' table. 2. Use the sum operation to calculate the total balance across all checking accounts by adding up all the values in the 'balance' column. Calculation: - Sum of balances = 10000.0 + 2000.0 + 3000.0 + 7000.0 + 10000.0 + 77.0 + 7.0 Final Answer: The total balance across checking accounts is 32,084.0.

Table 29: Results generated by the Sub-table Filling and Answer Generation for the given question and tables.