Piece of Table: A Divide-and-Conquer Approach for Selecting Subtables in Table Question Answering

Wonjin Lee[*1] **Kyumin Kim**[*2] **Sungjae Lee**[1] **Jihun Lee**[1] **Kwang In Kim**[1]
[1]POSTECH [2]LG CNS
{wonjin0403,leeeesj,gmindflow,kimkin}@postech.ac.kr
kyuminkim@lgcns.com

Abstract

Applying language models (LMs) to tables is challenging due to the mismatch between the two-dimensional structure of tables and the one-dimensional inputs expected by LMs. This mismatch forces linearization, making LMs particularly sensitive to irrelevant cells. Subtable selection mitigates this challenge by isolating question-relevant content prior to answer generation. However, existing approaches either rely on independent row or column selection, failing to capture cross-row and cross-column dependencies, or attempt global reasoning and face challenges similar to holistic table QA under noisy contexts. We propose *PieTa* (Piece of Table), a divide-and-conquer subtable selection framework that progressively aggregates locally selected evidence without requiring explicit global reasoning. *PieTa* uses an iterative, window-based multi-resolution process to construct compact subtables that capture global dependencies while limiting LM exposure to irrelevant content. Extensive experiments demonstrate that *PieTa* consistently outperforms prior subtable-based and holistic table QA approaches. Our code is available at https://github.com/MLV-group/Piece-of-Table.git.

1 Introduction

Tables effectively mitigate data complexity by organizing information into curated rows and columns, making them more accessible and comprehensible than plain text. This structured representation has inspired a range of natural language processing tasks, including table question answering (QA) (Pasupat and Liang, 2015; Zhong et al., 2017; Iyyer et al., 2017; Chen et al., 2020; Zhu et al., 2021; Nan et al., 2022; Cheng et al., 2022b; Pang et al., 2024; Wu et al., 2025a,b).

Despite their expressiveness, tables pose fundamental challenges for language model (LM) readers (Wang et al., 2024). LMs are primarily trained to process linear, one-dimensional token sequences, whereas tables are inherently two-dimensional and require reasoning across rows and columns, often without a natural sequential structure (Chen et al., 2023; Li et al., 2024). This mismatch necessitates linearization, forcing heterogeneous table cells into a single sequence.

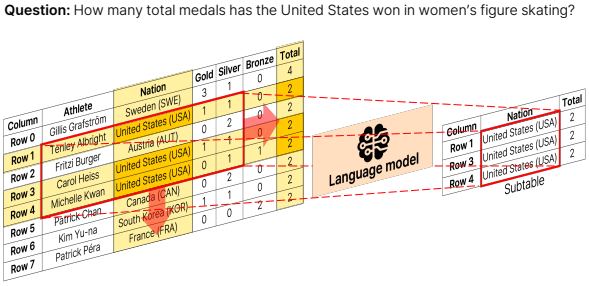

Figure 1: Overview of the proposed *PieTa* (Piece of Table) framework. Given an input table and a question, it synthesizes a subtable by iteratively dividing the table into smaller windows, selecting relevant cells within each window using language models, and merging the resulting subwindows. This process is repeated until the final subtable is formed.

Such linearization places relevant and non-evidential cells in a shared context, amplifying the impact of irrelevant information and distracting LM reasoning. In table QA, only a small subset of cells typically provides the evidence needed to answer a question, while the remainder introduces noise (Patnaik et al., 2024). Even models with table-aware pretraining or advanced prompting often struggle to reliably identify relevant evidence under such noisy conditions (Kumar et al., 2023; Chen et al., 2024).

Prior work has sought to mitigate this challenge by improving LMs' understanding of tabular structure, e.g. through table-specific pretraining objectives (Liu et al., 2022; Jiang et al., 2022; Cheng et al., 2022a), specialized positional embeddings (Herzig et al., 2020; Yang et al., 2022; Yin et al., 2020;

[*]Equal contribution.

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 31672–31688
July 2-7, 2026 ©2026 Association for Computational Linguistics

Wang et al., 2021), or table-specific prompting with large commercial LMs (Cheng et al., 2023; Zhang et al., 2024; Wang et al., 2024; Ji et al., 2024; Yang et al., 2025). However, these approaches typically operate over entire tables and therefore remain vulnerable to distraction from irrelevant content.

This motivates subtable selection, which aims to select a question-relevant subset of a table to reduce the noise presented to the QA reader. The QA reader then focuses on the selected subtable to construct the final answer.

For example, Lin et al. (2023)'s *inner table retriever (ITR)* selects rows and columns that align with a given question by computing similarity scores between the question and each individual row or column. This approach is effective for answering *simple* questions, but it struggles to capture dependencies across multiple rows and/or columns (Section 2). Ye et al. (2023)'s *Dater* addresses this limitation by using LM in-context learning to jointly select multiple rows. While this improves upon independent row selection, the method inherits limitations of applying LM readers to tabular data, as the LM-based *selector* itself can be distracted by noise present in the table.

Ideally, constructing a high-quality subtable requires reasoning over interactions among multiple rows and columns. However, capturing such interactions jointly, as in *holistic* QA approaches that reason directly over the entire table, remains challenging in practice due to the presence of excessive irrelevant information.

In this paper, we present *PieTa* (Piece of Table), a subtable selection framework for table QA. The key idea of *PieTa* is to progressively construct a subtable while ensuring that LMs are invoked only on moderate amounts of table context at each stage. *PieTa* follows an iterative, multi resolution procedure. In the *Divide* step, the input table is decomposed into smaller windows. In the subsequent *Conquer* step, an LM selects question-relevant cells within each window (Figure 1), forming subwindows. In the *Combine* step, these subwindows are merged into a new compact table. This process is repeated by treating the merged table as the input to the next iteration until the final subtable is obtained. By progressively selecting and combining informative table regions, *PieTa* captures dependencies across multiple rows and columns while avoiding LM reasoning under excessive irrelevant information.

Evaluated on the *WikiTableQuestions* (WikiTQ; Pasupat and Liang, 2015), WikiSQL (Zhong et al., 2017), and *Hierarchical Table* (HiTab; Cheng et al., 2022b) datasets, *PieTa* achieves significant improvements in both QA accuracy and subtable selection performance.

2 Related Work

In table QA, a reader, often implemented as a language model (LM), answers questions by reasonging over an entire table (*holistic* approaches) or a pre-selected subtable.

Holistic table QA readers. *TaPEx* mimics an SQL execution engine by synthesizing answers from tables using paired SQL queries (Liu et al., 2022). It adopts the *BART* backbone model (Lewis et al., 2020), which is initially pre-trained on synthetic SQL queries and subsequently fine-tuned for natural language table QA.

Constructing large-scale, high-quality table-SQL query pairs can be costly. To alleviate this burden, *OmniTab* replaces complex SQL queries with plain-text descriptions (Jiang et al., 2022). It assumes that each table is accompanied by a textual description, which is used for pre-training before fine-tuning on table QA tasks.

Prompt-based holistic methods have been extensively studied in the presence of large-scale commercial LMs. Chen (2023) adopts Chain-of-Thought (Wei et al., 2022) prompting to reason explicitly about table structure. Similarly, *Chain-of-Table* and *StructGPT* formulate step-by-step reasoning as sequences of tabular operations, enabling LMs to handle complex tables (Jiang et al., 2023; Wang et al., 2024). *Binder* and *ReActable* further employ LMs to generate executable SQL statements (Cheng et al., 2023; Zhang et al., 2024).

Despite their effectiveness, holistic approaches share a fundamental limitation: For most questions, only a small subset of table cells is relevant, while the remaining cells distract LM reasoning (Patnaik et al., 2024). This motivates approaches that aim to explicitly identify and isolate relevant table content.

Subtable selection(-based) approaches. Lin et al. (2023) fine-tune *dense passage retriever* encoders (Karpukhin et al., 2020) to select relevant rows and columns. Their *inner table retriever* (*ITR*) uses a supervised contrastive loss that labels rows and columns containing *answer* cells as positive. Originally designed to keep table inputs within an LM's token limit, this approach does not explicitly construct a *minimum* subtable that balances preci-

sion and recall. While extending this technique to filter irrelevant information is feasible, it cannot capture dependencies across multiple rows or columns. For instance, when answering the question `Which was the next aircraft that came into service after Cessna 404 Titan?`, *ITR* faces challenges because it must identify the `aircraft` that started the service after the `Cessna 404 Titan` without information on when it started the service.

Dater addresses independent selection by using LM in-context learning to joinlty synthesize a subtable and sub-query pair from a given table and question (Ye et al., 2023). However, the LM-based selector itself remains sensitive to irrelevant information in the table, which can interfere with reliable subtable construction, particularly when non-evidential content is prevalent (Section 4.2).

TabSQLify selects a subset of rows (with all columns) as input for LMs to generate SQL queries for subtable selection (Nahid and Rafiei, 2024). However, similar to *ITR*, it fails to capture relationships across rows. Moreover, even when applied to the entire table, it inherits limitations similar to those of *Dater*, as the LM must still reason in the presence of irrelevant information (Appendix C).

3 Divide-and-Conquer Subtable Selection

The table question answering (QA) task aims to generate an answer A from a table T and a human-understandable question Q. The answer A may take various forms (e.g., a number, word, phrase, or sentence) and is either extracted from T or synthesized by a reader $\mathcal{R}$.

In subtable-based QA approaches, a selector $\mathcal{S}$ first generates a subtable T_s from T, which is then fed to the reader $\mathcal{R}$ to generate the final answer:

$$T_s = \mathcal{S}(T,Q), \quad A = \mathcal{R}(T_s,Q).$$

Our study focuses on improving the selector $\mathcal{S}$ and does not modify the readers. However, in principle, joint tuning of $\mathcal{R}$ and $\mathcal{S}$ is feasible.

3.1 Subtable selection

Our subtable selection algorithm iteratively applies three main steps. In the *Divide* step, the input table T is partitioned into overlapping windows $\{W_i\}_{i=1}^N$ of size $w \times w$ using a sliding window approach (Algorithm 2). In the *Conquer* step, the (subwindow) selector $\mathcal{S}'$ extracts a subwindow $\widehat{V}_i$ within each window W_i, guided by the question Q and a prompt instruction $\mathcal{P}$ (see Appendix A.2):

$$\widehat{V}_i = \mathcal{S}'(\mathcal{P}(W_i,Q)).$$

The *Combine* step merges the generated subwindows $\{\widehat{V}_i\}_{i=1}^N$ into a new table T^t. Specifically, T^t is obtained as the set union of all cells in the subwindows $\{\widehat{V}_i\}_{i=1}^N$. These steps are repeated, taking T^t as the new input T for the *Divide* step in each iteration t, until the generated table T^t is no longer updated. The complete subtable selection process is outlined in Algorithm 1.

Algorithm 1 SUBTABLESELECTION ($\mathcal{S}$).

Input: Table T and question Q
Parameter: Window size w
Output: Final subtable T_s
$t \leftarrow 1$
$T^t \leftarrow T$; $T^0 \leftarrow \varnothing$
while $T^{t-1} \neq T^t$ **do**
  $T^{t-1} \leftarrow T^t$
  $\{W_i\}_{i=1}^N \leftarrow$ DIVIDETABLE(T^t; w)
    ▷ *Divide* step (see Algorithm 2)
  $T^t \leftarrow \varnothing$
  for W_i in $\{W_i\}_{i=1}^N$ **do**
    $\widehat{V}_i^t \leftarrow \mathcal{S}'(\mathcal{P}(W_i,Q))$ ▷ *Conquer* step
    $T^t \leftarrow T^t \cup \widehat{V}_i^t$ ▷ *Combine* step
  end for
  $t \leftarrow t+1$
end while
$T_s \leftarrow T^t$

3.2 Fine-tuning the selector

We construct the selector $\mathcal{S}'$ by fine-tuning the *Llama*[1] (Dubey et al., 2024) and *Qwen*[2] (Yang et al., 2024) models (Section 4). In principle, however, our method is applicable to any LM-based selector.

Data generation. The training data are generated by sampling input windows (of size $w \times w$) from the SQUALL (Shi et al., 2020) and WikiSQL (Zhong et al., 2017) training datasets, which consist of tables, questions, and the associated *target* subwindows (see Section 4). We define *condition columns* as columns containing cells that match the conditions specified in the question. Each cell in a condition column may or may not satisfy the given condition. Similarly, the column containing the exact cell values that the question seeks is referred to as the *answer column*. For example, in the example question presented in Figure 2, the columns `Horwood`, `Total`, and `Placement` are the condition columns, while `Goodman` is an answer column.

[1] `Llama3.1-8B-Instruct`, https://www.llama.com/.
[2] `Qwen2.5-7B-Instruct`, https://huggingface.co/Qwen/Qwen2.5-7B-Instruct.

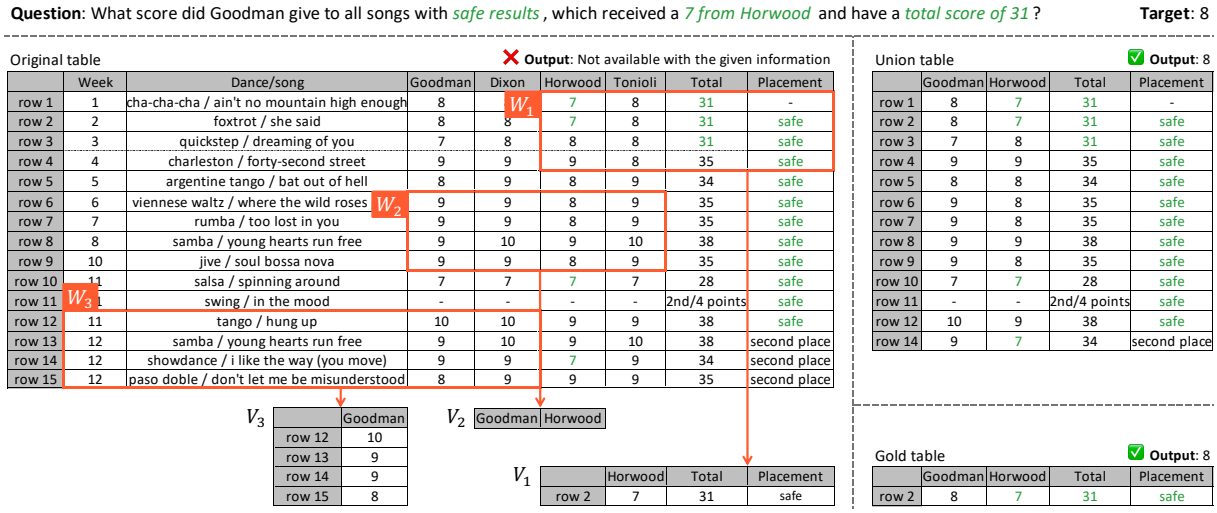

Figure 2: Example of training data generation for subtable selection with a window size $w = 4$. The question and table are taken from the WikiSQL training dataset (Zhong et al., 2017). Three conditions specified in the question and their corresponding cell values in the table are highlighted in green. The three sampled input windows, $\{W_i\}_{i=1}^{3}$ (orange boxes) and their corresponding targets subwindows $\{V_i\}_{i=1}^{3}$ are shown below. On the right, the derived union and gold tables are presented along with the resulting answers. The original table contains three condition columns (`Horwood`, `Total`, and `Placement`) and one answer column (`Goodman`). The union table aggregates all rows that satisfy at least one condition, whereas the gold table includes only rows that jointly satisfy all conditions. Estimating the gold table is significantly more challenging as it requires reasoning over the entire table. The *GPT-3.5* reader produces an incorrect answer when using the full table but generates correct answers when using either the union or gold table.

For each training window W_i sampled from a table T with a corresponding question Q, the target subwindow V_i is constructed by first selecting all condition and answer columns within W_i. Then, for these selected columns, only the rows that satisfy all the conditions present in W_i are included.

Figure 2 illustrates the training data generation process. The first window W_1 contains three condition columns, `Horwood`, `Total` and `Placement`. The target subwindow V_1 is then generated from the cells in these columns that jointly satisfy the conditions `7 from Horwood`, `total score of 31`, and `safe results` mentioned in the question. The second window W_2 includes a condition column `Horwood` and an answer column `Goodman`. To create V_2, the subset of these columns satisfying the condition `7 from Horwood` is included. However, since no cells meet this condition, V_2 is constructed as an empty table containing only the column headers. The third window W_3 contains only an answer column, and as no conditions are imposed, all cells from this column are included in V_3.

This target window generation strategy enables the selector $\mathcal{S}'$ to focus exclusively on the input windows W when making predictions (see Appendix B.5 for details).

3.3 Representing subtables

Common representations for LMs in subtable generation include the *index representation*, which identifies selected rows and columns using their indexes and headers, and the *table representation*, which organizes selected cells into a structured table format. Both representations have inherent limitations that impact their effectiveness.

The index representation struggles when the information required to identify relevant content is embedded within cells rather than in row or column headers. Because it relies solely on headers, it cannot directly capture cell content, making it difficult for selectors to learn how to align queries with embedded information. For example, as shown in Figure 3(a), the column `Placement` is not explicitly mentioned in the question, while the content matching the condition `safe` appears within a cell. This mismatch prevents the LM from recognizing the column `Placement` as relevant.

In contrast, the table representation is prone to errors caused by the autoregressive nature of the token generation process. Since each token prediction depends on previous predictions, early errors in the sequence can propagate and compound, significantly reducing accuracy (Ross and Bagnell, 2010; Bengio et al., 2015; Lamb et al., 2016; Bachmann

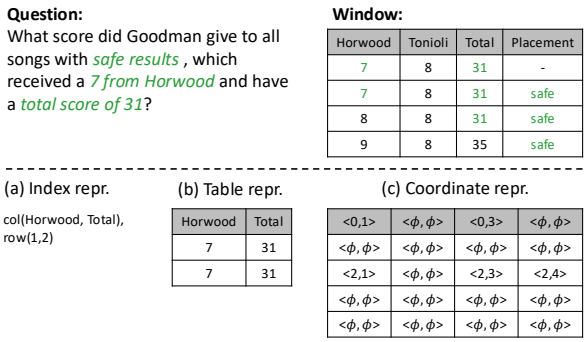

Figure 3: Examples of subtable representations (W_1 in Figure 2; $w = 4$). The information matching the condition `safe` is located in a cell rather than in the column header. Both the index and table representations fail to select the cell `safe` because the column `Placement` is not selected. The proposed coordinate representation overcomes this limitation by preserving the input window structure as well as relevant cell contents. $\langle\phi,\phi\rangle$ denotes an empty token, explicitly indicating that a cell was not selected.

and Nagarajan, 2024). For instance, in Figure 3(b), failure to predict the column header `Placement` results in the exclusion of the entire column.

To overcome these limitations, we propose the *coordinate representation*, which preserves the input subwindow structure and uses *coordinate tokens* (`<row_index,column_index>`) to indicate the positions of selected cells. Unselected cells are denoted by empty tokens (`<ϕ,ϕ>`). This approach improves the ability of the model to identify relevant cells, even when their columns are initially overlooked, as demonstrated in Figure 3(c) with the successful identification of the cell `safe`.

3.4 Discussion

The crux of our approach lies in the multi-resolutional table formation technique built upon table (sub)windows. In images, neighboring pixels often exhibit strong correlations, justifying the use of small windows e.g. convolutional kernels in convolutional neural networks and attention windows in vision transformers. However, tables do not necessarily exhibit such spatial structures: For example, in the original table in Figure 2, neighboring cells like `31` and `35` are more similar within the `Total` column. In contrast, cells in `row 2` lack a comparable relationship: the value `31` in the `total` column appears next to `safe`, but it is not semantically more related to `safe` than to other cell values. From this perspective, the use of subwindows may initially appear to impose unnecessary constraints.

Our *union*-based subwindow combination approach effectively overcomes such potential restrictions while ensuring focused answer generation by constructing moderately-sized final subtables. An ideal subtable for answering the question in Figure 2 would intersect the cells meeting the conditions `7 from Horwood`, `total score of 31` and `safe results`, along with their corresponding cell values in the `Goodman` column, as illustrated in the bottom table in the right column of Figure 2.

Directly identifying such a *gold* table would require a comprehensive understanding of the entire table. Instead, our algorithm iteratively merges subwindows identified within each window. For example, it selects cells matching `7` or `31` or `safe` and combines them into a subtable, effectively taking their *union* (as shown in the table in the upper right column of Figure 2).

This union approach offers significant advantages. First, identifying relevant subwindows within small windows is much simpler than pinpointing the gold table, which requires intersecting conditions across the entire table: The selection module (LMs) can focus exclusively on the cells within each window, avoiding the challenges posed by long context inputs. Our training target generation process aligns with this design: Each target subwindow is constructed solely based on the conditions and outputs present within the corresponding input window.

We also empirically validated that the resulting union subtables are significantly smaller than the original tables, facilitating downstream answer generation. On average, they contain only 13.91% of the original number of cells. Although slightly larger than the gold tables, they achieve comparable QA performance. Table 1 presents examples of subtable construction, demonstrating substantial improvements in final table QA performance.

Dataset	Reader	Original	Union	Gold
WikiTQ	*OmniTab*	62.52	64.39	67.79
WikiSQL	*OmniTab*	88.42	92.64	92.65
	Llama3.1	73.03	78.10	79.01

Table 1: Table QA performance (exact match, %): Union tables achieve performance comparable to gold tables while being easier to construct.

4 Experiments

We evaluated the proposed *PieTa* on the WikiTQ (Pasupat and Liang, 2015), WikiSQL (Zhong

Selector	Precision (P)	Recall (R)
ITR	13.45	97.84
TabSQLify	75.47	87.36
Holistic LM	89.27	93.20
PieTa (1 iter.)	19.22	99.65
PieTa (2 iter.)	56.88	99.22
PieTa (3 iter.)	60.13	99.15
PieTa (Final)	60.88	99.12

Table 2: Performance of subtable selection algorithms for the subtable generation sub-task on WikiSQL, evaluated in precision (%) and recall (%). Hereafter, (P) and (R) denote precision and recall, respectively. The last four rows highlight the effectiveness of our iterative subtable formation approach (*Llama*; see Section 3.1). The total number of iterations varies across instances, as *PieTa* terminates once the subtable remains unchanged.

# cond. & ans. columns	(P)	(R)	# ans. rows	(P)	(R)
1	35.32	100.00	0–5	60.83	99.13
2	66.68	98.90	5–10	65.14	97.83
3	48.37	99.54	10–15	67.09	98.82
4	51.53	99.69	15–20	77.54	99.34
5	42.76	99.53	20–270	72.06	94.36

Table 3: Subtable generation performance across varying numbers of condition and answer columns (left) and answer rows (right) on WikiSQL.

et al., 2017), and HiTab (Cheng et al., 2022b) datasets. WikiTQ and WikiSQL are standard table QA benchmarks that support broad algorithmic comparison, while HiTab presents more complex structures, including nested headers (dataset details are provided in Appendix B).

Our evaluation is conducted from two perspectives. First, for subtable selection, we evaluate the identification of condition and answer rows and columns using annotated labels from WikiSQL. Precision and recall are used as metrics, where cells belonging to the underlying gold table are treated as *positive* and all others as *negative*.

Second, we assess the end-to-end table QA performance by feeding the subtables generated by *PieTa* to table QA readers $\mathcal{R}$. Specifically, we use *OmniTab* (Jiang et al., 2022) and *TaPEx* (Liu et al., 2022), fine-tuned on WikiTQ and WikiSQL, respectively. We also consider general-purpose readers that are not fine-tuned for table QA, including *GPT-3.5*, *Gemini 2.0 Flash-Lite* and *DP&Agent* (Liu et al., 2024) (see Appendix A.3 for details).

For the HiTab dataset, we use *GPT-3.5* and *Gemini 2.0* readers. All evaluations are based on the exact-match (EM) accuracy, which measures the proportion of predictions that exactly match the ground-truth answers.

4.1 Subtable selection results

To evaluate *PieTa* against existing approaches, we conducted experiments using *ITR* (Lin et al., 2023), *TabSQLify* (Nahid and Rafiei, 2024), and a simple baseline algorithm that fine-tunes an LM to generate a subtable directly from the input table (referred to as *Holistic LM*).[3] For *PieTa* and *Holistic LM*, we fine-tune *Llama3.1* for one epoch (see Section 3.2). For *TabSQLify*, we initially evaluated both *Llama3.1* and *GPT-3.5*, and report results for the better-performing model (as *GPT-3.5* was used in (Nahid and Rafiei, 2024)).

Table 2 summarizes the results. *ITR*, which evaluates rows and columns independently, exhibits substantially lower recall than *PieTa*. By effectively capturing dependencies across rows and columns, *PieTa* also improves precision by 47.43%.

Similar to *PieTa*, *TabSQLify* can model cross-column dependencies. It first extracts a small set of rows (specifically, the first three, following Nahid and Rafiei, 2024) and then identifies condition and answer columns. However, if the initial row selection does not contain all required conditions, the method may select irrelevant columns, leading to suboptimal subtable construction.

Increasing the number of extracted rows could partially mitigate this issue, but it also places additional burden on the LM selector by introducing more irrelevant information. As a result, performance quickly plateaus and remains significantly lower than that of *PieTa* (see Appendix C).

Another drawback of approaches that generate SQL queries, including *TabSQLify*, is the risk of producing inoperable SQL code (Xie et al., 2024). We empirically observed that *TabSQLify* generated unexecutable SQL code for 425 out of 15,878 pairs in the WikiSQL dataset and 83 out of 4,344 pairs in the WikiTQ dataset.

In subtable selection, high recall is essential, as missing relevant rows or columns leads to incomplete evidence and consequently degrades

[3]Both *Dater* and *TabSQLify* rely on in-context learning, whereas *PieTa* adopts fine-tuning. To our knowledge, existing fine-tuning approaches do not target subtable selection. *Holistic LM* can thus be viewed as a fine-tuning-based analogue of *Dater* for this task.

Dataset	Selector	(P)	(R)
WikiTQ	*ITR*	7.80	86.67
	Dater	47.29	83.41
	TabSQLify	58.33	74.17
	PieTa	46.97	91.67
HiTab	*ITR*	3.54	100.00
	Dater	22.63	97.23
	TabSQLify	9.55	91.66
	PieTa	19.56	100.00

Table 4: Subtable selection results on WikiTQ and HiTab.

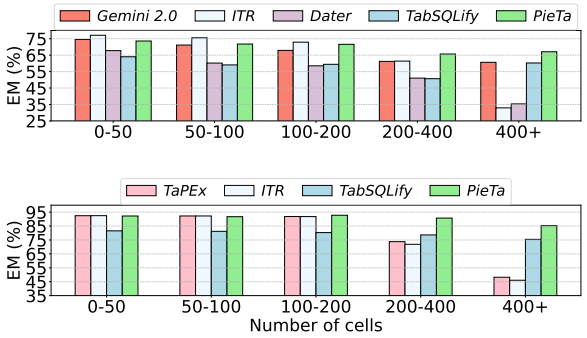

Figure 4: Table QA performance (EM; %) across varying number of cells on WikiTQ (top) and WikiSQL (bottom).

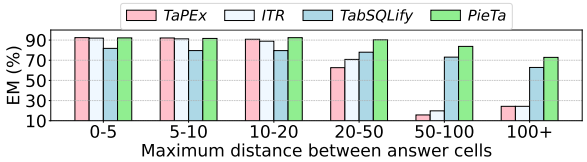

Figure 5: Table QA accuracy (EM; %) on WikiSQL across varying maximum distances between answer cells in a table.

downstream table QA performance. For example, despite achieving high precision in Table 2, *Holistic LM* attains only 45.79% EM (with *OmniTab* reader), substantially lower than our results of 91.17%.

Among the compared subtable selection algorithms, only *ITR* inherently produces high recall under its default settings provided by the authors. Other approaches lack a direct mechanism to balance precision and recall. While *PieTa* is not explicitly optimized for high recall, it consistently demonstrates a tendency to achieve higher recall compared to other approaches.

Table 3 shows that *PieTa* maintains high recall across different numbers of answer columns and rows. Recall also remains high even as the number of condition columns increases, reflecting constraints specified in the question. This demonstrates that *PieTa* effectively captures relevant information even for complex multi-condition queries, which is essential for maintaining QA accuracy.

We also conducted a small-scale human annotation study on the WikiTQ and HiTab test sets.[4] We randomly sampled 30 examples from each test set and asked annotators to determine whether the predicted subtable contains all rows and columns necessary to answer the question (Appendix D).

As shown in Table 4, *PieTa* achieves the highest recall on WikiTQ and matches the highest recall on HiTab, indicating that it consistently preserves the evidence required for answering. Precision is generally lower for all methods under this strict manual evaluation. On HiTab, Dater attains slightly higher precision, whereas *PieTa* achieves higher recall, reflecting a different precision-recall trade-off.

[4]WikiSQL provides SQL annotations from which gold tables can be constructed, whereas such annotations are not available for WikiTQ and HiTab, making automatic recall evaluation difficult.

4.2 Table QA results

Table 5 summarizes exact-match QA accuracy on WikiTQ, WikiSQL, and HiTab. The top section shows results for selector-based QA where different subtable selectors are combined with various readers (*TaPEx*, *OmniTab*, *GPT-3.5*, *Gemini 2.0*, and *DP&Agent*) that are commonly used in prior subtable selection research. A dash (–) indicates the holistic setting where a reader processes the full table without subtable selection. The bottom section presents additional holistic QA models, including *Binder* (Cheng et al., 2023), *StructGPT* (Jiang et al., 2023), *CABINET* (Patnaik et al., 2024), *Chain-of-Table* (Wang et al., 2024), and *ReAcTable* (Zhang et al., 2024), for reference.

On WikiTQ, *PieTa* (using either the *Llama* or *Qwen* backbone) consistently improves reader performance by generating focused and contextually relevant subtables. Across most reader configurations, *PieTa* outperforms both holistic readers and other selectors, achieving substantial gains over using entire tables. The only exception occurs with *TaPEx*, where *TabSQLify* performs slightly better. A similar trend is observed on WikiSQL, where *PieTa* again provides the highest accuracy for most readers. In contrast, competing selectors often degrade performance relative to the holistic baseline, while *PieTa* consistently enhances it.

For HiTab, zero-shot evaluation is performed

Selector-based QA: subtable selectors with different readers							
Dataset	Reader	Selector					
		–	*ITR*	*Dater*	*TabSQLify*	*PieTa* (*Llama*)	*PieTa* (*Qwen*)
WikiTQ	*TaPEx*	56.54	56.86	53.73	**58.75**	58.01	57.51
	OmniTab	63.01	63.47	57.09	60.01	**64.78**	63.79
	GPT-3.5	60.75	59.13	57.78 (52.80)	59.21 (59.71)	**63.65**	63.05
	Gemini 2.0	68.58	70.58	57.78	58.61	70.81	**73.14**
	DP&Agent	73.60	70.79	65.96	62.94	**74.15**	74.08
WikiSQL	*TaPEx*	89.31	89.09	83.01	80.68	**91.83**	89.97
	OmniTab	88.42	89.71	82.85	81.05	**91.17**	89.46
	GPT-3.5	69.50	67.60	68.97	69.58 (70.25)	**75.49**	72.71
	Gemini 2.0	63.97	61.69	61.83	**70.68**	69.76	66.21
HiTab	*GPT-3.5*	40.59	39.97	35.98	33.44	**46.72**	46.65
	Gemini 2.0	52.15	55.11	38.51	46.97	62.06	**63.32**

Holistic QA: full tables without subtable selection							
Dataset	WikiTQ					WikiSQL	
Reader	*Binder*	*StructGPT*	*CABINET*	*Chain-of-Table*	*ReAcTable*	*StructGPT*	*CABINET*
	55.40	52.20	60.00	59.94	52.50	65.60	89.17

Table 5: Exact-match accuracy (%) of table QA methods on WikiTQ, WikiSQL, and HiTab. (Top) Selector-based QA results, where subtable selectors are combined with different readers. A dash (–) denotes the holistic setting in which readers process the full table without subtable selection. (Bottom) Results of additional holistic QA models for comparison. For *Dater* and *TabSQLify*, we also report results using their original *GPT-3.5* readers (numbers in parentheses), while selectors remain unchanged (see Appendix A.3). For each reader, the best and second-best selectors are highlighted in **bold** and underlined, respectively.

with *GPT-3.5* and *Gemini 2.0*. Despite being trained only on WikiTQ, *PieTa* surpasses all other selectors and holistic readers on this challenging dataset. It is particularly effective on hierarchical tables with nested headers, which are common in HiTab, demonstrating its robustness and adaptability to diverse table structures.

Overall, selector-based approaches achieve higher accuracy than the holistic QA models shown in the bottom section of the table, with *PieTa* consistently delivering the largest improvements. Furthermore, as shown in Figure 4, *PieTa* maintains strong table QA performance across varying table sizes. This robustness can be attributed to its ability to identify and merge dispersed relevant cells using a union-based selection strategy combined with multi-resolution integration. Further analysis based on the distance between answer cells (Figure 5) shows that existing approaches struggle when answer cells are far apart, whereas *PieTa* remains effective even in these challenging cases.

4.3 Ablation study

Fine-tuning vs. in-context Learning. We compare our fine-tuning approach for the LM selector with an alternative in-context learning method using a two-shot setup. As shown in Table 6, in-context learning results in significantly lower performance compared to fine-tuning. This is primarily because LMs that are not specifically configured for subtable selection, often deviate from the intended task, e.g. attempting to answer questions directly based on the subwindow content rather than generating relevant subtables. In contrast, fine-tuning aligns the selector's functionality with the subtable selection task and adapts the output format to a coordinate

Learning	Target repr.	Window size	EM
Fine-tune	Coord.	4×4	90.11
In-context	Coord.	4×4	87.60
Fine-tune	Index	4×4	84.85
Fine-tune	Table	4×4	89.44
Fine-tune	Coord.	3×3	90.11
Fine-tune	Coord.	5×5	89.71
Fine-tune	Coord.	$4\times n$	86.03

Table 6: Table QA performance across variations of our algorithm on the WikiSQL validation set (*TaPEx* reader). The first row represents our final method, integrating selector LM fine-tuning and a coordinate-based subtable representation. The results show that these design choices enhance performance individually and collectively.

Table size	Runtime (s)				Tokens (10^3)			
(# cells)	*ITR*	*Dater*	*TabSQLify*	*PieTa*	*ITR*	*Dater*	*TabSQLify*	*PieTa*
$0-50$	0.28	9.79	1.29	3.28	0.49	4.79	2.36	6.05
$50-100$	0.67	11.65	1.83	6.15	0.86	4.87	2.36	11.05
$100-200$	0.67	10.97	2.31	6.97	1.66	4.94	2.39	23.51
$200-400$	0.67	9.89	3.36	12.70	3.47	4.87	2.55	43.44
$400+$	0.70	9.81	8.22	34.93	10.28	4.82	2.34	182.24

Table 7: Average runtime and token usage for subtable selection algorithms across different table sizes on WikiTQ.

representation. This eliminates the need for lengthy prompts and improves overall performance.

Window size w. This is the only hyperparameter in *PieTa* and is set to 4×4 by default. To assess its impact, we evaluated window sizes of 3×3, 5×5, and $4\times n$, where $4\times n$ spans all columns (averaging approximately six columns; see Appendix B). As shown in Table 6, both 4×4 and 3×3 achieve strong performance, while 5×5 shows a slight decline, and $4\times n$ degrades noticeably. This suggests that larger windows make it more challenging for the selector to identify cells that satisfy all conditions.

Subtable representation. Table 6 shows that the proposed coordinate-based subtable representation significantly outperforms existing index and table representations in table QA performance.

4.4 Runtime and token consumption

The computational cost of *PieTa* scales linearly with the size of the input table. Table 7 presents the average runtime (in seconds) across different table sizes. For *PieTa*, over 99% of the total runtime is spent on selector inference. The runtime of *ITR* and *PieTa* is measured on a server with eight RTX 6000 Ada GPUs, while that of *Dater* and *TabSQLify* is measured using the gpt-3.5-turbo API. While *PieTa* incurs higher computational overhead than baseline methods, it delivers substantially stronger performance. Its iterative design also enables a flexible trade-off between accuracy and efficiency by adjusting the number of iterations (Table 12).

Token usage is also presented in Table 7, measured as the total number of tokens per query (input prompt plus generated output) using each language model's native tokenizer.

PieTa is well suited for settings where improved reliability justifies additional computation. It operates within a few seconds for small to moderate tables, with latency increasing for larger inputs. Recent cost- and latency-aware evaluations of tool use and planning in LLM-based QA highlight the trade-off between latency and reasoning robustness, with reported latencies spanning a similar range (Ghoshal and Al-Bustami, 2026). In practice, *PieTa* can be applied when higher accuracy is required and integrated with lighter baselines to balance latency and performance.

5 Conclusion

We have explored the challenges of subtable selection in table question answering, focusing on the limitations of existing approaches in capturing joint dependencies across rows and columns, and managing lengthy token sequences in language models. Our algorithm bypasses these challenges by dividing input tables into small windows and independently constructing subtables therein. The resulting multi-resolution windowing strategy effectively combines subtables through simple union operations, streamlining subtable selection while preserving long-range relationships without imposing unnecessary operational assumptions. Evaluated on the WikiTQ, WikiSQL and HiTab datasets, our method demonstrated significant improvements over state-of-the-art methods.

Limitations

Our study focused on improving the subtable selection process independently of the reader that generates the final answer. This approach offers the advantage of flexibility, as the selection process can be easily combined with any type of reader. For instance, our algorithm can directly enhance the performance of readers that generate both natural language answers and SQL queries. However, in principle, tailoring the subtable selector specifically for the reader could further improve performance. Future research should explore the possibility of jointly optimizing the subtable selection process and the reader.

Acknowledgments

This work was supported by Samsung Electronics Co., Ltd. (IO240508-09825-01) and by the Korea government (MSIT) through the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant (No. RS-2019-II191906, Artificial Intelligence Graduate School Program (POSTECH)) and the National Research Foundation of Korea (NRF) grant (No. RS-2024-00337559).

References

Gregor Bachmann and Vaishnavh Nagarajan. 2024. The pitfalls of next-token prediction. In *ICML*.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. 2015. Scheduled sampling for sequence prediction with recurrent neural networks. In *NeurIPS*.

Jiawei Chen, Hongyu Lin, Xianpei Han, and Le Sun. 2024. Benchmarking large language models in retrieval-augmented generation. In *AAAI*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code. *arXiv:2107.03374*.

Pei Chen, Soumajyoti Sarkar, Leonard Lausen, Balasubramaniam Srinivasan, Sheng Zha, Ruihong Huang, and George Karypis. 2023. HyTrel: Hypergraph-enhanced tabular data representation learning. In *NeurIPS*.

Wenhu Chen. 2023. Large language models are few(1)-shot table reasoners. In *Findings of EACL*.

Wenhu Chen, Hanwen Zha, Zhiyu Chen, Wenhan Xiong, Hong Wang, and William Yang Wang. 2020. HybridQA: A dataset of multi-hop question answering over tabular and textual data. In *Findings of EMNLP*.

Zhoujun Cheng, Haoyu Dong, Ran Jia, Pengfei Wu, Shi Han, Fan Cheng, and Dongmei Zhang. 2022a. FORTAP: Using formulas for numerical-reasoning-aware table pretraining. In *ACL*.

Zhoujun Cheng, Haoyu Dong, Zhiruo Wang, Ran Jia, Jiaqi Guo, Yan Gao, Shi Han, Jian-Guang Lou, and Dongmei Zhang. 2022b. HiTab: A hierarchical table dataset for question answering and natural language generation. In *ACL*.

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 *ICLR*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, and 82 others. 2024. The Llama 3 herd of models. *arXiv:2407.21783*.

Subha Ghoshal and Ali Al-Bustami. 2026. When do tools and planning help llms think? A cost-and latency-aware benchmark. *arXiv:2601.02663*.

Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Eisenschlos. 2020. TaPas: Weakly supervised table parsing via pre-training. In *ACL*.

Mohit Iyyer, Wen-tau Yih, and Ming-Wei Chang. 2017. Search-based neural structured learning for sequential question answering. In *ACL*.

Deyi Ji, Lanyun Zhu, Siqi Gao, Peng Xu, Hongtao Lu, Jieping Ye, and Feng Zhao. 2024. Tree-of-Table: Unleashing the power of LLMs for enhanced large-scale table understanding. *arXiv:2411.08516*.

Jinhao Jiang, Kun Zhou, Zican Dong, Keming Ye, Xin Zhao, and Ji-Rong Wen. 2023. StructGPT: A general framework for large language model to reason over structured data. In *EMNLP*.

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 *NAACL*.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *EMNLP*.

Vishwajeet Kumar, Yash Gupta, Saneem Chemmengath, Jaydeep Sen, Soumen Chakrabarti, Samarth Bharadwaj, and Feifei Pan. 2023. Multi-row, Multi-span distant supervision for Table+Text question answering. In *ACL*.

Alex Lamb, Anirudh Goyal, Ying Zhang, Saizheng Zhang, Aaron Courville, and Yoshua Bengio. 2016. Professor forcing: A new algorithm for training recurrent networks. In *NeurIPS*.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *ACL*.

Peng Li, Yeye He, Dror Yashar, Weiwei Cui, Song Ge, Haidong Zhang, Danielle Rifinski Fainman, Dongmei Zhang, and Surajit Chaudhuri. 2024. Table-GPT: Table fine-tuned GPT for diverse table tasks. In *Proc. ACM Management of Data (PACMMOD)*.

Weizhe Lin, Rexhina Blloshmi, Bill Byrne, Adria de Gispert, and Gonzalo Iglesias. 2023. An inner table retriever for robust table question answering. In *ACL*.

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 *ICLR*.

Tianyang Liu, Fei Wang, and Muhao Chen. 2024. Rethinking tabular data understanding with large language models. In *NAACL*.

Md Mahadi Hasan Nahid and Davood Rafiei. 2024. TabSQLify: Enhancing reasoning capabilities of LLMs through table decomposition. In *NAACL*.

Linyong Nan, Chiachun Hsieh, Ziming Mao, Xi Victoria Lin, Neha Verma, Rui Zhang, Wojciech Kryściński, Nick Schoelkopf, Riley Kong, Xiangru Tang, Murori Mutuma, Ben Rosand, Isabel Trindade, Renusree Bandaru, Jacob Cunningham, Caiming Xiong, and Dragomir Radev. 2022. FeTaQA: Free-form table question answering. In *ACL*.

Chaoxu Pang, Yixuan Cao, Chunhao Yang, and Ping Luo. 2024. Uncovering limitations of large language models in information seeking from tables. In *Findings of ACL*.

Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables. In *ACL-IJCNLP*.

Sohan Patnaik, Heril Changwal, Milan Aggarwal, Sumit Bhatia, Yaman Kumar, and Balaji Krishnamurthy. 2024. CABINET: Content relevance-based noise reduction for table question answering. In *ICLR*.

Stephen Robertson, Hugo Zaragoza, and 1 others. 2009. The probabilistic relevance framework: BM25 and beyond. *Found. Trends Inf. in Information Retrieval*.

Stephane Ross and Drew Bagnell. 2010. Efficient reductions for imitation learning. In *ICML*.

Tianze Shi, Chen Zhao, Jordan Boyd-Graber, Hal Daumé III, and Lillian Lee. 2020. On the potential of lexico-logical alignments for semantic parsing to SQL queries. In *Findings of EMNLP*.

Zhiruo Wang, Haoyu Dong, Ran Jia, Jia Li, Zhiyi Fu, Shi Han, and Dongmei Zhang. 2021. TUTA: Tree-based transformers for generally structured table pre-training. In *KDD*.

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. 2024. Chain-of-Table: Evolving tables in the reasoning chain for table understanding. In *ICLR*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-Thought prompting elicits reasoning in large language models. In *NeurIPS*.

Jian Wu, Linyi Yang, Dongyuan Li, Yuliang Ji, Manabu Okumura, and Yue Zhang. 2025a. MMQA: Evaluating LLMs with multi-table multi-hop complex questions. In *ICLR*.

Xianjie Wu, Jian Yang, Linzheng Chai, Ge Zhang, Jiaheng Liu, Xeron Du, Di Liang, Daixin Shu, Xianfu Cheng, Tianzhen Sun, Guanglin Niu, Tongliang Li, and Zhoujun Li. 2025b. TableBench: A comprehensive and complex benchmark for table question answering. In *AAAI*.

Yuanzhen Xie, Xinzhou Jin, Tao Xie, Mingxiong Lin, Liang Chen, Chenyun Yu, Lei Cheng, Chengxiang Zhuo, Bo Hu, and Zang Li. 2024. Decomposition for enhancing attention: Improving LLM-based text-to-SQL through workflow paradigm. In *ACL*.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, and 22 others. 2024. Qwen2.5 technical report. *arXiv:2412.15115*.

Jingfeng Yang, Aditya Gupta, Shyam Upadhyay, Luheng He, Rahul Goel, and Shachi Paul. 2022. TableFormer: Robust transformer modeling for table-text encoding. In *ACL*.

Zhen Yang, Ziwei Du, Minghan Zhang, Wei Du, Jie Chen, Zhen Duan, and Shu Zhao. 2025. Triples as the Key: Structuring makes decomposition and verification easier in LLM-based TableQA. In *ICLR*.

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 *SIGIR*.

Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. TaBERT: Pretraining for joint understanding of textual and tabular data. In *ACL*.

Yunjia Zhang, Jordan Henkel, Avrilia Floratou, Joyce Cahoon, Shaleen Deep, and Jignesh M. Patel. 2024. ReAcTable: Enhancing ReAct for table question answering. In *Proc. VLDB Endow.*

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2SQL: Generating structured queries from natural language using reinforcement learning. *arXiv:1709.00103*, arXiv:1709.00103.

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 *ACL*.

A Implementation Details

A.1 Subwindow sampling process

Algorithm 2 details the window generation process from an input table. The DIVIDETABLE algorithm

generates a set of subwindows $W = \{W_i\}_{i=1}^{N}$ by sliding a window of size $w \times w$ across the input table T with a stride of 1.

Algorithm 2 DIVIDETABLE

Input: Table T with R rows and C columns
Parameters: Window size w
Output: Set of windows $W = \{W_i\}_{i=1}^{N}$
$W \leftarrow \varnothing$
for $i = 0$ to R **by** 1 **do**
  for $j = 0$ to C **by** 1 **do**
    if $i + w > R$ **then**
      $i = R - w$
    end if
    if $j + w > C$ **then**
      $j = C - w$
    end if
    $W_i = T[i : i + w, j : j + w]$ ▷ Window slicing
    $W = W \cup \{W_i\}$
  end for
end for

A.2 Input prompt structure for selector $\mathcal{S}$

The input prompt for selector $\mathcal{S}$ consists of both fixed and variable components, as shown in Fig. 6. The variable components, highlighted in blue and green, represent the question and the table window, respectively. The fixed components remain unchanged across all tables and questions.

A.3 Configurations of the general-purpose readers *GPT-3.5* and *Gemini 2.0*

For our experiments with the general-purpose readers *GPT-3.5* and *Gemini 2.0* in both the holistic setting and subtable-based QA (Table 4 of the main paper), we configure the models to generate both SQL queries and textual output answers, following the strategy of *DP&Agent* (Liu et al., 2024): When SQL execution returns a single-cell result, the cell's content is taken as the final answer. If the query produces multiple cells or fails to execute, we instead use the generated textual output as the answer.

We apply in-context learning for both output types: 10-shot learning for SQL outputs following (Nahid and Rafiei, 2024), and two-shot learning for textual outputs, consistent with the experimental setup for holistic readers in (Nahid and Rafiei, 2024). The prompts used in these experiments are shown in Figure 7.

This setup differs from the original configuration of *GPT-3.5* when combined with subtable selection in *TabSQLify* (Nahid and Rafiei, 2024) and *Dater* (Ye et al., 2023). In *TabSQLify*, the SQL query used to extract the subtable is incorporated into the reader prompt. In addition, when the extracted subtable contained only a single cell, it is omitted as input for the reader, and the cell itself was taken directly as the answer. In contrast, *Dater*'s reader processes not only the subtable but also the outputs of a question decomposer, emphasizing question decomposition to effectively handle complex queries.

For a fair comparison, we also report results using the original *GPT-3.5* configurations from *TabSQLify* and *Dater*, shown in parentheses in Table 4 of the main paper. For *DP&Agent*, we mitigate hallucinations in *GPT-3.5* by providing the agent with the following additional guideline: `**Stay focused on the given question**: You must base your answer strictly on the given question. After the observation step, do not generate or infer new questions`.

B Additional Experimental Details

B.1 Training settings

All experiments are conducted on a server with 4×NVIDIA RTX 6000 Ada GPUs, 1TiB RAM, and an AMD EPYC 7763 CPU running Ubuntu 20.04.6. We use PyTorch 2.5.1, Torchtune 0.6.0, vLLM 0.6.4, and HuggingFace Transformers 4.48.0 with CUDA 12.1.

The selector is trained for a single epoch with the AdamW optimizer with an initial learning rate of 10^{-5}, a batch size of 8, and 4 gradient accumulation steps. The training process takes approximately two hours for WikiSQL and WikiTQ datasets.

B.2 Datasets

The WikiSQL and WikiTQ datasets are built from tables extracted from Wikipedia, covering diverse domains. WikiTQ includes operations such as superlatives, aggregations, and comparisons, with 4,344 test examples. On average, the test tables contain 6.3 columns and 25.8 rows, with some tables reaching up to 21 columns and 517 rows. WikiSQL features simpler questions involving aggregation, column selection, and conditions. It contains 15,878 test examples, with an average table size of 6.4 columns and 18.6 rows, and a maximum of 23 columns and 1,950 rows. Both datasets are widely used in table QA research, including (Lin et al., 2023; Nahid and Rafiei, 2024).

We generate training data by extracting condition and answer columns from SQL queries annotated in WikiSQL (Zhong et al., 2017) and SQUALL (Shi et al., 2020). These columns define the target subwindow V within each window W (Section 3.2).

Prompt

Below is an instruction that describes a task, paired with an input that provides further context.
Write a response that appropriately completes the request.

Instruction:
This is a task of extracting subtable which has gold information to answer the question from a given table. Specifically, you have to select rows and columns which have the information you need. Then, generate the subtable with those rows and columns. Each cell is indicated through <row_index,column_index>. Indicate unnecessary cells as <empty,empty>.

Input:
Question: Who played in the Toronto Raptors from 1995-96?
Table:
Col: <0,1> Player | <0,2> School/Club Team | <0,3> Years in Toronto | <0,4> No.
<1,1> patrick o'bryant | <1,2> bradley | <1,3> 2009-10 | <1,4> 13.0
<2,1> jermaine o'neal | <2,2> eau claire high school | <2,3> 2008-09 | <2,4> 6.0
<3,1> dan o'sullivan | <3,2> fordham | <3,3> 1995-96 | <3,4> 45.0
<4,1> hakeem olajuwon | <4,2> houston | <4,3> 2001-02 | <4,4> 34.0

Subtable

Response:
<0,1> | <empty,empty> | <0,3> | <empty,empty>
<empty,empty> | <empty,empty> | <empty,empty> | <empty,empty>
<empty,empty> | <empty,empty> | <empty,empty> | <empty,empty>
<3,1> | <empty,empty> | <3,3> | <empty,empty>
<empty,empty> | <empty,empty> | <empty,empty> | <empty,empty>

Figure 6: An example of a prompt. Given a question Q and a window W, the prompt generator $\mathcal{P}$ creates a textual prompt (top). This prompt is fed to the subwindow selector $\mathcal{S}'$ to construct the output subwindow (bottom). The complete question Q and input window W are highlighted in blue and green, respectively. In the instruction, the input and target windows are referred to as a *table* and *subtable*, respectively.

SQUALL provides SQL annotations for 11,468 questions in WikiTQ.

HiTab complements WikiSQL and WikiTQ by introducing hierarchical tables with multi-level headers and nested structures, commonly found in real-world documents such as financial reports and government statistics. It comprises 1,584 test examples across 540 unique tables, with an average of 9.9 columns and 12.8 rows, and up to 22 columns and 62 rows. HiTab emphasizes numerical reasoning and the interpretation of structurally complex tables.

Our experiments follow the experimental protocol and evaluation metric of Jiang et al. (2022), using a single run for each of the WikiTQ and WikiSQL datasets on clearly separated training, validation, and test sets.

B.3 Algorithms compared

For the table QA experiments (Table 4 of the main paper), we reproduced the results for *OmniTab*, *CABINET*, *GPT-3.5*, *ITR*, *TabSQLify* and *Dater* (excluding the original *GPT* reader configuration used in *Dater*) using the code and services (for *GPT*-based models) provided by the respective authors.

For *Dater* and *Binder*, the original studies used *Codex* (Chen et al., 2021), which was not available during our experiments. Instead, we report the results from (Nahid and Rafiei, 2024) obtained with *GPT-3.5*. For other algorithms, we report the results as presented in their original papers.

B.4 Data augmentation

We augmented the selector training data by labeling cells with the same value as target cells within the window as targets in the training dataset. This augmentation was applied to 8% of the data in WikiTQ and 21% in WikiSQL.

B.5 Taining data sampling details

In our preliminary experiments, we observed that randomly sampling training data (pairs of input W_i and the corresponding target V_i subwindows) often caused $\mathcal{S}'$ to generate meaningless single-column target tables. This issue arises due to an imbalance in the distribution of target window sizes.

For a given input window W of size $w \times w$, there are $w(w+1)$ possible target window sizes: $\{[1,\,1],\ldots,[1,\,w],\ldots,[w,\,1],\ldots,[w,\,w]\}$. Cases such

Prompt

Based on the table information below, find the answer to the given question correctly.

Table_title: {table title}
Table:
{table or subtable}

Question: {question}
A: {reasoning process}
Answer: {short-form answer}

Prompt

Generate SQL for selecting the required rows and columns, given the question and table to answer the question correctly.
SQLite table properties:

Table: {table title} ({table column})
{table or subtable}
3 example rows:
select * from T limit 3;
{first 3 rows of table}
Q: {question}
SQL:

Figure 7: Template prompts for the holistic *GPT-3.5* reader to generate textual answers (top) and SQL queries (bottom), adapted from (Nahid and Rafiei, 2024). Each {} placeholder in the prompt is replaced with the corresponding variable components.

as $[0, 0]$, $[1, 0]$, $[2, 0]$, ..., and $[w, 0]$ are excluded, as a table must contain at least one column. Among the valid cases, single-column target windows dominate (Figure 8), while cases where the entire input window W becomes the target are rare. The latter scenario requires all columns in W to be either condition or answer columns, with every cells satisfying the given question. To address this imbalance, we explicitly adjust the target window size distribution when sampling the training set.

To achieve a balanced distribution of the number of rows and columns in target subtables, we evenly match their occurrences. We first determine the number of columns $n \in \{1, 2, \ldots, w\}$ in the target subtable and designate them as either condition or answer columns. The remaining $w - n$ columns are filled with non-condition, non-answer columns. Similarly, for the rows, we select m rows that satisfy the condition and fill the remaining $w - m$ rows with cells that do not meet the condition. By evenly sampling n and m, we can generate a well-balanced dataset. To ensure reproducibility, we fix the random seed to 42 during the sampling process.

Selector		EM (%)
	# rows	
TabSQLify	3 (original)	59.71
	10	60.52
	20	61.58
	Full	61.79
PieTa (Ours)		63.65

Table 8: Table QA performance of *TabSQLify* (EM; %) across different numbers of rows, compared with our method (*PieTa*). Both methods use *GPT-3.5* readers (with the original *TabSQLify* configuration). See Table 4 of the main paper for additional comparisons.

C Additional Results

C.1 Additional results for *TabSQLify*

TabSQLify selects a subset of rows (specifically, the first three rows with all columns) as input for LMs to generate SQL queries for subtable selection. A key limitation of this approach is that if the initial rows do not contain all necessary conditions, the algorithm may sample irrelevant columns, leading to suboptimal selections. Increasing the number of extracted rows could help mitigate this issue, but it also

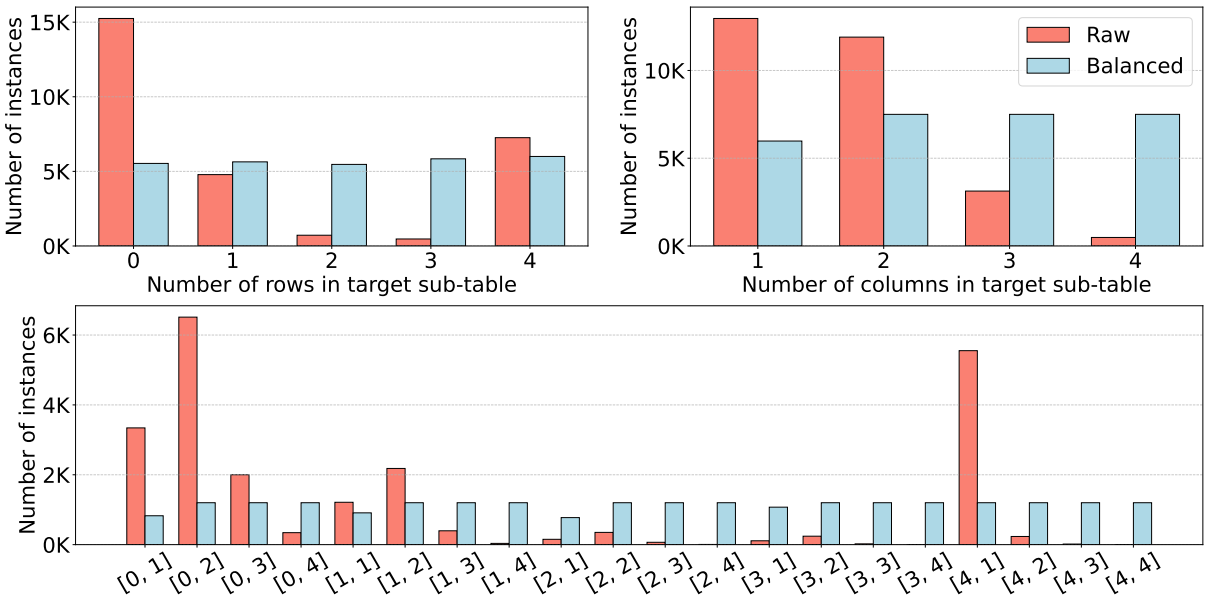

Figure 8: Histograms depicting the distribution of the generated 4×4 window training data for WikiSQL. The 'raw' histogram represents the original, unfiltered data, while the 'balanced' histogram shows the adjusted data used to mitigate class imbalance. In the bottom row, the x-axis represents [the number of answer rows, the number of condition & answer columns]. Note that the y-axis scales vary across the histograms.

burdens the LM selector with longer context processing, ultimately resulting in plateauing performance that remains significantly lower than ours (Table 8).

C.2 Performance by query types and failure cases

We analyze the behavior of subtable selection algorithms across different question types in Table 9. Overall, *PieTa* consistently achieves high recall across all categories, demonstrating its effectiveness in preserving question relevant information during subtable construction.

At the same time, the results reveal systematic variations in precision across question types. In particular, for *Find* type questions, where the ground truth subtable often comprises only a few cells and sometimes a single cell, *PieTa* tends to generate larger subtables than necessary. This behavior stems from its union based construction strategy, which prioritizes coverage over compactness.

Despite this limitation, *PieTa* substantially outperforms *ITR* and *Dater* in precision while maintaining near perfect recall. Although *TabSQLify* achieves higher precision overall, it does so at the cost of recall, which is critical in table QA settings. In contrast, *PieTa* provides a more favorable precision recall trade off, effectively filtering out irrelevant content while reliably retaining all information required to answer the query.

C.3 Simple baselines for subtable selection

To assess the inherent difficulty of subtable selection in table QA and to contextualize the benefits of LM-based approaches, we evaluate two simple, non-learned baselines: a rule-based lexical matching method and an iterative binary search strategy. These baselines rely solely on surface-level similarity and serve as reference points for understanding the limitations of straightforward selection heuristics.

Rule-based selection using BM25. We first consider a rule-based retrieval baseline based on the Best Matching 25 (*BM25*) algorithm (Robertson et al., 2009). *BM25* is an information retrieval method that ranks documents according to their lexical similarity to a query. In our setting, the question serves as the query, while each table row or column is treated as a document. Rows or columns whose *BM25* scores exceed a predefined threshold are selected.

We evaluate three variants: *Rule-Row*, which selects rows; *Rule-Col*, which selects columns, and *Rule-Inter*, which takes the intersection of the selected rows and columns. Table 10 reports results on the WikiSQL test set under varying thresholds: All rule-based variants exhibit substantially lower recall than LM-based methods, including *PieTa* (Table 2). High recall is critical for table QA,

Type	*ITR*		*Dater*		*TabSQLify*		*PieTa (Llama)*	
	(P)	(R)	(P)	(R)	(P)	(R)	(P)	(R)
Find	4.15	97.68	9.41	92.33	70.71	87.74	56.44	99.33
Count	23.50	97.84	27.39	91.88	52.27	57.97	72.41	96.43
Max	38.36	98.29	42.35	97.10	79.35	89.72	72.41	99.56
Min	35.56	98.09	41.04	96.55	81.24	92.42	73.03	99.36
Average	49.51	98.63	51.94	98.00	86.53	90.98	76.42	99.86
Sum	54.15	98.91	56.61	98.61	83.85	86.90	80.24	99.91

Table 9: Performance of subtable selection algorithms across query types. (P) and (R) denote precision and recall, respectively.

Rule-based selection									*PieTa (Llama)*	
Threshold	Avg. # rows	Avg. # columns	*Rule-Row*		*Rule-Col*		*Rule-Inter*			
			(P)	(R)	(P)	(R)	(P)	(R)	(P)	(R)
0.5	18.28	2.59	22.94	69.02	17.69	86.21	42.81	80.58	60.88	99.12
1.0	10.04	2.29	24.01	65.34	18.02	81.74	47.01	76.47		
1.5	5.57	1.79	22.40	55.85	16.81	59.62	42.15	59.29		
2.0	4.74	1.53	21.04	48.21	15.72	45.51	37.43	48.81		
2.5	4.07	1.33	19.21	39.97	14.62	36.74	31.88	41.71		
3.0	3.60	1.15	17.41	32.85	13.61	30.20	26.73	35.72		

Table 10: Performance of rule-based subtable selection on the WikiSQL test set with varying *BM25* thresholds. We evaluate row-wise selection (*Rule-Row*), column-wise selection (*Rule-Col*), and the intersection of the selected rows and columns (*Rule-Inter*).

Tree depth	(P)	(R)
1	15.41	78.79
2	18.85	64.72
3	23.50	54.92
4	28.81	47.60
5	50.24	28.62

Table 11: Performance of the iterative binary search based subtable selection strategy on the WikiSQL test set at five different tree depths.

as missing relevant rows or columns leads to incomplete evidence and limits downstream reasoning. Increasing the *BM25* threshold reduces table size but consistently degrades recall, while lowering the threshold substantially reduces precision. This illustrates a fundamental limitation of purely lexical filtering, which lacks semantic understanding.

Iterative binary search. To contrast with our window-based iterative framework, we also evaluate a binary search strategy that greedily reduces table size. At each iteration, the table is divided in half along both row and column dimensions. For each row-wise split, we compute the Jaccard similarity between the query and each half. The same procedure is applied to the column-wise split. Among these resulting candidate subtables, the one with the highest Jaccard similarity is selected. This process is repeated for a predefined number of iterations.

Table 11 summarizes the results on the WikiSQL test set across five different search depths. While shallow searches preserve relatively high recall, deeper searches quickly discard relevant content, leading to pronounced recall degradation due to the brittleness of greedy, similarity-based partitioning, where early decisions irreversibly remove necessary evidence.

D Manual Annotation of WikiTQ and HiTab Test Sets

We recruited graduate student annotators in NLP with direct experience working with LLMs. Each annotator was provided with an instruction along with

	PieTa (Llama)								*ITR*	*Dater*	*TabSQLify*
Max iteration	1	2	3	4	5	6	7	∞	–	–	–
Runtime	4.18	5.98	6.62	6.74	6.78	6.81	6.84	6.87	0.30	9.85	1.91
Accuracy	64.41	64.73	64.64	64.71	64.78	64.78	64.78	64.78	63.47	57.09	60.01

Table 12: Table QA accuracy (EM, %) and runtime (in seconds) of *PieTa* on WikiTQ across different maximum iteration settings. ∞ denotes running the algorithm until convergence, without an explicit iteration cap.

a table-question pair and asked to identify all header and content cells required to answer the question (Fig. 9). The annotation criterion focuses on identifying the minimal set of cells necessary to derive the correct answer. Each example was independently annotated. No compensation was provided.

```
Given a table and a question, identify the relevant headers and cells required to answer the question.

Represent each selected cell by its coordinate in the form (row, column), where both row and column indices start from 0.

Output only the set of selected coordinates.

Table: {table}
Question: {question}
```

Figure 9: Instruction provided to human annotators for identifying all header and content cells required to answer a given question.

E Potential Risks

Since *PieTa* relies on pre-trained language models (LMs) to extract relevant subtables, it may inherit and reflect any biases present in these models. Likewise, as *PieTa* learns to identify subtables based on training data, it may inadvertently reinforce existing biases within the dataset.

While we do not foresee specific problematic scenarios, subtable selection inherently carries the risk of introducing unintended bias due to the selective sampling of table contents. Furthermore, in adversarial settings, subtable selection systems could be exploited to selectively present data that supports a particular agenda while omitting contradictory evidence.