NOAH: A System for Analyzing Non-relational Tables Using Natural Language

LINAN ZHENG, University of Arizona, USA
CHUAN LEI, NTT DATA AIVista, USA
LEI CAO, University of Arizona, USA

Non-relational tables, such as spreadsheets and web tables, are widely used to store structured data, yet they are primarily designed for human reading rather than machine-driven analysis. These tables often contain hierarchical headers, merged cells, and implicit spatial semantics, making it difficult for large language models (LLMs) to correctly interpret their structure. Moreover, real-world non-relational tables are often large, leading to prohibitive token costs and poor scalability when processed directly by LLMs. To address these challenges, we present NOAH, a system for accurate and scalable analysis of large non-relational tables using natural language queries. Rather than loading the entire table into an LLM or transforming it into a relational format, NOAH constructs a Hierarchical Semantic Index (HSI) in an offline phase that explicitly captures the structure of row and column headers. HSI enables query-driven attribute linking, allowing NOAH to retrieve only the data regions relevant to a given query. Building on HSI, NOAH employs a context-aware data analysis pipeline that progressively resolves complex analytical queries through dynamic operator selection. To further improve efficiency, NOAH incorporates embedding-based hierarchy search, together with a recall-guaranteed dynamic thresholding strategy. We evaluate NOAH on real-world benchmarks for non-relational table analysis. Experimental results demonstrate that NOAH outperforms the best-performing baseline by 31.2% in accuracy, while scaling effectively to large tables and reducing LLM token consumption by at least 3.3×.

CCS Concepts: • **Information systems** → **Data management systems**.

Additional Key Words and Phrases: Non-relational Tables, Natural Language Queries, Semantic Index

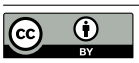
ACM Reference Format:
Linan Zheng, Chuan Lei, and Lei Cao. 2026. NOAH: A System for Analyzing Non-relational Tables Using Natural Language. *Proc. ACM Manag. Data* 4, 4 (SIGMOD), Article 282 (September 2026), 28 pages. https://doi.org/10.1145/3837120

1 Introduction

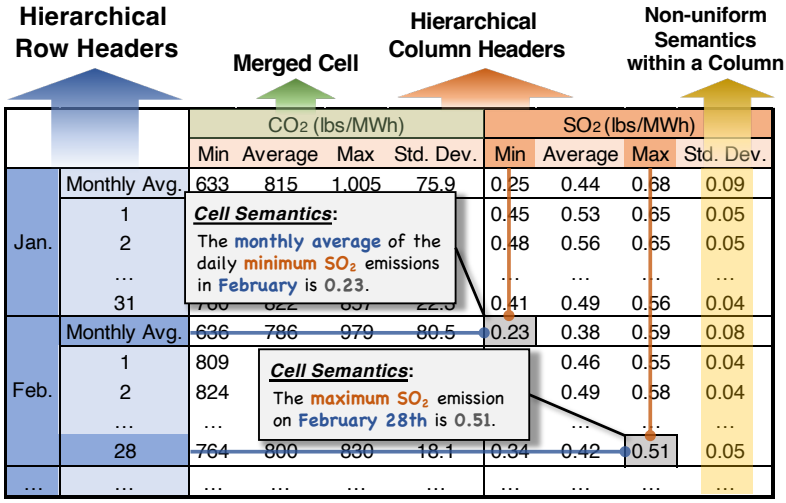
A vast amount of structured data in science, engineering, and the public sectors is stored in tables that are primarily designed to be *human-readable*, such as spreadsheets and web tables [7, 20]. These tables are widely used to organize and communicate information ranging from product sales and economic indicators to scientific measurements and population statistics. However, their layout and schema conventions are optimized for human readability rather than automated machine interpretation and analysis.

This mismatch between *human-readable table design* and *machine-readable data analysis* has become increasingly problematic in the era of AI. Recent advances in large language models (LLMs) have significantly improved natural language interfaces for data analysis, allowing users to pose complex analytical questions without writing code or SQL [4, 36, 41]. While these techniques

Authors' Contact Information: Linan Zheng, University of Arizona, USA, linanzheng@arizona.edu; Chuan Lei, NTT DATA AIVista, USA, chuan.lei@nttdata-aivista.com; Lei Cao, University of Arizona, USA, lcao@csail.mit.edu.



This work is licensed under a Creative Commons Attribution 4.0 International License.
© 2026 Copyright held by the owner/author(s).
ACM 2836-6573/2026/9-ART282
https://doi.org/10.1145/3837120



		CO_2 (lbs/MWh)				SO_2 (lbs/MWh)			
		Min	Average	Max	Std. Dev.	Min	Average	Max	Std. Dev.
Jan.	Monthly Avg.	633	815	1,005	75.9	0.25	0.44	0.68	0.09
	1					0.45	0.53	0.65	0.05
	2					0.48	0.56	0.65	0.05
	…					…	…	…	…
	31					0.41	0.49	0.56	0.04
Feb.	Monthly Avg.	636	786	979	80.5	0.23	0.38	0.59	0.08
	1	809					0.46	0.55	0.04
	2	824					0.49	0.58	0.04
	…	…					…	…	…
	28	764	800	830	18.1	0.34	0.42	0.51	0.05
…	…	…	…	…	…	…	…	…	…

Fig. 1. A non-relational table with complex schema.

have shown strong performance on relational tables via NL2SQL pipelines [38, 49, 52], analyzing non-relational tables using natural language remains challenging due to their *complex schema* and the *context length limitation* of LLMs.

Unlike relational tables, where explicit schema and rich metadata facilitate semantic understanding and reasoning, the schemas of non-relational tables are often implicit, hierarchical, and spatially encoded through multi-level headers, merged cells, indentation, or visual grouping, as shown in Figure 1. As a result, LLMs struggle to correctly interpret table structure and to identify relevant data regions when reasoning directly over such inputs. This gap raises a fundamental question: *How can we transform human-readable non-relational tables into representations that are both easy for AI systems to construct and effective for AI systems to analyze?*

A natural approach is to convert non-relational tables into relational tables prior to analysis. Recent work, such as AutoTables [23], explores automatically relationalization of spreadsheets to enable downstream SQL-based querying. However, hard spreadsheets often do not neatly fit a single relational schema, and automatic relationalization becomes increasingly difficult as transformation complexity grows. For example, AutoTables reports that the relationalization success rate on ATBench drops from 59.2% for tables requiring one transformation step to 38.5% for tables requiring two steps [24]. On RealHiTBench [60], a benchmark of real-world spreadsheets, AutoTables successfully relationalizes only 6.67% of tables. Even when relationalization succeeds, it can introduce unnatural redundancy. For example, the "Monthly Avg." column in Figure 1, which aggregates daily metrics, would be repeated across many daily rows. These limitations suggest that forcing non-relational tables into the relational paradigm is often insufficient.

Beyond schema complexity, scalability poses an additional challenge. Non-relational tables are often large, leading to excessive token consumption or even exceeds the context limits of LLMs. Existing LLM-based methods for non-relational table analysis, such as SheetAgent [11] and SheetCopilot [22], inject the full table into the LLM context for every query, resulting in high token usage and poor scalability as table size increases. Moreover, simply adopting long-context LLMs does not fundamentally resolve this issue. Prior work shows that model performance often degrades as context length grows [18, 26], due to the "lost in the middle" issue [30, 42] and the misleading irrelevant information in the table. Consequently, how to analyze large non-relational tables *efficiently* while preserving accuracy remains a critical open challenge.

Our Proposal. To address these challenges, we propose NOAH, an LLM-based system for accurate and scalable analysis of non-relational tables. Rather than directly processing the raw table or attempting to transform an entire non-relational table into an alternative format (e.g., a relational schema) [24, 46, 50, 56], we build a *Hierarchical Semantic Index* (HSI) that captures the *hidden attributes* in a non-relational table and model their hierarchical relationships using a tree structure. NOAH transforms an implicit, human-readable schema into an explicit and machine-interpretable representation. Moreover, by serving as an index, HSI enables on-demand access to query-relevant schema elements and data regions. Because only a small subset of a table is typically relevant to a given analytical query, this design avoids placing the full table in the LLM context and directly addresses the scalability challenges of LLM-based data analysis. Excluding irrelevant information also improves analysis accuracy by reducing spurious distractions in the LLM context. NOAH consists of two main components: *offline schema indexing* and *dynamic context-aware data analysis*.

Offline Schema Indexing. NOAH constructs the HSI offline once per table using an *Iterative Hierarchy Construction* algorithm, which recursively identifies and assembles header cells into an HSI tree based on their spatial and semantic relationships, without requiring LLM processing over the full table. At query time, HSI offers an *attribute linking* operator that evaluates the semantic relevance between a user query and HSI nodes corresponding to implicit table attributes, selectively retrieving only the data regions covered by the most relevant attributes. This design avoids full-table loading and transformation, substantially reducing the computational cost while preserving analytical accuracy.

As the HSI itself may be large, NOAH does not expose the entire index to the LLM during attribute linking. Instead, it employs an embedding-based similarity search to filter the HSI nodes irrelevant to the query. We further introduce a dynamic thresholding mechanism that adaptively determines the similarity cutoff based on the score distribution, pruning as many nodes as possible while guaranteeing high recall of relevant nodes.

To further facilitate LLM reasoning over the retrieved subtable [5], NOAH does not present it directly to the LLM. Instead, it organizes the retrieved content into an *AI-ready* representation that explicitly annotates each data cell with its semantic context, eliminating the need for the LLM to reason over the original complex schema.

Dynamic Context-aware Data Analysis. To efficiently support complex analytical queries over large non-relational tables, NOAH adopts an *Dynamic Context-aware Data Analysis* paradigm. Instead of attempting to answer a query in a single step, NOAH models data analysis as an iterative reasoning process that progressively refines both the query intent and the accessed data. NOAH conducts analysis through a sequence of context-driven operation rounds, where each round incrementally refines the analytical focus, e.g., rewriting a complex query, retrieving relevant data, or generating intermediate answers, based on the evolving analytical context.

In each round, NOAH orchestrates a set of analytical operators, including *Query Rewrite*, *Data Retrieval*, and *Answer Generation*. A key design principle is that analytical operators are selected dynamically based on the current context, rather than following a fixed, pre-computed execution plan. Unlike traditional query processing pipelines or static LLM-based workflows, where the operator sequence is determined upfront, analysis over non-relational tables is inherently exploratory. Therefore, NOAH continuously adapts its operator choices according to the evolving context, which captures the query intent, retrieved data and intermediate results.

Beyond deciding *which* operator to perform, NOAH also determines *how* each operator should be instantiated. For example, when precise numerical computation or structured transformation is required, NOAH prefers code generation to ensure accuracy and reliability. In contrast, when the task involves interpretation, summarization, or reasoning, NOAH invokes the LLM to generate textual responses. These decisions are made on the fly by examining the evolving analytical context.

Contributions. The main contributions of this paper include:

- To the best of our knowledge, NOAH is the first framework that jointly address accuracy and scalability for natural language-based analysis of non-relational tables.
- We propose the Hierarchical Semantic Index (HSI), which captures the complex schema of non-relational tables and enables efficient, query-driven data retrieval. A hybrid search strategy further reduces LLM reasoning cost while preserving high recall.
- We design a context-driven analysis framework that progressively resolves complex analytical queries through adaptive operator orchestration, augmented with context pruning to improve both efficiency and answer accuracy.
- Our extensive experiments on real-world non-relational tables demonstrate that NOAH achieves high accuracy in data analysis while scaling effectively to large tables. In particular, NOAH outperforms the best-performing baseline by 31.2% in accuracy, while reducing LLM token usage per query by at least 3.3×.

2 NOAH Overview

2.1 Problem Formulation

Given a non-relational table $\mathcal{T}$ and a natural language query q, the goal of NOAH is to correctly answer q using the data in $\mathcal{T}$ while satisfying the following key requirements:

- **Accuracy:** correctly infer the query intent and capture the table semantics to produce accurate answers.
- **Scalability:** support large, non-relational tables whose size exceeds the context length of LLMs.
- **Cost Efficiency:** minimize LLM-related costs, including token usage and inference latency.

Supported Tables. We focus on hard non-relational tables.

Definition 2.1 (Hard Non-relational Table)**.** A non-relational table is a "hard" table if it satisfies at least one of the following properties: (a) *Hierarchical headers:* headers form multi-level parent–child structures, so that multiple headers across one or both dimensions jointly determine the semantics of a data cell; (b) *Bi-dimensional hierarchical headers:* the header cells defining the spreadsheet schema are organized along both the row and column dimensions; or (c) *Non-uniform column semantics:* cells within the same column contain heterogeneous types of content, such as raw values or aggregates.

Such non-relational tables are common: all 90 RealHiTBench [60] tables are hard tables, and prior work reports that fewer than 3% of WebSheet tables have a clear relational schema [16].

Supported Queries. We focus on analytical queries over a single table that can be expressed as compositions of common data analysis operations, primarily `aggregation` and `filter`. Specifically, NOAH supports standard aggregate functions such as `Min`, `Max`, and `Sum`, as well as filtering predicates including comparisons, range conditions, and conjunctions or disjunctions of multiple predicates. For example, NOAH can answer queries such as: *"Among January and February, which month exhibits the smallest overall variation in SO_2 emission and which day within that month records the highest average CO_2 level?"*. The final answer is formatted as a structured relational table; for this query, the table has columns "Month" and "Day", and one row: "February" and "4th".

2.2 System Workflow

As illustrated in Figure 2, NOAH enables natural language–based analysis of non-relational tables through a two-phase workflow that combines *offline schema indexing* with *online, context-aware query execution*. At a high level, NOAH builds a *Hierarchical Semantic Index (HSI)* for each table offline, and leverages this index at query time to drive an LLM-powered, adaptive query execution pipeline.

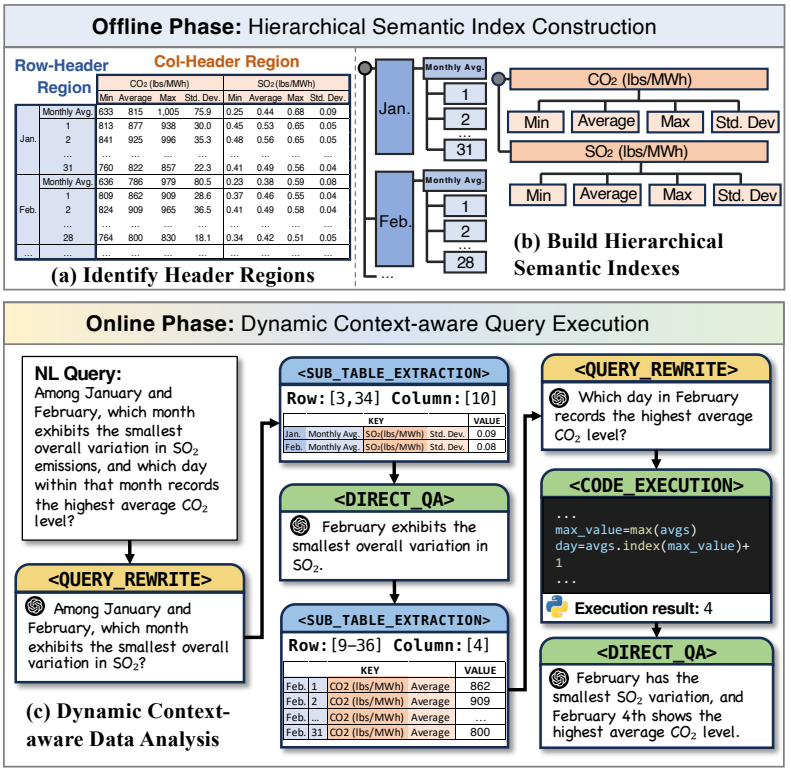


Fig. 2. The end-to-end workflow of NOAH.

Offline Phase: Hierarchical Semantic Indexing. NOAH constructs an HSI that explicitly encodes the latent hierarchical structure of a non-relational table. Given a raw table, NOAH first performs *header identification*, using LLMs to detect header cells and infer their spatial organization. To balance accuracy and cost, NOAH employs a pattern-based filtering strategy to avoid feeding the full-table to LLMs. This process identifies both row headers and column headers, shown as *blue* and *orange* regions in Figure 2(a), respectively. Row headers form a calendrical hierarchy (e.g., month, monthly average, day), while column headers encode metric-related hierarchies (e.g., emission types and aggregation functions).

Based on the identified headers, NOAH builds the HSI as a tree-based structure. Each node corresponds to a header cell and stores its textual content together with the span of rows or columns it governs in the table. Edges capture hierarchical relationships between headers. To simplify the index structure, NOAH decouples row and column semantics and independently constructs a Row-HSI and a Col-HSI. The resulting index enables efficient mapping from high-level header semantics to concrete data cells, and serves as the foundation for query-driven data access.

Online Phase: Dynamic Context-aware Query Execution. In the online analysis phase, NOAH processes each user query using an LLM-powered, context-aware execution engine. Rather than prompting an LLM to directly generate a final answer, NOAH resolves a query iteratively through a sequence of analysis steps, dynamically adapting its execution based on intermediate results.

Given the query shown in Figure 2(c), NOAH first invokes the *Query Rewrite* operation to derive a simpler sub-query that isolates part of the analytical intent. Next, NOAH performs *Data Retrieval*, which consults the HSI to conduct *attribute linking*: it identifies the query-relevant attributes and retrieves only the corresponding data regions from the table. To ensure efficiency, attribute linking

combines embedding-based similarity search with LLM-based semantic matching, allowing NOAH to avoid loading all attributes into the LLM context. The retrieved data is then transformed into an *AI-ready representation* that makes cell-level semantics explicit and facilitates downstream LLM reasoning.

Finally, NOAH executes the *Answer Generation* operation to answer the current sub-query. Depending on the characteristics of the sub-query and the size of the retrieved data, NOAH dynamically selects between *direct question answering*, which uses LLM reasoning for lightweight analytical tasks (e.g., selection or comparison), and *code execution*, which synthesizes Python programs for deterministic computation (e.g., aggregation). In the running example, NOAH selects direct question answering, as the sub-query does not require complex numerical processing.

NOAH iterates the above process, refining the query intent and reusing intermediate results, until the original user query is fully resolved. We next present the technical details of HSI (Sec. 3), followed by the dynamic, context-aware analysis framework (Sec. 4).

3 Hierarchical Semantic Index

We first define the Hierarchical Semantic Index (HSI), then describe how to construct it from raw tables, and finally present the HSI-based attribute linking operator used during query execution.

3.1 Definition of Hierarchical Semantic Index

A typical non-relational table comprises three components: column headers, row headers, and data cells [64]. Unlike relational schemas, column and row headers in non-relational tables often exhibit nested hierarchical relationships. Accurately identifying and modeling these latent hierarchies is essential for NOAH to correctly and efficiently answer analytical queries. Therefore, NOAH preprocesses each table by identifying its headers and encoding them into the AI-ready HSI. Compared to relational schemas, this tree-based structure can be constructed directly from raw non-relational tables and is more amenable to LLM-based reasoning.

Row headers and column headers typically encode different semantic dimensions of a table [6]. In real tables, row headers often describe entities or temporal structures (e.g., dates or categories), while column headers describe attributes, metrics, or measurement types. For example, as shown in Figure 2(a), the row hierarchy captures a calendrical structure, whereas the column hierarchy represents emission types and aggregation metrics. Therefore, HSI adopts a *dual-tree hierarchical structure*, consisting of a Row-HSI and a Col-HSI that separately model the row- and column-header hierarchies. This simplifies the index structure and reduces the complexity of attribute linking.

Definition 3.1 (Node in HSI)**.** A node in an HSI is defined as a tuple $I = (I.\text{children}, I.\text{value}, I.\text{span})$, where (a) $I.\text{children}$ is an ordered list of child HSI nodes, denoted as $[I_1, I_2, \ldots, I_k]$, representing child-levels in the hierarchy; (b) $I.\text{value}$ is the semantic content of the node, corresponding to a header cell in the table; and (c) $I.\text{span}$ denotes the contiguous range of row or column indices in the original table governed by the node, represented as a pair $(start, end)$.

In Figure 2(b), the node `Jan.` contains a value "Jan.", a set of child nodes corresponding to individual days in January, and a span of (3, 34) indicating the rows associated with January. Through such nodes, HSI encodes hierarchical header relationship and establishes a direct mapping from headers to data regions, thus enabling efficient attribute discovery and date retrieval during query execution.

3.2 Construction of HSI

NOAH first *identifies header region* and then *builds HSI*, progressively extracting and organizing the latent structural and semantic information from a non-relational table into a dual-tree structure.

3.2.1 Identify Header Regions

It determines which rows and columns contain header cells[15] that encode the hierarchical semantics of the table, similar to the attributes of relational tables. Due to the diversity and complexity of table structures, it is challenging to reliably detect header regions using fixed rules or handcrafted heuristics.

We formulate header detection as a *classification problem*. For column header detection, each row is classified as either a header row or a non-header row; row header detection is handled analogously. A naive solution would prompt an LLM with the entire table and classify each row (or column) independently. However, this is inefficient and does not scale to large tables.

To address this challenge, we exploit the following observation: rows or columns that contain header cells exhibit *distinct structural patterns* compared to data rows. For example, in Figure 2(a), the first two rows are column headers, where all cells are either textual or empty, whereas subsequent rows primarily contain numerical data. Based on such row/column patterns discovered from data, NOAH employs an iterative detection strategy to automatically identify headers without relying on predefined table-layout templates.

We describe column header detection below; row header detection follows the same procedure. First, we define the *pattern* of a row as the collection of the characteristics of the cells in that row.

Definition 3.2 (Row Pattern)**.** The pattern of a row r in table T with n columns is defined as a vector of cell features:

$$\text{Pattern}(r) = [f(c_1), f(c_2), \ldots, f(c_n)],$$

where c_i is the cell at column i in row r, and $f(c_i)$ is a function that captures key characteristics of cell c_i as follows:

$$f(c_i) = (\textsc{IsMergedCell}(c_i), \textsc{DataType}(c_i), \textsc{Formatting}(c_i))$$

$\textsc{IsMergedCell}(c_i) \in \{1, 0\}$; $\textsc{DataType}(c_i) \in \{\text{Text, Numeric, Empty}\}$; $\textsc{Formatting}(c_i)$ is a 3-dimensional binary vector indicating whether c_i uses *bold*, *italic*, and *underline* formatting, extracted using `openpyxl` [2], a widely used library for processing spreadsheets.

NOAH maintains a set of observed *non-header patterns* $\mathcal{P}_{\text{non-header}}$. Rows are accessed sequentially from top to bottom. For each row r, if $\text{Pattern}(r) \in \mathcal{P}_{\text{non-header}}$, NOAH immediately classifies r as a non-header row without invoking the LLM. Otherwise, NOAH classifies r using an LLM-based classifier, taking the row content and its neighboring rows as input. If a row r is classified as a non-header row and its $\text{Pattern}(r)$ has been observed more than K (empirically set to 5) times, the $\text{Pattern}(r)$ is added to $\mathcal{P}_{\text{non-header}}$. We track *non-header patterns* $\mathcal{P}_{\text{non-header}}$ rather than *header patterns* because non-header rows are typically more consistent and homogeneous (e.g., numeric cells with consistent formatting). This strategy allows NOAH to identify header regions without requiring loading the full table into the LLM context. On RealHiTBench[60], a real world non-relational table corpora, NOAH skips LLM classification for 88.3% of rows on average without sacrificing detection accuracy.

3.2.2 Build Hierarchical Semantic Index (HSI)

NOAH constructs the HSI by organizing the identified headers into a tree that captures their latent hierarchical relationships. We observe that, in most human-designed tables, even those with complex hierarchical structures, header layouts tend to follow a consistent pattern: within a row header region, headers that appear earlier from left to right usually correspond to higher-level concepts in the semantic hierarchy, while later columns represent finer-grained levels. An analogous pattern

Algorithm 1: Iterative Hierarchy Construction

Input: Table T, header region $\mathcal{R}_{\text{header}}$, primary axis prim $\in \{\text{row}, \text{col}\}$, secondary axis sec $\in \{\text{row}, \text{col}\}$
Output: Hierarchical Semantic Index I

1 $I \leftarrow$ tree with root node $\langle \text{ROOT} \rangle$;
// Process header cells in positional order (primary axis first)
2 $\mathcal{S} \leftarrow \text{LexSortCells}(\mathcal{R}_{\text{header}}, \text{prim}, \text{sec})$;
3 **foreach** *cell* $u \in \mathcal{S}$ **do**
// Collect cells from previous/current levels (as candidates)
4 $\mathcal{P} \leftarrow \text{GetCandidates}(u, \text{prim}, \text{sec})$;
// Validate parent-child relation using two heuristics
5 $\mathcal{P}_{\text{valid}} \leftarrow \emptyset$;
6 **foreach** $p \in \mathcal{P}$ **do**
7 **if** $\mathit{SpatialValid}(p, u) \ \vee \ \mathit{SemanticValid}(p, u)$ **then**
8 $\mathcal{P}_{\text{valid}} \leftarrow \mathcal{P}_{\text{valid}} \cup \{p\}$;
// Select the best candidate parent using a positional rule (primary first)
9 **if** $\mathcal{P}_{valid} \neq \emptyset$ **then**
10 $p^{\star} \leftarrow \arg\max_{p \in \mathcal{P}_{\text{valid}}} \text{LexScore}(p, \text{prim}, \text{sec})$;
11 $\text{AddEdge}(I, p^{\star}, u)$;
12 **else**
13 $\text{AddEdge}(I, \langle \text{ROOT} \rangle, u)$;
14 **return** I;

holds for column header regions. For example, in Figure 2(b), the left-to-right ordering of the row header columns indicates that month labels (e.g., "Jan.", "Feb.") are higher-level headers than day-level entries (e.g., "Monthly Average", "1", "2", ...) in the semantic hierarchy. In the column header region, the higher-level headers are positioned above the lower-level headers.

This observation guides us to design an **Iterative Hierarchy Construction** algorithm (Algorithm 1) to efficiently and effectively infer parent–child relationships among header cells. Given a row header region, the algorithm processes header cells in positional order, prioritizing the primary axis (left-to-right for row headers, top-to-bottom for column headers) (Line 2). To accurately establish parent–child relationships, NOAH identifies a set of candidate parent headers from previously processed cells (Line 4). For example, to form the parent–child pairs in Figure 2(a), when processing the day value "2" (column 2, row 5), the header cells located in its upper-left region, namely "Jan.", "Monthly Average", and "1", are treated as candidate parent headers. To validate parent–child relationships, each candidate pair is evaluated using two criteria (Lines 6–8):

- **Spatial Validity.** This rule leverages the *spatial relationship* of header cells. If a candidate parent header spatially *covers* the current header with a merged-cell layout, the current header is assigned as its child. For example, in Figure 2(b), the header "Jan." spans multiple rows and thus spatially covers day values "Monthly Average", "1", "2", etc., making it a valid parent candidate.
- **Semantic Validity.** This rule explores the *semantic relationship* of header cells. If the textual content of a candidate header *semantically subsumes or describes* the current header, the current header is regarded as its child. For example, in Figure 2(b), the header cell "Monthly Average" semantically subsumes day values such as "1" and "2", and is therefore a valid parent for these nodes.

Spatial validity is evaluated using deterministic layout checks, whereas semantic validity relies on LLMs. To avoid repeatedly invoking LLMs during the hierarchy construction process, NOAH prompts the LLM once per header region to analyze header texts and identify all semantically valid parent–child pairs in advance. During construction, a candidate parent–child pair is considered valid if it satisfies either condition (Lines 5–7). When multiple valid parent candidate exist, NOAH

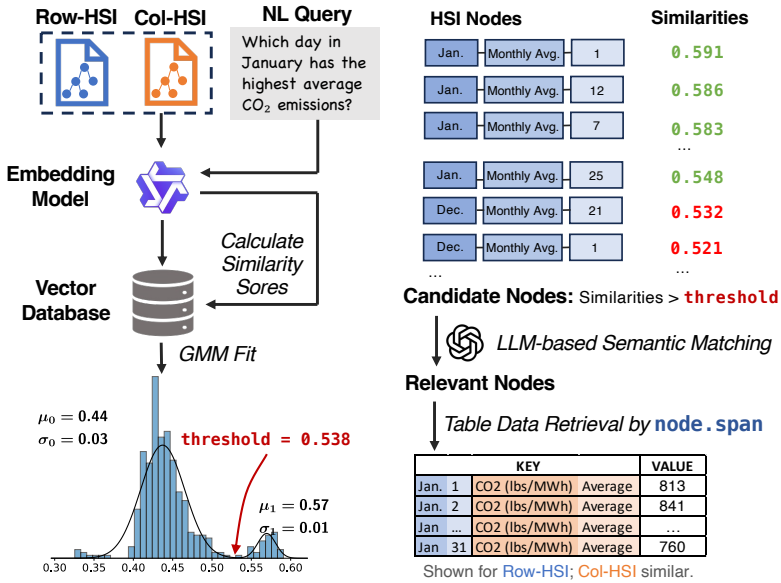


Fig. 3. Example of Attribute Linking with HSI.

selects the most appropriate one using a positional rule that favors spatial proximity along the primary axis. For example, in Figure 2(b), both ("Monthly Average", "1") and ("Jan.", "1") are valid parent–child pairs. However, "Monthly Averag" is selected as the parent of "1", because it appears closer in the positional order, reflecting a finer-grained hierarchical relationship.

Next, NOAH assigns a *span* to each HSI node, indicating the contiguous range of rows or columns in the original table governed by that header. For the Row-HSI, the table is partitioned into contiguous row segments based on row header positions; the same procedure applies to columns for the Col-HSI. For example, in Figure 2(b), the node "1" governs only row 36, yielding a span of $(36, 36)$, while "Jan." spans rows $(3, 34)$.

Finally, the complete HSI is serialized into a JSON-like representation, where each node is encoded as a (*value*, *span*) pair, and each node recursively references its list of child nodes.

3.3 Attribute Linking with HSI

Unlike relational tables, where attributes are explicitly defined as columns, attributes in non-relational tables are *implicit* and embedded within complex header hierarchies. The HSI, constructed offline, exposes an attribute linking operation to the query execution engine during online analysis. Given a query, attribute linking identifies the HSI nodes that correspond to the implicit attributes relevant to the query intent, and retrieves only the associated data regions from the table. This operation plays a role analogous to schema linking in NL2SQL [33, 53, 55], critical to both the accuracy and efficiency of non-relational table analysis.

An approach is to traverse all HSI nodes and invoke an LLM to assess their relevance to the query. Although effective in principle, it is prohibitively expensive for large tables with deep hierarchies. Therefore, we design a two-stage strategy that combines *embedding-based similarity search* with *LLM-based semantic matching*.

3.3.1 *Two-stage Search Strategy*

NOAH first embeds the incoming query and each HSI node into a vector space [19] using a pre-trained embedding model [67]. The embedding for a node is constructed by concatenating the textual values of all ancestors along the path from the root to the node. For example, in Figure 2(b), the

Row-HSI node "1" is represented as "`January > Monthly Average > 1`", allowing the embedding to encode both local and hierarchical semantics. NOAH computes similarity scores between the query embedding and all HSI node embeddings, only retaining nodes whose similarity exceeds a *cut-off threshold*, thus yielding a small candidate set for further inspection.

Next, NOAH applies LLM-based semantic matching to the candidate set. The LLM is prompted in a Chain-of-Thought [21] manner to reason about whether each candidate node aligns with the query intent and select the most relevant nodes.

Example 1. As illustrated in Figure 3, for the query *"Which day in January has the highest average CO_2 emissions?"*, embedding-based similarity search identifies candidate nodes such as (January, Monthly Avg., Day 1), (January, Monthly Avg., Day 7) ... LLM-based semantic matching then validates these candidates and identifies the true positive nodes corresponding to all days in January. Using the `span` information stored in the selected HSI nodes, NOAH retrieves the corresponding data cells, producing a compact sub-table containing only the daily average CO_2 values in January.

3.3.2 Embedding-based Similarity Search

The effectiveness of attribute linking highly depends on selecting an appropriate similarity threshold. Ideally, it should be high to minimize the number of candidates passed to the LLM, while ensuring that *all* relevant attributes are retained.

A common practice is to apply one empirically chosen threshold to all queries. Such a fixed threshold is brittle, as the number of relevant nodes vary substantially across different queries and tables.

We propose a **Dynamic Thresholding** strategy that automatically determines the retrieval threshold based on the distribution of similarity scores. Prior work [39, 54] shows that cosine similarity scores between queries and documents often exhibit a bimodal distribution, corresponding to relevant and irrelevant items. Thus, we model the similarity scores with a Gaussian Mixture Model (GMM) and derive a threshold to separate relevant and irrelevant nodes.

Formally, let X denote the similarity score between the query and an HSI node, and let $z \in \{0, 1\}$ be a latent variable indicating whether the node is relevant ($z = 1$) or irrelevant ($z = 0$). We model the distribution as:

$$\begin{aligned} X \mid z = 1 &\sim \mathcal{N}(\mu_1, \sigma_1^2) && \text{(relevant nodes)} \\ X \mid z = 0 &\sim \mathcal{N}(\mu_0, \sigma_0^2) && \text{(irrelevant nodes)} \\ P(z = 1) &= \pi, \quad P(z = 1) = 1 - \pi \end{aligned} \tag{1}$$

where π is the prior probability that a node is relevant. We fit the GMM parameters to the similarity scores of all HSI nodes using the Expectation-Maximization (EM) algorithm [14].

Next, NOAH select a threshold τ and retrieve all nodes with similarity scores $X \geq \tau$. To balance efficiency and recall, it finds the **largest** threshold τ that retrieves *all* relevant nodes with high confidence:

$$\max_{\tau} \quad \tau \qquad \text{s.t.} \quad \mathbb{P}(\text{recall} = 1 \mid X \geq \tau) \geq \gamma \tag{2}$$

where γ is the confidence level (e.g., 0.9). Note that γ does not represent the recall rate itself; rather the probability that all relevant nodes are retrieved. This distinction is crucial: missing even a single relevant node (i.e., recall < 1) can fail downstream query execution.

The following lemma characterizes an upper bound on the retrieval threshold τ that satisfies the recall constraint.

Lemma 2 (Dynamic Thresholding). *The threshold τ satisfying the recall constraint in* (2) *is upper bounded by*

$$\tau \leq \mu_1 + \sigma_1 \Phi^{-1}\left(\frac{1 - \gamma^{1/N}}{\pi}\right), \tag{3}$$

where $\Phi(\cdot)$ is the standard normal CDF and N is the number of HSI nodes.

Proof. We first define a "bad event" A as the event that a relevant node is missed during retrieval:

$$A := \{z = 1, X < \tau\} \tag{4}$$

The probability of this event is:

$$\mathbb{P}(A) = \mathbb{P}(X < \tau \mid z = 1) \cdot P(z = 1) = F_{X|z=1}(\tau) \cdot \pi \tag{5}$$

$$F_{X|z=1}(\tau) = \Phi\left(\frac{\tau - \mu_1}{\sigma_1}\right) \tag{6}$$

where $F_{X|z=1}(\tau)$ is the cumulative distribution function (CDF) of the Gaussian component for relevant nodes, and $\Phi(\cdot)$ is the CDF of the standard normal distribution.

Therefore, $\mathbb{P}(\text{recall} = 1 \mid X \geq \tau)$ can be expressed as:

$$\mathbb{P}(\text{ Recall } = 1) = \mathbb{P}\left(\bigcap_{i=1}^{N} A_i^c\right) = \prod_{i=1}^{N} \left(1 - \mathbb{P}\left(A_i\right)\right) = \left(1 - pF_1(T)\right)^N \tag{7}$$

where A_i^c is the complement of the bad event for node i, and N is the total number of nodes in the HSI. Thus, the constraint $\mathbb{P}(\text{recall} = 1 \mid X \geq \tau) \geq \gamma$ can be rewritten as:

$$\left(1 - \pi F_{X|z=1}(\tau)\right)^N \geq \gamma \tag{8}$$

We can solve this inequality for τ and yield:

$$\tau \leq \mu_1 + \sigma_1 \Phi^{-1}\left(\frac{1 - \gamma^{1/N}}{\pi}\right) \tag{9}$$

□

NOAH sets τ to the upper bound in Eq. 3 and retrieve all nodes with similarity scores above τ. This yields a compact candidate set while providing a probabilistic guarantee that all relevant attributes are retained for subsequent LLM-based semantic matching.

With this dynamic thresholding method, NOAH remains effective even if the embedding-similarity distribution does not exhibit a clean bimodal structure, for example, when many HSI nodes share long common header prefixes. In such cases, the similarity distribution may be weakly separable. More specifically, when the fitted components are not well separated, the upper-bound threshold in Eq. 3 will become more conservative, resulting in a lower retrieval threshold and thus a larger candidate set.

3.4 Computational Cost Analysis

We analyze the computational cost of NOAH in terms of *table access*, *LLM tokens*, and *embedding/index operations*. Assume a table has m rows and n columns, with m_c column-header rows and n_r row-header columns. For Header Region Identification, the *table-access cost* is $1 + m + n$. The worst-case *LLM input token cost* is $O(mn)$. However, in practice, repeated row/column patterns allow many calls to be skipped; on RealHiTBench, only 11.7% of cells are sent to the LLM in this step. *The LLM output cost* is negligible because each call returns only a binary label. For HSI construction, the worst-case *LLM token cost* and *embedding/index cost* are both $O(m_c n + n_r m)$, corresponding to detected header cells and their valid relationships. These costs are incurred once per table and amortized across multiple queries over the same table.

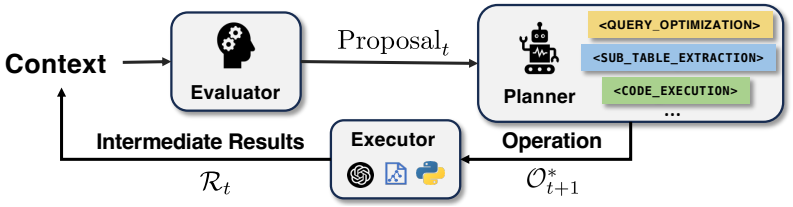


Fig. 4. Workflow of the query execution engine in NOAH.

4 Dynamic Context-aware Data Analysis

The NOAH query execution engine offers a set of operators and leverages an LLM as a planner to produce a customized analysis pipeline for each query. Rather than generating a complete execution plan upfront, NOAH adopts a progressive plan-and-execute paradigm: it incrementally selects and executes operators based on intermediate results obtained so far, until a satisfactory answer is produced. The workflow is illustrated in Figure 4. As a result, complex queries may trigger multiple rounds of planning and execution, while simpler queries can often be answered in only a few steps. This dynamic, context-aware execution model enables NOAH to effectively handle diverse analytical queries over complex non-relational tables.

Overall Workflow. NOAH provides three operators: *Query Rewrite*, *Data Retrieval*, and *Answer Generation*. The Query Rewrite operator leverages LLM reasoning to transform a complex query into a more manageable sub-query in natural language. Note that the rewritten query need not be a strict logical sub-query; rather, it represents an intermediate analytical goal. Given an analysis intent, the Data Retrieval operator invokes the *attribute linking* in HSI to identify the subset of data required to answer that intent. Finally, the Answer Generation operator produces intermediate or final results based on the retrieved data and the intermediate execution context. Next, we discuss this interactive data analysis process shown in Algorithm 2.

Let $C_t = (q, H_t)$ denote the analysis context at step t, where q is the original user query and H_t represents the history of executed operators and their results up to step t. Let $O = \{O_1, O_2, \ldots, O_n\}$ denote the set of available operators, namely *Query Rewrite*, *Data Retrieval*, and *Answer Generation*.

At each step, NOAH evaluates the current context C_t and produces a *Proposal*, which is a compact intermediate reasoning state generated by the LLM that summarizes the current analysis progress and identifies the next analytical operator (Line 4). Taking as input the current context C_t, the generated Proposal$_{t+1}$, and the set of possible operators O, NOAH invokes the LLM as a planner to select the most appropriate operator O^*_{t+1} for execution (Line 5).

For example, suppose the current context C_t contains the original query q and the result of a data retrieval step that extracts a relevant sub-table from the original table. The corresponding *Proposal* may summarize the progress as: *"I have extracted the relevant data for the query. Next, I need to analyze this data to generate an answer."* Based on the current context and the *Proposal*, the Planner evaluates the suitability of each operator in O and selects the *Answer Generation* operator as the optimal next operator O^*_{t+1}.

NOAH then executes the selected operator O^*_{t+1} to generate new intermediate results $\mathcal{R}_{t+1}$ (Line 6) and update the analysis context accordingly. Specifically, the tuple (Proposal$_{t+1}$, O^*_{t+1}, $\mathcal{R}_{t+1}$) is appended to the execution history to form the updated context C_{t+1} (Lines 12–13). Depending on the selected operator, this step may involve retrieving additional data from the input table $\mathcal{T}$ or generating an intermediate or final answer to the query.

This process is repeated iteratively until a final answer is produced (Lines 14–15) or a predefined maximum number of steps is reached (Line 3), in which case NOAH returns NoAnswer (Line 17).

Algorithm 2: Dynamic Context-aware Data Analysis

Input: Operator set O and initial context $C = (q, \emptyset)$
Output: Final answer

1 $t \leftarrow 1$;
2 $\mathcal{P} \leftarrow \emptyset$; // Set of pruned step ids
3 **while** *maximum steps not reached* **do**
4 $\quad \text{Proposal}_t \leftarrow \text{LLM}_{\text{evaluator}}(C)$;
5 $\quad O_t^* \leftarrow \arg\max_{O_i \in O} P(O_i \mid C, \text{Proposal}_t)$; // Select operator
6 $\quad \mathcal{R}_t \leftarrow \text{Apply}(O_t^*, C)$; // Execute operator
$\quad$ // Prune non-informative steps
7 $\quad$ **if** O_t^* *is* Data Retrieval *or* Answer Generation **then**
8 $\quad\quad$ **if** $LLM_{assessor}(\mathcal{R}_t) = \textit{False}$ **then**
9 $\quad\quad\quad \mathcal{P} \leftarrow \mathcal{P} \cup \{t\}$;
10 $\quad$ **if** O_t^* *is* Query Rewrite **then**
11 $\quad\quad$ Prune step t' from C if $t' \in \mathcal{P}$;
12 $\quad H_t \leftarrow C$.history;
13 $\quad C \leftarrow (q, H_t \oplus (\text{Proposal}_t, O_t^*, \mathcal{R}_t))$; // Update context
14 $\quad$ **if** $\textit{IsFinal}(\mathcal{R}_t, C)$ **then**
15 $\quad\quad$ **return** $\mathcal{R}_t$;
16 $\quad t \leftarrow t + 1$;
17 **return** NoAnswer;

Throughout the analysis process, NOAH accesses the input table $\mathcal{T}$ and the HSI exclusively through the *Data Retrieval* operator. As a result, all subsequent LLM reasoning is performed only on query-relevant subsets of the table, substantially improving efficiency and reducing LLM computation cost. In Algorithm 2, the only step relying on the LLM is operator selection (Line 5), while other steps, are performed by deterministic functions.

Context Management. As data analysis progresses, the length of the analysis context grows over time. However, not all intermediate steps contribute equally to producing the final answer. For example, some steps may generate Python code with run-time errors. Such intermediate results can be uninformative or even harmful for subsequent reasoning, as they may distract the LLM and degrade performance [28, 44].

To address this issue, we introduce a *context pruning* strategy that selectively removes less useful steps from the analysis context. As illustrated in Algorithm 2, after executing a *Data Retrieval* or *Answer Generation* operator, NOAH invokes an LLM-based assessor to evaluate the usefulness of the resulting intermediate result (Line 8). If the output is deemed unhelpful (e.g., erroneous code or irrelevant data), NOAH marks the corresponding step for pruning (Line 9).

Importantly, NOAH does not immediately remove such steps from the context. Instead, pruning is deferred until the next *Query Rewrite* operator is selected. This design allows potentially informative failure signals, such as error messages or incorrect intermediate results, to remain available for diagnosing and correcting mistakes in subsequent reasoning steps. When the *Query Rewrite* operator is triggered, it indicates that the analysis of the previous intermediate intent has concluded and the system is transitioning to a new analytical intent. At this point, previously marked steps can be safely pruned from the context (Lines 10–11).

As shown in Section 5.4, context pruning not only helps reduce the context length, but also improves the accuracy of the final answers. In the following subsections, we describe each operator in NOAH's query execution engine in detail.

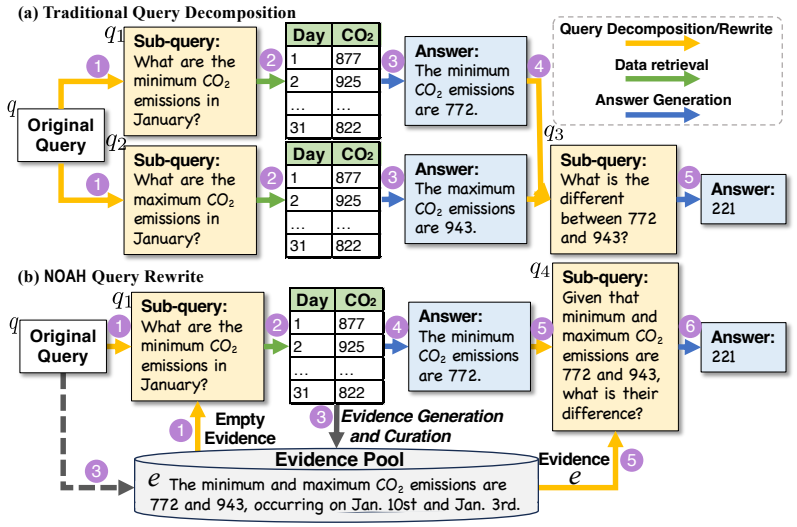


Fig. 5. Query Rewriting with Evidence Pool

4.1 Query Rewrite Operator

The original user query may be complex and multifaceted, often requiring multi-hop reasoning over both table structure and content, which makes it difficult for an LLM to resolve in a single step [9, 62]. The Query Rewrite operator (<QUERY_REWRITE>) addresses this challenge by rewriting a complex data analysis query into a more focused analytical intent that is easier for the LLM to handle. NOAH invokes this operator iteratively, resolving each rewritten intent in sequence and progressively accumulating intermediate results, ultimately producing a comprehensive answer to the original query.

In contrast, existing approaches [52, 63] typically decompose a complex query into multiple sub-queries, and execute them independently, either in parallel or sequentially. This independent execution model tends to fail to exploit shared reasoning and reusable intermediate results across sub-queries, leading to redundant computation and unnecessary data access.

For example, consider a user query q: *"What is the difference between the minimum and maximum CO_2 emissions in January?"* A typical decomposition yields two sub-queries: q_1: *"What is the minimum CO_2 emission in January?"* and q_2: *"What is the maximum CO_2 emission in January?"* Both sub-queries require retrieving and aggregating the same set of January CO_2 emissions records. Executing them independently results in repeated data retrieval and aggregation, as illustrated in Figure 5 (a). Since retrieving relevant data from non-relational tables is already costly, such redundant access further amplifies the overall computational cost.

To address this limitation, rather than decomposing q into multiple sub-queries parallelly, NOAH performs query rewriting in an incremental manner. Specifically, NOAH generates a simplified sub-query q_1 and attempts to resolve it first. Based on the intermediate results obtained from answering q_1, NOAH decides whether a subsequent query q_2 is necessary, using the results of q_1 as evidence.

For example, as illustrated in Figure 5 (b), to answer sub-query q_1, NOAH retrieves the relevant data, i.e., the January CO_2 emission records. Then, from these records it derives a new evidence e, including the minimum and maximum values as byproducts, since both are useful for answering the original query. Therefore, it is unnecessary to generate sub-query for the maximum value, thereby avoiding redundant retrieval and reprocessing. In this example, although the immediate

subquery asks only for the minimum value, NOAH produces the maximum value as a byproduct, as it is contained in the retrieved cells. Because this information is useful for answering the original query, NOAH retains it in the evidence pool.

To enable such evidence-driven decision making, NOAH maintains a *Query-aware Evidence Pool E* throughout the analysis process [48, 65, 70]. The evidence pool is continuously augmented with intermediate findings that contribute to resolving parts of the original query q. Based on the current contents of E, NOAH determines whether additional sub-queries are required and generates the next sub-query to address the remaining unresolved aspects of q.

NOAH manages the evolving evidence pool using two LLM-powered primitives: *Evidence Generation* and *Evidence Curation*.

Evidence Generation. After each analysis operation (e.g., *Data Retrieval* or *Answer Generation*), NOAH invokes *Evidence Generation* to extract new evidence from the intermediate results. Each evidence item e is represented as a textual statement that captures a factual finding relevant to the original query.

Evidence Curation. Newly generated evidence is incorporated into the pool through *Evidence Curation*. Rather than naively appending all evidence, it identifies and removes redundant entries to maintain a compact and informative evidence set. For instance, suppose the evidence pool E already contains e_i: *"The CO_2 emission in Jan. is 19,550 lbs/MWh."*. After a new evidence item e: *"The CO_2 emission in Jan. is 19,550 lbs/MWh, with a maximum of 943 and a minimum of 772."* is generated, NOAH retains only e and discards e_i.

Sub-query Proposal. If <QUERY_REWRITE> is triggered, NOAH synthesizes a simplified sub-query q_{sub} that targets the analytical objective not yet answered by the current evidence pool. It then adds this sub-query to the analysis context for subsequent processing. Figure 5 (b) shows the evidence-aware query rewriting process.

4.2 Data Retrieval Operator

The *Data Retrieval* operator extracts query-relevant information from a table using the HSI, and converts it into an AI-ready representation to facilitate downstream analysis.

Data Retrieval. To support different retrieval requirements, different retrieval requirements, NOAH offers two physical implementations of this operator—<SUB_TABLE_EXTRACTION> and <HIERARCHY_EXTRACTION>—analogous to physical operators in relational databases.

<SUB_TABLE_EXTRACTION> supports queries that require accessing specific data values from the table, such as *"What is the average CO_2 emissions in February?"*. Given a query, this operator invokes the Attribute Linking operation in HSI to identify the relevant attributes and the data cells they govern, and then extracts a sub-table containing only those cells.

<HIERARCHY_EXTRACTION> supports queries that can be answered directly based on the HSI without having to access the underlying data cells. For example, a user may ask *"How many emission types are tracked in the table?"*. In such cases, extracting raw data values is unnecessary. Instead, NOAH directly retrieves the relevant hierarchy from the Col-HSI or Row-HSI and computes the answer by operating on the hierarchy structure (e.g., counting the number of leaf nodes). By avoiding data cell extraction, this operator substantially reduces the number of input tokens passed to the LLM. Similar to <SUB_TABLE_EXTRACTION>, this operator also relies on the Attribute Linking operation in HSI, but only extracts the corresponding hierarchy layers from HSI.

AI-ready Representation. Although LLMs demonstrate strong capabilities in natural language understanding, they are not primarily trained to reason over two-dimensional tabular data with complex hierarchical layouts [25, 45, 47]. Rather than directly feeding an extracted sub-table, often containing nested headers and implicit structure, into an LLM or flattening it into an unstructured

text sequence, NOAH encodes the retrieved data into a *cell-annotated, AI-ready representation* to help LLMs better understand the semantics.

Specifically, each data cell is represented as a <key, value> pair, where the key is the concatenated paths of the cell in the Col/Row-HSI trees, and the value is the cell content. For example, in Figure 3, the data cell "813" is encoded as "<January–Monthly Average–1, CO_2(lbs/MWh)–Average>, 813", making the cell semantics explicit, and allowing LLMs to reason over individual values without interpreting complex table structures. Unlike relational tables, which are also easy to analyze but expensive to derive from non-relational inputs, this representation can be generated efficiently from HSI without schema inference or table normalization.

4.3 Answer Generation Operator

The *Answer Generation* operator produces an answer to a given query, which may correspond to the original user query or an intermediate sub-query generated by the *Query Rewrite* operator.

There are two common paradigms for answering natural language queries with LLMs: (1) *direct question answering (QA)*, where the LLM directly generates an answer from the provided context, and (2) *program synthesis*, where the LLM generates executable analysis code (e.g., Python) whose execution yields the answer [8, 17]. These two paradigms are suitable for different analytical scenarios. Program synthesis, using data analysis library such as Pandas, is well suited for queries that involve numerical computation, aggregation, filtering, or access to a large number of data cells, as executing generated code over structured data is more scalable and reliable than having the LLM reason directly over large contexts. In contrast, direct QA is effective for queries that require semantic interpretation, comparison, or lightweight reasoning over a small set of values, such as matching entities or searching answers [10].

Based on the above observations, NOAH implements two physical operators for answer generation: <DIRECT_QA> and <CODE_EXECUTION>, corresponding to direct question answering and program synthesis, respectively. At runtime, NOAH dynamically selects between these two operators based on the characteristics of the query and the retrieved data.

Specifically, NOAH selects <CODE_EXECUTION> if either of the following conditions holds: (1) **Data Volume**: the relevant data volume is large (empirically, over 30 cells); and (2) **Computation Complexity**: the query involves numerical computation, aggregation, filtering, or other data manipulations. If neither condition is met, NOAH selects <DIRECT_QA> to generate the answer directly.

5 Experiments

Our experiments aim to answer the following research questions:

- **RQ1: End-to-end Performance vs. Baselines.** How does NOAH compare with existing approaches in terms of answer accuracy and token consumption on non-relational table analysis tasks?
- **RQ2: Efficiency Analysis.** How efficiently does NOAH reduce the number of HSI nodes processed by the LLM through embedding-based attribute linking, and how much does NOAH reduce token consumption compared with baseline methods?
- **RQ3: Ablation Study of Main Components.** How effective are the key design choices in NOAH, including HSI, dynamic context-aware data analysis, and context pruning, in improving overall system performance?
- **RQ4: Robustness and Sensitivity Analysis.** How robust is NOAH across different LLMs, and how sensitive is our header region identification to the threshold K and formatting cues?

5.1 Experimental Setup

Datasets. To evaluate NOAH, we conduct experiments on two benchmarks: RealHiTBench [60] and MiMoTable [27]. We select these benchmarks for two main reasons: (1) Both contain real-world non-relational tables from diverse domains, paired with analytical natural-language queries; (2) they differ substantially in table scale, with RealHiTBench containing longer tables with 372 rows on average, while MiMoTable contains shorter tables with 26 rows on average. The two datasets are summarized as follows:

- **RealHiTBench [60]** is a challenging benchmark for evaluating LLMs on complex tabular tasks. In our experiments, we focus on the *numerical reasoning* task, as it is most relevant to NL-based data analysis. This task consists of 90 long tables and 203 queries.
- **MiMoTable [27]** is designed to evaluate LLM-based question answering over spreadsheets. The dataset covers seven domains and includes a variety of spreadsheet layouts. The dataset contains 150 tables and 591 queries.

Note that we have also evaluated NOAH on the SSTQA [50] benchmark, which was provided in a baseline method (ST-Raptor) [50] and has simpler table structures. NOAH wins on this benchmark. Due to space limit, we do not present the detailed results.

Evaluation. We evaluate NOAH by measuring *Execution Accuracy* (EA), defined as follows:

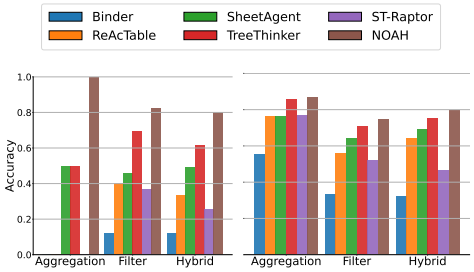
$$EA = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}(A_i = A_i^{\text{pred}}) \tag{10}$$

where N is the number of queries, A_i is the ground-truth answer, and A_i^{pred} is the predicted answer. Following [31], we use Claude 3.5 Sonnet [3] as an LLM-based judge to assess answer correctness.

Baselines. NOAH is an inference-time LLM-based framework that does not rely on any task-specific training or fine-tuning. For a fair comparison, we evaluate it against inference-time baselines designed for table analysis.

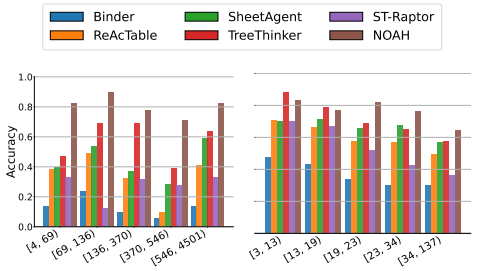
- **Bare LLM.** We prompt a black-box LLM with the table and the query and ask it to either (i) directly output the answer or (ii) generate a Python program whose execution yields the answer.
- **Chain of Thought (CoT)[58].** We augment the Bare LLM prompting with an in-context Chain-of-Thought demonstration.
- **Binder[12].** Binder is a training-free neuro-symbolic framework that maps task inputs to executable programs augmented with LLM API calls, enabling programmatic reasoning.
- **ReAcTable[66].** ReAcTable answers a query via multi-step execution, where each step generates code to produce intermediate artifacts used in subsequent steps until the final answer is obtained.
- **SheetAgent [11].** SheetAgent integrates multiple agentic modules (e.g., Planner, Informer, and Retriever) to perform step-by-step data analysis.
- **TreeThinker [60].** TreeThinker decomposes a query into multiple key words and finds the matching headers from a table to build a keyword-header tree, which will be incorporated into the prompt to generate the answer.
- **ST-Raptor[50].** ST-Raptor prompts a Vision Language Model to turn a table into a tree, decomposes a query into multiple concurrent sub-queries, and generates pipelines of a set of pre-defined tree operations to solve the problem.
- **DocETL [43].** DocETL optimizes LLM-based document-processing pipelines. We adapt DocETL by serializing tables as documents and carefully define a pipeline to process them.

Note that to evaluate the impact of context length, for Bare LLM and CoT, we test the cases that select the first k rows of the table as input context, where k sets to 30 and 90. To prevent the planner from entering an infinite loop, Alg. 2 sets the maximum number of steps conservatively to 30.



(a) RealHiTBench (b) MiMoTable

Fig. 7. Execution accuracy by query type.



(a) RealHiTBench (b) MiMoTable

Fig. 8. Execution accuracy by table size.

Table 1. Execution accuracy (%) of different models on RealHiTBench and MiMoTable.

Method [a]	**RealHiTBench**	**MiMoTable**
Bare LLM (30) [b]	19.70	36.20
Bare LLM (90)	16.25	37.22
Bare LLM (full)	15.27	37.22
CoT (30)	12.21	39.25
CoT (90)	19.21	38.74
CoT (full)	17.24	38.74
Binder	11.83	36.72
ReAcTable	34.98	60.24
ST-Raptor	27.59	54.82
DocETL	70.37	74.28
SheetAgent	43.84	66.33
SheetAgent (Gemini 2.5 Flash)	24.13	43.65
TreeThinker	61.57	71.57
TreeThinker (Gemini 2.5 Flash)	61.08	72.41
NOAH	**80.78**	76.65
NOAH (Gemini 2.5 Flash)	74.87	**80.54**

[a] Unless otherwise noted, we use Claude 3.5 Sonnet [3] as the underlying LLM for all methods to ensure fair comparison.

[b] **Bare LLM (k)** and **CoT (k)** indicate that the model is prompted with the first k rows of the table; **full** denotes the entire table.

5.2 RQ1: Accuracy Comparison with Baselines

We conduct extensive experiments to evaluate the effectiveness of NOAH. To ensure a fair comparison, we use the same underlying LLM (i.e., Claude 3.5 Sonnet) for all baselines as well as our approach. To show the performance of NOAH on different LLMs, we evaluate the three most performant methods on Gemini 2.5 Flash [13]. The results in Table 1 lead to the following observations.

First, NOAH consistently outperforms all baselines on both benchmarks, demonstrating its effectiveness in answering analytical queries over non-relational tables. We observe that DocETL is the best-performing baseline. However, its performance is sensitive to the user-defined pipeline. In our setting, we use a carefully designed two-stage pipeline that first extracts query-relevant cells and then generates the final answer based on the extracted content. If DocETL uses a simpler pipeline to directly generate answers from the serialized raw table, its accuracy drops to 60.5% on RealHiTBench and 74.2% on MiMoTable. Among tree-based approaches (i.e., TreeThinker and

ST-Raptor), TreeThinker achieves better performance than ST-Raptor. This is because TreeThinker constructs the keyword-header tree in plain text through end-to-end prompting, enabling inference of the tree structure of the headers. In contrast, ST-Raptor relies heavily on explicit layout cues (e.g., merged cells) to capture relationships between headers and data cells. In real-world non-relational tables, such layout structures are often absent and thus must be inferred from content.

Second, several baselines that generate code to process tables perform substantially worse than NOAH. These approaches primarily focus on addressing the complexity of query reasoning, while overlooking the complexity of non-relational table structures. For, example, although Binder can generate code to process tabular data, its core operation, i.e., adding new columns to a table, is poorly aligned with non-relational table analysis. Thus, Binder struggles to correctly identify and operate on the relevant data regions.

Third, feeding more rows to a bare LLM does not necessarily improve performance. On RealHiTBench, the Bare LLM achieves higher execution accuracy when using only the first 30 rows (19.70%), suggesting that excessive rows may obscure key structural information, such as headers, due to "lost-in-the-middle" [30].

Finally, replacing Claude 3.5 Sonnet with Gemini 2.5 Flash significantly degrades the code-generation-based SheetAgent, while the QA-based TreeThinker remains relatively stable. Since Gemini 2.5 Flash has weaker code-generation capability than Claude 3.5 Sonnet [1, 59], NOAH shows some performance drop on the more data-intensive RealHiTBench, but remains stable on the MiMoTable.

Accuracy Breakdown by Different Query Types. To further understand the behavior of NOAH across different analytical workloads, we categorize queries into three classes: *Aggregation*, *Filter*, and *Hybrid*, where Hybrid queries combine aggregation and filtering operations. In RealHiTBench, the query distribution consists of 1.1% *Aggregation*, 31.5% *Filter*, and 67.4% *Hybrid* queries; in MiMoTable, the proportions are 25.9%, 40.0%, and 34.1%. As shown in Figure 7, NOAH consistently outperforms all baselines across both benchmarks and all query types. We observe that *Filter* queries are generally more challenging than *Aggregation* and *Hybrid* queries, because they require precise identification of implicit attributes and fine-grained data regions in non-relational tables. In contrast, *Aggregation* queries often involve more uniform or repetitive values, making relevant headers and data easier to identify.

Accuracy Breakdown by Different Table Sizes. We evaluate NOAH' robustness by analyzing accuracy across tables of different sizes, i.e., the number of rows. We group tables into size-based bins, with each bin containing an equal number of tables. The results are shown in Fig. 8. As table size increases, several baselines exhibit a decline in execution accuracy, reflecting their limited ability to scale to larger tables. NOAH maintains relatively stable performance across all table size bins, demonstrating its scalability and robustness. The performance of some baselines does not strictly decrease with table size, as seen in Fig. 8. This behavior can be attributed to the fact that certain large tables exhibit simpler structures or more regular data patterns, making their implicit schemas easier for LLMs to infer despite increased size.

Performance on Challenging Cases. Real-world non-relational tables can be structurally irregular and large. To evaluate NOAH on such challenging cases, we construct a stress-test set of 100 large tables, each with more than 10,000 rows, and 300 natural-language queries.

It covers two atypical layouts: *sparse-header*, where more than 80% of cells in a header row or column are blank, and *interior-header*, where header cells appear inside the table rather than only in top-left regions. The queries cover common analytical operations, such as filtering and aggregation. On average, each query involves 2,629.32 relevant rows in the sparse-header subset and 2,196.09 rows in the interior-header subset.

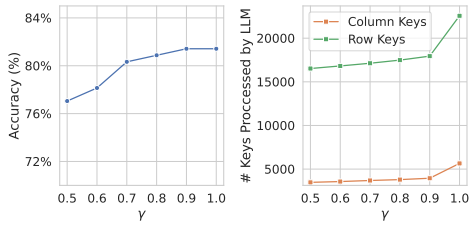


Fig. 9. Impact of γ values on execution accuracy and number of hierarchy nodes processed by LLM on RealHiTBench.

As shown in Table 2, NOAH achieves 51.33% execution accuracy on sparse-header tables and 47.33% on interior-header tables, substantially outperforming the best baselines, DocETL and TreeThinker. Sparse-header tables obscure header-value correspondence due to many blank header cells, while interior-header tables require recognizing header regions beyond conventional top-left positions. These results show that NOAH's HSI and context-aware execution can better capture atypical table structures at scale. We also observe that *interior-header* tables are more challenging than *sparse-header* tables, because sparse-header tables still preserve a relatively clear boundary between header and data regions, whereas interior-header tables place headers inside the table body, making it harder to distinguish headers from data cells and determine their scope.

Table 2. Execution accuracy on challenging cases.

Method	Sparse-header	Interior-header
DocETL	22.67	18.67
TreeThinker	8.67	7.33
NOAH	**51.33**	**47.33**

5.3 RQ2: Efficiency Analysis

Efficiency Analysis of Attribute Linking. We evaluate the efficiency of the embedding-based attribute linking in reducing the computational cost of LLM reasoning. As discussed in Section 3.3.2, a larger γ value increases the likelihood of retrieving all relevant hierarchy nodes and the corresponding data from the table, which can improve execution accuracy, but at the expense of higher computational overhead. To quantify this trade-off, we vary γ and examine its impact on both execution accuracy and efficiency, measured by the number of hierarchy nodes processed by the LLM.

As shown in Fig. 9, as γ increases, accuracy improves, indicating that retrieving more hierarchy nodes helps the LLM generate more accurate results. However, beyond a certain point (e.g., 0.7), the accuracy gains begin to diminish. Meanwhile, the number of hierarchy nodes processed by the LLM also grows, reflecting increased computational cost. Notably, this increase is not linear: when γ rises from 0.5 to 0.9, the number of processed nodes grows slowly, whereas setting $\gamma = 1$ (i.e., recalling all hierarchy nodes) leads to a sharp increase in cost. These results indicate that a moderate γ value (e.g., $\gamma = 0.9$) achieves a good balance between execution accuracy and computational efficiency.

We further observe that the number of row hierarchy nodes processed by the LLM is substantially larger than that of column nodes. This reflects the characteristics of long non-relational tables, which typically contain many more rows than columns. Column headers usually correspond to a small set of metrics or attributes, whereas row headers often encode categorical or indexing information and thus grow proportionally with table length. Despite these differences in scale

Table 3. Token and cell consumption per query on RealHiT and MiMo benchmarks.

Benchmark	Method	Cells / Query	Tokens / Query
RealHiT	ReAcTab	10,684.62	59,971.13
	End-to-End Prompting[a]	5,473.04	21,807.71
	NOAH	**478.01**	**4,356.66**
MiMo	ReAcTab	1,471.27	10,504.96
	End-to-End Prompting[a]	295.60	1,495.61
	NOAH	**90.08**	**952.72**

[a] End-to-end prompting methods require prompting the LLM with the entire table at least once, e.g., Bare LLM, CoT, SheetAgent, TreeThinker, ST-Raptor and DocETL. Here we report the results of one round of prompting.

and semantics, the embedding-based hierarchy search effectively reduces both row and column candidate sets.

Token Consumption Efficiency. We measure the average number of table cells accessed and the corresponding LLM input tokens on RealHiTBench and MiMoTable, with results summarized in Table 3. By leveraging HSI to selectively retrieve query-relevant data, NOAH avoids prompting the LLM with the entire table while preserving answer accuracy, leading to substantially lower token consumption than baseline methods. Each <SUB_TABLE_EXTRACTION> operation requires one table access, one query-embedding computation, and one HSI similarity search, and is invoked 1.27 times per query on average on RealHiTBench. As shown in Table 3, NOAH uses only 19.97% of one-shot full-table prompting. The output token cost is small because outputs are concise, such as operator names, rewritten subqueries, or analytical code.

5.4 RQ3: Ablation Study of Main Components

Ablation Study of HSI and Interactive Operations. To assess the effectiveness of the HSI and the context-aware data analysis paradigm, we conduct an ablation study on the RealHiTBench benchmark. For efficiency, we report the number of operations, i.e., the number of LLM steps executed to reach the final answer. The average number of steps NOAH executes is only 5.38, much smaller than the upper bound limit (30). NOAH reaches this limit only once on RealHiTBench.

To evaluate the impact of the HSI, we remove it entirely and prompt the LLM with the full table, allowing it to dynamically generate operations to answer the query. To evaluate the impact of the dynamic context-aware data analysis, we perform two sets of ablations. First, we disable the dynamic analysis process and require the LLM to generate a complete response in one shot. Second, we disable each operator described in Section 4 (e.g., <HIERARCHY_EXTRACTION>, <QUERY_REWRITE>, etc.) one at a time, while keeping the rest of the system unchanged. The results are summarized in Table 4.

We observe that disabling the context-aware analysis leads to the largest degradation in accuracy (a drop of 22.96%), highlighting the critical role of interactive planning and execution in NOAH. Removing the HSI also results in a substantial accuracy decrease (22.30%), demonstrating that explicit hierarchical modeling provides valuable structural guidance for LLM reasoning over non-relational tables.

Furthermore, removing HSI significantly increases the number of operations (by 82.72%). This indicates that, without explicit hierarchical structure, the LLM requires more trial-and-error steps to identify relevant data. In contrast, HSI enables NOAH to generate more accurate and efficient execution plans.

Table 4. Ablation study of Hierarchical Semantic Index and interactive operations on RealHiTBench.

NOAH	Acc. (%)	Total Operations
Base (γ=1)	**81.42**	**565**
w/o Hierarchical Semantic Index	62.84	1089
w/o Context-aware Data Analysis	62.30	-
w/o <SUB_TABLE_EXTRACTION>	66.12	1061
w/o <QUERY_REWRITE>	79.78	593
w/o <DIRECT_QA>	78.14	723
w/o <CODE_EXECUTION>	78.68	536

To further evaluate <QUERY_REWRITE>, we compare NOAH with and without this operator on SSTQA-hard [50], containing complex multi-hop queries. Removing <QUERY_REWRITE> degrades accuracy siginficantly from 60.65% to 50.82%.

Disabling <CODE_EXECUTION> causes only a small accuracy drop, since many queries involve limited retrieved data and can be answered through direct LLM reasoning. Specifically, the number of tuples processed per query ranges from 1 to 1324, with a median of 20. However, <CODE_EXECUTION> becomes particularly useful in data-intensive settings: on the challenging benchmark in Section 5.2, where each query involves 2,412.71 relevant rows on average, removing it reduces accuracy from 49.33% to 28.67%.

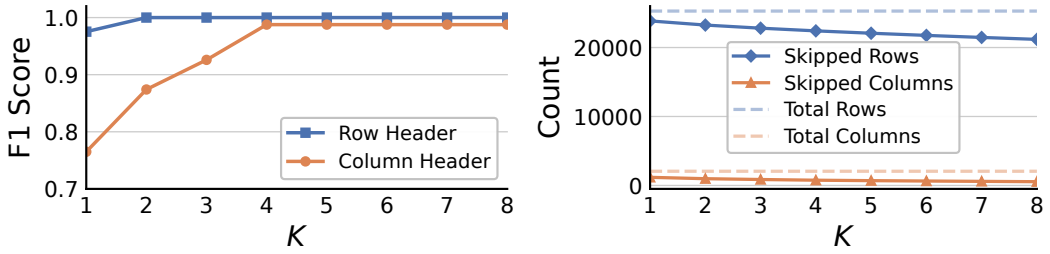
Ablation Study of Context Pruning. We evaluate the impact of context pruning under different values of γ. As shown in Table 5, context pruning improves efficiency without sacrificing execution accuracy, especially when γ is small. A smaller γ makes attribute linking more selective, which reduces LLM processing cost but may retrieve incomplete data and lead to erroneous intermediate steps, as discussed in Section 5.3. Context pruning mitigates this issue by removing such misleading steps from the context before subsequent reasoning and operator selection.

Table 5. NOAH with and w/o context pruning.

γ	NOAH	Acc. (%)	Total Operations
0.7	w/o Context Pruning	78.81	908
	w/ Context Pruning	80.29	665 (↓26.76%)
0.8	w/o Context Pruning	79.31	784
	w/ Context Pruning	80.78	596 (↓23.98%)
0.9	w/o Context Pruning	81.28	639
	w/ Context Pruning	81.28	580 (↓9.23%)

5.5 RQ4: Robustness and Sensitivity Analysis

Robustness to Different Underlying LLMs. To evaluate the robustness of NOAH across LLMs, we test it with Claude 3.5 Sonnet, Claude 4.6 Sonnet, Gemini 2.5 Flash, Gemini 3.1 Pro, and GPT-4.1. As shown in Table 6, NOAH achieves strong performance with all models. The accuracy drop with GPT-4.1 is due to its weaker code-generation capability compared with other models [32]. More specifically, each prompt in NOAH consists of a static system prompt and a dynamic context prompt. The same static prompt is used across all LLMs, while the dynamic prompt is constructed from query-relevant data. Thus, the consistently strong performance across model families suggests that NOAH is not tightly coupled to model-specific prompt tuning. Instead, its performance mainly



(a) Header region identification performance w.r.t. threshold K.

(b) Numbers of skipped and total rows/columns w.r.t. threshold K.

Fig. 10. Vary threshold K in header region identification.

depends on core components such as HSI-based retrieval and context-aware execution, as further supported by the significant performance drop when HSI is removed in Table 4.

Table 6. Robustness of NOAH under different underlying LLMs.

Underlying Model	RealHiTBench	MiMoTable
GPT-4.1	70.89%	74.28%
Claude 3.5 Sonnet	80.78%	76.65%
Gemini 2.5 Flash	74.87%	80.54%
Claude 4.6 Sonnet	82.01%	77.32%
Gemini 3.1 Pro	75.66%	81.72%

Sensitivity Analysis w.r.t. Threshold K. To evaluate sensitivity to the threshold K, we vary K from 1 to 8. Intuitively, a smaller K allows more rows/columns with repeated patterns to skip LLM-based classification, while a larger K makes skipping more conservative. However, if K is too small, more header-related patterns may be mistakenly treated as non-header patterns, which can reduce recall. Therefore, K should be chosen to balance identification accuracy and efficiency. As shown in Fig. 10a, the F1-score remains high when $K \geq 4$. This is because NOAH maintains a set of non-header patterns rather than header patterns, thus a smaller K causes more patterns to be classified as non-header patterns, which can reduce recall. In contrast, increasing K reduces the number of patterns. As shown in Fig. 10b, as K increases, the number of skipped rows and columns decreases, which leads to more LLM calls. To guarantee accuracy, we conservatively set $K = 5$ in our experiments, which achieves a good F1 score of 1.0 for column header identification and 0.9938 for row header identification, while skipping 88.19% of rows and 35.21% of columns on average on RealHiTBench.

Sensitivity Analysis w.r.t. Formatting Cues. 91.1% of tables in RealHiTBench contain formatting cues (defined in Def. 3.2, e.g., bold font, underline) that can help identify header regions. To evaluate the robustness to the absence of formatting cues, we remove all formatting information from the input tables. We find that NOAH's F1-score remain unaffected, while the number of skipped rows and columns slightly decreases to 87.29% and 33.75%, respectively. This suggests that our pattern-based method is robust to scenarios where formatting cues may be missing.

6 Related Work

Our work is closely related to natural language-based tabular data analysis and tabular data processing.

Natural Language Based Tabular Data Analysis. For relational tables, extensive research has focused on NL2SQL methods [34, 37, 38, 52, 61] that translate natural language queries into

executable SQL queries. However, non-relational table analysis is more challenging due to the absence of explicit schemas, and has therefore received comparatively less attention. Recent work leverages LLMs to generate and execute code to analyze tables. For example, Binder [12] uses LLMs as callable APIs, ReAcTable [66] and Chain-of-Table [57] iteratively transform tables, and some methods [11, 68, 69] generate Python code for table analysis.

Beyond table-specific systems, recent LLM-based data processing systems use LLMs as semantic operators, query planners, or pipeline optimizers. For example, Palimpzest [29] provides a declarative framework for optimizing AI-powered analytical queries; CAESURA [51] uses LLMs to translate natural-language queries into executable query plans; Galois [40] explores how to execute SQL queries over LLMs using database-style physical operators; Lotus [35] introduces semantic operators for AI-powered data processing and DocETL [43] optimizes complex pipelines through query rewriting and evaluation; These systems are broadly related to <SUB_TABLE_EXTRACTION> in NOAH, as they also aim to select, transform, or extract query-relevant information. However, their target inputs are typically unstructured or weakly structured documents, such as PDFs. In contrast, <SUB_TABLE_EXTRACTION> is designed for non-relational tables, whose complex cell-header relationships require table-specific retrieval. Moreover, systems such as Lotus and DocETL provide declarative interfaces for specifying processing pipelines, whereas NOAH allows users to directly ask NL questions about tables. Unlike pipeline-level rewriting in DocETL, NOAH's <QUERY_REWRITE> operator incrementally generates subqueries based on current evidence.

More recently, TreeThinker [60] and ST-Raptor [50] construct tree-structured representations of non-relational tables for LLM-based reasoning. However, they often rely on heuristic layout cues, such as merged cells, which limits robustness and generalizability. In contrast, NOAH constructs HSI to explicitly model table structure without task-specific training data, while avoiding scanning full-table with LLMs at query time for better scalability and efficiency.

Tabular Data Processing. Tabular data processing focuses on transforming, manipulating, and extracting information from tables to support downstream analysis tasks. Prior work has explored a variety of techniques for understanding and processing tabular data. For example, Dong et al. [15] propose using deep neural networks to infer the semantic types of table cells, which can aid table understanding and normalization. Other approaches attempt to convert non-relational tables into relational representations. Auto-Tables [24] trains specialized neural networks to convert non-relational tables into relational formats, enabling the use of NL2SQL techniques [34, 37, 38, 52, 61] for analysis. TUTA [56] trains a Transformer-based model to construct a bi-dimensional coordinate tree to model hierarchical information in non-relational tables. However, they typically require substantial labeled data for training, which limits their applicability to diverse table formats.

7 Conclusion and Future Work

In this paper, we present NOAH, a system for scalable natural language analysis of large non-relational tables. To handle complex and implicit table schemas, NOAH constructs HSI offline to make hierarchical semantics explicit. HSI enables query-relevant data retrieval, reducing the amount of table content fed to LLMs while improving accuracy. Built on HSI, NOAH uses a dynamic, context-aware workflow that iteratively rewrites queries, retrieves relevant data, and generates answers. Experiments on real-world benchmarks show that NOAH improves accuracy by 31.2% over state-of-the-art methods while reducing LLM token consumption by at least 3.3×.

Acknowledgments

This research was supported in part by NSF under grants DBI-2327954 and CCF-2106621 and OAC-2608717.

References

[1] [n. d.]. LiveBench: A Challenging, Contamination-Free LLM Benchmark. https://livebench.ai/. Accessed: 2026-01-15.
[2] [n. d.]. A Python library to read/write Excel 2010 xlsx/xlsm files. https://openpyxl.readthedocs.io/en/stable/. Accessed: 2026-01-15.
[3] Anthropic. 2024. Claude 3.5 Sonnet. https://www.anthropic.com/news/claude-3-5-sonnet. Accessed: 2024.
[4] Dhananjay Ashok and ZacharyC. Lipton. 2023. PromptNER: Prompting For Named Entity Recognition. (May 2023).
[5] Muhammad Imam Luthfi Balaka, David Alexander, Qiming Wang, Yue Gong, Adila Krisnadhi, and Raul Castro Fernandez. 2025. Pneuma: Leveraging llms for tabular data representation and retrieval in an end-to-end system. *Proceedings of the ACM on Management of Data* 3, 3 (2025), 1–28.
[6] Douglas Burdick, Marina Danilevsky, Alexandre V Evfimievski, Yannis Katsis, and Nancy Wang. 2020. Table extraction and understanding for scientific and enterprise applications. *Proc. VLDB Endow.* 13, 12 (2020), 3433–3436. doi:10.14778/3415478.3415563
[7] Michael J Cafarella, Alon Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. 2008. Webtables: exploring the power of tables on the web. *Proceedings of the VLDB Endowment* 1, 1 (2008), 538–549.
[8] Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023. Program of Thoughts Prompting: Disentangling Computation from Reasoning for Numerical Reasoning Tasks. *Transactions on Machine Learning Research* (2023).
[9] 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 the Association for Computational Linguistics: EMNLP 2020*. 1026–1036.
[10] Yongchao Chen, Harsh Jhamtani, Srinagesh Sharma, Chuchu Fan, and Chi Wang. 2025. Steering Large Language Models between Code Execution and Textual Reasoning. In *The Thirteenth International Conference on Learning Representations*.
[11] Yibin Chen, Yifu Yuan, Zeyu Zhang, Yan Zheng, Jinyi Liu, Fei Ni, Jianye Hao, Hangyu Mao, and Fuzheng Zhang. 2025. SheetAgent: towards a generalist agent for spreadsheet reasoning and manipulation via large language models. In *Proceedings of the ACM on Web Conference 2025*. 158–177.
[12] Zhoujun Cheng, Tianbao Xie, Peng Shi, Chengzu Li, Rahul Nadkarni, Yushi Hu, Caiming Xiong, Dragomir Radev, Mari Ostendorf, Luke Zettlemoyer, et al. 2023. Binding Language Models in Symbolic Languages. In *The Eleventh International Conference on Learning Representations*.
[13] Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261* (2025).
[14] A. P. Dempster, N. M. Laird, and D. B. Rubin. 2018. Maximum Likelihood from Incomplete Data Via the EM Algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)* 39, 1 (12 2018), 1–22.
[15] Haoyu Dong, Shijie Liu, Zhouyu Fu, Shi Han, and Dongmei Zhang. 2019. Semantic structure extraction for spreadsheet tables with a multi-task learning architecture. In *Workshop on Document Intelligence at NeurIPS 2019*.
[16] Haoyu Dong, Shijie Liu, Shi Han, Zhouyu Fu, and Dongmei Zhang. 2019. TableSense: spreadsheet table detection with convolutional neural networks. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence* (Honolulu, Hawaii, USA) *(AAAI'19/IAAI'19/EAAI'19)*. AAAI Press, Article 9, 8 pages. doi:10.1609/aaai.v33i01.330169
[17] Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Pal: Program-aided language models. In *International Conference on Machine Learning*. PMLR, 10764–10799.
[18] Peyman Hosseini, Ignacio Castro, Iacopo Ghinassi, and Matthew Purver. 2025. Efficient solutions for an intriguing failure of llms: Long context window does not mean llms can analyze long sequences flawlessly. In *Proceedings of the 31st international conference on computational linguistics*. 1880–1891.
[19] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick SH Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense Passage Retrieval for Open-Domain Question Answering.. In *EMNLP (1)*. 6769–6781.
[20] Oliver Lehmberg, Dominique Ritze, Robert Meusel, and Christian Bizer. 2016. A large public corpus of web tables containing time and context metadata. In *Proceedings of the 25th international conference companion on world wide web*. 75–76.
[21] Chengshu Li, Jacky Liang, Andy Zeng, Xinyun Chen, Karol Hausman, Dorsa Sadigh, Sergey Levine, Li Fei-Fei, Fei Xia, and Brian Ichter. 2024. Chain of code: reasoning with a language model-augmented code emulator. In *Proceedings of the 41st International Conference on Machine Learning* (Vienna, Austria) *(ICML'24)*. JMLR.org, Article 1134, 19 pages.
[22] Hongxin Li, Jingran Su, Yuntao Chen, Qing Li, and Zhao-Xiang Zhang. 2023. Sheetcopilot: Bringing software productivity to the next level through large language models. *Advances in Neural Information Processing Systems* 36 (2023), 4952–4984.

[23] Peng Li, Yeye He, Cong Yan, Yue Wang, and Surajit Chaudhuri. 2023. Auto-Tables: Synthesizing Multi-Step Transformations to Relationalize Tables without Using Examples. *Proceedings of the VLDB Endowment* 16, 11 (2023), 3391–3403.

[24] Peng Li, Yeye He, Cong Yan, Yue Wang, and Surajit Chaudhuri. 2023. Auto-Tables: Synthesizing Multi-Step Transformations to Relationalize Tables without Using Examples. *Proc. VLDB Endow.* 16, 11 (July 2023), 3391–3403. doi:10.14778/3611479.3611534

[25] 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. *Proc. ACM Manag. Data* 2, 3, Article 176 (May 2024), 28 pages. doi:10.1145/3654979

[26] Tianle Li, Ge Zhang, Quy Duc Do, Xiang Yue, and Wenhu Chen. 2025. Long-context LLMs Struggle with Long In-context Learning. *Transactions on Machine Learning Research* (2025).

[27] Zheng Li, Yang Du, Mao Zheng, and Mingyang Song. 2025. MiMoTable: A multi-scale spreadsheet benchmark with meta operations for table reasoning. In *Proceedings of the 31st International Conference on Computational Linguistics*. 2548–2560.

[28] Zeju Li, Jianyuan Zhong, Ziyang Zheng, Xiangyu Wen, Zhijian Xu, Yingying Cheng, Fan Zhang, and Qiang Xu. 2025. Compressing chain-of-thought in llms via step entropy. *arXiv preprint arXiv:2508.03346* (2025).

[29] Chunwei Liu, Matthew Russo, Michael Cafarella, Lei Cao, Peter Baile Chen, Zui Chen, Michael Franklin, Tim Kraska, Samuel Madden, Rana Shahout, and Gerardo Vitagliano. 2025. Palimpzest: Optimizing AI-Powered Analytics with Declarative Query Processing. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*.

[30] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the Middle: How Language Models Use Long Contexts. *Transactions of the Association for Computational Linguistics* 12 (2024), 157–173. doi:10.1162/tacl_a_00638

[31] Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. 2023. G-Eval: NLG Evaluation using Gpt-4 with Better Human Alignment. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Singapore, 2511–2522. doi:10.18653/v1/2023.emnlp-main.153

[32] LiveCodeBench. [n. d.]. LiveCodeBench Leaderboard. https://livecodebench.github.io/leaderboard.html. Accessed: 2026-04-22.

[33] Peixian Ma, Boyan Li, Runzhi Jiang, Ju Fan, Nan Tang, and Yuyu Luo. 2024. A plug-and-play natural language rewriter for natural language to sql. *arXiv preprint arXiv:2412.17068* (2024).

[34] Md Mahadi Hasan Nahid and Davood Rafiei. 2024. TabSQLify: Enhancing Reasoning Capabilities of LLMs Through Table Decomposition. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, Kevin Duh, Helena Gomez, and Steven Bethard (Eds.). Association for Computational Linguistics, Mexico City, Mexico, 5725–5737. doi:10.18653/v1/2024.naacl-long.320

[35] Liana Patel, Siddharth Jha, Melissa Pan, Harshit Gupta, Parth Asawa, Carlos Guestrin, and Matei Zaharia. 2025. Semantic Operators and Their Optimization: Enabling LLM-Based Data Processing with Accuracy Guarantees in LOTUS. *Proceedings of the VLDB Endowment* 18, 11 (2025), 4171–4184. doi:10.14778/3749646.3749685

[36] PengLi and TianxiangSun ect all. 2023. CodeIE: Large Code Generation Models are Better Few-Shot Information Extractors. (May 2023).

[37] Mohammadreza Pourreza, Hailong Li, Ruoxi Sun, Yeounoh Chung, Shayan Talaei, Gaurav Tarlok Kakkar, Yu Gan, Amin Saberi, Fatma Ozcan, and Sercan O Arik. 2025. CHASE-SQL: Multi-Path Reasoning and Preference Optimized Candidate Selection in Text-to-SQL. In *The Thirteenth International Conference on Learning Representations*. https://openreview.net/forum?id=CvGqMD5OtX

[38] Mohammadreza Pourreza and Davood Rafiei. 2023. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *Advances in Neural Information Processing Systems* 36 (2023), 36339–36348.

[39] Stephen Robertson. 2007. On score distributions and relevance. In *Proceedings of the 29th European Conference on IR Research* (Rome, Italy) *(ECIR'07)*. Springer-Verlag, Berlin, Heidelberg, 40–51.

[40] Mohammed Saeed, Nicola De Cao, and Paolo Papotti. 2024. Querying Large Language Models with SQL. In *Proceedings of the 27th International Conference on Extending Database Technology (EDBT)*. 365–372. doi:10.48786/edbt.2024.32

[41] Oscar Sainz, Iker Garcia-Ferrero, Rodrigo Agerri, OierLopezde Lacalle, German Rigau, and Eneko Agirre. 2023. GoLLIE: Annotation Guidelines improve Zero-Shot Information-Extraction. (Oct 2023).

[42] Nikolaus Salvatore, Hao Wang, and Qiong Zhang. 2025. Lost in the Middle: An Emergent Property from Information Retrieval Demands in LLMs. arXiv:2510.10276 [cs.LG] https://arxiv.org/abs/2510.10276

[43] Shreya Shankar, Tristan Chambers, Tarak Shah, Aditya G. Parameswaran, and Eugene Wu. 2025. DocETL: Agentic Query Rewriting and Evaluation for Complex Document Processing. *Proceedings of the VLDB Endowment* 18, 9 (2025), 3035–3048. doi:10.14778/3746405.3746426

[44] Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed H Chi, Nathanael Schärli, and Denny Zhou. 2023. Large language models can be easily distracted by irrelevant context. In *International Conference on Machine*

Learning. PMLR, 31210–31227.

[45] Ananya Singha, José Cambronero, Sumit Gulwani, Vu Le, and Chris Parnin. 2023. Tabular Representation, Noisy Operators, and Impacts on Table Structure Understanding Tasks in LLMs. In *NeurIPS 2023 Second Table Representation Learning Workshop*.

[46] Yuan Sui, Mengyu Zhou, Mingjie Zhou, Shi Han, and Dongmei Zhang. 2024. Table Meets LLM: Can Large Language Models Understand Structured Table Data? A Benchmark and Empirical Study. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining* (Merida, Mexico) *(WSDM '24)*. Association for Computing Machinery, New York, NY, USA, 645–654. doi:10.1145/3616855.3635752

[47] Yuan Sui, Mengyu Zhou, Mingjie Zhou, Shi Han, and Dongmei Zhang. 2024. Table meets llm: Can large language models understand structured table data? a benchmark and empirical study. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*. 645–654.

[48] Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. [n. d.]. Dynamic cheatsheet: Test-time learning with adaptive memory, 2025. *URL https://arxiv. org/abs/2504.07952* ([n. d.]).

[49] Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. 2024. Chess: Contextual harnessing for efficient sql synthesis. *arXiv preprint arXiv:2405.16755* (2024).

[50] Zirui Tang, Boyu Niu, Xuanhe Zhou, Boxiu Li, Wei Zhou, Jiannan Wang, Guoliang Li, Xinyi Zhang, and Fan Wu. 2025. ST-Raptor: LLM-Powered Semi-Structured Table Question Answering. *Proc. ACM Manag. Data* 3, 6, Article 364 (Dec. 2025), 27 pages. doi:10.1145/3769829

[51] Matthias Urban and Carsten Binnig. 2024. CAESURA: Language Models as Multi-Modal Query Planners. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*. https://www.cidrdb.org/cidr2024/papers/p14-urban.pdf

[52] Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Linzheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, et al. 2025. MAC-SQL: A Multi-Agent Collaborative Framework for Text-to-SQL. In *Proceedings of the 31st International Conference on Computational Linguistics*. Association for Computational Linguistics, Abu Dhabi, UAE, 540–557.

[53] Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020. Rat-sql: Relation-aware schema encoding and linking for text-to-sql parsers. In *Proceedings of the 58th annual meeting of the association for computational linguistics*. 7567–7578.

[54] Qiuchen Wang, Ruixue Ding, Zehui Chen, Weiqi Wu, Shihang Wang, Pengjun Xie, and Feng Zhao. 2025. Vidorag: Visual document retrieval-augmented generation via dynamic iterative reasoning agents. *arXiv preprint arXiv:2502.18017* (2025).

[55] Tianshu Wang, Xiaoyang Chen, Hongyu Lin, Xianpei Han, Le Sun, Hao Wang, and Zhenyu Zeng. 2025. Dbcopilot: Natural language querying over massive databases via schema routing. *Preprint* (2025).

[56] 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 *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 1780–1790.

[57] 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 *The Twelfth International Conference on Learning Representations*.

[58] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems* 35 (2022), 24824–24837.

[59] Colin White, Samuel Dooley, Manley Roberts, Arka Pal, Benjamin Feuer, Siddhartha Jain, Ravid Shwartz-Ziv, Neel Jain, Khalid Saifullah, Sreemanti Dey, Shubh-Agrawal, Sandeep Singh Sandha, Siddartha Venkat Naidu, Chinmay Hegde, Yann LeCun, Tom Goldstein, Willie Neiswanger, and Micah Goldblum. 2025. LiveBench: A Challenging, Contamination-Limited LLM Benchmark. In *The Thirteenth International Conference on Learning Representations*. https://openreview.net/forum?id=sKYHBTAxVa

[60] Pengzuo Wu, Yuhang Yang, Guangcheng Zhu, Chao Ye, Hong Gu, Xu Lu, Ruixuan Xiao, Bowen Bao, Yijing He, Liangyu Zha, et al. 2025. RealHiTBench: A Comprehensive Realistic Hierarchical Table Benchmark for Evaluating LLM-Based Table Analysis. In *Findings of the Association for Computational Linguistics: ACL 2025*. Association for Computational Linguistics, Vienna, Austria, 7105–7137. doi:10.18653/v1/2025.findings-acl.371

[61] 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 *Findings of the Association for Computational Linguistics: ACL 2024*. Association for Computational Linguistics, Bangkok, Thailand, 10796–10816. doi:10.18653/v1/2024.findings-acl.641

[62] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*. 2369–2380.

[63] 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*. 174–184.

[64] Richard Zanibbi, Dorothea Blostein, and James R Cordy. 2004. A survey of table recognition: Models, observations, transformations, and inferences. *Document Analysis and Recognition* 7, 1 (2004), 1–16.

[65] Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, et al. 2025. Agentic context engineering: Evolving contexts for self-improving language models. *arXiv preprint arXiv:2510.04618* (2025).

[66] Yunjia Zhang, Jordan Henkel, Avrilia Floratou, Joyce Cahoon, Shaleen Deep, and Jignesh M. Patel. 2024. ReAcTable: Enhancing ReAct for Table Question Answering. *Proc. VLDB Endow.* 17, 8 (April 2024), 1981–1994. doi:10.14778/3659437.3659452

[67] Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. 2025. Qwen3 Embedding: Advancing Text Embedding and Reranking Through Foundation Models. *arXiv preprint arXiv:2506.05176* (2025).

[68] Zhehao Zhang, Yan Gao, and Jian-Guang Lou. 2024. e5: Zero-shot hierarchical table analysis using augmented LLMs via explain, extract, execute, exhibit and extrapolate. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*. 1244–1258.

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

[70] Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. 2024. Memorybank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 19724–19731.

Received 17 January 2026; revised 19 May 2026; accepted 12 June 2026