SHEET-RAPTOR: RECONSTRUCTING CONTEXT FOR SPREADSHEET REASONING

Anonymous authors
Paper under double-blind review

ABSTRACT

Spreadsheets are widely used in real-world data analysis. However, their multi-sheet organization and irregular table structures require understanding both structure and semantics within and across files, which conventional database techniques do not adequately support. Existing LLM-, RAG-, and agent-based methods struggle with lengthy and redundant spreadsheet content, fragmented context from row-level retrieval, and limited flexibility across analysis tasks. To address these challenges, we propose SHEET-RAPTOR, a unified system for context-critical spreadsheet analysis. First, it combines structure-aware preprocessing with layout-sensitive folding to reduce redundancy while preserving semantically meaningful table regions. Second, it integrates multi-granular spreadsheet embeddings, question rewriting, and progressive agentic retrieval to recover relevant context across files. Third, it decomposes queries into spreadsheet-specific atomic operations and dynamically coordinates semantic reasoning and programmatic execution. Experiments on four benchmarks show that SHEET-RAPTOR consistently outperforms state-of-the-art baselines, improving accuracy by up to 18.74%.

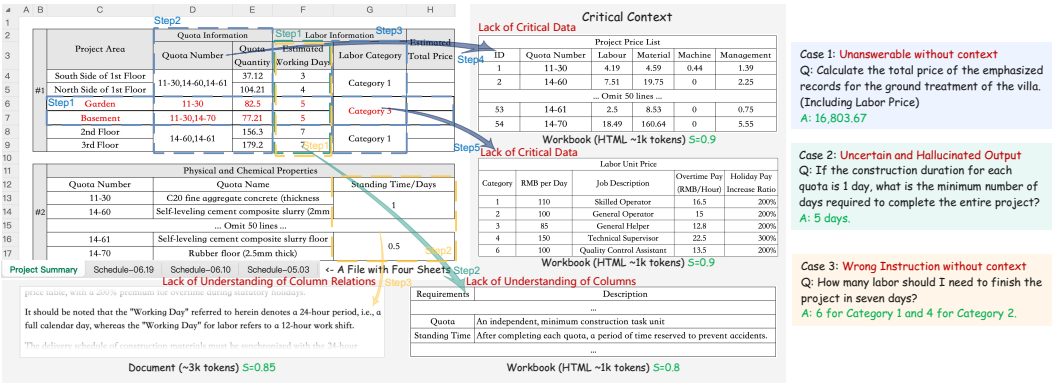

Figure 1: **Context-critical spreadsheet analysis.** Evidence from other sheets and files is required to recover missing values, interpret domain-specific terms, and execute the intended operation.

1 INTRODUCTION

Spreadsheets serve as a primary interface for operational data analysis across a wide range of domains (Li et al., 2017; Tang et al., 2026b). Unlike relational databases, however, spreadsheets lack both predefined schemas and a clear separation between data and presentation. A single workbook may contain multiple sheets, where spatial layout, adjacent tables, merged headers, and formatting jointly encode task-relevant structure and semantics. Moreover, spreadsheet semantics often extend beyond individual files through explicit references and implicit semantic dependencies. Our analysis of more than 5,000 enterprise and public spreadsheets spanning over 30 scenarios and 10 domains shows that each workbook contains an average of 2.23 sheets, with 1.43 tables per sheet. Notably, 60.58% of the tables exhibit inter-file dependencies, including implicit semantic relations in 53.46% and explicit links in 35.13%. As a result, answering even a seemingly localized analytical query

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

may require identifying the relevant table region, interpreting its layout-dependent structure, and integrating supporting evidence from semantically related spreadsheets.

Existing approaches address only parts of this problem. Text-based and table-specialized models mix relevant cells with repetitive content and weaken layout cues (Zhang et al., 2025). Vision-language models preserve layout but scale poorly and remain unreliable for exact computation (Zheng et al., 2024). RAG supplies evidence, yet row-wise chunks detach values from headers while whole-table chunks add noise. Tool-using agents support computation and editing, but their plans depend on correctly identified regions and cross-file relations (Zhang et al., 2024; Xiong et al., 2025; Wang et al., 2025; Chen et al., 2025). Errors propagate when structure is handled separately from retrieval and execution.

We study *context-critical spreadsheet analysis*, where a system uses auxiliary files to analyze a target spreadsheet and fulfill a user request. Failures in Figure 1 motivate three challenges. First, large sheets contain repetitive content that obscures implicit table boundaries, while a spreadsheet file contains several spreadsheets. Understanding them requires identifying coherent regions and folding redundancy while preserving layout semantics **(C1)**. Achieving effective spreadsheet context retrieval remains a challenge. Evidence scattered across sheets and files must be retrieved as complete context. Row-level fragmentation can detach values from headers, while irrelevant content introduces noise **(C2)**. What's more, different types of tasks require diverse reasoning strategies. Semantic QA needs contextual reasoning, computation and visualization require program execution, while spreadsheet editing requires iterative feedback and replanning as operations change spreadsheet state **(C3)**.

We address these challenges with SHEET-RAPTOR. Its structure-aware preprocessor recognizes table regions from a location-enhanced serialization and folds repetition while preserving layout cues. A hierarchical retriever uses these regions to rewrite requests and progressively acquires sufficient context. A routing agent decomposes each request and dispatches subtasks to solvers that combine semantic reasoning with exact program execution.

Using SHEET-RAPTOR, we find that access to relevant evidence does not automatically translate into correct spreadsheet reasoning. Retrieval is particularly important for inter-file QA, where the answer depends on information outside the target spreadsheet. Yet program-oriented QA remains substantially weaker under single-shot VLM inference even when retrieval is retained. This contrast points to a gap between locating evidence and using it correctly. Together, these findings suggest that reliable reasoning depends not just on finding evidence, but on preserving its table structure and connecting its semantics to executable operations.

2 PROBLEM SETUP AND SYSTEM OVERVIEW

2.1 CONTEXT-CRITICAL SPREADSHEET ANALYSIS

A spreadsheet file $F = (n, \mathcal{S})$ has a name n and a set of sheets $\mathcal{S} = \{S_1, \ldots, S_m\}$. We represent a table as $T = (H, D, M)$, where H contains header cells, D contains data cells, and $M = (F, S, R)$ identifies the source and rectangular range. Each cell retains its coordinate and value along with formatting that may determine the user's intent.

Given a request q, a primary table T_p, and an auxiliary spreadsheet collection $\mathcal{F}$, the relevant context is $\mathcal{C} = \mathcal{C}_{\text{intra}} \cup \mathcal{C}_{\text{inter}}$. Here, $\mathcal{C}_{\text{intra}}$ contains other tables in T_p's file, whereas $\mathcal{C}_{\text{inter}}$ contains task-relevant tables from other files. The system first selects $\mathcal{C}$ and then maps $(q, T_p, \mathcal{C})$ to an output o. This output can be a direct answer, a modified spreadsheet, or plotting code with its visualization. The formulation therefore covers both semantic questions and state-changing tasks while making context selection explicit.

2.2 SYSTEM OVERVIEW

Figure 2 shows the three-stage architecture. **SheetPreprocessor** turns raw sheets into compact tables that preserve location. An LLM recognizes implicit boundaries from a coordinate-augmented serialization before repeated regions are folded. **ContextRetriever** builds a four-level semantic index over auxiliary spreadsheets. It rewrites the request using the primary table and progressively

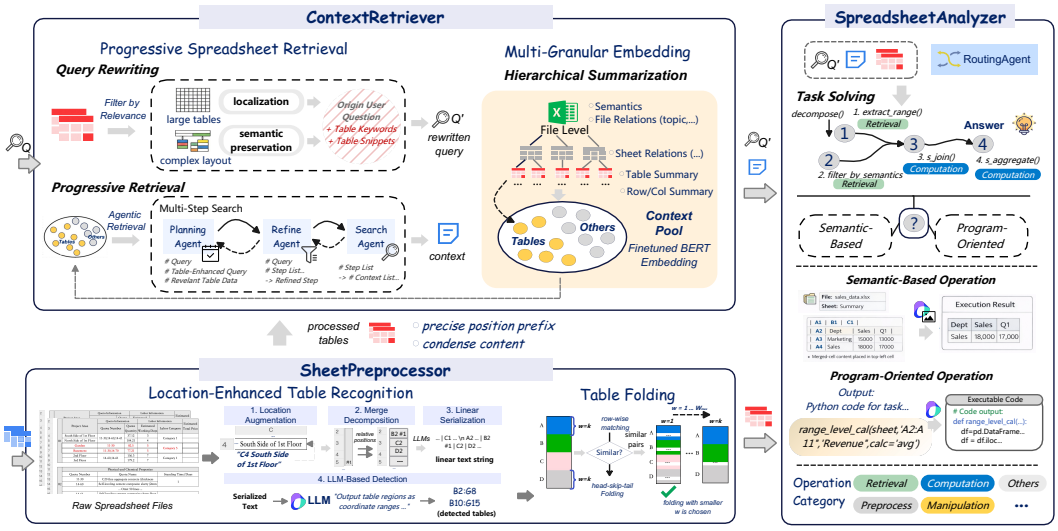

Figure 2: **Overview of SHEET-RAPTOR.** The system exposes structure during preprocessing, retrieves context at multiple spreadsheet granularities, and executes each subtask with the appropriate semantic or programmatic solver.

collects complete contextual tables. **SpreadsheetAnalyzer** represents the enriched request as a directed acyclic graph of operations. A routing agent assigns each subtask to a task-specific solver and chooses between semantic and programmatic execution.

For the running example in Figure 1, preprocessing detects five tables and folds a 56-row material table to five representative rows. The retriever uses the target-table semantics to locate two external price tables. The routing agent then builds a plan where relevant ground-treatment rows are selected before quantities are joined with their unit prices. An exact aggregation produces the final total. This workflow illustrates why the stages must share table boundaries and metadata. Retrieval identifies the operands that execution subsequently manipulates.

3 METHOD

SHEET-RAPTOR shares a location-aware table representation across all three modules. The preprocessor converts each raw spreadsheet into compact tables. Their hierarchy and summaries allow the retriever to acquire auxiliary evidence, after which the analyzer executes a task graph over the primary and retrieved tables. This coupling prevents an early structural error from being hidden by a later generic retrieval or planning stage.

3.1 STRUCTURE-AWARE SPREADSHEET PREPROCESSING

Location-enhanced table recognition. A sheet can contain adjacent or nested tables whose boundaries are implicit. Treating the whole sheet as one table conflates headers with data, whereas rule-based blank-row segmentation fails on merged cells and decorative gaps. We serialize every non-empty cell with its coordinate before its value. For example, the content of cell `C4` is rendered as `C4` followed by that content. A vertical bar separates cells and a newline separates rows, so the model sees both the values and their original layout. A merged region is expanded into individual coordinates, with its content in the top-left cell and coordinate markers in the remaining cells. Given this representation, an LLM predicts table ranges such as `B2:G8` and `B10:G15`. We materialize each range as an independent table whose identifier records the source file, sheet, and range. The identifier distinguishes two neighboring regions even when they contain similar headers, and the coordinates remain attached throughout retrieval and execution.

Structure-aware folding. Even after recognition, long sheets contain repeated structures that consume model context without adding new layout evidence. At row i, a sliding window tests block heights $w \in [1, W_{\max}]$ and counts structurally similar adjacent blocks (Algorithm 1). For two rows R_a, R_b with m columns, exact values receive a per-column score s_j of 1, while a mismatch in merged-cell layout receives 0. Other pairs receive 0.8 when both cells are numeric, 0.7 when text is compared with a number or date, and 0.6 when only one cell is empty. We define

$$\mathrm{sim}(R_a, R_b) = \frac{1}{m}\sum_{j=1}^{m} s_j, \qquad R_a \simeq R_b \iff \mathrm{sim}(R_a, R_b) \geq \tau, \tag{1}$$

subject to identical merge spans. We set $\tau = 0.9$ and reject a match when opposing empty/nonempty pairs exceed both five cells and 30% of the row. The same procedure handles column-wise repetition by transposing the table.

The scan begins at the first row and compares adjacent blocks of height w, incrementing a repetition count while their corresponding rows satisfy Equation 1. When the count reaches $\rho_{\min}$, the matched region is recorded and the scan resumes after it. Otherwise, the scan advances one row. This prevents a repetitive block from being detected again at each of its interior positions. For every detected region, we replace the repeated middle with a folding marker while preserving the first two rows and the last row. This *head–skip–tail* view exposes the repeated pattern and its boundary without discarding the full table, which remains available to programmatic operations. For N rows, the average complexity is $O(W_{\max}^2 N)$ with $W_{\max} < 10$ in our implementation. A 36-row monthly table can therefore be rendered with its first two rows, a folding marker, and the final row while later code retains access to every exact value.

Folding is a change to the visual view, not a value-level deduplication rule. Two rows can share a layout while containing different numbers, so the omitted cells must not be replaced by the representative values during computation. The analyzer pairs the folded image with a textual table representation and uses the original spreadsheet for exact range operations. The image communicates the repeated structure, while coordinates identify the cells that later code reads. This separates the cost of presenting a long table from the need to preserve its complete data.

3.2 Hierarchical Context Retrieval

Conventional RAG partitions a spreadsheet into independent rows or fixed token chunks. This breaks the association between values and their structural context. Indexing a whole table has the opposite problem because irrelevant cells can overwhelm the retriever. SHEET-RAPTOR instead represents each auxiliary spreadsheet with a four-level summary hierarchy.

$$\mathcal{I}_F = \{\mathcal{L}^f, \mathcal{L}^s_i, \mathcal{L}^t_{i,j}, \mathcal{L}^{rc}_{i,j,k}\}, \tag{2}$$

where file summaries describe global topics and sheet relations. Sheet summaries capture local topics and relations between tables, while each table summary describes one recognized region. The lowest level contains row- or column-oriented slices of 100 rows by default. A retrieval hit returns the unit associated with its summary. If several levels from one spreadsheet match, we keep the finest unit to avoid duplicate evidence while retaining its ancestry metadata.

The hierarchy separates the text used for matching from the evidence used to answer the task. A file-level summary can match a broad topic even when a particular sheet name is absent from the question. A row-level hit can instead expose only the relevant part of a very long table. Both hits retain the parent file and sheet identity. Returning the underlying unit, rather than asking the analyzer to reason from a summary alone, preserves the original cells and their coordinates for subsequent operations.

Summary embedding and reranking. We train a RoBERTa-based dual encoder on about five million task–context pairs drawn from open and enterprise QA data, including 500K synthetic pairs. In-batch contrastive learning brings each task close to its relevant context and treats the other contexts in the minibatch as negatives. At inference, approximate nearest-neighbor search returns $k = 200$ candidates. A separately trained cross-encoder scores each concatenated task–context pair and keeps $c = 10$. Its training contrasts the relevant context against the other retrieved candidates,

allowing the first stage to favor recall and the second to resolve close semantic matches. The hierarchy remains compact because a 164,664-row spreadsheet produces about 1.6K embeddings and requires roughly 8 MB.

Table-conditioned question rewriting. The user's request can be underspecified without the primary table, but appending the entire table adds noise. We therefore combine the request with table-topic keywords and question-relevant snippets. For a large, regular table, lexical matching locates task-related records and includes nearby rows so that the matched values retain their headers and local context. When layout is important, a VLM instead reads the folded image and coordinate-enhanced text to recover relationships that text matching may miss, such as a session nested below day and track headers. An LLM then rewrites the request as a self-contained search question. For example, a request for contacts of selected drugs is augmented with the matched drug rows and relevant contact fields instead of all catalog rows. The rewrite narrows the search while preserving the entities needed to match auxiliary files.

Progressive retrieval. A fixed number of search rounds cannot anticipate whether a task needs one lookup or a multi-hop join. We maintain accumulated evidence $\mathcal{K}$ within a loop that separates planning from execution (Algorithm 2). The planning agent proposes focused search steps, which a refine agent checks before the search agent executes them against the hierarchy. The plan may be refined again up to a bounded planning limit before search proceeds. After each round, the refine agent checks the combined old and new evidence for sufficiency. If a needed entity or relation remains unresolved, the next plan uses both the original question and $\mathcal{K}$, rather than repeating the initial query. Retrieval ends when the evidence is sufficient or the maximum-round guard is reached. Every returned unit retains its complete location identity throughout this loop.

More explicitly, let $\mathcal{P}_r$ be the finalized search plan at round r, with $\mathcal{K}_0 = \emptyset$. The evidence update is

$$(\mathcal{K}_{r+1},\, b_r) = \mathrm{RefineEvidence}\left(\mathcal{K}_r,\, \bigcup_{s \in \mathcal{P}_r} \mathrm{Search}(s)\right), \tag{3}$$

where b_r is the sufficiency decision. If it is false, the next planning state is $(q, \mathcal{K}_{r+1})$. The original request remains in this state so that evidence about an intermediate entity does not replace the user's goal. For a conference-schedule question, the first search may identify a speaker through a session entry, after which a later search uses that speaker to locate additional information. The analyzer receives the final evidence list only after this loop, so search refinement and spreadsheet execution remain distinct stages.

3.3 Task-Adaptive Agentic Analysis

Spreadsheet tasks mix ambiguous semantic predicates (e.g., "high-risk customers") with deterministic operations (e.g., summing `B4:C6`). A single modality is therefore brittle to layout semantics or imprecise for computation. Our operations use two complementary templates, with their categories detailed in Appendix B.2.

Semantic and program-oriented operations. The *semantic template* supplies a VLM with the operation description and the location-aware table representation. It also includes the folded image and representative examples. This template handles operations where meaning must be inferred from context. For instance, `filter_by_semantics` identifies regions that satisfy a description such as "high-risk customers" by reading their headers and spatial alignment. A semantic join can align corresponding fields across tables even when their column names differ. The *program-oriented template* replaces the image with spreadsheet-specific code examples and asks an LLM to produce executable code. It handles exact operations on ranges, formulas, and cells. An `extract_range` call uses explicit cell coordinates to select a rectangle, and `range_level_cal` then computes over that selection. Because both templates implement the same logical interface, a plan can combine semantic row selection with deterministic range-level calculation without turning the selected region into an ungrounded textual description.

This distinction matters for non-relational layouts. A heading in one merged cell may apply to several rows, so semantic filtering must interpret that layout. Once the region is selected, code

extracts its exact cells without asking the model to copy values into a prompt. Execution templates are selected per operation, not per request.

Routing and solver agents. The routing agent receives $(q, T_p, \mathcal{C})$ and emits typed subtasks in a constrained schema, where q is the user request, T_p contains the primary tables, and $\mathcal{C}$ is the retrieved context. A QA agent chooses direct semantic reasoning for a lightweight question or decomposes a computation that needs several table operations. Editing requests go to a manipulation agent, which applies one scoped change at a time and verifies the updated spreadsheet state before proposing another edit. A visualization agent identifies the relevant data and chart specification before generating and validating plotting code. New solver types can be added without changing the preprocessing or retrieval interfaces.

The execution strategy also depends on whether the output changes spreadsheet state. QA and visualization use a plan-then-execute workflow, where the requested result determines the necessary data dependencies before execution. Manipulation instead follows reflection-then-action and replans from the updated sheet after each operation. This distinction matters because a structural edit can change the ranges needed by a subsequent edit. The solver therefore does not blindly execute an initial sequence against stale coordinates. For visualization, planning resolves the data scope before code generation, so the plotted values come from the selected table regions rather than being reconstructed from a natural-language summary.

DAG planning and execution. For complex subtasks, the solver constructs an operation DAG (Algorithm 3). Each node specifies an operation together with its inputs and execution template, while edges encode data dependencies. The execution engine follows a topological order and caches intermediate state for dependent nodes. In the running example, semantic row selection determines which price and quantity ranges are extracted. These ranges are then joined with the external price tables before an exact aggregation is executed. The output of each node, including selected coordinates and computed values, is passed to its successors. Explicit state prevents these artifacts from being lost in conversational summaries and allows the final answer to remain grounded in the same table regions identified at preprocessing time.

4 EXPERIMENTS

We evaluate SHEET-RAPTOR on both general and context-critical spreadsheet analysis. We then use controlled ablations to identify where its gains originate.

4.1 EXPERIMENTAL SETUP

Benchmarks. ContextSheetBench contains 259 professionally annotated tasks over 147 production-derived spreadsheets. Among them, 199 are question-answering requests, while manipulation and visualization account for 18 and 42 tasks. The spreadsheets contain irregular layouts and tables with up to 164,664 rows. A task may depend on as many as four context files. We also evaluate on RealHiTBench (Wu et al., 2025b), TableBench (Wu et al., 2025c), and TableEval (Zhu et al., 2025). These three public benchmarks cover complementary forms of table analysis and reasoning.

ContextSheetBench separates capabilities that aggregate QA would otherwise conflate. Inter-file reasoning is the most common contextual requirement, while program-oriented analysis appears in nearly half of the tasks. The categories overlap because a request may require several capabilities. Appendix A.1 gives the full capability and operation profile.

Baselines and protocol. We compare against ten baselines that span foundation models, table-finetuned models, RAG methods, and spreadsheet agents. Seed-1.6 provides the foundation LLM and VLM (ByteDance Seed, 2025), while TableLLM and TableLLaVA represent table-specific tuning (Zhang et al., 2025; Zheng et al., 2024). The retrieval baselines are Vanilla RAG and DeepSearch (Singh et al., 2025). We further include representative spreadsheet agents (Zhang et al., 2024; Xiong et al., 2025; Wang et al., 2025; Chen et al., 2025). All agent baselines and SHEET-RAPTOR use the same Seed-1.6 backbone. The LLM receives Markdown tables, whereas the VLM

Table 1: Accuracy (%) on four benchmarks. DA, FC, NR, VA, IR, TR, QA, and MA denote data analysis, fact checking, numerical reasoning, visualization, information retrieval, table reasoning, question answering, and manipulation. Unsupported tasks are marked "–" and excluded from the corresponding total. Best results are bold, and strongest baselines are underlined.

	RealHiT				TableBench					TableEval					ContextSheetBench			
Method	Total	DA	FC	NR	Total	DA	FC	NR	VA	Total	DA	NR	IR	TR	Total	QA	MA	VA
Seed-1.6 LLM	69.34	44.84	77.73	73.02	69.04	43.15	89.58	86.62	–	85.16	78.75	89.83	89.32	81.16	23.62	23.62	–	–
Seed-1.6 VLM	70.00	46.59	77.58	74.12	70.17	46.65	90.62	85.61	–	84.56	77.00	90.11	89.01	79.71	24.62	24.62	–	–
TableLLM	22.28	14.82	28.84	17.06	22.49	15.45	72.92	17.68	–	25.45	23.58	17.77	33.28	28.99	3.03	3.03	–	–
TableLLaVA	9.65	9.94	9.78	9.24	8.70	9.91	28.12	3.79	–	5.53	9.75	3.11	3.61	2.90	0.51	0.51	–	–
ReAcTable	31.34	12.05	46.29	41.15	31.62	15.74	66.67	36.87	–	48.06	68.41	42.49	37.05	19.57	4.55	4.55	–	–
TableZoomer	37.48	12.05	42.41	27.21	56.41	29.15	82.29	73.34	–	42.10	52.98	39.93	36.01	24.64	10.83	11.11	–	–
SheetBrain	65.23	33.17	76.82	68.75	58.88	26.37	85.42	76.95	87.76	80.93	83.84	81.14	79.91	70.29	32.92	25.76	31.25	66.67
SheetAgent	20.91	14.50	23.02	22.01	29.27	20.07	35.42	36.11	22.00	65.10	62.10	71.61	64.26	57.97	13.68	11.62	37.50	13.33
Vanilla RAG	69.34	44.84	77.73	73.02	69.04	43.15	89.58	86.62	–	85.16	78.75	89.83	89.32	81.16	31.31	31.31	–	–
DeepSearch	69.34	44.84	77.73	73.02	69.04	43.15	89.58	86.62	–	85.16	78.75	89.83	89.32	81.16	32.83	32.83	–	–
SHEET-RAPTOR	**74.22**	**53.66**	**82.58**	**75.23**	**76.05**	**55.98**	**91.67**	**87.91**	**89.04**	**89.04**	**86.77**	**91.39**	**89.48**	**88.41**	**51.66**	**47.47**	**43.75**	**74.41**

receives rendered images. We first apply exact matching, where a prediction must contain the ground truth. Remaining cases are assessed by Seed-1.6 as a binary judge, and task accuracy is the fraction judged correct. Scores are percentages.

For the folding scores in Equation 1, we sweep non-exact match values from 0 to 1 on 87 manually annotated tables and select the configuration with the best Tree Edit Distance Similarity. The chosen $(0.8, 0.7, 0.6)$ scores outperform the second-best configuration by 7.27%.

4.2 MAIN RESULTS

General spreadsheet analysis. Table 1 shows that SHEET-RAPTOR is best on all four benchmarks. Relative to the strongest baseline, its total-score gains are 4.22 points on RealHiTBench, 5.88 on TableBench, and 3.88 on TableEval. The data-analysis breakdown reveals larger improvements of 7.07, 9.33, and 2.93 points, respectively. These tasks require several transformations over spreadsheet regions, which benefits from explicit table boundaries and executable operations. The smaller TableEval margin is consistent with its strongest baseline already reaching 83.84% on this category. The public benchmarks provide no auxiliary context pool, so the RAG variants reduce to the Seed-1.6 LLM and share its scores.

The finer-grained results show that the aggregate gains do not come from one favorable task category. On RealHiTBench, SHEET-RAPTOR improves fact checking by 4.85 points and numerical reasoning by 1.11 points over the strongest baselines. On TableBench, it establishes new best results for fact checking, numerical reasoning, and visualization, where the margins are 1.05, 1.29, and 1.28 points. The largest category-level improvement appears on TableEval table reasoning, where the score rises from 81.16% to 88.41%. Information retrieval on TableEval is nearly saturated among the strongest systems, yet SHEET-RAPTOR still obtains the highest result. This pattern is consistent with the method design. Structure-aware preprocessing helps preserve the evidence needed for checking and retrieval, while programmatic execution benefits calculations and table reasoning.

Context-critical analysis. The gap is largest on ContextSheetBench, where SHEET-RAPTOR reaches 51.66% total accuracy and exceeds the strongest baseline by 18.74 points. It is also the only method that leads across every task type. Generic RAG raises Seed-1.6 QA from 23.62% to between 31.31% and 32.83%, yet remains far below SHEET-RAPTOR at 47.47%. This gap shows that retrieval alone does not resolve spreadsheet fragmentation. SheetBrain is the strongest full-task baseline and trails SHEET-RAPTOR by 18.74 points overall. The gap to SheetAgent is 37.98 points. These results support exposing the same table boundaries to retrieval and execution instead of attaching a generic tool loop to raw files. The absolute score of 51.66% also shows that context-critical spreadsheets remain far from solved when structural and execution capabilities must work together.

Figure 3 localizes the improvement. On inter-file questions, SHEET-RAPTOR reaches 46.8%, compared with 26.8% for DeepSearch and 24.6% for Vanilla RAG. This gap directly measures the benefit of spreadsheet-specific retrieval. The corresponding gains over the strongest baselines are 13.7

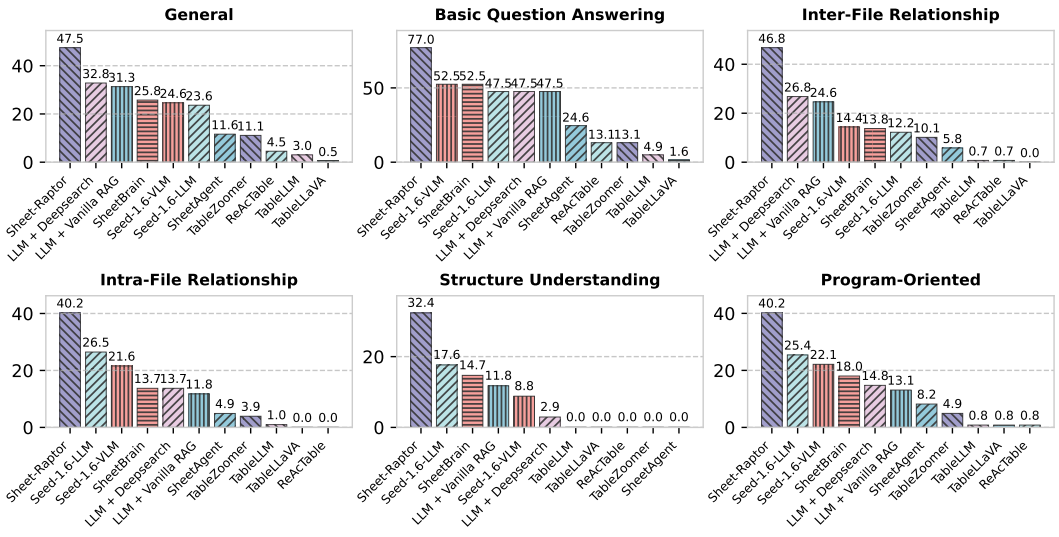

Figure 3: ContextSheetBench QA accuracy by capability. The same question can belong to multiple categories. SHEET-RAPTOR leads in every category.

points for intra-file reasoning and 14.8 points for structure understanding. Program-oriented accuracy improves by the same 14.8-point margin. The gains therefore extend beyond one favorable task family. Preprocessing benefits questions that depend on local structure, retrieval contributes most strongly across files, and task-adaptive execution supports program-oriented questions.

4.3 ABLATION STUDY

Table 2: Component ablations on ContextSheetBench QA (accuracy, %). The QA total corresponds to the QA column in Table 1. Manipulation and visualization are excluded, and each removal uses the same backbone as the full system.

Method	QA total	Basic	Inter	Intra	Struct.	Program
Full system	**47.47**	**77.05**	**46.76**	**40.20**	**32.35**	**40.16**
w/o preprocessing	44.61	63.93	44.78	31.63	18.18	35.59
w/o context retrieval	35.18	72.13	16.55	14.71	11.76	20.49
w/o agentic analysis	38.67	55.74	27.54	22.55	17.65	23.77
w/o spreadsheet agent	41.71	34.43	45.32	36.27	25.27	34.43

Table 2 aligns each gain with a claimed component. Removing preprocessing reduces structure understanding by 14.17 points and intra-file reasoning by 8.57 points. This result confirms that region recognition and folding preserve layout evidence that downstream models cannot recover from raw sheets. Context retrieval has the largest effect because its removal lowers inter-file accuracy from 46.76% to 16.55% and QA total by 12.29 points. Replacing agentic analysis with single-shot VLM inference reduces QA total by 8.80 points and program-oriented accuracy by 16.39 points. A general agent also loses 5.76 QA points relative to the spreadsheet-specific agent. Together, these results show that every module contributes and that retrieved evidence alone is insufficient without spreadsheet-aware execution.

4.4 BACKBONE SENSITIVITY

Table 3 tests whether the system depends on its default Seed-1.6 backbone (ByteDance Seed, 2025). We retain the same pipeline while replacing the model with a lightweight Seed variant, Qwen3-VL-Plus (Bai et al., 2025), or GPT models (OpenAI, 2025). Qwen3-VL-Plus reaches 46.73%, which is close to the default system despite changing the model family and modality. GPT-5.2 achieves the best total score of 57.79% and leads on every context-dependent capability, while Seed-1.6 remains

Table 3: ContextSheetBench QA accuracy (%) with different LLM/VLM backbones. All variants use the same SHEET-RAPTOR pipeline.

Backbone	Total	Basic	Inter	Intra	Struct.	Program
Seed-1.6	47.47	**77.05**	46.76	40.20	32.35	40.16
Seed-1.6-flash	40.20	62.30	29.50	19.61	20.59	20.49
Qwen3-VL-Plus	46.73	63.93	39.57	29.41	29.41	31.15
GPT-5	44.22	60.66	37.41	26.47	23.53	30.33
GPT-5.2	**57.79**	62.71	**53.24**	**43.14**	**36.26**	**45.90**

strongest on basic QA. The 7.27-point drop from Seed-1.6 to its flash variant also exposes a clear capacity–quality trade-off. These results show that the spreadsheet-centric pipeline transfers across backbones, although the remaining performance range indicates that model capability still matters.

This comparison is intended as a sensitivity analysis rather than a model leaderboard. It uses one configuration per backbone and does not tune the retrieval index or agent prompts separately for each model. The consistent performance across model families supports the architectural contribution, while the variation in absolute accuracy motivates future work on backbone-specific adaptation.

5 RELATED WORK

Spreadsheet understanding and agents. Early spreadsheet research addresses structural understanding and transformation through tasks such as region detection or formula prediction (Dong et al., 2019; Chen et al., 2021; He et al., 2023; Li et al., 2023b; Bendre et al., 2015). Recent systems use LLMs to analyze longer and less regular sheets. SpreadsheetLLM compresses sheets into structural skeletons (Dong et al., 2025), while ST-Raptor retrieves regions within a fixed analysis pipeline (Tang et al., 2026a). SheetCopilot operates spreadsheets through atomic interface actions (Li et al., 2023a). Other agents improve execution by combining structured views with either SQL-like operations or symbolic planning (Chen et al., 2025; Wang et al., 2025). These systems improve scale or executability but generally treat a workbook as an isolated input. SHEET-RAPTOR instead integrates cross-sheet and cross-file evidence throughout the analysis process.

Table reasoning. Table reasoning commonly assumes tables with explicit schemas (Liu et al., 2022; Cheng et al., 2023). Prompting methods decompose questions into transformations (Ye et al., 2023; Wang et al., 2024), while tool-using agents refine tables with SQL or Python (Zhang et al., 2024). TableZoomer reduces cost through query-dependent sub-schema selection (Xiong et al., 2025). Multi-table QA methods instead follow key-based relations (Wu et al., 2025a). Real spreadsheets violate these assumptions because merged cells and hierarchical headers interact with formatting and free-form regions. Our preprocessor recovers explicit, location-aware table units before applying retrieval or tools.

Table retrieval and agentic RAG. RAG grounds generation in external evidence (Lewis et al., 2020). Table-specific variants further expand queries and retrieve signals tied to schemas or cells (Chen et al., 2024). Self-RAG and Adaptive RAG allow a model to decide whether and how to retrieve (Asai et al., 2024; Jeong et al., 2024). Fixed row chunks, however, separate spreadsheet values from their structural context, whereas whole-table chunks often contain more information than a request needs. SHEET-RAPTOR addresses both problems through a hierarchical summary index, table-conditioned question rewriting, and progressive stopping.

6 CONCLUSION

We presented SHEET-RAPTOR, which shares location-aware table units across preprocessing, retrieval, and task-specific execution. Recognition and folding expose compact, grounded tables. Hierarchical retrieval supplies cross-file evidence, while semantic and programmatic operations support diverse tasks. Across four benchmarks, the system outperforms strong baselines, including an 18.74-point gain on ContextSheetBench.

AI USE STATEMENT

AI models are integral to SHEET-RAPTOR, where LLMs and VLMs interpret spreadsheet structure and coordinate context retrieval with task execution. Seed-1.6 is the default backbone and also serves as the model of LLM-as-a-judge. We additionally evaluate Seed-1.6-flash, Qwen3-VL-Plus, GPT-5, GPT-5.2 as alternative backbones, as reported in Table 3. TableLLM and TableLLaVA serve as table-finetuned baselines in Section 4. Context retrieval uses a RoBERTa-based dual encoder whose training data include 500K synthetic task–context pairs, as described in Section 3.2.

We used ChatGPT to polish the manuscript's language and improve readability. This writing assistance did not introduce new experimental measurements. The authors remain responsible for the scientific claims and the final content of the paper.

ETHICS STATEMENT

This work studies spreadsheet analysis using public benchmarks and ContextSheetBench, which contains professionally annotated tasks over production-derived spreadsheets. For ContextSheetBench, the enterprise data are used with authorization, and sensitive information is processed to protect confidentiality. We carefully filter out any spreadsheets that may contain personal information, and ensure that all data is processed in full compliance with the opensource standards before the final release.

REPRODUCIBILITY STATEMENT

To make SHEET-RAPTOR reproducible, the source code, benchmark data, and reproduction instructions are provided at `https://anonymous.4open.science/r/Sheet-Raptor/README.md`. Section 3 describes the preprocessing algorithms and the interfaces connecting hierarchical retrieval to agentic execution, including the folding criteria and retrieval candidate counts. Section 4 specifies the benchmarks, baseline comparisons, and evaluation protocol, including the shared Seed-1.6 backbone for agent-based methods. Appendix A provides the ContextSheetBench capability profile, focused controls, and per-request efficiency measurements. The results can be reproduced by following the procedures described in the paper and the accompanying anonymous GitHub repository.

REFERENCES

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. Self-RAG: Learning to retrieve, generate, and critique through self-reflection. In *Proceedings of the International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=hSyW5go0v8`.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, and et.al. Qwen3-vl technical report, 2025. URL `https://arxiv.org/abs/2511.21631`.

Mangesh Bendre, Bofan Sun, Ding Zhang, Xinyan Zhou, Kevin Chen-Chuan Chang, and Aditya G. Parameswaran. DATASPREAD: unifying databases and spreadsheets. *Proc. VLDB Endow.*, 8 (12):2000–2003, 2015.

ByteDance Seed. Seed1.6. `https://seed.bytedance.com/en/seed1_6`, 2025. Accessed: 2026-01-16.

Si-An Chen, Lesly Miculicich, Julian Martin Eisenschlos, Zifeng Wang, Zilong Wang, Yanfei Chen, Yasuhisa Fujii, Hsuan-Tien Lin, Chen-Yu Lee, and Tomas Pfister. TableRAG: Million-token table understanding with language models. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, Red Hook, NY, USA, 2024. Curran Associates, Inc. URL `https://papers.nips.cc/paper_files/paper/2024/hash/88dd7aa6979e352fda7c4952ca8eac59-Abstract-Conference.html`.

Xinyun Chen, Petros Maniatis, Rishabh Singh, Charles Sutton, Hanjun Dai, Max Lin, and Denny Zhou. Spreadsheetcoder: Formula prediction from semi-structured context. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021. URL https://openreview.net/forum?id=S80ayYOfK6y.

Yibin Chen, Yifu Yuan, Zeyu Zhang, YAN ZHENG, Jinyi Liu, Fei Ni, Jianye HAO, Hangyu Mao, and Fuzheng Zhang. Sheetagent: Towards a generalist agent for spreadsheet reasoning and manipulation via large language models. In *THE WEB CONFERENCE 2025*, 2025. URL https://openreview.net/forum?id=AKy9ryZwi3.

Zhoujun Cheng, Tianbao Xie, Peng Shi, Chengzu Li, Rahul Kumar, Jill Fain Lehman, Chengxiang Zhai, Lu Chen, Kai-Wei Chang, Pengtao Xie, Jidong Xia, Lingpeng Kong, and Shaojun Wang. Binding language models in symbolic languages. In *Proceedings of the International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=9H0Z9K8N9h.

Haoyu Dong, Shijie Liu, Shi Han, Zhouyu Fu, and Dongmei Zhang. Tablesense: Spreadsheet table detection with convolutional neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 69–76, 2019. doi: 10.1609/aaai.v33i01.330169. URL https://doi.org/10.1609/aaai.v33i01.330169.

Haoyu Dong, Jianbo Zhao, Yuzhang Tian, Junyu Xiong, Shiyu Xia, Mengyu Zhou, Yun Lin, José Cambronero, Yeye He, Shi Han, and Dongmei Zhang. Spreadsheetllm: Encoding spreadsheets for large language models, 2025. URL https://arxiv.org/abs/2407.09025.

Wanrong He, Haoyu Dong, Yihuai Gao, Zhichao Fan, Xingzhuo Guo, Zhitao Hou, Haoyin Chen, Zhanyi Sun, and Dongmei Zhang. HermEs: Interactive spreadsheet formula prediction via hierarchical formulet expansion. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pp. 8375–8394. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.acl-long.465. URL https://aclanthology.org/2023.acl-long.465.

Soyeong Jeong, Jinheon Baek, Sukmin Cho, Sung Ju Hwang, and Jong Park. Adaptive-RAG: Learning to adapt retrieval-augmented large language models through question complexity. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 7036–7050, Mexico City, Mexico, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.naacl-long.389. URL https://aclanthology.org/2024.naacl-long.389/.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, volume 33, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html.

Hongxin Li, Jingran Su, Yuntao Chen, Qing Li, and Zhaoxiang Zhang. Sheetcopilot: Bringing software productivity to the next level through large language models. In *Advances in Neural Information Processing Systems*, 2023a. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/d8e3cc58032f9136e099611681b4904-Paper-Conference.pdf.

Keqian Li, Yeye He, and Kris Ganjam. Discovering enterprise concepts using spreadsheet tables. In *KDD*, pp. 1873–1882. ACM, 2017.

Peng Li, Yeye He, Cong Yan, Yue Wang, and Surajit Chaudhuri. Auto-tables: Synthesizing multi-step transformations to relationalize tables without using examples. *Proc. VLDB Endow.*, 16(11): 3391–3403, 2023b.

Qian Liu, Bei Chen, Jiaqi Guo, Morteza Ziyadi, Zeqi Lin, Weizhu Chen, and Jian-Guang Lou. TAPEX: Table pre-training via learning a neural SQL executor. In *Proceedings of the International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=O50443AsCP.

OpenAI. Gpt-5 introduction. https://openai.com/index/introducing-gpt-5/, 2025. Accessed: 2026-01-16.

Aditi Singh, Abul Ehtesham, Saket Kumar, and Tala Talaei Khoei. Agentic retrieval-augmented generation: A survey on agentic rag, 2025. URL https://arxiv.org/abs/2501.09136.

Zirui Tang, Boyu Niu, Xuanhe Zhou, Boxiu Li, Wei Zhou, Jiannan Wang, Guoliang Li, Xinyi Zhang, and Fan Wu. St-raptor: Llm-powered semi-structured table question answering. *Proc. ACM Manag. Data*, 2026a.

Zirui Tang, Xuanhe Zhou, Yumou Liu, Linchun Li, Yukai Wu, Weizheng Wang, Hongzhang Huang, Wei Zhou, Jun Zhou, Jiachen Song, Shaoli Yu, Jinqi Wang, Zihang Zhou, Hongyi Zhou, Yuting Lv, Jinyang Li, Jiashuo Liu, Ruoyu Chen, Chunwei Liu, GuoLiang Li, Jihua Kang, and Fan Wu. Workspace-bench 1.0: Benchmarking ai agents on workspace tasks with large-scale file dependencies, 2026b. URL https://arxiv.org/abs/2605.03596.

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. Chain-of-table: Evolving tables in the reasoning chain for table understanding. In *Proceedings of the International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=4L0xnS4GQM.

Ziwei Wang, Jiayuan Su, Mengyu Zhou, Huaxing Zeng, Mengni Jia, Xiao Lv, Haoyu Dong, Xiaojun Ma, Shi Han, and Dongmei Zhang. Sheetbrain: A neuro-symbolic agent for accurate reasoning over complex and large spreadsheets, 2025. URL https://arxiv.org/abs/2510.19247.

Jian Wu, Linyi Yang, Dongyuan Li, Yuliang Ji, Manabu Okumura, and Yue Zhang. MMQA: Evaluating LLMs with multi-table multi-hop complex questions. In *Proceedings of the International Conference on Learning Representations*, 2025a. URL https://openreview.net/forum?id=GGlpykXDCa.

Pengzuo Wu, Yuhang Yang, Guangcheng Zhu, Chao Ye, Hong Gu, et al. Realhitbench: A comprehensive realistic hierarchical table benchmark for evaluating llm-based table analysis, 2025b. URL https://arxiv.org/abs/2506.13405.

Xianjie Wu, Jian Yang, Linzheng Chai, Ge Zhang, Jiaheng Liu, Xinrun Du, Di Liang, Daixin Shu, Xianfu Cheng, Tianzhen Sun, Guanglin Niu, Tongliang Li, and Zhoujun Li. Tablebench: A comprehensive and complex benchmark for table question answering, 2025c. URL https://arxiv.org/abs/2408.09174.

Sishi Xiong, Ziyang He, Zhongjiang He, Yu Zhao, Changzai Pan, Jie Zhang, Zhenhe Wu, Shuangyong Song, and Yongxiang Li. Tablezoomer: A collaborative agent framework for large-scale table question answering, 2025. URL https://arxiv.org/abs/2509.01312.

Yunhu Ye, Binyuan Hui, Min Yang, Binhua Li, Fei Huang, and Yongbin Li. Large language models are versatile decomposers: Decomposing evidence and questions for table-based reasoning. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 174–184. Association for Computing Machinery, 2023. doi: 10.1145/3539618.3591708. URL https://doi.org/10.1145/3539618.3591708.

Xiaokang Zhang, Sijia Luo, Bohan Zhang, Zeyao Ma, Jing Zhang, Yang Li, Guanlin Li, Zijun Yao, Kangli Xu, Jinchang Zhou, Daniel Zhang-Li, Jifan Yu, Shu Zhao, Juanzi Li, and Jie Tang. TableLLM: Enabling tabular data manipulation by LLMs in real office usage scenarios. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 10315–10344, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.538. URL https://aclanthology.org/2025.findings-acl.538/.

Yunjia Zhang, Jordan Henkel, Avrilia Floratou, Joyce Cahoon, Shaleen Deep, and Jignesh M. Patel. Reactable: Enhancing react for table question answering. *Proc. VLDB Endow.*, 17(8):1981–1994, April 2024. ISSN 2150-8097. doi: 10.14778/3659437.3659452. URL https://doi.org/10.14778/3659437.3659452.

Mingyu Zheng, Xinwei Feng, Qingyi Si, Qiaoqiao She, Zheng Lin, Wenbin Jiang, and Weiping Wang. Multimodal table understanding. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9102–9124, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.493. URL https://aclanthology.org/2024.acl-long.493/.

Junnan Zhu, Jingyi Wang, Bohan Yu, Xiaoyu Wu, Junbo Li, Lei Wang, and Nan Xu. Tableeval: A real-world benchmark for complex, multilingual, and multi-structured table question answering, 2025. URL https://arxiv.org/abs/2506.03949.

A Additional Experimental Details

A.1 ContextSheetBench Profile

Table 4 reports the capability annotations and execution profile of ContextSheetBench. Capability counts overlap because a task may require, for example, both inter-file reasoning and program-oriented analysis. Operator usage is computed over tasks, whereas operation shares count individual invocations. The average of 2.59 operations per task indicates that a typical request needs a sequence of steps rather than one isolated lookup.

Table 4: ContextSheetBench composition and execution profile. Capability counts may overlap. Operator usage is the percentage of tasks whose trace invokes the operation.

Capability	Tasks
Structure understanding	34
Intra-file reasoning	102
Inter-file reasoning	139
Program-oriented analysis	122

Execution statistic	Value
`filter_by_semantics` tasks	45.6%
`decompose` tasks	46.7%
`extract_range` tasks	64.5%
Semantic-operation share	65.9%
Program-operation share	34.1%
Operations per task	2.59

The trace profile motivates the mixed execution design. Deterministic spreadsheet actions account for more than one third of invocations, but semantic operations are more frequent. Keeping the same coordinate-bearing table units available to both execution paths avoids losing the link between selected regions and exact cell values.

A.2 Focused Ablations

Isolating table folding. We first isolate the contribution of retrieved evidence. Supplying SheetRaptor's context raises a vanilla Seed-1.6 LLM from 12.23% to 30.43% on ContextSheetBench QA, and raises its VLM counterpart from 14.39% to 26.09%. The full system reaches 47.47%, which is 17.04 points above the stronger context-augmented baseline. This control shows that context is valuable but does not by itself provide the structural interpretation and execution required by the task.

To isolate folding, we keep the vanilla VLM and its surrounding evaluation fixed and change only whether repetitive regions are folded. Table 5 reports the resulting QA accuracy. Folding raises the total by 2.52 points and basic QA by 10.32 points. It also improves inter-file, structure, and program-oriented accuracy, while intra-file accuracy is unchanged. These results are consistent with removing visually repetitive rows while retaining representative layout cues. They do not imply that folding alone accounts for the full-system improvement, since the context controls and component ablations measure different interventions.

Table 5: Table-folding ablation on ContextSheetBench QA with a vanilla VLM (accuracy, %).

Setting	Total	Basic	Inter	Intra	Struct.	Program
VLM w/o folding	40.91	55.74	34.78	23.53	11.76	25.41
VLM w/ folding	**43.43**	**66.06**	**36.23**	23.53	**14.71**	**27.05**

A.3 EFFICIENCY

Table 6 compares systems using the same Seed-1.6 backbone. SHEET-RAPTOR takes about 70 seconds and 36K tokens per request on average, at a cost of $0.0077. It is faster than each spreadsheet-agent baseline, whose latency ranges from 92 to 154 seconds. The lower latency does not come at the expense of ContextSheetBench accuracy, where SHEET-RAPTOR reaches 51.66% and the strongest agent baseline reaches 32.92%.

Table 6: Average per-request latency and cost with Seed-1.6, alongside ContextSheetBench total accuracy.

Method	Latency (s)	Cost ($)	Accuracy (%)
TableZoomer	154	0.0052	10.83
SheetBrain	148	0.0136	32.92
SheetAgent	92	0.0101	13.68
SHEET-RAPTOR	70	0.0077	51.66

The cost comparison is less uniform than the latency comparison. SHEET-RAPTOR costs less than SheetBrain and SheetAgent, but TableZoomer is cheaper by $0.0025 per request while trailing by 40.83 accuracy points. Thus the measurements support a favorable accuracy–latency trade-off, not a claim that SHEET-RAPTOR minimizes every resource measure.

B AGENTIC RETRIEVAL AND SPREADSHEET OPERATIONS

B.1 PROGRESSIVE AGENTIC RETRIEVAL

The retrieval agents acquire external evidence before spreadsheet analysis. Their input is the table-conditioned question from Section 3.2, where relevant snippets supply entities and structural context missing from the original request. The procedure initializes an empty evidence list $\mathcal{K}$ and a state containing the question. Separate bounds, `MAX_ITER` and `MAX_PROUND`, limit retrieval rounds and plan-refinement rounds, respectively.

Planning and plan refinement. The Planning Agent proposes a search plan from the current state. Each step identifies information to seek from the auxiliary context rather than an operation to perform on the target spreadsheet. The Refine Agent checks the proposed plan and revises it before search. This inner loop terminates when the plan is accepted or the planning bound is reached, after which the current plan is passed to the Search Agent. The planning decision therefore depends on evidence already acquired, not only on the initial wording of the request.

Search and evidence refinement. The Search Agent executes each search step against the summary hierarchy. Embedding search supplies candidates and reranking selects the underlying evidence units, as described in Section 3.2. Results from all steps form a temporary evidence list. The Refine Agent then combines these results with $\mathcal{K}$ and assesses whether the accumulated evidence is sufficient. Plan acceptance and evidence sufficiency are separate decisions. A well-formed search plan can still return insufficient evidence, requiring another retrieval round.

State update and termination. If evidence remains insufficient, the next state contains both the original question and the updated $\mathcal{K}$. This allows a newly discovered entity to become the subject of a later search without losing the user's goal. For example, a session entry can identify a speaker whose details must then be found in another file. The procedure stops on evidence sufficiency or the retrieval-round bound and returns the accumulated list. The bound limits retrieval effort but does not certify that all required evidence has been found. Retrieved units retain their source identities, allowing downstream analysis to refer to the corresponding spreadsheet regions rather than summary text alone.

B.2 SPREADSHEET OPERATION CATEGORIES

External context retrieval and spreadsheet retrieval operations have different roles. The former searches auxiliary files to construct $\mathcal{C}$, whereas the latter selects data within the primary or retrieved tables during analysis. Table 7 organizes representative analysis operations into five functional categories. A category describes what an operation does, while its execution template determines whether it relies on semantic interpretation or generated code.

Table 7: Representative spreadsheet operations by functional category. Semantic operations use layout-aware model reasoning, while program-oriented operations execute generated code over spreadsheet data.

Category and operation	Template	Role
Retrieval		
`filter_by_semantics`	Semantic	Select regions satisfying a contextual predicate.
`extract_range`	Program	Read a region specified by cell coordinates.
Transformation		
`s_format`	Semantic	Normalize heterogeneous content or header structure.
`transpose`	Program	Reorient a specified table range.
Computation		
`s_join`	Semantic	Align related fields across tables using context.
`s_aggregate`	Semantic	Resolve an aggregation expressed in natural language.
`range_level_cal`	Program	Apply a function to an explicit cell range.
Manipulation		
`add_cells`	Program	Insert data or extend spreadsheet structure.
`cell_split`	Program	Decompose composite cells or table regions.
`cell_merge`	Program	Consolidate cells or table regions.
Orchestration and presentation		
`plan`	Semantic	Choose an execution path or the next operation.
`decompose`	Semantic	Construct a dependency-aware operation DAG.
Plot generation	Program	Render selected data through executable plotting code.

Retrieval. Spreadsheet data access must account for implicit schemas and spatial relationships. A semantic filter can interpret "high-risk customers" using header meaning and nearby cells, where a fixed predicate may not identify the intended records. Once a region has been localized, `extract_range` reads its exact coordinates, such as `A1:C10`. Program-oriented retrieval also supports explicit filtering and formula dependency tracing. For a cell containing `SUM(A1:A10)`, the referenced range identifies the source values needed by a later computation.

Transformation. These operations make irregular spreadsheet content usable without assuming that every sheet already has a relational schema. The semantic operation `s_format` interprets a normalization instruction, such as flattening a multi-level header into one level. In contrast, `transpose` swaps rows and columns in a specified region when headers run vertically. The first requires interpreting the intended representation, while the second performs a deterministic structural change.

Computation. Semantic computation resolves which fields and relationships a request refers to. The operation `s_join` aligns related columns when their headers differ, and `s_aggregate` interprets a request such as "total revenue by region". The program-oriented operation `range_level_cal` applies a specified function to a grounded range, such as summing `B4:C6`. Semantic alignment and exact arithmetic can therefore occupy different nodes in the same plan. Keeping the original cells available avoids substituting a folded visual sample for the complete numeric input.

Manipulation. These operations change spreadsheet content or structure. The operation `add_cells` inserts data or extends a region, while `cell_split` decomposes composite content

and `cell_merge` consolidates selected cells. They use program-oriented execution because the target coordinates and resulting structural changes must be explicit. The manipulation solver observes the updated spreadsheet before choosing another action, rather than assuming that all coordinates in the initial plan remain valid.

Orchestration and presentation. The operation `plan` selects between direct reasoning and a multi-step workflow. For a complex request, `decompose` constructs a DAG whose edges specify which intermediate results subsequent operations need. Visualization then uses plotting code over the selected data. Unlike the data-access operators, these operations organize execution or present its result. They connect a high-level request to a sequence of spreadsheet-grounded actions.

B.3 Operation Templates and State Propagation

Both templates receive an operation specification with spreadsheet metadata and a coordinate-enhanced preview. Merged-cell content is anchored in the top-left cell. The semantic template additionally supplies the folded image and operation examples, and the model response is parsed into the result. The program-oriented template substitutes spreadsheet-specific code examples for the image. Generated programs read the original data, not a numeric reconstruction of the visual preview.

The routing agent selects a solver, which either reasons directly or builds a dependency-aware plan. Each DAG node uses its designated template and passes results to dependent nodes. Semantic selection can thus identify relevant rows before code reads their exact cells and computes an aggregate. QA and visualization plan toward a fixed output, whereas manipulation replans after edits change spreadsheet state. Coordinates connect evidence acquisition to subsequent execution.

C ALGORITHM DETAILS

This appendix gives the procedures for structure-aware folding and progressive retrieval described in Section 3. It also summarizes task-adaptive execution, including the state propagation discussed in Appendix B.3. Routine names denote the corresponding agent roles or operations rather than additional interfaces.

C.1 STRUCTURE-AWARE FOLDING

Algorithm 1 scans rows from top to bottom and tests candidate block heights in ascending order. The predicate SIMILAR compares corresponding rows using Equation 1, including the merge-span and empty-cell checks. The loop requires both adjacent blocks to fit within the sheet. When a repeated region is found, the scan skips that entire region rather than detecting it again from an interior row.

Algorithm 1 Structure-aware spreadsheet folding

Require: Sheet $\mathcal{S}$ with N rows, maximum block height $W_{\max}$, repetition threshold $\rho_{\min}$
Ensure: Folded view $\mathcal{S}'$, with the full spreadsheet retained for execution

$\mathcal{F} \leftarrow \emptyset, i \leftarrow 1$
while $i \leq N$ **do**
  $found \leftarrow$ false
  for $w = 1, \ldots, \min(W_{\max}, N - i + 1)$ **do**
    $\rho \leftarrow 1, j \leftarrow i$
    while $j + 2w - 1 \leq N$ **and** SIMILAR($\mathcal{S}[j : j + w - 1], \mathcal{S}[j + w : j + 2w - 1]$) **do**
      $\rho \leftarrow \rho + 1, j \leftarrow j + w$
    end while
    if $\rho \geq \rho_{\min}$ **then**
      $\mathcal{F} \leftarrow \mathcal{F} \cup \{(i, w, \rho)\}$
      $i \leftarrow i + w\rho$
      $found \leftarrow$ true
      break
    end if
  end for
  if $\neg found$ **then**
    $i \leftarrow i + 1$
  end if
end while
$\mathcal{S}' \leftarrow$ HEADSKIPTAILVIEW($\mathcal{S}, \mathcal{F}$)
return $\mathcal{S}'$

For a region recorded as (i, w, ρ), the repeated span contains $w\rho$ rows. The view keeps its first two rows and final row, replacing only a removable middle with a folding marker. This procedure reduces visual redundancy without substituting representative values for omitted cells. Column-wise folding applies the same scan to the transposed table, and program-oriented operations continue to read the complete data.

C.2 PROGRESSIVE AGENTIC RETRIEVAL

Algorithm 2 formalizes the two nested loops in Appendix B.1. The input q is the table-conditioned question. The hierarchy $\mathcal{I}$ contains the summaries and their associated evidence units. The positive budgets R and B bound retrieval rounds and planning attempts, respectively. Equivalently, an implementation with zero-based inclusive guards uses $R =$`MAX_ITER`$+1$ and $B =$`MAX_PROUND`$+1$.

Algorithm 2 Progressive table-context retrieval

Require: Question q, hierarchy $\mathcal{I}$, retrieval budget R, planning budget B
Ensure: Accumulated evidence list $\mathcal{K}$

$\mathcal{K} \leftarrow [\,]$, $state \leftarrow (q, \mathcal{K})$
for $r = 1, \ldots, R$ **do**
  for $b = 1, \ldots, B$ **do**
    $\mathcal{P} \leftarrow$ PLANNINGAGENT$(state)$
    $(\mathcal{P}, accepted) \leftarrow$ REFINEPLAN$(\mathcal{P})$
    if $accepted$ **then**
      break
    end if
  end for
  $\mathcal{K}_{new} \leftarrow [\,]$
  for search step s in $\mathcal{P}$ **do**
    $hits \leftarrow$ SEARCHAGENT$(s, \mathcal{I})$
    $\mathcal{K}_{new} \leftarrow$ EXTEND$(\mathcal{K}_{new}, hits)$
  end for
  $(\mathcal{K}, sufficient) \leftarrow$ REFINEEVIDENCE$(\mathcal{K}, \mathcal{K}_{new})$
  if $sufficient$ **then**
    break
  end if
  $state \leftarrow (q, \mathcal{K})$
end for
return $\mathcal{K}$

REFINEPLAN and REFINEEVIDENCE are two uses of the Refine Agent. The first checks the search plan, while the second combines evidence and assesses sufficiency. If the planning budget is exhausted, the current plan is executed. Exhausting the retrieval budget returns the evidence acquired so far, not a guarantee of completeness. Each search uses the embedding and reranking stages from Section 3.2 and returns evidence with its file, sheet, and range identity.

C.3 TASK-ADAPTIVE EXECUTION AND STATE PROPAGATION

Algorithm 3 is a high-level procedural summary of the routing, planning, and execution workflow in Section 3.3. The routing agent emits pairs (a, u), where a identifies a solver and u specifies its subtask. State X holds the current spreadsheet data and retrieved context, while $\mathcal{M}$ caches a solver's intermediate results. QA and visualization use direct reasoning or a dependency-aware plan. Manipulation instead proposes one scoped edit and replans from the observed spreadsheet state.

Algorithm 3 Task-adaptive analysis with explicit spreadsheet state

Require: Request q, primary tables T_p, retrieved context $\mathcal{C}$
Ensure: Task outputs $\mathcal{O}$

$X \leftarrow (T_p, \mathcal{C}), \mathcal{O} \leftarrow [\,]$
$routes \leftarrow \textsc{RoutingAgent}(q, X)$
for solver–subtask pair (a, u) in $routes$ **do**
  $\mathcal{M} \leftarrow \emptyset$, $P \leftarrow \textsc{Plan}(a, u, X)$
  if a is a manipulation solver **then**
    while P specifies a further edit **do**
      $(y, X) \leftarrow \textsc{ExecuteOperation}(P, X, \mathcal{M})$
      $\mathcal{M} \leftarrow \textsc{CacheResult}(\mathcal{M}, P, y)$
      $X \leftarrow \textsc{ObserveSpreadsheet}(X)$
      $P \leftarrow \textsc{Plan}(a, u, X)$
    end while
    $\mathcal{O} \leftarrow \textsc{Extend}(\mathcal{O}, \text{edited primary spreadsheet in } X)$
  else
    if P selects direct reasoning **then**
      $(y, X) \leftarrow \textsc{ExecuteOperation}(P, X, \mathcal{M})$
    else
      $G \leftarrow \textsc{Decompose}(P)$
      for node v in $\textsc{TopologicalOrder}(G)$ **do**
        $o \leftarrow \textsc{BindInputs}(v, \mathcal{M}[\text{pred}(v)])$
        $(y, X) \leftarrow \textsc{ExecuteOperation}(o, X, \mathcal{M})$
        $\mathcal{M}[v] \leftarrow y$
      end for
      $y \leftarrow \textsc{TerminalResults}(G, \mathcal{M})$
    end if
    $\mathcal{O} \leftarrow \textsc{Extend}(\mathcal{O}, y)$
  end if
end for
return $\mathcal{O}$

EXECUTEOPERATION selects the template attached to the operation. Semantic execution receives the coordinate-enhanced representation and folded image. Program-oriented execution generates and runs code over the complete current data. The routine returns both its result and the spreadsheet state, which remains unchanged for read-only operations. DAG successors receive cached results from their predecessors, including selected coordinates, instead of relying on an implicit conversation summary.