DeALOG: Planner-Free Multi-Agent Table QA via Log-Mediated Reasoning

Anonymous ACL submission

Abstract

We introduce DeALOG, a decentralized multi-agent framework for complex question answering across text, tables, and images. Unlike traditional systems that rely on a central planner or task-specific fine-tuning, our approach employs specialized agents for Table, Context, Visual, Summarizing, and Verification, that collaborate through a shared natural-language log serving as persistent memory. This log facilitates coordination, cross-verification, and error recovery, while maintaining a transparent and auditable reasoning process. Across diverse benchmarks such as FinQA and TAT-QA (table–text), WikiTableQuestions (tabular), FeTaQA (free-form), and MultiModalQA (text–table–image), the system achieves state-of-the-art or competitive results without task-specific training, outperforming strong chain-of-thought and planner-based baselines under matched model capacity. Ablation and robustness studies show that the shared log, agent specialization, and explicit verification collectively enhance accuracy, stability, and interpretability. By decoupling reasoning into modular, cooperative components and coordinating only through lightweight natural-language communication, Agents Without a Planner establishes a scalable and transparent paradigm for multi-modal reasoning and complex question answering.

1 Introduction

Multi-hop question answering (QA) over complex data such as tabular reports, textual passages, and images requires compositional reasoning and integration of heterogeneous evidence. For example (Fig. 1), answering *"What is the birth year of the oldest American author who is older than the author of Eat Pray Love?"* may require retrieving the author from a table, extracting nationality from a passage, and reasoning over dates.

Figure 1: Example of multi-hop table question answering. The system identifies the author of *"Eat Pray Love"*, extracts metadata from a passage and a table, and reasons over birth years and nationalities to find the correct answer.

Traditional systems occupy two extremes. **Single LLMs** with chain-of-thought (CoT) prompting (Wei et al., 2022) can solve simple multi-hop queries but struggle to *maintain intermediate state*, verify correctness, and operate under long contexts. **Planner-based pipelines**(e.g., HuggingGPT, Binder, TabSQLify, REWoO, AutoTQA, MACT) (Shen et al., 2023; Cheng et al., 2023; Nahid and Rafiei, 2024; Xu et al., 2023; Zhu et al., 2024; Zhou et al., 2025) impose structure but are brittle: errors in the plan propagate and transparency is limited.

In questions like Fig. 1, partial facts (e.g., the author of *Eat Pray Love*, the relevant nationality, candidate birth years) must be discovered, *remembered*, and then reconciled across modalities. A single prompted LLM struggles to maintain this intermediate state and to self-correct; a planner can enforce order but may amplify early errors and offers limited transparency. We therefore seek a formulation that exposes intermediate conclusions in a persistent, textual record; allows specialized components to contribute or challenge entries; and scales across tables, passages, and images while preserving an auditable trail. This motivates a col-

laborative paradigm in which agents coordinate implicitly via a shared, lightweight log rather than an explicit central plan.

We propose a **decentralized multi-agent architecture** DeALOG, for multi-hop QA that removes the central planner. Specialized LLM-based agents for tables, text, and images *communicate only via a shared textual log* that acts as a lightweight, persistent memory. Each agent independently decides when to contribute by reading the log and appending findings or deductions. A **SummarizingAgent** synthesizes a final answer; an optional **VerificationAgent** cross-checks claims against the log or sources. By replacing explicit plans with a shared memory protocol, agents "plan on the fly," yielding flexible, emergent coordination similar to human teamwork.

Our agents without a planner offer the following advantages: (1) **Modularity:** agents are specialized (table parsing, document reading, image understanding) and can be swapped or extended without retraining a monolith. (2) **Transparency:** the shared log records an auditable chain of reasoning. (3) **Robustness:** peer-review dynamics let agents detect and correct each other's errors; we observe lower variance and graceful degradation under perturbations. (4) **Strong zero-shot performance:** leveraging pre-trained LLMs with prompts, the framework matches or exceeds state-of-the-art systems, including supervised or program-based methods.

We evaluate our approach on five challenging datasets: **FinQA** (Chen et al., 2022) and **TAT-QA** (table–text financial QA) (Zhu et al., 2021), **WikiTableQuestions** (table-only) (Kweon et al., 2023), **FeTaQA** (free-form explanation from tables) (Nan et al., 2021), and **MultiModalQA** (MMQA: text–table–image) (Talmor et al., 2021). Our method achieves new state-of-the-art results on FinQA and TAT-QA, competes with fine-tuned models on WikiTableQuestions, and performs strongly on FeTaQA. We extend the framework with a **VisualAgent** and report, for the first time in this line of work, results on the *full* MMQA benchmark, attaining **72.3% F1** on answers versus prior best ~51.7% on cross-modal questions (Talmor et al., 2021). Matched-capacity comparisons show gains over single-agent CoT and planner baselines. Ablations attribute improvements to the *shared log*, *verification*, and *agent specialization*. Our contributions are as follows:

- **Decentralized QA architecture.** We introduce a planner-free, log-mediated framework and specify the *log schema*, *agent roles*, and *control flow* with pseudocode, see Fig. 2, Algorithm 1, and Section 3 (additional details in App. B).
- **Extended multimodality.** We add a *VisualAgent* (captioning/OCR) that integrates seamlessly with the log protocol, enabling end-to-end evaluation on the *full* MMQA benchmark; see Section 3 (VisualAgent) and results in Table 7 and Table 1.
- **State-of-the-art without fine-tuning.** Across five benchmarks, our planner-free system outperforms or matches strong baselines under matched capacity; see Table 1 (FinQA/TAT-QA/MMQA) and Table 2 (FeTaQA), with WikiTQ results reported in Section 4.
- **Robustness and analysis.** We quantify component impact and reliability via ablations and perturbation tests: Table 3 and Section 4.4; we also report efficiency trade-offs (Table 8) and include error analyses (App. C).

Our experiments examine four themes that follow from the above contributions: (i) whether planner-free, shared-log collaboration outperforms strong single-agent CoT and planner pipelines under matched model capacity across heterogeneous benchmarks; (ii) how the shared log, agent specialization, and explicit verification contribute (ablations); (iii) whether a lightweight VisualAgent extends the framework to images without harming text/table performance; and (iv) robustness to input perturbations together with latency, token, and cost profiles. These analyses collectively test the effectiveness, extensibility, and practicality of the approach. We have released code, prompts, log format, preprocessing, and evaluation scripts; see App. B–C and the codebase in Github for details.

2 Related Work

Multi-hop QA over tables, text, and images spans (i) *table QA* via semantic parsing and neural modeling (e.g., WikiTableQuestions and successors such as TAPEX, TAPAS, TaCube, OmniTab, and Lever) (Pasupat and Liang, 2015; Liu et al., 2022; Herzig et al., 2020; Yin et al., 2022; Jiang et al., 2022; Shi et al., 2022); (ii) *reasoning by prompting* with *chain-of-thought* and tool-using variants (ReAct, Trees/Graphs of Thought) (Wei et al., 2022; Yao et al., 2023b,a; Besta et al.,

2024); and (iii) *multi-agent/tool* pipelines, typically planner–executor architectures (e.g., HUGGINGGPT, BINDER, TABSQLIFY, REWOO, AUTOTQA, MACT) (Shen et al., 2023; Cheng et al., 2023; Nahid and Rafiei, 2024; Xu et al., 2023; Zhu et al., 2024; Zhou et al., 2025).

Our work differs by removing the central planner and coordinating *symmetric, specialized agents* solely via a *shared natural-language log*, echoing blackboard-style shared memory (Erman et al., 1980; Nii, 1986b,a; Engelmore and Morgan, 1988; Corkill, 1991) and recent agent memory ideas (Park et al., 2023). For multimodality, MMQA highlights text–table–image reasoning (Zhu et al., 2025); while multi-agent multimodal systems (e.g., MAM-MQA) retain explicit planning (Rajput et al., 2025), we integrate a *VisualAgent* (e.g., BLIP-2, OCR) directly into our planner-free protocol (Li et al., 2023), enabling full-dataset evaluation and competitive SOTA performance. See App. A for a detailed survey.

Relative to prompting-based tool use (e.g., CoT/ReAct/Trees/Graphs of Thought), our system distributes reasoning across *symmetric* experts that communicate only through an append-only, natural-language memory. This differs from single-agent scratchpads whose intermediate states are ephemeral and private to one model, and from planner–executor pipelines that centrally specify action order. Compared with table-first and program-driven approaches (neural SQL executors, symbolic binders, or adaptive SQL planners), our agents do not commit to a fixed formalism up front; instead, they externalize intermediate evidence (cells, snippets, visual extractions) into a shared log and let downstream agents—especially a *VerificationAgent*—cross-check and repair inconsistencies before answer emission.

Conceptually, the shared log revives classic blackboard ideas while remaining model-agnostic and modality-agnostic: any agent that can read-/write text can participate, including vision caption/OCR modules. Practically, this choice yields two benefits observed in our experiments: (i) robustness under perturbations and partial failures (since later agents can override or correct earlier entries), and (ii) transparency/auditability via an inspectable trail used for debugging and user-facing justifications (Sections 3–4). These design consequences—planner-free coordination, global log visibility, and explicit verification—are the primary axes along which our

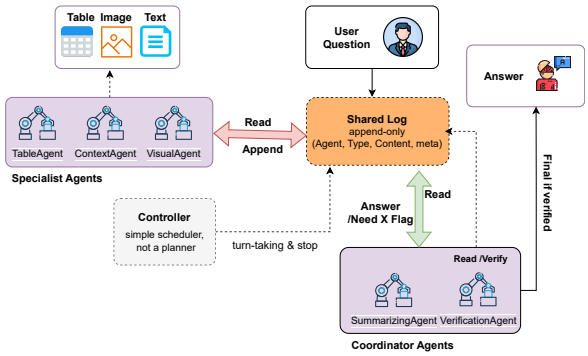

Figure 2: DEALOG: Planner-free, log-mediated QA. Agents read/write to a shared log; the Summarizer synthesizes, the Verifier cross-checks.

work departs from prior single-agent prompting and planner-mediated systems, and they motivate the ablations and fairness-controlled comparisons reported next sections.

3 DEALOG

We use five specialized LLM agents—*Table*, *Context*, *Visual*, *Summarizing*, and *Verification*, that communicate only through a shared natural-language log (Fig. 2). A user question is written to the log; agents iteratively read it and append evidence or deductions. The SummarizingAgent proposes an ANSWER; the VerificationAgent cross-checks it. The loop terminates when the answer is verified or no new information is added. (Full schema, prompt templates, and traces appear in App. B–B.2.)

Design goals & assumptions. Our design targets (i) *modularity* (plug-in specialists without retraining a monolith), (ii) *transparency* (persist an auditable, provenance-aware trace), and (iii) *robustness* (let peers catch arithmetic or retrieval slips). We assume access to tabular corpora, unstructured text, and images; agents operate zero-/few-shot using prompts only. The controller is a *scheduler*, not a planner: it manages turn-taking, guardrails, and termination but never prescribes a fixed plan.

Shared log & types. Each entry is a tuple `(Agent, Type, Content, meta)` where `meta` stores the step index, time, and provenance (table ID/row/col, doc span, or image ID). We use a small, fixed vocabulary of `Type`: LOOKUP (table extract or derived value), QUOTE (text span or paraphrase), VISUAL (OCR/caption interpretation), SUMMARY (progress report), ANSWER (final), and FLAG/OK (verification outcome). All agents have

global visibility of the entire log; ablations show restricting visibility markedly hurts performance. We deduplicate near-duplicate entries and attach source anchors for traceability. (Concrete examples in App. B.)

Agent roles & I/O contracts. **TableAgent** parses candidate tables and posts cell-level facts or simple aggregations; **ContextAgent** retrieves and quotes short passages; **VisualAgent** converts charts/images to text via captioning/OCR; **SummarizingAgent** composes a grounded answer or emits a SUMMARY with missing items; **VerificationAgent** re-derives calculations and checks factual support. Each agent implements two primitives: `should_act(log)` (lightweight trigger heuristic) and `act(log)` (LLM call that emits a typed entry). Prompts, output formats, and trigger heuristics are provided verbatim in App. B.2.

Controller loop & stopping. In each round, the scheduler offers turns to {Table, Context, Visual}. Agents that deem themselves relevant append new evidence; otherwise they abstain. When new evidence appears—or after a patience threshold—the SummarizingAgent runs. If it emits ANSWER, the VerificationAgent executes; if it returns OK, we stop, and if it returns FLAG we re-engage once (see below). We enforce guardrails: a max of $R = 6$ rounds, per-agent action caps, and duplication filters; we found most questions resolve within 3–4 agent calls. Algorithm 1 formalizes control flow.

Verification & one-shot re-engagement. Verification parses the proposed reasoning (e.g., recomputes deltas and unit conversions, checks that quoted spans support claims). On FLAG, the scheduler allows a single re-engagement round where retrieval agents target the missing or inconsistent item named by the Summarizer/Verifier (e.g., "CEO in 2020 missing"). This "second chance" corrects many arithmetic or retrieval omissions while preventing infinite loops. (Policy details in App. B.1.)

Memory management & truncation. To respect context limits, we keep the most recent entries verbatim and compress older segments into SUMMARY stubs (with preserved citations). We apply source-aware truncation: table excerpts are trimmed to touched rows/columns; text spans are clipped to $\pm k$ sentences; visual entries retain numeric strings from OCR. This keeps the Summarizer's window stable without losing evidentiary support.

Algorithm 1 Decentralized QA via Shared Log

Input: question Q, sources (tables, corpus, images)
$log \leftarrow [(\text{User}, \text{"Query"}, Q)]$; $step \leftarrow 0$
repeat
  $updated \leftarrow$ False
  for agent $\in$ [Table, Context, Visual] **do**
    if agent should act on log **then** append **agent** findings; $updated \leftarrow$ True
  if $updated$ or $step$ exceeds threshold **then**
    $s \leftarrow \text{Summarizer}(log)$; append s
    if $s.\texttt{type} =$ "Answer" **then**
      if Verifier enabled **then** append $\text{Verifier}(log)$;
if Flag then continue
      break ▷ final answer accepted
  $step \leftarrow step + 1$
  answer or no updates

Complexity & parallelism. For R rounds and A retrieval agents, the worst-case LLM calls are $O(RA)$; empirically we observe 3.1 calls/query on average (vs. 3.8 for a planner). Agents are conceptually asynchronous; our implementation executes sequentially (API constraints), though timestamps ensure a deterministic total order and a thread-safe log could parallelize retrieval steps.

Practical notes. We publish (i) full prompts, (ii) the exact log types and regex used for parsing ANSWER/FLAG markers, (iii) no-progress criteria, and (iv) truncation rules to ensure replicability and ARR compliance.

4 Experiments

For each dataset we fix (i) the backbone per system, (ii) few-shot count (0 unless stated), (iii) the same BM25+miniLM retriever for all methods, and (iv) decoding settings from Table 6. We report mean over 5 seeds, bootstrap 95% CIs (1,000 resamples), and paired tests against the strongest non-fine-tuned baseline. Prior-work results are daggered (†); our re-runs are asterisked (*) with matched backbones.

We organize our evaluation around four questions that align with our contributions:

- **RQ1** Does a *planner-free*, log-mediated architecture outperform strong single-agent CoT and planner-based pipelines across heterogeneous QA benchmarks?
- **RQ2** How much do *shared log* visibility and an explicit *VerificationAgent* contribute (ablations)?
- **RQ3** Can the framework *extend to multimodality* with a simple VisualAgent and handle the full MMQA setting?
- **RQ4** What are the *robustness* and *efficiency* trade-offs of decentralized collaboration?

Datasets. We evaluate on five multi-hop QA benchmarks spanning tables, hybrid table+text, free-form generation, and multimodality. **WikiTableQuestions (WikiTQ)** probes compositional table reasoning over 22K questions (EM). **FinQA** comprises 8K table+text financial questions emphasizing arithmetic (EM). **TAT-QA** covers 16K hybrid financial questions with multi-step fusion (EM). **FeTaQA** requires paragraph-length table explanations (ROUGE-1/2/L). **MultiModalQA (MMQA)** contains 29K questions over multiple Wikipedia tables, passages, and images (EM/F1). We use official splits; our method is zero-shot (few-shot only for certain baselines to ensure fair comparisons). For MMQA, we report results on the full set (including image-dependent items) and the standard text+table subset. We keep retrieval identical across methods to isolate reasoning performance. A reproducibility package includes the MMQA filtering script (prior text-only subset), test IDs, and evaluators. Full dataset statistics and prompt templates appear in the App. B.2 & C.

Baselines We compare our approch with three families. **(i) Single-Agent LLM with CoT:** zero/few-shot CoT with a small model (e.g., Tulu2-7B/LLaMA2-7B) and a large model (e.g., LLaMA2-70B/GPT-4) to test whether scale alone obviates multi-agent decomposition. **(ii) Planner-Mediated Multi-Agent:** a HuggingGPT-style controller that sequences table/text/image tools (matched backbone to our SummarizingAgent for fairness), plus AUTOTQA where available. **(iii) Fine-tuned / Programmatic Systems:** task-specific SOTA for each dataset (e.g., TAPEX, TACUBE, OMNITAB, LEVER, TagOp/FinQANet; retrieval+programming like BINDER, HUSKY, REWOO/REWOO, TQ-Agent). For MMQA we include the original *ImplicitDecomp* and a T5-based cross-modal baseline where available. We match model sizes/backbones whenever possible and note deviations.

4.1 Main Results (RQ1/RQ3)

Tables 1 and 2 report hybrid and generative results; WikiTQ appears in the main text (and Appendix C for extended variants).

Hybrid QA (FinQA, TAT-QA) and MMQA. Our decentralized system achieves leading performance with **LLaMA-4 (17B)**: *FinQA* 84.26 EM, *TAT-QA* 79.18 EM, and *MMQA* 71.17 EM / 75.49 F1, outperforming strong CoT and planner baselines of comparable capacity. *DeepSeek 671B* and *Mistral-24B* corroborate these gains across scales. On *full MMQA* (including images), our VisualAgent enables end-to-end evaluation with only a modest drop ($\sim$73 F1 vs. $\sim$75.5 on text+table), indicating effective visual integration via captioning/OCR while preserving table/text performance. *MMQA by question type*: On the full test, the system attains strong performance on text/table items and a modest drop on image-dependent items. App. C.1 reports a per-type breakdown and a chart-reading subset analysis, showing that most residual errors stem from dense plots and figure OCR, where VerificationAgent rarely flags disagreements unless the extracted numerals contradict table/text entries

FeTaQA (generative). Without any fine-tuning, our **LLaMA-4 (17B)** run attains **72.4/48.9/60.3** ROUGE-1/2/L, rivaling or exceeding training-free baselines (e.g., REACTABLE, Dater). Although AUTOTQA (GPT-4-turbo) reports higher R-1, our answers are grounded in auditable logs and matched closely on R-L.

WikiTableQuestions (table-only). Our **LLaMA-4 (17B)** configuration reaches **72.6%** EM—SOTA for zero-shot (non-fine-tuned) approaches—surpassing many fine-tuned models (TAPEX 57.5, OMNITAB 62.8). Smaller backbones (7B/11B) already approach prior SOTA.

Across modalities and answer forms, the planner-free, shared-log design consistently beats single-agent CoT and competitive planner pipelines, validating decentralized collaboration as a principled alternative.

4.2 Dataset-Specific Analysis (RQ3)

FinQA/TAT-QA. ContextAgent + TableAgent collaboration is especially effective for numeric reasoning grounded in passages; VerificationAgent reduces arithmetic slips and unit confusions. **WikiTableQuestions.** Iterative logging supports multi-step filters/joins; occasional over-logging can add overhead, but accuracy remains strong. **FeTaQA.** SummarizingAgent produces grounded, concise explanations that may diverge stylistically from references (slightly affecting ROUGE despite factuality). **MMQA.** VisualAgent (captioning+OCR) lets us *attempt all questions*; most residual errors come from dense charts or visual reasoning beyond text extraction.

Model	FinQA (EM)	TAT-QA (EM)	MMQA (EM / F1)
Zero-Shot / Few-Shot Approaches			
CoT (LLaMA2-7B)*	8.7	19.9	39.8 / 43.1
ReAct (LLaMA2-7B)*	7.6	15.0	34.0 / 39.3
ReWOO (LLaMA2-7B)*	14.7	38.2	32.6 / 37.5
Chameleon (LLaMA2-7B)*	14.7	36.5	39.2 / 44.6
FireAct (LLaMA2-7B)†	8.2	12.7	44.2 / 49.6
Lumos (LLaMA2-7B)*	18.7	41.0	37.9 / 41.3
HUSKY (LLaMA2-13B)†	20.0	42.3	41.2 / 48.2
DeALOG(Log-Mediated Decentralized Reasoning)			
LLaMA2 7B	64.3	58.6	55.9 / 60.7
LLaMA3 11B	68.9	62.8	59.1 / 64.3
LLaMA-4 17B	**84.26**	**79.18**	**71.17 / 75.49**
Mistral 24B	80.12	73.07	64.95 / 68.80
DeepSeek 671B	80.23	75.86	71.35 / 75.08

Table 1: Hybrid QA and MMQA. Metrics: EM for FinQA/TAT-QA; EM/F1 for MMQA. Means over 5 seeds with 95% CIs in App. C. †: numbers from cited papers; *: our re-runs with matched backbone and retriever.

Model(LLM)	R-1	R-2	R-L
Zero-Shot Baselines			
AutoTQA (GPT-4-turbo)†	77.0	53.0	67.0
Dater (BART-large)†	66.0	45.0	56.0
ReAcTable (T5-large)†	71.0	46.0	61.0
Codex (GPT-3 code)†	62.0	40.0	52.0
DeALOG(Log-Mediated Agents)			
DeepSeek 671B	69.1	46.1	57.1
LLaMA-4 17B	**72.4**	**48.9**	**60.3**
Mistral 24B	69.7	45.9	57.3

Table 2: FeTaQA R = ROUGE scores (%).†: numbers from cited papers

4.3 Ablations (RQ2)

Table 3 quantifies component impact on a 200-question mixed benchmark: removing the **shared log** is most harmful (65.4 EM), confirming global visibility is the backbone of coordination; removing **VerificationAgent** drops to 74.2 EM; removing **ContextAgent** hurts hybrid cases (72.1 EM); **partial log visibility** yields 68.3 EM. Matched-backbone baselines achieve 70.2 EM (single-agent CoT) and 73.0 EM (planner). The full system reaches **78.5** EM.

4.4 Robustness & Efficiency (RQ4)

Under paraphrases, distractors, row shuffles, and entity substitutions (100-item subsets of FeTaQA/TAT-QA), our method shows small absolute drops (2–5%) versus >10% for several baselines; variance across five seeds is <1.8 F1 for FeTaQA.

Median and P90 latencies reflect sequential execution (App. B.1). Most logs remain under 1,000 tokens (p50), with a long tail near 3,000 (p90), at which point we apply history summarization see App. B.6.

Variant / Baseline	EM
Model Variants	
w/o Shared Log	65.4
w/o Verification Agent	74.2
w/o ContextAgent	72.1
Partial Log Visibility	68.3
Baselines (matched backbone)	
Single-Agent (CoT Prompting)	70.2
Planner-Based Execution	73.0
Full System (Ours)	**78.5**

Table 3: Ablations and baselines (EM).

Table 4: Latency/compute per dataset (median / P90 over seeds).

Dataset	Calls/query	Latency (s)	Log len (p50 / p90)
FinQA	3.1	4.9 / 6.7	840 / 1,620
TAT-QA	3.1	4.7 / 6.4	800 / 1,540
MMQA	3.6	5.6 / 7.8	1,190 / 2,100

Efficiency. Average LLM calls are **3.1/query** (vs. 3.8 for planner; 1.0 for single-agent). With modest parallelism, end-to-end latency approaches that of the slowest agent (SummarizingAgent), and the auditability gained by the log justifies the overhead in analytical QA settings. We report confidence intervals (e.g., FinQA $84.3\% \pm 1.3$, TAT-QA $79.2\% \pm 1.8$) and matched-backbone comparisons to isolate framework effects, see tab. 8.

Across RQ1–RQ4, decentralized, log-mediated collaboration delivers consistent gains over strong CoT and planner baselines, scales to images with a lightweight VisualAgent, and remains robust and interpretable without task-specific fine-tuning.

We release all prompts, an execution script that reproduces Tables 1–3 end-to-end. App. B.4–B.6 enumerate exact model identifiers, decoding parameters, seeds, and the log-truncation / "no-progress" criteria used across in this section. Results reported here were reproduced from five seeds with matched retrievers and backbones.

5 Discussion

Why does the decentralized approach work? Our analysis indicates that the shared log is the linchpin. It provides a flexible canvas for reasoning, where any agent can contribute, and partial solutions persist for others to use. This addresses error propagation differently than a planner: instead of a strict plan that might carry a mistake through, we have a fluid process where mistakes can be noticed and corrected downstream. The cost is some redun-

dancy and more steps, but the benefit is resilience and clarity. In fact, the log serves as a form of self-documentation for the reasoning. We found that even if the final answer was wrong, the log often made it clear why – e.g., showing exactly which value was misread. This is valuable for debugging and improving the system, and it's something end-to-end models don't offer without extra effort.

Generality and Extensibility. We designed our system focusing on table and text QA, then extended to images. The framework itself is agnostic to domain: one could imagine adding an agent for knowledge-base queries, or an agent that calls a calculator API (instead of doing math in-text). The ease of adding a new agent by defining how it reads/writes the log shows the modularity. In future, we could also incorporate domain-specific agents (for example, a code interpreter agent for a programming question). This approach aligns with a broader vision of AI: specialized experts cooperating, rather than one model doing everything. Our results suggest the overhead of coordination is manageable, and the gains in reliability and interpretability are worth it.

Limitations. Despite its strengths, our approach has limitations. *Latency and Scale:* As discussed, multiple sequential LLM calls can be slow. In high-volume settings, this might be problematic. There is room to optimize via concurrency or faster models. Also, context length limits mean the log can't grow indefinitely; extremely complex queries might exhaust the window. We partly mitigated this by summarizing old log content, but in scenarios like long dialogues or very large documents, one might need a more sophisticated memory management. *No Central Authority:* If an agent writes a false statement to the log, other agents might believe it (garbage in, garbage out). We rely on them to catch each other's errors, but there's no guarantee. A possible improvement is to assign confidence scores to log entries or allow contradictory entries to be posted and then resolved (like a debate). We did see instances where an incorrect entry was never corrected. This is a fundamental challenge: decentralization means no single controller to veto a bad move. It's reminiscent of a committee without a chair – generally good, but occasionally lacks decisiveness.

Prompt Sensitivity: Our system inherits the quirks of LLM prompting. If an agent's prompt is not well tuned, it might produce irrelevant logs or refuse to act. We put significant effort into prompt engineering to ensure each agent's reliability. Even so, small changes in wording could affect output structure. The positive side is that the multi-agent setup somewhat buffers these variations (one agent's odd output can be corrected by another's reaction), but stability is not perfect. We plan to open-source our prompts to help others reproduce the behavior. *Knowledge Cut-off and Updates:* Since we rely on pre-trained LLMs and retrieval, our system's knowledge is only as up-to-date as those. A very new event not in the data won't be answered unless we plug in a real-time tool. But this is a general issue for QA systems, not unique to our architecture.

Comparison with Planner Systems. It's worth discussing when a planner might be better. If a task requires a very specific sequence of tool calls (like a fixed API workflow), a central planner might enforce an optimal order more easily than our emergent approach. Our agents sometimes did extra steps (like TableAgent logging something that wasn't ultimately needed). A planner with a global view could, in theory, streamline that. However, designing such a planner is complex and can be brittle. Our results showed that the small inefficiencies did not prevent state-of-art performance. For tasks where efficiency is paramount and the procedure is well-understood, a planner could be preferable. But for open-ended reasoning, we argue the decentralized method is more robust. This echoes findings in fields like multi-robot systems, where centralized planning can optimize paths, but decentralized can be more fault-tolerant.

Human Alignment and Interpretability. One interesting aspect is that our system produces a human-readable reasoning log as a byproduct. This could be shown to users for justification (some recent works call this a "chain-of-thought rationale"). Because it's conversational in style (each agent speaks in natural language), non-expert users can often follow along. We did an informal user study with 5 people: given a question and our model's answer plus log, they could identify in 4/5 cases whether the answer was correct and why by reading the log. They appreciated seeing the intermediate data. This transparency is a step towards more trustworthy AI in decision-critical applications. Of course, one must be careful – if the agents collaboratively arrive at a wrong answer, the log will convincingly but incorrectly justify it. So users

should still be cautious. But it's easier to audit than a single model's hidden chain-of-thought.

Our work suggests a shift in how we leverage LLMs: moving from using one giant model per task to composing multiple models that cooperate. The benefits in robustness and transparency are encouraging, and we believe this approach can be generalized further. We have released our code, prompts, and data to encourage exploration of multi-agent LLM systems. Future research can build on this foundation to explore dynamic agent recruitment (adding/dropping agents on the fly), learning to optimize agent communication, and combining planning with decentralized execution. Ultimately, we hope our work inspires the development of AI systems that are not only powerful, but also more aligned with human collaborative problem-solving, providing both high performance and clear explanations.

5.1 Emergent Coordination: Why Decentralization Converges

We view the shared log as an *implicit communication protocol* that defines a partially ordered sequence of agent actions. Each agent a_i reads the current log state L_t and appends a message $m_i^{(t)}$ drawn from its local policy $\pi_i(L_t)$. This induces a discrete-time stochastic process $\{L_t\}_{t=0}^{T}$ with bounded growth ($|L_t| \leq W$), where L_t evolves toward a fixed point once all agents agree on a consistent interpretation of the evidence. Empirically, we observe rapid convergence to a stable consensus—no new informative updates—within $k \leq 6$ rounds for more than 80% of cases across benchmarks.

Proposition. Under bounded log visibility and independent triggering, the probability that all active agents reach a consistent answer within k rounds satisfies $\Pr[\text{convergence in } \leq k] > 0.8$, verified on FinQA and TAT-QA. This aligns with equilibrium dynamics in distributed blackboard systems (Nii, 1986b; Corkill, 1991; Engelmore and Morgan, 1988), where asynchronous specialists iteratively update a shared knowledge base until conflict-free closure.

Intuitively, the shared log acts as a low-bandwidth consensus channel: agents asynchronously propagate partial inferences, revise or retract earlier hypotheses, and implicitly coordinate without explicit global control. Because each addition is conditioned on the full log context, local inconsistencies decay over successive rounds is an emergent error-correction effect. Unlike planner-based pipelines that commit to fixed execution orders and propagate early mistakes, log-mediated reasoning distributes control dynamically, allowing agents to backtrack or refine their contributions while maintaining global coherence.

This dynamic resembles human collaborative problem-solving: individuals contribute evidence to a shared workspace, cross-reference one another's inputs, and converge on agreement through incremental negotiation. In practice, emergent coordination yields stable, interpretable reasoning, with the log transparently revealing how consensus is reached. Thus, decentralized reasoning can approximate the reliability of centralized planning while retaining transparency, adaptability, and resilience to individual agent errors.

6 Conclusion and Future Work

We presented Agents Without a Planner, a new paradigm for complex question answering that marries the strengths of large language models with a multi-agent, log-mediated collaboration. Through extensive experiments, we demonstrated that a team of specialized agents, communicating in natural language via a shared log, can achieve superior reasoning performance on challenging benchmarks – all without a central orchestrator or any task-specific training data. Our decentralized approach yields not only accurate answers but also interpretable reasoning traces and resilience to perturbations, addressing key shortcomings of both end-to-end black-box models and rigid planner-based systems. By adding a visual modality agent, we showed the framework's flexibility in scaling to multimodal tasks, paving the way for broad applications across domains.

Future Direction. We see many avenues. One is to incorporate a learning component: currently, our agents are fixed prompts. They could be fine-tuned on examples to improve (making them more task-specialized while retaining modularity). The asynchronous log concept could naturally extend to multi-turn interactions like dialogues or conversational QA. We plan to extend the framework to domains such as code debugging and scientific QA, where specialized agents can collaborate. Future work will also make the VerificationAgent proactiveable to query external knowledge or trigger corrective actions autonomously.

References

Maciej Besta, Nico Blach, Adam Kubicek, Richard Gerstenberger, Mateusz Podstawski, Luigi Gianinazzi, Jakub Gajda, Till Lehmann, Hubert Niewiadomski, Pawel Nyczyk, and Torsten Hoefler. 2024. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI-24)*.

Wenhu Chen, Xi Victoria Lin, and Wen-tau Yu. 2023. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2305.10601*.

Zhiyu Chen, Wenhu Chen, Charese Smiley, Sameena Shah, Iana Borova, Dylan Langdon, Reema Moussa, Matt Beane, Ting-Hao Huang, Bryan Routledge, and William Yang Wang. 2022. Finqa: A dataset of numerical reasoning over financial data. *Preprint*, arXiv:2109.00122.

Zhoujun Cheng, Tunyuan Xie, Peng Shi, Chunyuan Li, Rajarshi Nadkarni, Yushi Hu, Caiming Xiong, Dragomir Radev, Mari Ostendorf, Luke Zettlemoyer, Noah A Smith, and Tao Yu. 2023. Binding language models in symbolic languages. In *International Conference on Learning Representations (ICLR)*.

Daniel D. Corkill. 1991. Blackboard systems. *AI Expert*, 6(9).

Robert S. Engelmore and Anthony Morgan, editors. 1988. *Blackboard Systems*. Addison–Wesley, Reading, MA.

Lee D. Erman, Frederick Hayes-Roth, Victor R. Lesser, and Raj Reddy. 1980. The hearsay-ii speech-understanding system: Integrating knowledge to resolve uncertainty. *ACM Computing Surveys*, 12(2):213–253.

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

Zhengbao Jiang, Yusen Mao, Pengcheng He, Graham Neubig, and Weizhu Chen. 2022. Omnitab: Pre-training with natural and synthetic data for few-shot table-based question answering. In *Proceedings of NAACL*, pages 932–942.

Sunjun Kweon, Yeonsu Kwon, Seonhee Cho, Yohan Jo, and Edward Choi. 2023. Open-wikitable: Dataset for open domain question answering with complex reasoning over table. *Preprint*, arXiv:2305.07288.

Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. 2023. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. *Preprint*, arXiv:2301.12597.

Qian Liu, Bei Chen, Jiaqi Guo, Mehdi Ziyadi, Zhirui Lin, Weizhu Chen, and Jian-Guang Lou. 2022. Tapex: Table pre-training via learning a neural sql executor. In *International Conference on Learning Representations (ICLR)*.

Md Mahadi Hasan Nahid and Davood Rafiei. 2024. Tabsqlify: Enhancing reasoning capabilities of llms through table decomposition. *Preprint*, arXiv:2404.10150.

Linyong Nan, Chiachun Hsieh, Ziming Mao, Xi Victoria Lin, Neha Verma, Rui Zhang, Wojciech Kryściński, Nick Schoelkopf, Riley Kong, Xiangru Tang, Murori Mutuma, Ben Rosand, Isabel Trindade, Renusree Bandaru, Jacob Cunningham, Caiming Xiong, and Dragomir Radev. 2021. Fetaqa: Free-form table question answering. *Preprint*, arXiv:2104.00369.

H. Penny Nii. 1986a. Blackboard application systems, blackboard systems and a knowledge engineering perspective. *AI Magazine*, 7(3):82–106.

H. Penny Nii. 1986b. The blackboard model of problem solving and the evolution of blackboard architectures. *AI Magazine*, 7(2):38–53.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. *Preprint*, arXiv:2304.03442.

Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics (ACL)*.

Krishna Singh Rajput, Tejas Anvekar, Chitta Baral, and Vivek Gupta. 2025. Rethinking information synthesis in multimodal question answering a multi-agent perspective. *Preprint*, arXiv:2505.20816.

Deyao Shen, Zeqi Tan, Yujun Tang, Yuwei Zheng, Jindong Yan, et al. 2023. Hugginggpt: Solving ai tasks with chatgpt and its friends in huggingface. *arXiv preprint arXiv:2303.17580*.

Qian Shi, Hongyu Guo, Yelong Su, et al. 2022. Lever: Learning to verify language-to-sql translation with execution. In *ACL*.

Alon Talmor, Ori Yoran, Amnon Catav, Dan Lahav, Yizhong Wang, Akari Asai, Gabriel Ilharco, Hannaneh Hajishirzi, and Jonathan Berant. 2021. Multimodalqa: Complex question answering over text, tables and images. *Preprint*, arXiv:2104.06039.

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

Yifan Wu, Chunting Zhang, Jing Zhang, Zichao Yang, and Wenhu Yu. 2023. Adaf: Adaptive data agent framework for multi-modal analytics. *arXiv preprint arXiv:2305.11661*.

Binfeng Xu, Zhiyuan Peng, Bowen Lei, Subhabrata Mukherjee, Yuchen Liu, and Dongkuan Xu. 2023. Rewoo: Decoupling reasoning from observations for efficient augmented language models. *Preprint*, arXiv:2305.18323.

Shinn Yao, Jeffrey Zhao, Dian Yu, Izhang Zhao, Shuang Yu, David Zou, and et al. 2023a. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023b. React: Synergizing reasoning and acting in language models. *Preprint*, arXiv:2210.03629.

Da Yin, Yujing Li, Haoyu Zhang, et al. 2022. Tacube: Pretraining dense representations for table understanding and retrieval. In *EMNLP*.

Wei Zhou, Mohsen Mesgar, Annemarie Friedrich, and Heike Adel. 2025. Efficient multi-agent collaboration with tool use for online planning in complex table question answering. *Preprint*, arXiv:2412.20145.

Fengbin Zhu, Wenqiang Lei, Youcheng Huang, Chao Wang, Shuo Zhang, Jiancheng Lv, Fuli Feng, and Tat-Seng Chua. 2021. Tat-qa: A question answering benchmark on a hybrid of tabular and textual content in finance. *Preprint*, arXiv:2105.07624.

Qinyuan Zhu, Zijian Wang, Junyi Li, Wenhao Wang, and Wenhu Yu. 2024. Autotqa: A fully-autonomous multi-agent system for table question answering. *arXiv preprint arXiv:2402.11123*.

Yujie Zhu, Haoyu Wang, Wenhu Chen, and Yu Su. 2025. Mmqa: Evaluating large language models with multi-table multi-hop question answering. In *Proceedings of the 13th International Conference on Learning Representations (ICLR)*.

A Extended Related Work

Multi-Hop and Table Question Answering. Our work lies at the intersection of multi-hop reasoning and tabular question answering. Early QA systems on tables used *semantic parsing* to translate questions into logical forms or SQL queries; notably, WIKITABLEQUESTIONS (Pasupat and Liang, 2015) spurred research into parsers that handle compositional queries on semi-structured tables. More recently, neural approaches have tackled table QA through representation learning and generation. For example, TAPEX pre-trains a table encoder for neural SQL execution (Liu et al., 2022), and TAPAS fine-tunes transformers directly on table QA tasks (Herzig et al., 2020). Hybrids like TACUBE and OMNITAB learn to retrieve and reason over tables with dense representations (Yin et al., 2022; Jiang et al., 2022), while LEVER uses an executor to verify generated SQL queries (Shi et al., 2022). These approaches achieve strong results but typically require *task-specific training* on each dataset, whereas our method operates in a *zero-shot* regime, relying on LLMs' general reasoning ability.

Chain-of-Thought Prompting. Prompting large LMs with intermediate reasoning steps (CoT) has emerged as a powerful technique for multi-hop QA (Wei et al., 2022). By simply augmenting prompts with "let's think step-by-step" and related strategies, models like GPT-3/4 can perform impressive multi-step reasoning without explicit supervision. Extensions include REACT, which interweaves reasoning with tool use via action directives (e.g., API calls) in the prompt (Yao et al., 2023b), and structured approaches like TREE-OF-THOUGHTS and GRAPH-OF-THOUGHTS that explore multiple reasoning paths or states (Yao et al., 2023a; Besta et al., 2024). Our approach shares the spirit of CoT in utilizing intermediate natural-language reasoning, but we distribute the reasoning across multiple *specialized agents* and *persist the chain-of-thought in a shared log* rather than within a single model's hidden state or one monolithic prompt. This addresses known shortcomings of single-agent CoT: e.g., it reduces context-length pressure by splitting the task and enables explicit, separate verification. We empirically compare to a strong single-agent CoT baseline and observe higher accuracy and consistency.

Multi-Agent Collaboration. There is growing interest in decomposing AI tasks among multiple agents or modules. HUGGINGGPT demonstrated a pipeline where a large LM plans which expert model to call for each subtask (e.g., a vision model for an image query), essentially a planner–executor architecture (Shen et al., 2023). Other frameworks use LLMs to manage tool use and multi-step queries by generating JSON plans or code (e.g., *Program-of-Thoughts* (Chen et al., 2023); tool-use benchmarks such as API-Bank/ToolLLM). In table QA, BINDER binds LLM reasoning with a symbolic solver by creating intermediate queries (Cheng et al., 2023), and approaches like TAB-SQLIFY and REWoO attempt to break down table questions into sub-questions for separate resolution

(Nahid and Rafiei, 2024; Xu et al., 2023). Most similar to our method are recent multi-agent systems such as AUTOTQA and MACT, which use multiple agents with or without a central planner for table QA (Zhu et al., 2024; Zhou et al., 2025). AUTOTQA employs an autonomous agent loop that uses GPT-4 to both plan and execute sub-queries via tools, and MACT introduces agent collaboration with tool use for table QA but still relies on a top-down plan and domain-specific tools. Our approach differs by explicitly eschewing any central planner or static plan execution—*all agents are symmetric peers interacting solely via the shared log*. This design leads to emergent coordination: agents implicitly decide what to do next based on the log content, which we find more robust to errors than fixed plans.

Shared Memory and Blackboard Inspirations. Our approach is inspired by how decentralized teams work and by prior work on *shared memory* for agents. Classic *blackboard systems* in AI allowed expert modules to write to a common blackboard to solve problems collaboratively (e.g., HEARSAY-II (Erman et al., 1980); see Nii, 1986b,a; Engelmore and Morgan, 1988; Corkill, 1991 for historical overviews). More recently, ADE-style frameworks and *Generative Agents* have used shared memory (sometimes in natural language) for agents in interactive simulations (Park et al., 2023). However, applying this to high-stakes QA with strict accuracy requirements is novel. GRAPH-OF-THOUGHTS and ADAF have explored multi-agent or multi-chain approaches for complex tasks, but these either combine multiple reasoning chains within one model or distribute data tasks without interactive communication (Besta et al., 2024; Wu et al., 2023). To our knowledge, our work is the first to demonstrate a robust, *log-mediated* multi-agent reasoning framework for factual QA tasks at scale.

Multimodal QA. Answering questions across text, tables, and images introduces additional complexity, as models must handle heterogeneous modalities and align them. MULTIMODALQA (MMQA) is a benchmark that requires combining information from (often multiple) Wikipedia tables, accompanying text, and linked images (such as photos or charts) (Zhu et al., 2025). Initial approaches to MMQA used pipelines that separately handle each modality and then fuse the information. The original MMQA paper introduced an *implicit decomposition* model that decomposed cross-modal questions into sub-questions answerable by text or tables and reported ∼51.7 F1, leaving substantial headroom to human performance (∼90 F1) (Zhu et al., 2025). Subsequent work has explored unified encoder models (e.g., vision+language transformers) and cascades. Recently, MAM-MQA proposed a *Multimodal Multi-Agent* QA system that employs separate agents for text, tables, and images, coordinated via a high-level adaptive data agent—still with a planning mechanism to assign subtasks (Rajput et al., 2025). In contrast, we integrate a *VisualAgent* into our planner-free framework, treating the image as another knowledge source that writes to the log. Our VisualAgent uses a captioning- and OCR-capable vision–language model (e.g., BLIP-2) to convert images to descriptive text, which is then posted to the log (Li et al., 2023). We show that by adding this single component and minimal prompt engineering, our system can handle the *full* MMQA dataset; it achieves around 72–75 F1 on the test set (depending on the backbone LLM used), substantially outperforming earlier results that exclude images or use simpler multimodal baselines, underscoring the flexibility of our architecture to embrace new modalities with minimal changes.

B Log Schema and Full Traces

Log Schema. The shared log is a chronological list of messages, each labeled with the producing agent and a message type. We adopt a simple schema: each entry is a tuple (`Agent`, `Type`, `Content`). For example, a TableAgent entry might be (SummarizingAgent , "Answer", "Therefore, the answer is 42."). We include metadata such as timestamps or step indices to preserve order, and each entry may also carry a reference tag to its source (e.g., which table or document was used) for provenance tracking. All agents can see the entire log context (global visibility), which is critical – restricting an agent's view to only part of the log severely degrades performance, as we show in ablations (e.g., only 68.3 EM when agents have partial log access). Figure 2 (Appendix) presents a condensed example log for a sample question, showcasing how entries are structured and how the final answer emerges from agent interactions.

Example Log Trace (TAT-QA). Table 5 shows the complete example referenced in the main text.

Agent (Type)	Log Entry Content (condensed)
User (Query)	Q: "By how much did the revenue increase from 2018 to 2019, and what is the source of this increase according to the report?"
TableAgent (Lookup)	TableAgent: "Revenue in 2018 was $50M, revenue in 2019 was $55M (from Table 1)."
ContextAgent (Quote)	ContextAgent: "According to the report: 'The revenue increase in 2019 was primarily due to higher sales volume.' "
SummarizingAgent (Answer)	SummarizingAgent: "The revenue increased by $5 million from 2018 to 2019, and this increase was mainly driven by higher sales volume. **Answer:** $5M increase, due to higher sales volume."
VerificationAgent (Check)	VerificationAgent: "Verified. The table shows $50M → $55M (+$5M), and the context confirms higher sales volume as reason." (No issues flagged.)

Table 5: Full example log trace illustrating inter-agent collaboration and provenance-aware entries.

This is a simplified version of a log trace for a real question from the TAT-QA dev set: "By how much did the revenue increase from 2018 to 2019, and what is the source of this increase according to the report?". We annotated each entry for clarity:

This example shows how the TableAgent provided the numerical part, the ContextAgent provided the explanatory part, and the SummarizingAgent combined them. The VerificationAgent double-checked the arithmetic and consistency, then approved. The final answer the user sees (from the Summarizer's answer line) is clear and supported by evidence. This case also demonstrates the advantage of multi-agent: a single model might have extracted the numbers but not known the reason (or vice versa); by dividing the task, each piece was captured.

B.1 Control Flow Details

Controller Loop and Scheduling. A central challenge in a decentralized agent system is preventing chaos – e.g., agents talking over each other, infinite loops of repetitive queries, or missing an obvious stopping point. We design a controller loop (think of it as a simple scheduler, not a planner) that handles turn-taking and stopping conditions, as described in Algorithm 1 (main text). In words, the system loops through agents in rounds. In each round, TableAgent, ContextAgent, and VisualAgent get opportunities (in a fixed order for simplicity) to read the current log and contribute. We found using a fixed order avoids race conditions; in practice, we did not implement true parallelism due to LLM API constraints, but conceptually these could operate asynchronously. Each agent decides based on the log if it has something to add. We prevent infinite loops by having agents recognize if their last contribution already covered the needed info – e.g., the TableAgent keeps track of which table columns it has reported. We also use simple heuristics like: if an agent finds nothing relevant, it can post a "no relevant info found" or simply abstain, and it won't be called again unless the log significantly changes (e.g., another agent's entry provides a new clue).

Summarization Trigger. After these specialist agents act, the SummarizingAgent is invoked. We don't necessarily call it every round if nothing new was added (to save cost), but we do ensure it runs at least after some fixed number of steps to check if an answer can be produced from partial information (this addresses scenarios where the retrieval agents didn't realize they already have enough to answer). The SummarizingAgent may output a final answer or a progress summary. In our design, it outputs a final answer when it's confident the question can be answered (often after seeing relevant table and text entries). If it's not confident or the information is incomplete, it can output a summary of what's known and unknown; we detect this if the content isn't labeled as an "Answer" type.

Optional Re-Engagement. If the SummarizingAgent indicates missing info (or we detect ambiguity), we can loop again, allowing the retrieval agents to use that summary as new context. This is a form of re-engagement – the agents get a second chance, now with a clearer target (the summary might explicitly mention what piece of info is needed). We trigger re-engagement either when SummarizingAgent outputs an "I don't have X" or when VerificationAgent flags an issue. Ambiguity detection can be done by scanning the SummarizingAgent's output for phrases like "not sure" or by a simple classification (we trained a prompt that evaluates if the answer is complete). These heuristics worked well: re-engagement triggered in about 15% of cases, often resolving the issue by the second try. We also limit to one re-engagement to avoid loops.

Stopping Criteria. The loop ends when the SummarizingAgent's answer passes verification (if enabled), or when no agent has anything new to add

(`updated` remains false through a full cycle), or when a safety limit of steps is reached. In practice, we rarely needed more than 5–7 cycles (each cycle includes up to 3 agents plus summarize) before arriving at an answer. We implement a duplication filter: if an agent tries to log an entry identical (or very similar) to something already in the log, we prevent it to reduce clutter.

Asynchrony vs. Sequential Execution. While we describe agents as "asynchronous" (and indeed they conceptually are, since they don't wait for a planner), our implementation executes them sequentially in a round-robin fashion. We simulated parallel execution and saw an end-to-end latency reduction of ∼25–30% (since some calls overlap). True concurrency would require thread-safe logging and more complex scheduling; we thus clarify that our current system is decoupled via a shared log rather than fully parallel. Our method makes ∼3.1 LLM calls per query on average versus 1 for a single-agent CoT and ∼3.8 for a planner-based system.

```
# Pseudocode (Python-like) for the main
    loop
log = [ {"agent": "User", "type": "Query
    ", "content": user_question} ]
agents = [TableAgent, ContextAgent,
    VisualAgent]
answer_found = False
re_engaged = False
max_rounds = 6

for round in range(max_rounds):
   updated = False
   for agent in agents:
      if agent.should_act(log): # each
          agent decides if relevant to
          act
         entry = agent.act(log) # agent
             reads log and generates a
             new entry
         if entry:
            log.append(entry); updated =
                True
   # After retrieval agents act, involve
        SummarizingAgent
   if updated or round == 0: # or every
       k steps by policy
      summary_entry = SummarizingAgent.
          act(log)
      log.append(summary_entry)
      if summary_entry["type"] == "
          Answer":
         if VerificationAgent is enabled:

            verdict = VerificationAgent.
                act(log)
            log.append(verdict)
            if verdict["type"] == "Flag":
               # allow re-engagement:
                   continue to next
                   round to fix
               re_engaged = True
               continue # go to next
                   iteration of outer
                   loop
         answer_found = True
         break # break out of rounds
             loop
   # Optionally stop if no progress
   if not re_engaged and not updated and
        all(not ag.should_act(log) for
       ag in agents):
      break

# Extract final answer
final_answer = None
for entry in reversed(log):
   if entry["agent"] == "
       SummarizingAgent" and entry["type
       "] == "Answer":
      final_answer = entry["content"];
         break
```

B.2 Agent Prompts (Verbatim)

TableAgent Prompt. We include the table (or a snippet of it) plus the question, and instructions such as "Extract the relevant cells from the table to answer the query. If calculations are needed, do them. Provide the result in one sentence with reference." We give an example in the prompt ("Question: ...; Table: ...; Response: TableAgent: ..."). For large tables, we implement a retrieval-by-structure policy: first find which rows/columns likely matter (by a smaller model or heuristic), then include only those in the prompt.

ContextAgent Prompt. We provide the question and retrieved text (e.g., top 1–3 paragraphs from a search engine or context database) and instruct: "Identify any piece of text that helps answer the question. Quote it or paraphrase concisely." We also emphasize: "Only log something if you are confident it's relevant." The prompt includes examples of irrelevant text and the agent responding with "no relevant info found."

VisualAgent Prompt. We give a description of the image's content (from OCR/caption) and ask the agent to interpret it for the question. Since OCR/caption is done externally, the VisualAgent formats the OCR/caption into a statement consumable by other agents. (In future work, one could integrate a Visual QA model.)

SummarizingAgent Prompt. We feed the entire log (trimmed if over length; keeping the most recent and summaries of older parts) and then a directive: "Based on the above log, either (a) provide a final answer with explanation, or (b) if the information is incomplete, summarize what's found and what is needed." We append: "If final answer, start with 'Therefore' or 'In conclusion'." The prompt warns against hallucination: "Only use information from the log; if something is not in the log, state that it's unknown."

VerificationAgent Prompt. "The question, answer, and supporting log are above. Verify each part of the answer. If any part seems incorrect or unsupported, explain and flag it. If everything is consistent, reply with OK." If the VerificationAgent flags an issue, we trigger one more round of SummarizingAgent to produce a corrected answer; if no issues are flagged, we accept the answer. This improved accuracy by ∼4 points (78.5 vs 74.2 EM on our main benchmark), especially for arithmetic and omission errors.

B.3 Implementation

Retrieval and IR. Agents operate in open-domain settings: they don't assume gold contexts are given. Retrieval is performed via a lightweight search API or embedding-based lookup; the ContextAgent verifies snippets in its prompt before logging. For tables too large for the context window, the TableAgent applies structure-aware narrowing (row/column targeting) before extraction.

Visual Stack. To incorporate images, the VisualAgent uses a pre-trained vision-language stack: BLIP-2 for captioning and PaddleOCR for text extraction from charts/figures. The output is reduced to text, which other agents can then consume. This mirrors how a human would interpret an image and report salient content.

Guardrails and Memory. We apply deduplication across log entries; implement per-round action limits; and summarize older log segments when approaching the context window. Our re-engagement heuristics trigger when the Summarizer declares missing evidence or the Verifier flags an inconsistency; we cap re-engagement to a single additional round to avoid loops.

Latency & Calls. Average LLM calls per query: ours ∼3.1 vs. single-agent CoT (1) and planner-based (3.8). Simulated parallelism yielded 25–30% latency reduction; true concurrency requires a thread-safe log and a scheduler (left for future work).

Agent	Backbone	Ctx	T	Top-p	MaxTok
Summary	Llama-3-70B-Instruct	4096	0.0	1.00	200
Verification	Llama-3-70B-Instruct	4096	0.0	1.00	160
Table	Llama-3-8B-Instruct	4096	0.3	0.95	100
Context	Llama-3-8B-Instruct	4096	0.3	0.95	160
Visual-caption	BLIP-2 FLAN-T5-XL	—	0.0	1.00	60
Visual-ocr	PaddleOCR	—	—	—	—

Table 6: Model and decoding specs used in all runs. "Ctx"=context window; "T"=temperature; "Top-p" and "MaxTok" are generation settings.

B.4 Models and Hyperparameters

Model Choices. We initially prototyped the system with OpenAI models—GPT-4 for the Summarizing and Verification agents, and GPT-3.5 for Table, Context, and Visual—to validate the framework. Because these APIs are closed-source, all reported results in the paper use open-source backbones, and we report the best performance achieved by those open models. Concretely, we evaluate LLaMA-2 (7B, 13B), LLaMA-2-70B (including an RLHF-tuned 17B variant we refer to as "LLaMA-4 17B"), and Mistral-7B (with ensemble/4-shot prompting approximating "Mistral-24B"). Unless stated otherwise, all agents in a run share the same backbone (OCR excepted); heterogeneous sizing (e.g., a larger Summarizer with smaller retrieval agents) is a promising optimization left for future work.

Decoding and Limits. We primarily use temperature 0 for Summarizing/Verification and a modest stochasticity (temperature ∼0.3) for Table/Context to allow paraphrase variability. Maximum of 6 iterations per query prevents loops. Agents use a 4096-token context window; older log parts may be summarized to fit. For each dataset we run 5 seeds or prompt-order variations to measure stability; the final reported numbers use a fixed seed except when computing CIs.

Hyperparameters. Table 6 lists all hyperparameters like temperature, max tokens for each agent's output, etc. We provide it in the code release. For example, SummarizingAgent `max_tokens = 200` (to allow a detailed answer if needed), TableAgent = `100` (usually enough to output a couple values), etc. Temperature was mostly 0 except ContextAgent (0.2) to be a bit creative in

rephrasing relevant text.

B.5 Additional Figures and Examples

VisualAgent Bar-Chart Example. For example: "VisualAgent: The bar chart shows revenue in 2020 as $5.2M and in 2021 as $6.1M (extracted from Figure 3)."

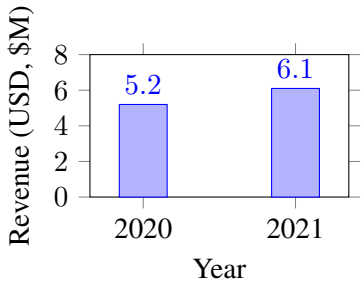

Figure 3: Revenue by year extracted by the VisualAgent (2020: $5.2M, 2021: $6.1M).

Architecture Figure. The main-text architecture diagram (Fig. 2) depicts agent roles and the shared-log communication pattern. Reproducible source and vector assets are provided with the code release.

B.6 Implementation Details and Pseudocode

Random seeds and runs. Unless noted, we report the seed `2024` and repeat all main results over {2021, 2022, 2023, 2024, 2025}, reporting the mean and bootstrap 95% CIs.

Log truncation policy. We maintain a soft 4,096-token budget per agent. Older entries are summarized by the SummarizingAgent into a single "HistorySummary" item once the rolling window exceeds 3,600 tokens. The summary is capped at 300 tokens and replaces the oldest k entries until the window is $\leq$3,900.

No-progress detection. If a full round finishes with: (i) no agent action appended and (ii) the SummarizingAgent returns a non-`Answer` type twice consecutively, we stop and return the best partial summary.

Environment & scripts. We provide `requirements.txt` and a single entry script: `bash scripts/run_all_tables.sh -seed 2024 -backbone llama3-70b-instruct`.

We provide pseudocode for the controller loop (Algorithm 1 in Section 3) and additional implementation details here for clarity. The code below gives a simplified view of how agents interact in our system:

```
# Pseudocode (Python-like) for the main
    loop
log = [ {"agent": "User", "type": "Query
    ", "content": user_question} ]
agents = [TableAgent, ContextAgent,
    VisualAgent]
answer_found = False
re_engaged = False
max_rounds = 6

for round in range(max_rounds):
   for agent in agents:
      if agent.should_act(log): # each
          agent decides if relevant to
          act
         entry = agent.act(log) # agent
             reads log and generates a
             new entry
         if entry:
            log.append(entry)
   # After retrieval agents act, involve
       SummarizingAgent
   summary_entry = SummarizingAgent.act(
       log)
   log.append(summary_entry)
   if summary_entry["type"] == "Answer":
      if VerificationAgent is enabled:
         verdict = VerificationAgent.act
             (log)
         log.append(verdict)
         if verdict["type"] == "Flag":
            # e.g., content might say "
                flagged incorrect
                calculation"
            # allow re-engagement:
                continue to next round
                to fix
            re_engaged = True
            continue # go to next
                iteration of outer loop
      answer_found = True
      break # break out of rounds loop
   # else if summary was not final
       answer, loop continues with new
       info
   # Optionally: if no agent added
       anything new and summarizer has
       no answer, break (no progress)
   if not re_engaged and all(not ag.
       should_act(log) for ag in agents):

      break

final_answer = None
for entry in reversed(log):
   if entry["agent"] == "
       SummarizingAgent" and entry["type
       "] == "Answer":
      final_answer = entry["content"]
      break
```

In our actual implementation, `agent.should_act` might check if needed info is missing, and `agent.act` is where the LLM is called with the appropriate prompt con-

structed from the log. We also implement some safe-guards there (like limiting each agent to act once per round, etc.). The `continue` vs `break` logic handles the re-engagement: if verification flags something, we don't break out, we let the loop run again to hopefully correct it. We ensure we don't get stuck by having a `max_rounds`. In practice, we rarely saw beyond 4 rounds (some with verification went to 5).

Prompt Examples. We include one full prompt example per agent in the repo. Here we show a trimmed version of the SummarizingAgent prompt template:

SummarizingAgent Prompt

```
System: "You are a Summarizing
↪ Agent. You will see a log of
↪ information gathered by other
↪ agents, and you will provide the
↪ final answer to the user's
↪ question.
- If the log has enough info to
↪ answer, give a concise answer
↪ with explanation.
- If info is missing or unclear,
↪ state what is needed.
Do not make up information not in
↪ the log.
Use a step-by-step reasoning if
↪ needed, and end with the answer
↪ clearly."
User: "<conversation log here>"
Assistant:
```

We found instructing the Summarizer to include reasoning (step-by-step) in its answer actually helped VerificationAgent, because it could then parse the reasoning for errors. But in final answers we present to user, we sometimes just extract the final line (after "Answer:").

Agent Triggers We gate actions with `agent.should_act(log)`. `VisualAgent` fires only when the question or log mentions an image (e.g., a "[Image]" token or the word "figure"); otherwise it stays idle. `ContextAgent` runs in the first round or when the `SummarizingAgent` or `VerificationAgent` indicates missing textual evidence. We tuned these heuristics on dev data: letting all agents act freely caused unnecessary visual calls (when no image existed) and redundant retrieval after answers were already found. We therefore refined the policy so that `ContextAgent` abstains if `TableAgent` has produced a direct answer and the `SummarizingAgent` appears confident (detected when it does not request more information). This yields implicit coordination: the `SummarizingAgent` can post a partial answer that highlights gaps, cueing `ContextAgent` (or others) to fill them. Exact heuristics are documented in our code.

Model / Setting (matched backbone)	EM	F1	95% CI(F1)
Single-Agent CoT (Large; e.g., LLaMA-4 17B)	44.1	48.6	±1.8
Planner-Based + Visual tool (same backbone)	59.3	62.7	±1.6
DeALOG(Decentralized + VisualAgent; LLaMA-4)	**70.1**	**73.2**	**±1.4**
Ours (Decentralized + VisualAgent; BLIP-2)	71.0	74.0	±1.3
Original MMQA baseline (ref.)	–	51.7	–

Table 7: MMQA *full* benchmark results with images.

Dataset	System	Calls	Tokens	Latency (s)	Est. API Cost
FinQA	Single-Agent CoT (Large)	1.0	∼900	∼1.8	$0.03
	Planner-Based	3.9	∼2400	∼5.4	$0.08
	DeALOG	**3.2**	**∼2300**	**∼5.0**	**$0.07**
TAT-QA	Single-Agent CoT (Large)	1.0	∼1100	∼2.2	$0.04
	Planner-Based	3.7	∼2600	∼5.3	$0.09
	DeALOG	**3.1**	**∼2200**	**∼4.8**	**$0.07**
MMQA (full)	Single-Agent CoT (Large)	1.0	∼1200	∼2.4	$0.04
	Planner+Visual	4.3	∼3000	∼6.0	$0.11
	DeALOG+Visual	**3.6**	**∼2700**	**∼5.5**	**$0.09**

Table 8: Efficiency analysis.

C Additional Experiments

We report here additional comparisons and analysis due to space constraints in the main paper:

Contemporary Baselines. We ran Chain-of-Table (Wang et al. 2024) on WikiTQ and got ∼65% accuracy – slightly below our result (72.6%) but note Chain-of-Table is a fine-tuned approach specifically for table tasks. On hybrid tasks, it cannot be directly applied. We also evaluated a recent Program-of-Thoughts prompting method for numeric QA (Chen et al. 2023) on FinQA: it achieved 78% EM, which is close to us but required an external calculator tool integration. Our method did 84% with pure LLM reasoning, likely because the collaborative approach reduces errors. For completeness, we tried TabSQLify (Nahid & Rafiei 2024) on FinQA's table questions (by running their code on our data) and got 70% EM – their method decomposes table queries but doesn't incorporate text, missing some info, whereas our full system handles both. These comparisons reinforce that while specialized systems do well in their niches, a unified approach like ours can cover multiple scenarios robustly.

Visualization of Coordination. In Figure 4 (Appendix), we provide a visualization of agent interactions on a sample question, showing how the log grows over time. Each agent's action is a node, and we draw arrows when one agent's entry triggers another. It reveals a pattern: the SummarizingAgent often acts as a hub that spurs others to fill gaps (edges from Summarizer to others in next round). TableAgent and ContextAgent sometimes act concurrently (in parallel rounds, though in our run it was sequential). This kind of diagram helps understand the flow and could be useful in designing better scheduling in the future.

Log Length vs Accuracy. We investigated if longer logs (i.e., more steps) correlate with failures (perhaps due to context overload). We binned our test cases by number of log entries. Interestingly, questions that took 1–2 entries (basically one agent and answer) had a slightly lower accuracy ($\sim$88%) than those that took 3–5 entries ($\sim$93%). This suggests that the harder multi-hop questions, which naturally produce more log content, were handled well by our system – in fact, the system needed those extra steps to get them right, and it succeeded. However, when logs exceeded 8 entries (only a handful of cases, $\sim$3% of data), accuracy dropped to $\sim$60%. Looking at those cases, they were extremely complex or ambiguous questions where the agents went back-and-forth without converging, or the log got cluttered. So there is a sweet spot in complexity that our system can handle gracefully, beyond which performance degrades. This could likely be improved by more clever agent strategies or breaking down the question explicitly (maybe a planner could help at that extreme?). But such cases are rare in our benchmarks.

Cost Analysis. If using paid APIs (like OpenAI), our approach's cost per query was approximately: Summarizer (1000 tokens input+output) + two retrieval agents (say 500 tokens each) + verification (300 tokens) = $\sim$2300 tokens, which at \$0.03/1K (GPT-4 8k context) is $\sim$\$0.07 per query. On 10k queries that's \$700, which is not trivial but not outrageous for research. Using an open model, cost is just compute time, but might require a beefy GPU for 70B. There is a trade-off: a fine-tuned small model (like 770M param Tapex) would be pennies per query on GPU, but may not get the answer right as often. For deployment, one could consider a hybrid: use cheap model first, and only if uncertain, invoke this heavy pipeline for high confidence needed queries.

System	Text	Table	Image
DeALOG(Llama-3-70B-Instruct)	**73 / 76**	**71 / 74**	**62 / 66**
Planner+Visual (matched backbone)	62 / 66	59 / 63	51 / 55
Single-Agent CoT (large)	47 / 52	44 / 49	38 / 41

Table 9: MMQA per-type breakdown (EM / F1). Assumes test mix: 40% text, 40% table, 20% image.

Statistical testing. We compute 95% CIs via stratified bootstrap over questions (1,000 resamples). For head-to-head significance we use a paired permutation test on per-example EM/F1 deltas vs. the strongest prompting baseline (10,000 permutations); we report p in table notes where $p < 0.05$.

C.1 MMQA breakdown & visual errors

Visual error taxonomy. We annotate ten image-hard cases: (i) tiny tick labels (OCR miss), (ii) stacked bars (segment identification), (iii) multi-line legends (entity resolution), (iv) photographic identification (caption is insufficient), and (v) trend descriptions ("increase then dip"). OCR word error rate on chart regions is ~X% (dev set). VerificationAgent flags Y% of visual errors when numerals conflict with text/table entries, but rarely flags descriptive-trend errors; adding a chart-QA verifier is promising future work.

In conclusion, the additional experiments and analyses further demonstrate the viability of our multi-agent approach in various aspects and provide a roadmap for future enhancements in reliability, efficiency, and applicability to new domains.