David vs. *Goliath*: Cost-Efficient Financial QA via Cascaded Multi-Agent Reasoning

Chenghao Liu[1], Qian Liu[1]*, Ziqin Zhu[1], Hao Fei[2], Aniket Mahanti[1]
[1]School of Computer Science, University of Auckland, New Zealand
[2]School of Computing, National University of Singapore, Singapore
{chenghao.liu, liu.qian, a.mahanti}@auckland.ac.nz, zhuziqin2000@gmail.com, haofei37@nus.edu.sg

Abstract

Large language models (LLMs) have demonstrated remarkable reasoning capabilities, including in financial question answering (FQA). However, the performance in FQA remains limited, particularly in questions that require deep financial knowledge and complex numerical reasoning. While supervised fine-tuning and closed-source LLMs have shown promise, they are often constrained by high costs or computational inefficiency. In this paper, we propose a low-cost yet effective framework, named FinMAN (**Fin**ancial **m**ulti-**a**ge**n**t framework), that enables small LLMs (e.g., 8B) to perform complex reasoning tasks without relying on expensive models or task-specific fine-tuning. FinMAN improves formula selection, extraction, and calculation to help small-scale models solve FQA tasks more accurately, with a lightweight verification mechanism to correct common errors. Experimental results show that FinMAN outperforms the best open-source model on BizBench by 23.96% and achieves competitive performance to GPTo3-mini using significantly fewer parameters. Our code and data are publicly available at `https://github.com/coenliu/MultiAgentFin`.

1 Introduction

Recent advances in LLMs (Touvron et al., 2023; OpenAI, 2023; Wu et al., 2024, 2025; Fei et al., 2024a,b, 2025) have led to remarkable progress across a wide range of natural language processing (NLP) tasks (Qin et al., 2024; Liu et al., 2025; Ju et al., 2025; Wang et al., 2025; Wei et al., 2024). However, their performance remains limited when tackling domain-specific and knowledge-intensive problems such as financial question answering (FQA) (Reddy et al., 2024). In real-world FQA scenarios, high performance is typically achieved with large, closed-source models (e.g., GPT-3.5 (Ye et al., 2023)), which are expensive and inaccessible to many users (Gao et al., 2023a). In contrast, smaller, open-source models (e.g., Llama (Meta, 2024)) are more affordable but underperform, particularly when dealing with long and complex financial documents. In this work, we aim to bridge this gap by developing an effective and affordable LLM-based method tailored for real-world FQA applications.

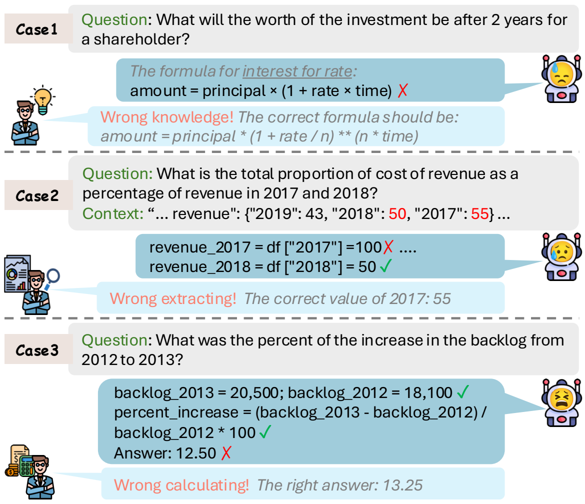

Figure 1: Illustration of error cases generated by smaller LLMs (e.g., Llama2) and correct answers by `FinMAN`.

Small-scale LLMs typically face three challenges in FQA. First, they often struggle to provide accurate and reliable financial knowledge. For instance, in *Case 1* of Fig. 1, a small LLM (e.g., Llama2) applied an incorrect formula to calculate *annual income*, demonstrating insufficient domain knowledge for answering professional financial questions. Second, they have difficulty in extracting key financial values from lengthy financial contexts. As shown in *Case 2*, they fail to extract the correct value for *Statement of Operations* from the given long financial document. Third, they underperform in multi-step reasoning and calculations, frequently making errors in intermediate steps or

*corresponding author

Findings of the Association for Computational Linguistics: EMNLP 2025, pages 4212–4229
November 4-9, 2025

the final computation (*Case 3*). Given that FQA involves complex, multi-step solutions, any type of these errors can lead to incorrect answers.

To address these challenges, a common approach is to apply supervised fine-tuning (SFT) (Zhu et al., 2024) on small LLMs for FQA. However, fine-tuning requires annotated data and substantial computational resources, making it costly and less accessible. In response to specific challenges, some studies have explored alternative strategies, such as incorporating external financial knowledge via retrieval-augmented generation (RAG) (Chen et al., 2024b) or enhancing numerical reasoning through external computational tools (Theuma and Shareghi, 2024). While these approaches show promise, they often depend heavily on external retrieval systems or tools and fall short of offering a unified solution to the broader challenges of FQA. Additionally, some efforts have been made to improve small LLMs' reasoning ability by knowledge distillation from large models like GPT-3.5 and DeepSeek (Tyen et al., 2024). However, distilling the intricate reasoning capabilities required for complex FQA tasks remains a significant hurdle, limiting the effectiveness of such methods in guiding meaningful improvements (Li et al., 2025).

In this work, we explore how smaller LLMs can achieve strong performance in FQA *without* relying on closed-source LLMs. To this end, we propose a cost-efficient method using cascaded multi-agent reasoning, named `FinMAN`. Specifically, we decompose FQA into modular subtasks, each assigned to a distinct agent specialized in a particular role. We design four agents: a `Formulator` (financial expert) for reasoning and formula planning, a `Resolver` (data analyst) for report comprehension and quantity extraction, an `Executor` that generates and runs code to compute the answer, and a cross-cutting `Evaluator` that verifies and corrects outputs at each stage. These subtasks become more facilitated once decomposed and clearly defined for each agent, particularly for the `data analyst` and `executor`, whose responsibilities are straightforward. Moreover, the `financial expert`, based on small LLMs, is required to master deep, domain-specific knowledge. To support this agent, we employ Monte Carlo Tree Search (MCTS) (Zhang et al., 2024a) to simulate and explore reasoning paths, and we integrate an `evaluator` agent that leverages external financial knowledge to score and verify each intermediate step. Consequently, `FinMAN` gains an accuracy of 25.5% in the out-of-domain FinanceMATH dataset (Zhao et al., 2024a), surpassing GPT-3.5.

Additionally, to ensure reliability and transparency, we incorporate a stepwise verification mechanism, making each output traceable and verifiable. Our method draws inspiration from the human strategy of breaking down complex tasks into manageable subtasks, allowing smaller LLMs to become proficient by addressing these subtasks[1].

To validate its effectiveness, we evaluate `FinMAN` on widely used FQA tasks, covering both quantity extraction and financial mathematical reasoning. By enabling cooperation among multiple smaller agents (e.g., Llama3-8B), our approach improves mathematical reasoning accuracy from 32.98% to 46.63%, reaching or even surpassing the performance of GPT-3.5 (i.e., 36.1%).

Our contributions are summarized as follows:

- We propose `FinMAN`, a cost-effective and open-source multi-agent framework that enables small LLMs to achieve strong performance on real-world FQA tasks.

- We design a cascade of four agents: `Formulator` (financial expert), `Resolver` (data analyst), `Executor`, and a cross-cutting `Evaluator`. The `Formulator` plans formulas with MCTS using retrieved financial knowledge, while the `Evaluator` performs stepwise verification at each stage. This architecture enables `FinMAN` to address multi-step, domain-specific financial question answering effectively.

- Experiments show that `FinMAN` achieves substantial performance gains, outperforming or matching larger closed-source models in both quantity extraction and FQA tasks.

2 Related Works

LLMs in FQA. Recent FQA tasks integrate both tabular and textual data, supported by expert-annotated datasets like FinQA (Chen et al., 2021) and ConvFinQA (Chen et al., 2022b), derived from S&P 500 annual reports. This challenging task has garnered significant attention, leading to the development of several notable models.

[1]Like the biblical tale of *David vs. Goliath*, where a young shepherd defeats a giant warrior with a sling, our method employs a multi-agent framework in which specialized smaller LLMs (*David*) collaborate to match the performance of much larger models (*Goliath*).

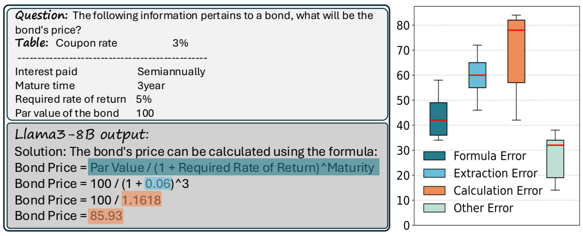

Figure 2: Illustration of error types for FQA (left) and average error rates (%) across seven small LLMs (right).

For example, FinBERT-21 (Liu et al., 2021) is a BERT-based model trained on general and financial texts; FinMA (Xie et al., 2023) fine-tunes Llama (7B/30B) on financial QA datasets (Chen et al., 2021, 2022b); InvestLM (Yang et al., 2023) adapts Llama-65B with curated financial instructions; and BloombergGPT (Wu et al., 2023), a BLOOM-style closed-source model trained on 363B tokens, demonstrates strong FQA proficiency. Following this line (Phogat et al., 2024), we introduce a multi-agent framework that leverages small LLMs for cost-efficient FQA.

Multi-agent Framework. Modern LLMs have demonstrated advanced capabilities that motivate the creation of agent systems tailored for specific tasks (Yao et al., 2023; Park et al., 2023). In the financial sector, StockAgent (Liu et al.) specializes in stock market analysis by forecasting price movements. FinAgent (Zhang et al., 2024b) provides comprehensive financial evaluations and strategic recommendations. FinMEM (Yu et al., 2024) employs memory-augmented models to combine historical market data with current conditions for long-term investment strategies. Li et al. (2024b) presents a factor mining agent using a neuro-symbolic approach to achieve superior performance with an annualized return. Han et al. (2024) investigated various multi-agent collaboration structures for analyzing investment research reports. Targeting FQA, we design a new lightweight multi-agent framework based on small LLMs, which simulate real-world division of labor. It comprises four agents: a `Formulator` (financial expert), a `Resolver` (data analyst), and an `Executor`, plus a cross-cutting `Evaluator` that enables step-by-step, verifiable reasoning.

MCTS in Reasoning. Compared to large-scale LLMs, small LLMs often lack the reasoning ability (Fei et al.; Xu et al., 2025; Chen et al., 2024a, 2025; Cheng et al., 2025; Wang et al., 2025) needed for complex tasks like formula generation in FQA. To mitigate this limitation, tree-search-based methods, particularly MCTS (Qi et al., 2024), have been explored to enhance reasoning, which is employed to simulate potential outcomes of various decisions, enabling the agent to navigate through different hypotheses or possible solutions. To improve financial reasoning, RAP (Hao et al., 2023a) focuses on simple reasoning using only internal knowledge. XOT (Ding et al., 2024) employs MCTS as an external tool to refine LLM-generated thoughts. Different from previous methods, our `expert` agent embeds MCTS directly within a domain-specific QA, leveraging a curated financial knowledge bank to model and explore relationships among financial concepts for accurate formula selection.

3 Preliminary

3.1 Task Definition

Following the widely adopted FinQA benchmark (Chen et al., 2021), our input formulation focuses exclusively on textual content and structured tables. Given a textual content E and a table T, along with a question Q, the FQA task is to generate the final answer Ans. To arrive at Ans, the model needs to produce a sequence of reasoning steps $S = \{s_0, s_1, s_2, \ldots, s_n\}$, where each s_i denotes an individual step required to derive the correct solution. The relationship among these components is formalized as follows:

$$P(Ans|T, E, Q) = \sum_{S_i \in S} P(S_i|T, E, Q) \qquad (1)$$

In addition to answering the question, FQA involves extracting quantities during reasoning steps. This subtask is called *quantity extraction*, which is identifying numerical values within a given context (Göpfert et al., 2022). A quantity typically consists of a numeric value and, when applicable, a unit of measurement. Modifiers such as "average," "approximately," or "above" can alter the meaning of a quantity and may be included within the quantity span (Sun et al., 2024). Quantities can appear as ranges, enumerations, uncertainties, or combinations (Srivastava et al.).

3.2 Probing Experiments

To gain deeper insights into the limitations of small-scale LLMs in FQA, we conduct a probing analysis on the widely used benchmark CodeFinQA (Krumdick et al., 2024). We test seven

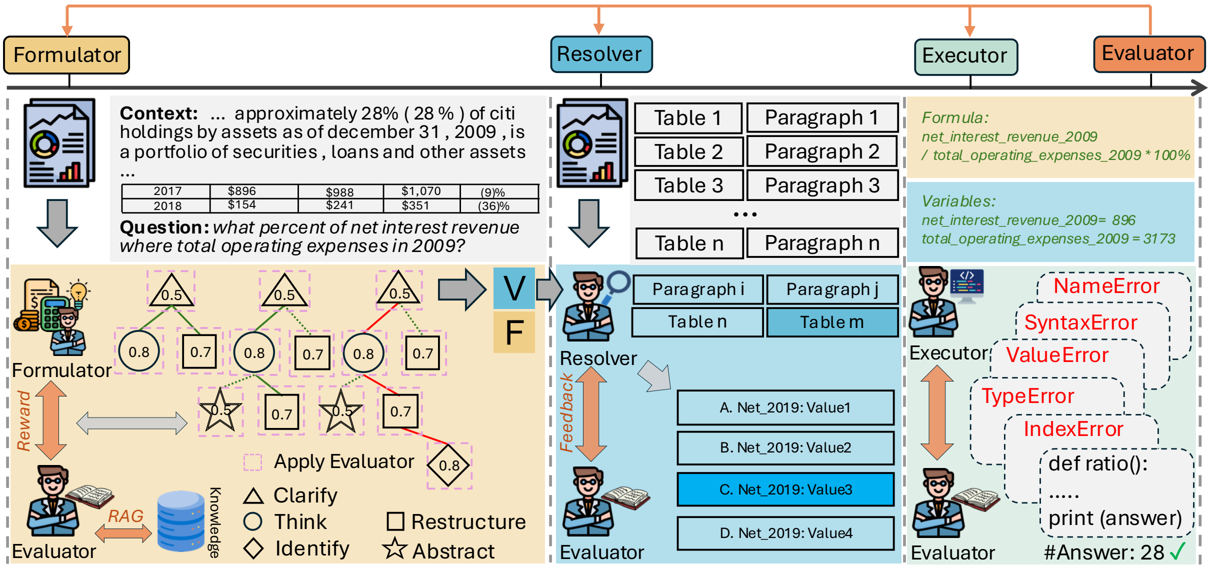

Figure 3: The overview of our proposed FinMAN. The top is the input data and the bottom is an example of workflow. The core pipeline operates in three stages (*Formulator*→*Resolver*→*Executor*); a cross-cutting *Evaluator* agent cross-checks and corrects outputs at each stage.

small LLMs in a few-shot setting, and randomly sampled 50 failed cases for each model, resulting in a total of 350 instances for manual error analysis. Details of this experiment are presented in Appendix C.

It is observed that LLM-generated responses to FQA tasks typically involve a three-stage process (as illustrated in Fig. 3): applying appropriate financial formulas, extracting relevant data, and performing calculations. Our probing experiment reveals that errors can occur at any of these stages, and a single failed case may involve multiple error types.

Then, we ask three experts[2] to annotate these cases, and the distribution of error types is shown in Figure 2. It is observed that **calculation errors**, which involve mistakes during intermediate or final computational steps, are the most common, accounting for 84% of failed cases. **Formula errors**, the second most frequent, occur when the model selects or applies incorrect financial formulas (Srivastava et al.). **Extraction errors** arise from inaccurately identifying or extracting relevant numerical values from provided texts or tables (Zhao et al., 2024a). Lastly, **other errors** include issues such as misunderstanding questions.

However, when considering the subtasks involved in these error types, such as extraction and calculation, it is notable that existing methods already perform well individually. For instance, 72% of failures are extraction errors, but treating extraction as a standalone task yields strong results; e.g., a simple fine-tuned GLM-3 achieves a high match rate (BERTScore F1 of 0.81) (Chen et al., 2024b). This gap suggests that the performance of complex FQA tasks can be improved by decomposing the workflow into specialized agents for formula selection, data extraction, and calculation.

4 Methodology

The overall architecture of our `FinMAN` framework is shown in Figure 3. The basic idea is a cascaded pipeline of four agents. The *Formulator* agent uses MCTS to explore reasoning paths and generate financial formulas and variables. Next, the *Resolver* agent retrieves relevant text and table segments to extract variable values. Then, the *Executor* agent converts these formulas and values into Python code and executes it to obtain the answer. Finally, the *Evaluator* agent performs error detection to verify and correct each intermediate output.

Formulator Agent with MCTS. In FQA, selecting the appropriate formula and its variables is crucial for obtaining the correct result. Given a question that includes context E, tables T, and query Q, the Formulator agent $M_{formulator}$ integrates MCTS to explore candidate reasoning trajectories composed of intermediate steps. We define an action set $\mathcal{A} = \{\mathbf{T}, \mathbf{C}, \mathbf{D}, \mathbf{I}, \mathbf{A}\}$ that represents *thinking*, *clarify*, *decompose*, *identify*, and *abstract*, where each child node represents an intermediate

[2]Senior Ph.D. students in finance.

Formulator Agent Prompt

Context: ...liquidity and capital resources our objective ...
Table: | (In millions) | 2017 | 2016 | 2015 | ....
Question: what is the net change in cash during 2016?
Instruction: Please choose one of the action to solve the question *<ACTION>*,*<T: Thinking>*, *<C: Clarify>*,*<D: Decompose>* *<I: Identify>*, *<A: Abstract>* ....

Figure 4: Prompting template for Formulator agent.

step s_i produced by $M_{formulator}$. A candidate trajectory is designed by concatenating the question Q and a sequence of s_i

$$t = Q \oplus s_1 \oplus s_2 \oplus \cdots \oplus s_i, \quad s_i \in \mathcal{A}. \tag{2}$$

Given t, the formulator agent outputs the formula set $\mathcal{F}$ and corresponding variables $\mathcal{V}$ using the template prompt as shown in Fig 4:

$$\mathcal{F}, \mathcal{V} = M_{formula}\Big(t\Big). \tag{3}$$

In this agent, MCTS search uses a default rollout depth of 20, and results for other rollout settings are discussed in Section 5.3.

Resolver Agent. Given the variables $\mathcal{V} = \{V_1, V_2, \cdots\}$, the *Resolver* agent extracts the corresponding numeric value from the input context for each variable $V_i \in \mathcal{V}$. Specifically, given textual content E and table T, we first parse the input into a set of paragraphs $E = \{e_1, e_2, \cdots\}$ and tables tables $T = \{t_1, t_2, \cdots\}$. For each variable[3] $V \in \mathcal{V}$, we use BM25 to identify top-k most relevant contexts from $E \cup T$ for value extraction:

$$\mathrm{Top}_k(V) \;=\; \underset{C \in E \cup T}{\arg\text{top-k}}\; \mathrm{BM25}(C, V) \tag{4}$$

where each C represents either a paragraph or a table. Unlike prior work such as FinQA (Chen et al., 2021), which relies on retrievers based on pretrained models (e.g., using a BERT encoder and a binary relevance classifier), we adopt BM25 (Robertson et al., 2009) due to its greater efficiency performance, as discussed in Appendix D.2. Next, M_{resolver} is designed to generate m candidate values and then select the final value v^* for V with the highest confidence score, using the prompt template illustrated in Fig. 5:

$$\{v_i\}_{i=1}^{m} \sim M_{\text{resolver}}\big(\mathrm{Top}_k(V),\, V\big) \tag{5}$$

$$v^* \;=\; \arg\max_{v_i}\; P\big(v_i \mid \mathrm{Top}_k(V),\, V\big) \tag{6}$$

[3] We omit subscripts for clarify.

Resolver Agent Prompt

Variables: $\{\mathcal{V}\}$
Retrieved chunk: $\{\mathrm{BM25}(Top_k\ (\mathcal{V}))\}$
Instruction: Please identify the key variables and their corresponding values from the provided chunk

Figure 5: Prompting template for Resolver agent.

Executor Agent Prompt

Formula: $\{\mathcal{F}\}$
Variables: $\{\mathcal{V}\}$: $\{v^*\}$
Instruction: You are tasked with writing Python code based on the provided context.

Figure 6: Prompting template for Executor agent.

Executor Agent. To ensure accurate and transparent calculation, the *Executor* agent uses a code-based model M_{executor} to generate Python code that is subsequently executed by an interpreter, following the framework of program-aided language models (Gao et al., 2023b; Chen et al., 2022a). Given formulas $\mathcal{F}$ and extracted values for the variables $\mathcal{V}$, *Executor* generates code using the prompt template illustrated in Fig. 6:

$$\text{code} \;=\; M_{\text{executor}}\big(\mathcal{F},\, \{(V_i : v_i^*) | V_i \in \mathcal{V}\}\big). \tag{7}$$

This code is then run by a local interpreter and the final computed answer Ans with no execution errors occur:

$$Ans \;=\; \mathrm{Interpreter}(\text{code}) \tag{8}$$

Evaluator Agent with RAG. LLM-as-a-judge is widely used in many applications (Li et al., 2024a); however, smaller models can yield inaccurate results. To address this issue, we introduce an Evaluator agent $M_{\text{evaluator}}$ that verifies the outputs of each module with task-specific strategies. The Evaluator is cross-cutting and provides stage-wise checks for Formulator, Resolver, and Executor outputs. For the *Formulator*, a key step in FQA is selecting formulas grounded in fundamental financial concepts such as revenue, ratios, and expenses. We therefore augment this agent with external financial knowledge. Specifically, we collect 48 corporate-finance formulas from FinancialAnalyst[4] and convert them into LaTeX. Using retrieval-augmented generation (RAG) (Lewis et al., 2020), $M_{\text{evaluator}}$ assigns an unsupervised reward $Q(s_i|s_i \in \mathcal{A})$ to each candidate

[4] https://365financialanalyst.com/templates-and-models/corporate-finance-formulas-cfa-level-1/

Dataset	Train	Test	QL (Med/Avg)	CL (Med/Avg)	RT (Med/Avg)
BizBench					
FinCode	7	47	62.8 / 65	–	–
CodeFinQA	4,668	795	16.8 / 16.0	685.1 / 688.0	7.7 / 7.0
CodeTAT-QA	2,856	288	14.2 / 14.0	–	7.1 / 6.0
ConvFinQA (E)	–	629	7.8 / 6.0	781.3 / 766.0	8.6 / 8.0
TAT-QA (E)	–	120	11.1 / 11.0	299.2 / 239.0	10.3 / 9.5
SEC-Num	6,846	2,000	6.1 / 6.0	810.5 / 781.0	23.5 / 22.0
FinanceMATH	200	1,000	54.0 / 61.8	–	3.0 / 3.0

Table 1: Statistics of tasks in **BizBench** and **FinanceMATH**. QL = Question Length, CL = Context Length, RT = Rows per Table.

trajectory produced by MCTS. For the *Resolver*, a binary classifier ensures that extracted values are supported by the context. For the *Executor*, the Evaluator detects and flags common programming errors (e.g., syntax errors) via a local interpreter. The complete `FinMAN` workflow is summarized in Algorithm 1.

5 Experiments

5.1 Experimental Setup

Dataset. We conduct evaluations on two representative benchmarks: BizBench (Krumdick et al., 2024) and FinanceMATH (Zhao et al., 2024b), which require multi-step reasoning over both textual and tabular financial data across domains, including quantitative finance, accounting, and derivatives. For the *FQA* task, we use **FinCode**, **CodeFinQA**, and **CodeTAT-QA** from **BizBench**, as well as **FinanceMATH**. For the *Quantity Extraction* task, we use **ConvFinQA**, **TAT-QA**, and **SEC-Num** from **BizBench**, which focus on accurate extraction of numerical values from text and tables. We evaluate only our *Resolver* agent on this task. Detailed statistics and dataset descriptions are provided in Table 1.

Baselines. We compare our approach against a diverse set of large language models (LLMs), grouped into three categories: (1) **Closed-source models**, represented by GPTo3-mini and GPT-3.5; (2) **Open-source models**, including CodeLlama (Roziere et al., 2023), Qwen 2.5 (7B) (Yang et al., 2025), Llama-3 (Llama3-8B, Llama3.2-3B/1B) (Meta, 2024), Gemma2 (9B) (Team et al., 2024), and DeepSeek-R1 (a distilled version of Qwen2.5-7B) (Guo et al., 2025); and (3) **Fine-tuned models**, including FinMA (Xie et al., 2023) and Llama3-SFT, obtained by fine-tuning Llama3-8B with LoRA (Kojima et al., 2022). We evaluate these models using three standard prompting strategies: Chain-of-Thought (**CoT**) (Wei et al., 2022), Program-of-Thought (**PoT**) (Chen et al., 2022a), and In-Context Learning (**ICL**) (Brown et al., 2020), following the BizBench evaluation framework. Our `FinMAN` utilizes Llama3-8B for the *Formulator*, *Resolver*, and *Evaluator* agents, while CodeLlama-13B is designated for the *Executor*, with a decoding temperature set to 0.1. Additionally, we experimented with Deepseek-R1 7B as the backbone for all four agents to compare backbone effects. Full prompt templates are provided in Appendix G.

Metrics. Following previous studies (Krumdick et al., 2024; Zhao et al., 2024b), we report average test *accuracy* for both *FQA* and *Quantity Extraction*.

5.2 Overall Performance

Performance on FQA. Table 2 reports FQA results. Our `FinMAN` achieves strong performance. Against the best *closed-source* baseline (GPTo3-mini, taking the strongest prompting per dataset), `FinMAN-7B` (ICL) yields 4.26% on FinCode (61.70 vs. 57.44) and 4.31% on CodeFinQA (82.64 vs. 78.33). Compared with open-source LLMs, `FinMAN-7B` (ICL) surpasses Llama3-8B-ICL on CodeFinQA by 13.34% and DeepSeek-R1-7B-PoT on CodeTAT-QA by 23.96%; for completeness, the gains on FinCode and FinanceMATH over the best open-source baselines are 17.02% and 13.50%, respectively. Finally, relative to GPT-3.5 (175B), our 7B model attains higher accuracy on FinCode (+23.41% under CoT) and CodeFinQA (+13.94% under CoT) while using far fewer parameters.

FinanceMATH targets knowledge-intensive financial math reasoning across domains including quantitative finance and derivatives, requiring multi-step reasoning over text and tables. In this dataset, `FinMAN-7B` (ICL) attains 37.50, exceeding the strongest open-source baseline DeepSeek-R1-7B (ICL, 24.00) by 13.50 and the best fine-tuned model Llama3-SFT-8B (ICL, 15.50) by +22.00. For completeness, this aligns with the summary row in Table 2: *Impr. over open-source LLMs* = **13.50** and *Impr. over fine-tuned LLMs* = **22.00**. Relative to the best closed-source baseline GPTo3-mini (ICL, 32.50), `FinMAN-7B` (ICL) is 5.00 higher while using far fewer parameters (7B vs. ∼200B). These results indicate strong generalization of `FinMAN` to FinanceMATH's challenging, out-of-

Model	#Para	Method	FQA Tasks				Quantity Extraction Tasks		
			FinCode	CodeFinQA	CodeTAT-QA	FinanceMATH	ConvFinQA (E)	TAT-QA (E)	SEC-Num
Closed-source LLMs									
GPTo3-mini	~200B	CoT	57.44	72.55	73.33	29.50	80.66	86.66	70.33
		PoT	46.80	73.33	80.33	31.00	-	-	-
		ICL	51.06	78.33	71.33	32.50	83.66	85.66	72.66
GPT-3.5	175B	CoT	38.29	68.70	57.39	23.50	91.41	84.17	78.50
		PoT	31.91	49.13	46.09	23.00	-	-	-
		ICL	36.10	67.50	87.60	24.60	92.40	84.20	76.00
Open-source LLMs									
CodeLlama	13B	CoT	23.40	28.99	21.48	5.85	56.12	75.83	57.5
		PoT	29.78	15.99	19.42	4.87	-	-	-
		ICL	21.28	31.95	35.53	7.00	78.01	81.25	62.00
Gemma2	9B	CoT	31.91	60.38	49.31	12.00	75.68	84.16	34.55
		PoT	27.78	61.50	50.00	13.50	-	-	-
		ICL	36.17	62.51	53.12	13.00	77.26	83.33	33.60
Llama3	8B	CoT	32.98	66.23	38.77	14.00	78.17	85.42	71.60
		PoT	40.42	60.12	50.34	15.50	-	-	-
		ICL	36.17	69.30	52.43	17.50	84.26	82.50	62.40
DeepSeek-R1	7B	CoT	42.55	68.55	59.38	19.50	75.83	83.33	74.25
		PoT	42.55	64.44	61.80	20.50	-	-	-
		ICL	44.68	65.96	57.99	24.00	79.17	79.17	70.90
Qwen2.5	7B	CoT	27.66	45.79	45.83	18.50	87.17	85.83	33.05
		PoT	29.78	48.67	41.66	20.00	-	-	-
		ICL	31.91	50.56	48.61	19.50	89.18	84.16	32.05
Llama3.2	3B	CoT	29.78	42.39	20.83	17.50	74.88	68.33	40.00
		PoT	19.14	47.79	23.95	15.50	-	-	-
		ICL	28.72	52.70	22.57	5.70	80.02	60.56	69.58
Llama3.2	1B	CoT	17.02	23.97	13.19	5.50	61.04	40.00	49.30
		PoT	14.89	20.68	10.76	4.00	-	-	-
		ICL	19.14	24.34	11.45	4.50	63.43	48.33	47.05
Fine-tuned LLMs									
FinMA	7B	ICL	11.55	35.28	11.11	2.50	81.17	66.39	69.45
Llama3-SFT	8B	ICL	25.53	61.33	54.51	15.50	81.39	82.9	70.14
`FinMAN`	13B & 8B	CoT	42.55	75.59	45.48	20.00	**90.62**	91.66	75.10
		PoT	40.42	76.85	42.01	21.50	-	-	-
		ICL	46.63	78.16	43.72	25.50	88.93	**92.50**	79.40
`FinMAN`	7B	CoT	59.57	79.57	72.22	29.50	82.82	83.33	71.20
		PoT	55.31	81.81	75.34	33.50	-	-	-
		ICL	**61.70**	**82.64**	**85.76**	**37.50**	86.96	90.83	**81.35**
Impr. over open-source LLMs			↑17.02	↑13.34	↑23.96	↑13.50	↑1.44	↑9.17	↑7.10
Impr. over fine-tuned LLMs			↑36.17	↑21.31	↑31.25	↑22.00	↑9.23	↑9.60	↑11.21

Table 2: Performance of various LLMs on FQA and Quantity Extraction tasks. Red indicates the best results among closed-source LLMs. Blue and Green mark the best results on open-source and fine-tuned models, respectively. **Bold** denotes the best results among our `FinMAN`. Impr. and Impr. denote improvement over open-source and fine-tuned LLMs, respectively. Results on the Quantity Extraction task are reported only for our *Resolver* agent.

domain financial reasoning tasks.

Comparison with fine-tuned models. As shown in Table 2, our training-free `FinMAN-7B` (ICL) surpasses the strongest fine-tuned baseline on all four FQA datasets by 36.17, 21.31, 31.25, and 22.00 percentage points on FinCode, CodeFinQA, CodeTAT-QA, and FinanceMATH, respectively. In contrast, training domain-specific financial LLMs typically demands substantial compute; for example, BloombergGPT (50B) was trained on **700B** tokens for **1.3M GPU hours** (Wu et al., 2023).

Results on Quantity Extraction. Our *Resolver* achieves strong results on quantity extraction. On TAT-QA (E) and SEC-Num, `FinMAN` (ICL) reaches 92.50 and 81.35, which are $+8.3\%$ and $+5.35\%$ points above GPT-3.5 (84.20 and 78.50), respectively. Compared with *open-source* LLMs, `FinMAN-7B` (ICL) shows the reported summary gains of 1.44, 9.17, and 7.10 on ConvFinQA (E), TAT-QA (E), and SEC-Num, respectively, as listed in Table 2. Overall, `FinMAN` delivers the top results among *open-source* and *fine-tuned* baselines across all three QE datasets, while remaining competitive with closed-source models on ConvFinQA (E).

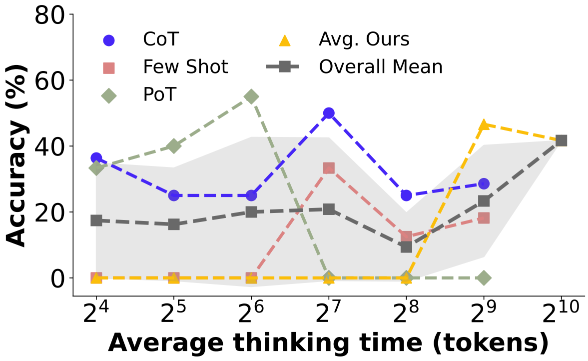

Figure 7: Test-time scaling on CodeFinQA with Llama3-8B.

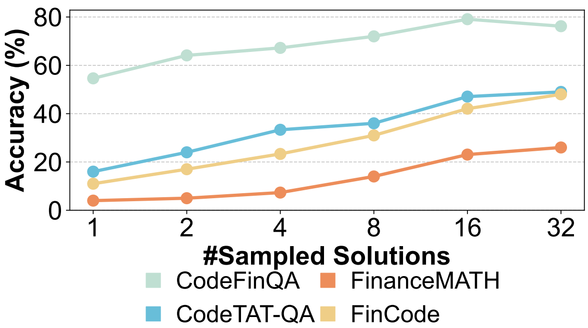

Figure 8: Performance comparison on the FQA dataset under different number of rollouts with FinMAN.

5.3 Further Analysis

Ablation Study. We examine each agent by disabling it in turn. *w/o Formulator*: skips MCTS and directly passes the original question and context to the Resolver. *w/o Resolver*: retains formulas and variable names from the Formulator but extracts numbers from the context using regular expressions. *w/o Executor*: does not generate or execute Python code; instead, the same LLM directly performs the calculation. *w/o Evaluator*: disables all error-checking and retry mechanisms.

Across BizBench, removing the *Formulator* results in the largest drops (e.g., -44.07% on **CodeFinQA**), confirming the centrality of formula planning. Turning off the *Evaluator* or *Executor* also leads to substantial declines (e.g., -31.64% and -18.75% on **CodeFinQA** and **CodeTAT-QA**, respectively), highlighting the importance of step-wise verification and code-based execution. Removing the *Resolver* consistently degrades performance across tasks (up to -14.62%). These results indicate that all four agents contribute meaningfully, with the *Formulator* having the largest impact. The *Evaluator* and *Executor* providing essential safeguards for extraction and computation.

Variants	CodeFinQA	FinanceMATH	CodeTAT-QA	FinCode
Ours	78.16 / –	25.50 / –	45.48 / –	46.63 / –
w/o Formulator	34.03 / ↓44.07	7.50 / ↓18.00	29.86 / ↓15.72	31.91 / ↓14.72
w/o Resolver	63.48 / ↓14.62	11.00 / ↓14.50	32.63 / ↓12.95	36.17 / ↓10.46
w/o Executor	68.06 / ↓10.04	17.50 / ↓8.00	26.73 / ↓18.75	38.29 / ↓9.34
w/o Evaluator	46.46 / ↓31.64	12.00 / ↓13.50	25.34 / ↓20.24	34.04 / ↓12.49

Table 3: Ablation study results. *w/o* indicates removing the corresponding agent.

Model	Acc. (%)	Time (s)
GPTo3-mini (∼200B)	32.50	14.72
GPT-3.5 (175B)	24.60	21.55
CodeLlama (13B)	7.00	20.11
Gemma2 (9B)	13.50	20.96
Llama3 (8B)	17.50	21.04
Qwen2.5 (7B)	20.00	19.26
DeepSeek-R1 (7B)	24.00	21.15
Llama3.2 (3B/1B)	17.50/5.50	15.66 / 15.04
FinMA (7B)	2.50	<u>0.63</u>
Llama3-SFT (8B)	15.50	28.76
FinMAN (7B)	**37.50**	18.34

Table 4: Accuracy and average time on FinanceMATH. **Bold** denotes the best result among all models; <u>underline</u> denotes the lowest inference time.

Test-time Scaling. To assess the performance of various prompting strategies, we adopt the approach from (Muennighoff et al., 2025), as shown in Figure 7. Specifically, when the token limit is reached, we insert an end of thought delimiter, optionally followed by "Final Answer:", to terminate reasoning and prompt the model to output its current best answer. Our findings indicate that the performance of both CoT and PoT improves with increased thinking time up to a point; however, when the thinking token exceeds 128, their performance declines, possibly because extended reasoning does not yield further benefits in the financial domain. In contrast, while the few-shot method peaks at 128 tokens, our `FinMAN` framework continues to benefit from additional test-time computation.

Effectiveness under Different Rollouts. The *Formulator* agent applies a rollout policy for expanding the MCTS tree, where increasing the number of rollouts generates more candidate solution trajectories at the expense of higher inference costs. Figure 8 compares the performance of various rollout counts in all FQA datasets. Our key observation is that the agent's performance generally improves with more rollouts, although the gains become marginal when the rollout reaches 32.

Time Efficiency. The experiment on runtime is shown in Table 4. Accuracy and mean inference time are reported for FinanceMATH dataset. Our approach achieves competitive results compared to GPTo3-mini and GPT-3.5 in FinanceMATH but with fewer parameters. Moreover, our framework achieves higher scores than other open-source models. In contrast, FinMA only produces numerical answers without intermediate reasoning steps, making it difficult to understand or analyze the cause of incorrect answers. In contrast, `FinMAN` supplies a transparent chain of intermediate reasoning steps, each accompanied by verification signals, enabling precise inspection. Details of our `FinMAN` runtime are provided in Appendix D.5

6 Conclusion

In this work, we propose FinMAN, a novel FQA method based on an autonomous agent framework designed to solve financial math problems. FinMAN offers a cost-effective solution by leveraging open-source LLMs while achieving performance comparable to that of large commercial models. Our multi-agent solution mimics human problem-solving by decomposing the FQA task into four specialized roles: `financial expert`, `data analyst`, `executor` and `evaluator`. In particular, to address challenging formulations, the designed *Formulator* agent that leverages external financial knowledge and employs MCTS to derive appropriate formulas. Experimental results demonstrate that `FinMAN` significantly enhances performance on both FQA and quantity extraction tasks. In future work, we plan to extend `FinMAN` to handle more complex financial scenarios and incorporate multi-modal data.

Limitations

Although FinMAN significantly improves the performance of small open-source models in FQA, our approach has several limitations. First, our work primarily focuses on FQA, which limits its applicability to broader financial tasks. Second, while MCTS plays a crucial role in enhancing reasoning performance, it is also the most time-consuming component of the framework. Finally, the framework's robustness across diverse, real-world financial scenarios remains to be fully validated. In future work, we plan to refine the `FinMAN` framework to handle more complex data and develop improved methods for capturing entity relations and ensuring robust performance.

References

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Qiguang Chen, Libo Qin, Jinhao Liu, Dengyun Peng, Jiannan Guan, Peng Wang, Mengkang Hu, Yuhang Zhou, Te Gao, and Wanxiang Che. 2025. Towards reasoning era: A survey of long chain-of-thought for reasoning large language models. *arXiv preprint arXiv:2503.09567*.

Qiguang Chen, Libo Qin, Jin Zhang, Zhi Chen, Xiao Xu, and Wanxiang Che. 2024a. M^3CoT: A novel benchmark for multi-domain multi-step multi-modal chain-of-thought. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8199–8221.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. 2022a. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*.

Yuemin Chen, Feifan Wu, Jingwei Wang, Hao Qian, Ziqi Liu, Zhiqiang Zhang, Jun Zhou, and Meng Wang. 2024b. Knowledge-augmented financial market analysis and report generation. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 1207–1217.

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. 2021. FinQA: A dataset of numerical reasoning over financial data. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3697–3711. Association for Computational Linguistics.

Zhiyu Chen, Shiyang Li, Charese Smiley, Zhiqiang Ma, Sameena Shah, and William Yang Wang. 2022b. ConvFinQA: Exploring the chain of numerical reasoning in conversational finance question answering. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 6279–6292, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Zihui Cheng, Qiguang Chen, Jin Zhang, Hao Fei, Xiaocheng Feng, Wanxiang Che, Min Li, and Libo Qin. 2025. Comt: A novel benchmark for chain of multi-modal thought on large vision-language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 23678–23686.

Ruomeng Ding, Chaoyun Zhang, Lu Wang, Yong Xu, Minghua Ma, Wei Zhang, Si Qin, Saravan Rajmohan, Qingwei Lin, and Dongmei Zhang. 2024. Everything of thoughts: Defying the law of penrose triangle for thought generation. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 1638–1662. Association for Computational Linguistics.

Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Baobao Chang, Xu Sun, Lei Li, and Zhifang Sui. 2024. A survey on in-context learning. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 1107–1128. Association for Computational Linguistics.

Hao Fei, Shengqiong Wu, Wei Ji, Hanwang Zhang, Meishan Zhang, Mong Li Lee, and Wynne Hsu. Video-of-thought: Step-by-step video reasoning from perception to cognition. In *Proceedings of the International Conference on Machine Learning, ICML*, pages 6373–6391.

Hao Fei, Shengqiong Wu, Hanwang Zhang, Tat-Seng Chua, and Shuicheng Yan. 2024a. Vitron: A unified pixel-level vision LLM for understanding, generating, segmenting, editing. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024*.

Hao Fei, Shengqiong Wu, Meishan Zhang, Min Zhang, Tat-Seng Chua, and Shuicheng Yan. 2024b. Enhancing video-language representations with structural spatio-temporal alignment. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Hao Fei, Yuan Zhou, Juncheng Li, Xiangtai Li, Qingshan Xu, Bobo Li, Shengqiong Wu, Yaoting Wang, Junbao Zhou, Jiahao Meng, et al. 2025. On path to multimodal generalist: General-level and general-bench. In *Proceedings of the International Conference on Machine Learning*.

Kaiyuan Gao, Sunan He, Zhenyu He, Jiacheng Lin, QiZhi Pei, Jie Shao, and Wei Zhang. 2023a. Examining user-friendly and open-sourced large gpt models: A survey on language, multimodal, and scientific gpt models. *arXiv preprint arXiv:2308.14149*.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023b. Pal: Program-aided language models. In *International Conference on Machine Learning*, pages 10764–10799. PMLR.

Jan Göpfert, Patrick Kuckertz, Jann M. Weinand, Leander Kotzur, and Detlef Stolten. 2022. Measurement extraction with natural language processing: A review. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 2191–2215. Association for Computational Linguistics.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Xuewen Han, Neng Wang, Shangkun Che, Hongyang Yang, Kunpeng Zhang, and Sean Xin Xu. 2024. Enhancing investment analysis: Optimizing ai-agent collaboration in financial research. In *Proceedings of the 5th ACM International Conference on AI in Finance*, pages 538–546.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. 2023a. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 8154–8173. Association for Computational Linguistics.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. 2023b. Self-specialization: Uncovering latent expertise within large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 8154–8173. Association for Computational Linguistics.

Hao Ju, Hu Zhang, and Zhedong Zheng. 2025. Anomalylmm: Bridging generative knowledge and discriminative retrieval for text-based person anomaly search. *arXiv preprint arXiv:2509.04376*.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213.

Michael Krumdick, Rik Koncel-Kedziorski, Viet Dac Lai, Varshini Reddy, Charles Lovering, and Chris Tanner. 2024. BizBench: A quantitative reasoning benchmark for business and finance. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8309–8332. Association for Computational Linguistics.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474.

Dawei Li, Bohan Jiang, Liangjie Huang, Alimohammad Beigi, Chengshuai Zhao, Zhen Tan, Amrita Bhattacharjee, Yuxuan Jiang, Canyu Chen, Tianhao Wu, et al. 2024a. From generation to judgment: Opportunities and challenges of llm-as-a-judge. *arXiv preprint arXiv:2411.16594*.

Yuetai Li, Xiang Yue, Zhangchen Xu, Fengqing Jiang, Luyao Niu, Bill Yuchen Lin, Bhaskar Ramasubramanian, and Radha Poovendran. 2025. Small models struggle to learn from strong reasoners. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 25366–25394. Association for Computational Linguistics.

Zhiwei Li, Ran Song, Caihong Sun, Wei Xu, Zhengtao Yu, and Ji-Rong Wen. 2024b. Can large language models mine interpretable financial factors more effectively? a neural-symbolic factor mining agent model. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 3891–3902.

Qian Liu, Hang Yu, Qiqi Wang, Qi Xu, Jinpeng Li, Zhuoqun Zou, Rui Mao, and Erik Cambria. 2025. Legal knowledge infusion for large language models: A survey. *Information Fusion*, page 103426.

Xinyi Liu, Chong Zhang, Mingyu Jin, Zhongmou Zhang, Zhenting Wang, Dong Shu, Suiyuan Zhu, Sujian Li, Mengnan Du, and Yongfeng Zhang. When ai meets finance (stockagent): Large language model-based stock trading in simulated real-world environments.

Zhuang Liu, Degen Huang, Kaiyu Huang, Zhuang Li, and Jun Zhao. 2021. Finbert: A pre-trained financial language representation model for financial text mining. In *Proceedings of the twenty-ninth international conference on international joint conferences on artificial intelligence*, pages 4513–4519.

AI Meta. 2024. Introducing llama 3.1: Our most capable models to date. *Meta AI Blog*, 12.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. 2025. s1: Simple test-time scaling. *arXiv preprint arXiv:2501.19393*.

R OpenAI. 2023. Gpt-4 technical report. arxiv 2303.08774. *View in Article*, 2(5).

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pages 1–22.

Karmvir Singh Phogat, Chetan Harsha, Sridhar Dasaratha, Shashishekar Ramakrishna, and Sai Akhil Puranam. 2023. Zero-shot question answering over financial documents using large language models. *arXiv preprint arXiv:2311.14722*.

Karmvir Singh Phogat, Sai Akhil Puranam, Sridhar Dasaratha, Chetan Harsha, and Shashishekar Ramakrishna. 2024. Fine-tuning smaller language models for question answering over financial documents. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 10528–10548. Association for Computational Linguistics.

Zhenting Qi, Mingyuan Ma, Jiahang Xu, Li Lyna Zhang, Fan Yang, and Mao Yang. 2024. Mutual reasoning makes smaller llms stronger problem-solvers. *arXiv preprint arXiv:2408.06195*.

Libo Qin, Qiguang Chen, Xiachong Feng, Yang Wu, Yongheng Zhang, Yinghui Li, Min Li, Wanxiang Che, and Philip S Yu. 2024. Large language models meet nlp: A survey. *arXiv preprint arXiv:2405.12819*.

Varshini Reddy, Rik Koncel-Kedziorski, Viet Dac Lai, Michael Krumdick, Charles Lovering, and Chris Tanner. 2024. Docfinqa: A long-context financial reasoning dataset. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics, 2024*, pages 445–458. Association for Computational Linguistics.

Stephen Robertson, Hugo Zaragoza, et al. 2009. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Pragya Srivastava, Manuj Malik, Vivek Gupta, Tanuja Ganu, and Dan Roth. Evaluating llms' mathematical reasoning in financial document question answering. In *Findings of the Association for Computational Linguistics, pages = 3853–3878, publisher = Association for Computational Linguistics, year = 2024,*.

Qi Sun, Kun Huang, Xiaocui Yang, Rong Tong, Kun Zhang, and Soujanya Poria. 2024. Consistency guided knowledge retrieval and denoising in llms for zero-shot document-level relation triplet extraction. In *Proceedings of the ACM Web Conference 2024*, pages 4407–4416.

Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. 2024. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*.

Adrian Theuma and Ehsan Shareghi. 2024. Equipping language models with tool use capability for tabular data analysis in finance. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2024 - Volume 2: Short Papers*, pages 90–103. Association for Computational Linguistics.

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

Gladys Tyen, Hassan Mansoor, Victor Carbune, Peter Chen, and Tony Mak. 2024. Llms cannot find reasoning errors, but can correct them given the error location. In *Findings of the Association for Computational Linguistics*, pages 13894–13908. Association for Computational Linguistics.

Yaoting Wang, Shengqiong Wu, Yuecheng Zhang, Shuicheng Yan, Ziwei Liu, Jiebo Luo, and Hao Fei. 2025. Multimodal chain-of-thought reasoning: A comprehensive survey. *arXiv preprint arXiv:2503.12605*.

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

Xiao Wei, Haoran Chen, Hang Yu, Hao Fei, and Qian Liu. 2024. Guided knowledge generation with language models for commonsense reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 1103–1136. Association for Computational Linguistics.

Shengqiong Wu, Hao Fei, Xiangtai Li, Jiayi Ji, Hanwang Zhang, Tat-Seng Chua, and Shuicheng Yan. 2025. Towards semantic equivalence of tokenization in multimodal LLM. In *The Thirteenth International Conference on Learning Representations*. OpenReview.net.

Shengqiong Wu, Hao Fei, Leigang Qu, Wei Ji, and Tat-Seng Chua. 2024. NExT-GPT: Any-to-any multimodal LLM. In *Proceedings of the International Conference on Machine Learning*, pages 53366–53397.

Shijie Wu, Ozan Irsoy, Steven Lu, Vadim Dabravolski, Mark Dredze, Sebastian Gehrmann, Prabhanjan Kambadur, David Rosenberg, and Gideon Mann. 2023. Bloomberggpt: A large language model for finance. *arXiv preprint arXiv:2303.17564*.

Qianqian Xie, Weiguang Han, Xiao Zhang, Yanzhao Lai, Min Peng, Alejandro Lopez-Lira, and Jimin Huang. 2023. Pixiu: A large language model, instruction data and evaluation benchmark for finance. *arXiv preprint arXiv:2306.05443*.

Jundong Xu, Hao Fei, Meng Luo, Qian Liu, Liangming Pan, William Yang Wang, Preslav Nakov, Mong-Li Lee, and Wynne Hsu. 2025. Aristotle: Mastering logical reasoning with a logic-complete decompose-search-resolve framework. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics.

An Yang, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoyan Huang, Jiandong Jiang, Jianhong Tu, Jianwei Zhang, Jingren Zhou, et al. 2025. Qwen2. 5-1m technical report. *arXiv preprint arXiv:2501.15383*.

Yi Yang, Yixuan Tang, and Kar Yan Tam. 2023. Investlm: A large language model for investment using financial domain instruction tuning. *arXiv preprint arXiv:2309.13064*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations ICLR*. OpenReview.net.

Junjie Ye, Xuanting Chen, Nuo Xu, Can Zu, Zekai Shao, Shichun Liu, Yuhan Cui, Zeyang Zhou, Chao Gong, Yang Shen, et al. 2023. A comprehensive capability analysis of gpt-3 and gpt-3.5 series models. *arXiv preprint arXiv:2303.10420*.

Yangyang Yu, Haohang Li, Zhi Chen, Yuechen Jiang, Yang Li, Denghui Zhang, Rong Liu, Jordan W Suchow, and Khaldoun Khashanah. 2024. Finmem: A performance-enhanced llm trading agent with layered memory and character design. In *Proceedings of the AAAI Symposium Series*, volume 3, pages 595–597.

Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. 2024a. Rest-mcts*: Llm self-training via process reward guided tree search. *Advances in Neural Information Processing Systems*, 37:64735–64772.

Wentao Zhang, Lingxuan Zhao, Haochong Xia, Shuo Sun, Jiaze Sun, Molei Qin, Xinyi Li, Yuqing Zhao, Yilei Zhao, Xinyu Cai, et al. 2024b. A multimodal foundation agent for financial trading: Tool-augmented, diversified, and generalist. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 4314–4325.

Yuxiang Zhang, Yuqi Yang, Jiangming Shu, Yuhang Wang, Jinlin Xiao, and Jitao Sang. 2024c. Openrft: Adapting reasoning foundation model for domain-specific tasks with reinforcement fine-tuning. *arXiv preprint arXiv:2412.16849*.

Yilun Zhao, Hongjun Liu, Yitao Long, Rui Zhang, Chen Zhao, and Arman Cohan. 2024a. Financemath: Knowledge-intensive math reasoning in finance domains. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12841–12858.

Yilun Zhao, Hongjun Liu, Yitao Long, Rui Zhang, Chen Zhao, and Arman Cohan. 2024b. Financemath: Knowledge-intensive math reasoning in finance domains. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12841–12858. Association for Computational Linguistics.

Fengbin Zhu, Ziyang Liu, Fuli Feng, Chao Wang, Moxin Li, and Tat Seng Chua. 2024. Tat-llm: A specialized language model for discrete reasoning over financial tabular and textual data. In *Proceedings of the 5th ACM International Conference on AI in Finance*, pages 310–318.

A Error cases

To qualitatively demonstrate the effectiveness of FinMAN, we present a case study along with error analyses in FQA, focusing on verification and mathematical calculations. Figure 9 compares three common error cases. As shown, Llama3-8B fails to perform accurate calculations (see Figure 9(c)) and its calculation process lacks verification. Additionally, Llama3-8B only extracts portions of the correct values from the provided context and generates incorrect financial formulas based solely on its internal knowledge, leading to incorrect answers in the financial domain.

Context:
... net revenue utility following is an analysis of the change in net revenue comparing 2015 to 2014. amount (in millions).

	Amount (In Millions)
2014 net revenue	5,735
Retail electric price	187
Volume/weather	95
Waterford 3 replacement steam generator provision	(32)
MISO deferral	(35)
Louisiana business combination customer credits	(107)
Other	(14)
2015 net revenue	5,829

Question:
what would 2015 net revenue have been in millions assuming there was no impact from both the retail electric price change and the impact of volume/weather in the year?

Ground Truth:
```
retail_electric_price = 187
volume_weather = 95
net_revenue_2015 = 5829 +
retail_electric_price + volume_weather
answer = net_revenue_2015
Answer:6111
```

Model Error Response:
```
2014_net_revenue = 5735
2015_net_revenue = 5829
retail_electric_price = 187
volume_weather = 95
2015_net_revenue_without_impact =
2015_net_revenue - (retail_electric_price +
volume_weather)  (wrong formula)
answer = 2015_net_revenue_without_impact
Answer: 5,732
```

(a) Error case 1: Problem-Solving Approach Error

Context:
{"Fiscal Years -- ": {"2019": 2019, "2018": 2018, "2017": 2017}, "Statements of Operations: -- Revenue": {"2019": 100, "2018": 100, "2017": 100}, "Statements of Operations: -- Cost of revenue": {"2019": 43, "2018": 50, "2017": 55}, "Statements of Operations: -- Gross profit": {"2019": 57, "2018": 50, "2017": 45}, "Operating expenses: -- Research and development": {"2019": 120, "2018": 79, "2017": 79}, "Operating expenses: -- Selling, general and administrative": {"2019": 86, "2018": 79, "2017": 81}, "Operating expenses: -- Loss from operations": {"2019": -149, "2018": -108, "2017": -115}, "Operating expenses: -- Interest expense": {"2019": -3, "2018": -1, "2017": -1}, "Operating expenses: -- Interest income and other expense, net": {"2019": 2, "2018": 1, "2017": "\u2014%"}, "Operating expenses: -- Loss before income taxes": {"2019": -150, "2018": -108, "2017": -116}, "Operating expenses: -- Provision for income taxes": {"2019": 1, "2018": 1, "2017": 1}, "Operating expenses: -- Net loss": {"2019": -151, "2018": -109, "2017": -117}}

Question:
What is the total proportion of cost of revenue as a percentage of revenue in 2017 and 2018?

Ground Truth:
```
revenve_cost_2017 = df["Statements of Operations:
-- Cost of revenue"]["2017"]
revenve_cost_2018 = df["Statements of Operations:
-- Cost of revenue"]["2018"]
answer = (revenve_cost_2017 + revenve_cost_2018)
Ans:105
```

Model Error Response:
```
cost_of_revenue_2017 = df["Statements of
Operations: -- Cost of revenue"]["2017"] = 55
revenue_2017 = df[“Statements of Operations: --
Revenue”][“2017”] = 100 (unrelated extract)
cost_of_revenue_2018 = df["Statements of
Operations: -- Cost of revenue"]["2018"] = 55
(wrong extract)
revenue_2018 = df["Statements of Operations: --
Revenue"]["2018"] = 100 (unrelated extract)
Answer: 55.00
```

(b) Error case 2: Extraction Error

Context:
... results of operations consolidated sales and earnings.

($ in millions)	2011	2010	2009
Net sales	8,630.9	7,630.0	6,710.4
Net earnings attributable to Ball Corporation	444.0	468.0	387.9

The increase in net sales in 2011 compared to 2010 was driven largely by the increase in demand for metal packaging in the prc …

Question: *what were average net sales in millions for the three years ending in 2011?*

Ground Truth:
```
table_row = [8630.9, 7630.0, 6710.4]
# row labeled net sales
a = sum(table_row) / len(table_row)
Answer: 7657.1000976562
```

Model Error Response:
```
average_net_sales = ( 8630.9 + 7630.0 +
6710.4 ) / 3
answer = average_net_sales
Answer: 7336.4333333 (wrong calculation)
```

(c) Error case 3: Calculation Error

Figure 9: Error case study.

B FinMAN Algorithm

C Error Distribution

Figure 10 presents the distribution of four major error types, formula, extraction, calculation, and other, across seven small open-source LLMs on the CodeFinQA dataset. Each horizontal bar indicates the percentage of questions for which a given model committed a specific error. Calculation errors are the most frequent for most models, confirming that multi-step numerical reasoning remains the primary bottleneck. Extraction errors are the second largest category, suggesting room for improvement in value retrieval. Formula selection mistakes are comparatively lower but still significant, especially pronounced in the CodeLlama models, highlighting the difficulty of choosing the correct financial equation. Other errors account for a small fraction of failures. This breakdown motivates our design of specialized agents in `FinMAN` to target the three dominant error sources: calculation, formula generation, and extraction.

Algorithm 1 FinMAN Framework

Require: Q, E, T ▷ Question, context, tables
Ensure: Ans ▷ Final answer
init $\mathcal{K}$
$\mathcal{A} \leftarrow \{T,\ C,\ D,\ I,\ A\}$
while optimal action sequence not found **do**
  $s_n \leftarrow M_{\text{formula}}(\text{choose from } \mathcal{A})$ ▷ MCTS
  $t \leftarrow Q \oplus s_1 \oplus \cdots \oplus s_n$
  reward $\leftarrow M_{\text{evaluate}}(t, \mathcal{K})$
  if reward indicates optimal sequence **then**
    break ▷ Optimal sequence found
  else
    update action ▷ Rollback candidate
  end if
end while
$\mathcal{F}, \mathcal{V} \leftarrow$ final formula & vars
for each $v_i \in \mathcal{V}$ **do** ▷ Value Extraction
  $\text{Top}_k \leftarrow \text{BM25}(E \cup T, v_i)$
  c $\leftarrow M_{\text{resolve}}(\text{Top}_k)$
  $v^* \leftarrow \arg\max_c P(c \mid \text{context})$
  if $\text{M}_{\text{evaluate}}(v^*, E \cup T)$ returns false **then**
    reExtract()
  end if
end for
code $\leftarrow M_{\text{execute}}(\mathcal{F}, \{v^*\})$ ▷ Code Execution
$(a_n,\ err) \leftarrow$ Interpreter(code)
if M_{evaluate}(err) detects error **then**
  commentAndReRun()
end if
return Ans

D Experiment Setup

D.1 Implementation Details

For all LLMs, the decoding temperature is set to 0.1 to ensure deterministic generation. To maintain the validity of our comparisons, we replicate the existing results reported in the literature and other

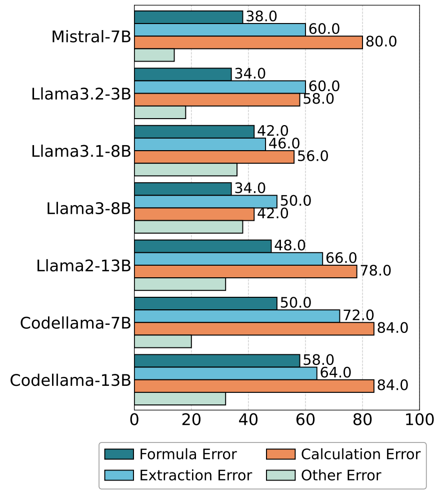

Figure 10: Error distribution in CodeFinQA

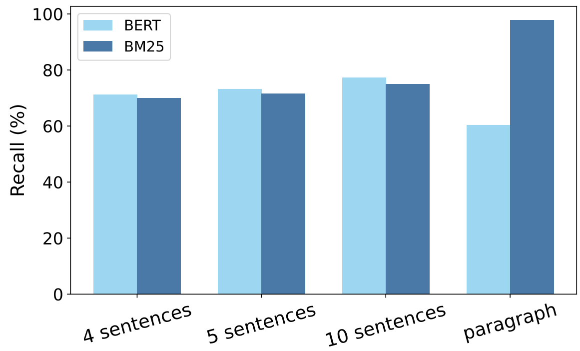

Figure 11: The recall rate of BM25 and BERT on SEC-NUM dataset.

baselines using the same experimental settings. All experiments are implemented using PyTorch and Huggingface libraries. All training and evaluation procedures are conducted on two NVIDIA A100 80GB GPUs.

D.2 Evaluating BERT and BM25 in Resolvers

We report our findings on the SEC-NUM dataset by comparing BERT and BM25, as shown in Figure 11. In this experiment, we split the input table into either sentences or paragraphs and evaluate recall performance. We observe that there is no significant difference in recall when retrieving the top 4, 5, or 10 sentences; both BERT and BM25 perform similarly in these settings. However, when using paragraph-level retrieval, BM25 significantly outperforms BERT.

D.3 RAG

For the RAG, we use the all-MiniLM-L6-v2 model as our embedding model. The *Evaluator* agent has the option to retrieve a financial formula; if it does not, it functions as a financial domain expert by providing feedback to the Formulator agent based solely on its internal knowledge. Conversely, if retrieval is chosen, this indicates that internal knowledge alone may be insufficient, prompting the incorporation of external knowledge for evaluation.

D.4 Comparison of SFT and RAG

Following OpenRFT (Zhang et al., 2024c), we constructed an SFT dataset consisting of 412 instruction data points and 100 sample data points with reasoning steps from CodeFinQA. The instructions and reasoning data were generated by GPT-o1 and subsequently verified by Ph.D. students in Finance.

By contrast, the knowledge bank required far less effort: we simply collected formulas from a publicly available website and converted them into LaTeX format without the need for in-depth validation or labeling by domain experts. This process involved minimal additional human effort and computational resources compared to the creation of the SFT dataset, which required both data generation and expert verification.

D.5 Time Efficiency

In terms of runtime, multi-agent coordination in `FinMAN` averages 21.96 seconds per question. The most time-intensive component is the MCTS within the *Formulator* agent, which explores the action space for optimal solutions, averaging 16.37 seconds. The verification process, which validates the outputs of the *Formulator* and other agents, requires approximately 1.65 seconds.

E Case Study

To effectively showcase the efficiency of FinMAN, we illustrate a case study in Figure 12 involving its application in FQA, specifically concentrating on verification and mathematical computations. After one iteration of reasoning, the Formulator agent produces an output that includes its thinking trajectory and the selected action sequence. The *Evaluator* agent then evaluates this output by its local knowledge base if the reasoning steps align with the basic formulas. For example, when a question is straightforward, the *Evaluator* can rely on existing knowledge and pertinent formulas to assign a high score to the *Formulator's* action, such as directly abstracting the formula. Conversely, for more challenging questions, the *Evaluator* may assign a low score to the*Formulator's* action, prompting further reasoning based on fundamental financial components.

F Comparison to Previous Work

To clarify the differences between FinMAN and other reasoning methods in FQA, we aim to address the following two questions:

What are the benefits of designing an autonomous agent in FQA? FQA is a domain-specific task that involves multiple steps. Previous approaches relying on LLMs' internal knowledge (e.g., CoT (Wei et al., 2022), PoT (Chen et al., 2022a), or in-context learning (Dong et al., 2024)) are limited by inherent model constraints. In contrast, our multi-agent framework extends LLM capabilities by incorporating a local knowledge bank and a code interpreter. Moreover, existing financial QA solutions often overlook fundamental financial concepts or rely heavily on training data (Zhu et al., 2024; Hao et al., 2023b). FinMAN overcomes these issues by implementing a multi-step verification process during reasoning and continuously updating its local knowledge bank, resulting in a more robust and adaptable solution.

Can smaller language models perform well in challenging FQA? Existing methods (Phogat et al., 2023) have heavily relied on powerful close-source language models, such as GPT-4, which lead to high computational costs and impose significant limitations on practical deployment. Furthermore, reliance on closed-source models can hinder error analysis, making it difficult to diagnose and improve performance. Our work demonstrates that by integrating autonomous agent capabilities and constructing a local knowledge bank, smaller language models can effectively address FQA tasks through clearly defined, step-wise verification. This approach significantly reduces the dependence on closed-source LLMs.

G Agent Prompt

This section describes how each specialized agent in the system is instructed and guided to perform its unique role. In particular, it outlines the tasks each agent must accomplish, the context in which those tasks occur, and the rules that govern the agent's interactions. By defining these prompts clearly, the section ensures that every agent, whether it is responsible for chunking input, extracting variables, writing or verifying code, or any other specialized function, works cohesively within the workflow.

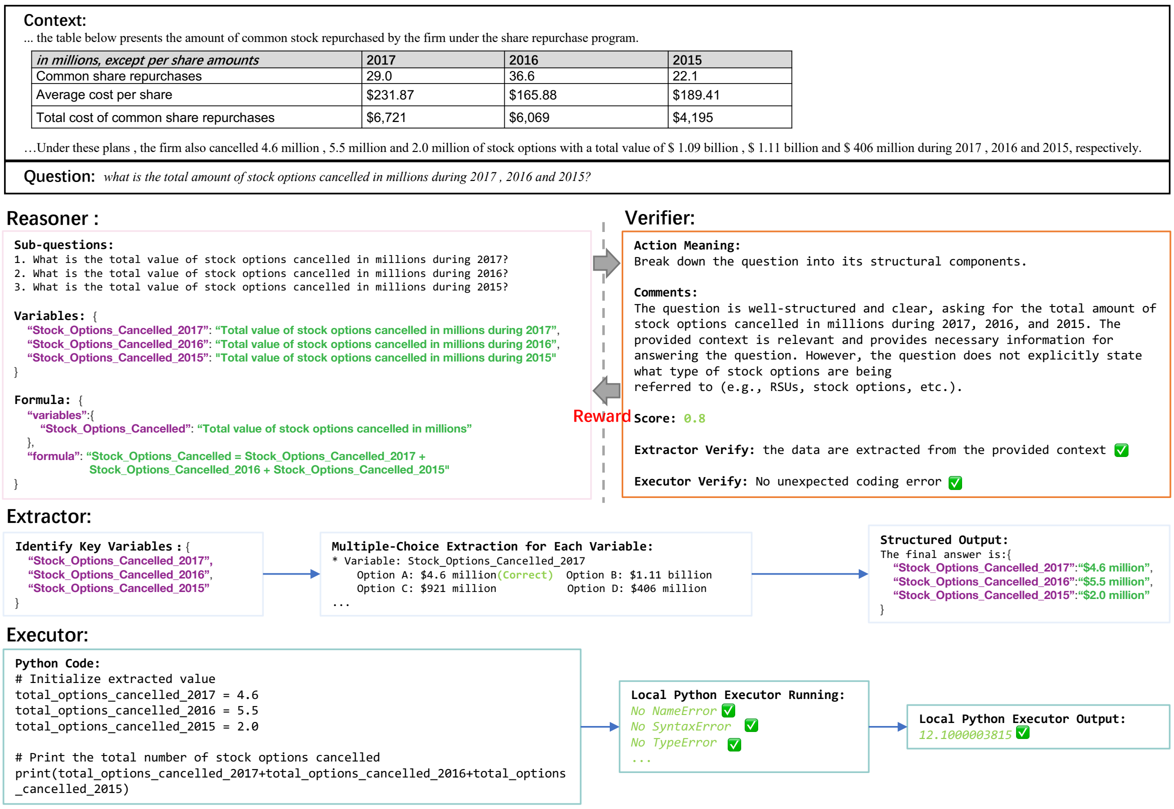

Figure 12: FinMAN case study

```
You are a Chartered Financial Analyst (CFA) expert.

Your task is to:
1. Generate relevant financial formulas.
2. Identify key variables based on the user's question and the provided context.
3. Choose one of the action to answer the question {ACTION SPACE}
**Constraints:**
- **Do not extract any numerical values** from the context.
- **Do not perform any calculations** in this step.
- **Ensure all formulas are based on the identified variables** and the user's question.

**Example:**
{
  "variables": {
    "Net_Revenue": "Total income generated from sales after deductions",
    "Gross_Revenue": "Total income generated from sales before any deductions",
    "Discounts": "Total discounts given to customers",
    "Returns": "Total value of returned goods",
    "Commissions": "Total commissions paid to sales personnel"
  },
  "formula": "Net_Revenue = Gross_Revenue - Discounts - Returns - Commissions"
}
**Notes:**
- Always keep the user's question in mind when generating formulas.
- Ensure that each formula directly relates to the identified variables.
- Output your answer in JSON format.
```

Figure 13: System prompt of Formulator agent to write formula and variables.

```
***Context:***
{INSERT_CHUNKS}

──────────────────────────────
Step 1: Identify Key Variables
──────────────────────────────
- From the text and tables, identify the numerical variables that are essential for the final calculation. For
instance, these might be values like revenue figures or impact amounts.
- Use the context provided by the text (keywords and surrounding phrases) to determine which numbers
correspond to each variable.
***Variables:***
{INSERT_VARIABLES}
──────────────────────────────
Step 2: Multiple-Choice Extraction for Each Variable
──────────────────────────────
For each key variable, do the following:
1. Generate four multiple-choice options labeled A, B, C, and D. One of these options must be the correct
value extracted from the text; the other three should be plausible distractors.
2. Briefly explain your reasoning (chain-of-thought) for selecting the correct option.

**Example:**

Suppose you encounter the following text snippet:

"According to the latest report, in 2011 the net re
venue reached $2,045 million, and in 2012 it was $1,854 million. Additionally, the report noted a $33 million
impact from nuclear volume changes."

For each variable, you might generate:

- **Variable: Net Revenue 2011**
  - Option A: $2,045 million  *(Correct)*
  - Option B: $1,854 million
  - Option C: $2,000 million
  - Option D: $2,100 million
  - **Explanation:** The phrase "in 2011 the net revenue reached" directly indicates that $2,045 million is
the correct value.

──────────────────────────────
Step 3: Structured Output
──────────────────────────────
Present your final answer in a structured format (e.g., JSON). Your output should include:
- The correct value chosen for each key variable.
- The computed results (like the revenue decrease and percentage).
- A brief summary of your reasoning for each step.

**For Example, the final output could be structured as:**

{{
  "net_revenue_2011": "$2,045 million",
  "net_revenue_2012": "$1,854 million",
  "nuclear_volume_effect": "$33 million",
  "net_revenue_decrease": "$191 million",
  "percentage_nuclear_volume": "17.3%"
}}

──────────────────────────────
Final Instructions:
──────────────────────────────
1. Confirm that you understand these instructions.
2. When processing any given input, first break it into paragraphs and tables.
3. Identify the key variables using contextual clues.
4. For each variable, create four multiple-choice options, select the correct one with a brief explanation,
and then use these values for your final computations.
5. Present your final answer in the structured format shown above.
```

Figure 14: Prompt of Resolver agent to extract value from chunk.

```
You are tasked with writing Python code based on the provided context. Follow these guidelines to
ensure the code is accurate, efficient, and free from common mistakes:

1. **Understand the Context:**
   - Carefully read and comprehend the provided context to grasp the requirements and objectives of the
code.

2. **Code Structure and Best Practices:**
   - Write clean, well-structured, and readable code.
   - Follow Python's best practices and PEP 8 style guidelines.
   - Use meaningful variable and function names that reflect their purposes.

3. **Avoid Common Mistakes:**
   - Ensure there are no syntax errors or logical flaws.
   - Optimize the code for performance without sacrificing readability.

4. **Output Format:**
   - Present the complete Python code without additional explanations or markdown formatting.
   - Ensure that the code is ready to run and doesn't require further modifications
**Instructions:**
- Based on the above context, write the required Python code adhering to all the guidelines mentioned.
- Do not include any explanations, just provide the Python code.

**Few-shot Example:**
```python
# initialize variables
net_interest_revenue_2009 = 896

# initialize variables
total_operating_expenses_2009 = 3173

# Final answer: percent of net interest revenue where total operating expenses in 2009
percent_2009 = net_interest_revenue_2009 / total_operating_expenses_2009

# Get the final answer
answer = percent_2009 * 100

# Print the total number of stock options cancelled
print(answer)
```
```

Figure 15: System prompt of Executor agent to write Python code.

```
    You need to evaluate the following action and provide a score based on its effectiveness
and correctness. \n
        Question: {CURRENT_QUESTION}
        Context: {CURRENT_CONTEXT}
        Action: {ACTION}
        **Action Meaning**: {ACTION_MEANING}
        **Provide your response as a JSON object with two keys:**
        - **"comments"**: A string containing your review comments.
        - **"score"**: A numerical value between 0 and 1, where 1 indicates full approval and 0
indicates disapproval.
```

Figure 16: Prompt of Evaluator agent to verify actions.

```
ACTION SPACE:

"REASON_ACTION_ClAIFY": "Clarify the question to ensure understanding.”

"REASON_ACTION_QUESTION_STRUCTURE": "Break down the question into its structural components.”

"REASON_ACTION_IDENTIFY_VAR": "Identify variables and its meaning involved in the question.”

"REASON_ACTION_THINKING_ONE_MORE": "Think through the relationships between the variables.”

"REASON_ACTION_DERIVE_ABSTRACT": "Derive an abstract formula or method to solve the
question."
```

Figure 17: Prompt of action space