FinanceReasoning: Make Financial Numerical Reasoning More Credible, Comprehensive, and Challenging

Anonymous ACL submission

Abstract

We introduce **FinanceReasoning**, a novel benchmark designed to evaluate the reasoning capabilities of large reasoning models (LRMs) in financial numerical reasoning problems. Compared to existing benchmarks, our work provides three key advancements. (1) **Credibility**: We update 15.6% of the questions from four public datasets, annotating 908 new questions with detailed Python solutions and rigorously refining evaluation standards. This enables an accurate assessment of the reasoning improvements of LRMs. (2) **Comprehensiveness**: FinanceReasoning covers 67.8% of financial concepts and formulas, significantly surpassing existing datasets. Additionally, we construct 3,133 Python-formatted functions, which enhances LRMs' financial reasoning capabilities through refined knowledge (*e.g.,* 83.2% → 91.6% for GPT-4o). (3) **Challenge**: Models are required to apply multiple financial formulas for precise numerical reasoning on 238 *Hard* problems. The best-performing model (*i.e.,* OpenAI-o1 with PoT) achieves 89.1% accuracy, yet LRMs still face challenges in numerical precision. We demonstrate that combining Reasoner and Programmer models can effectively enhance LRMs' performance (*e.g.,* 83.2% → 87.8% for DeepSeek-R1). Our work paves the way for future research on evaluating and improving LRMs in domain-specific complex reasoning tasks. Our code and datasets are publicly available[1].

Dataset Difficulty Distribution and Knowledge Coverage

Dataset	Easy	Medium	Hard	Knowledge Coverage
CodeFinQA (795)	559	239	6	45.7%
CodeTAT-QA (288)	189	99	0	14.66%
FinCode (47)	6	27	14	15.46%
FinanceMath (200)	34	104	62	18.89%
FinanceReasoning (2238)	**1000**	**1000**	238	**67.76%**

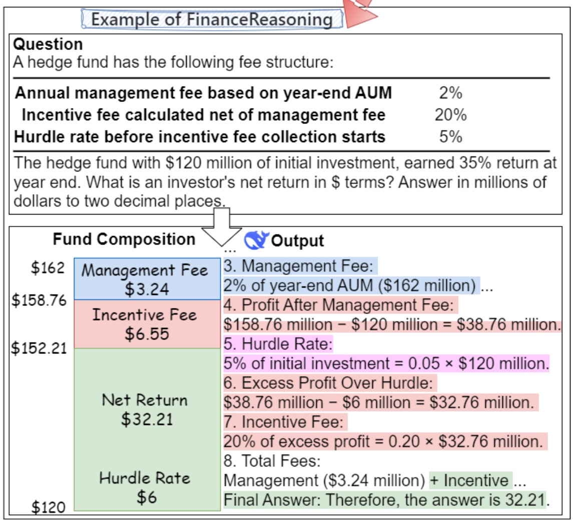

Figure 1: Statistics and an example of FinanceReasoning. The Knowledge Coverage is calculated as the proportion of financial calculations involved in the questions relative to the financial encyclopedia. To address the given problem, LRMs are required to first select appropriate financial formulas based on the given conditions (*e.g.,* hurdle rate) and perform step-by-step precise numerical computations with rounding requirements.

1 Introduction

Recently, combined with train-time scaling and test-time scaling (Kaplan et al., 2020; OpenAI, 2024b), large language models (LLMs) have exhibited remarkable reasoning capabilities (Xu et al., 2025), through a long reasoning process and effective reasoning strategies. These reasoning-enhanced models (*i.e.,* Large reasoning models (LRMs)) (OpenAI, 2024d,c, 2025; Guo et al., 2025; Team, 2024; Team et al., 2025; Gemini, 2025), are able to tackle complex tasks that require multi-step reasoning, such as code (Jain et al., 2024; Chen et al., 2021a), math (Mao et al., 2024; Lightman et al., 2023), and science (Lu et al., 2024; Yue et al., 2024; Wang et al., 2024).

However, as illustrated in Figure 1, more real-world domain-specific numerical reasoning tasks (*e.g.,* financial quantitative analysis) challenge LRMs to deeply understand and apply domain-

[1]Anonymous Github Code: https://anonymous.4open.science/r/finance-reasoning-CDB9

(a) Re-annotation				
Dataset	Subset	Answer Corrections	Question Disambiguations	Total
CodeFinQA	Test (795)	55 (6.9%)	58 (7.3%)	113 (14.2%)
CodeTAT-QA	Test (288)	19 (6.6%)	9 (3.2%)	28 (9.7%)
FinCode	Test (47)	6 (12.8%)	1 (2.1%)	7 (14.9%)
FinanceMATH	Val (200)	15 (7.5%)	45 (22.5%)	60 (30.0%)
(b) Re-evaluation				
Dataset	Deepseek-V3	Deepseek-R1	Δ (R1 - V3)	Growth
CodeFinQA-Silver	61.76	60.88	-0.88	-1.42%
CodeFinQA-Golden	85.41	87.42	2.01	2.35%
CodeTAT-QA-Silver	89.24	89.58	0.34	0.38%
CodeTAT-QA-Golden	91.67	93.75	2.08	2.27%
FinCode-Silver	80.85	82.98	2.13	2.63%
FinCode-Golden	87.72	95.74	8.02	9.14%
FinanceMATH-Silver	58.50	71.00	12.50	21.37%
FinanceMATH-Golden	59.50	83.50	24.00	40.34%

Table 1: (a) **Re-annotation**: For the test or validation subsets of four datasets, the proportion of updated examples ranges from 9.7% to 30%. (b) **Re-evaluation**: Silver denotes the Acc on the original dataset, while Golden represents the results on the re-annotated dataset. The results demonstrate that only after rigorous revised can the true performance of LRMs and the significant improvement of DeepSeek-R1 over DeepSeek-V3 be revealed.

specific knowledge, and perform intricate mathematical calculations based on hybrid contexts such as table and text (Plaat et al., 2024; Chen et al., 2023c; Wang and Zhao, 2024; Romera-Paredes et al., 2024). Specifically, in the high-stakes financial domain, where precision and transparent reasoning are paramount (Krumdick et al., 2024), the reasoning capabilities of LRMs must be further validated and accurately assessed. Existing numerical reasoning benchmarks for finance are limited in their notation quality, coverage of specific knowledge in the financial domain, and complexity of reasoning (Chen et al., 2021b, 2022; Zhu et al., 2021; Zhao et al., 2024; Krumdick et al., 2024). As illustrated in Table 1, DeepSeek-R1 have achieved greater accuracy 90% in easier datasets and are saturated due to annotation quality, making it difficult to objectively evaluate their actual reasoning capabilities and analyze their shortcomings.

Therefore, we propose **FinanceReasoning**, a credible, comprehensive, and challenging financial numerical reasoning benchmark to evaluate the reasoning capabilities of LRMs in the finance domain. The dataset comprises a total of 2,238 problems covering diverse financial knowledge, of which 1,420 problems have been reviewed and revised based on public datasets, while 908 problems were automatically generated by LLM (*i.e.,* GPT-4o) and subsequently annotated by experts. Each problem includes hybrid contexts, unambiguous questions, Python-formatted solutions, and precise answers, providing a reliable reference for accurately evaluating the complex numerical reasoning capabilities of LRMs. Additionally, we have collected and open-sourced a comprehensive financial function library containing 3,133 Python-formatted functions. Each function includes precise functional descriptions, parameter explanations, and step-by-step implementation code, offering a high-quality structured knowledge base to automatically build domain-specific reasoning problems and enhance LLMs' domain-specific reasoning capabilities through knowledge retrieval.

We evaluate six current open-source and proprietary LRMs (OpenAI, 2024d,c, 2025; Guo et al., 2025; Team, 2024; Gemini, 2025), using Chain-of-Thought (CoT) (Wei et al., 2022) and Program-of-Thought (PoT) (Chen et al., 2023b). Compared with traditional LLM without post-training, we also evaluate seven LLMs (Gemini, 2025; OpenAI, 2024a; Anthropic, 2024; DeepSeek-AI et al., 2024; AI@Meta, 2024b,a; Qwen et al., 2025).

Our experimental results demonstrate that the powerful LRM (*i.e.,* OpenAI-o1) with PoT achieves the best performance, with an accuracy of 89.1% on *Hard* subset, significantly outperforming other LLMs. However, current LRMs still faced incorrect formula application and imprecise numerical calculation on challenging domain-specific reasoning problems. Next, we explore various knowledge augmentation methods and combinations of models. Experiments demonstrate that integrating structured, refined reasoning knowledge and enabling model collaboration can further enhance the complex reasoning capabilities of LRMs.

Our contributions are summarized below:

- We propose FinanceReasoning, a credible financial numerical reasoning benchmark constructed from re-annotated public datasets and newly collected challenging data through Human-AI collaboration, demonstrating the superior reasoning capabilities of LRMs.

- We construct and open-source a comprehensive financial function library containing 3,133 Python-formatted functions, demonstrating the effectiveness of refined knowledge augmentation in enhancing domain-specific reasoning.
- We analyze the shortcomings of LRMs and propose a combination of **Reasoner** and **Programmer** models, effectively enhancing their performance on complex mathematical calculations.

2 FinanceReasoning Benchmark

In this section, we first update existing datasets like BizBench (Krumdick et al., 2024) and FinanceMATH (Zhao et al., 2024), addressing issues such as disambiguation and corrections. We then construct a financial function library by extracting articles. Expert annotators are guided to review and revise the model-generated problem.

2.1 Updates to Public Datasets

Following prior work (Krumdick et al., 2024; Zhao et al., 2024), we retain the format of questions with optional hybrid contexts as input, accompanied by Python-formatted solutions and program-executed numerical results. Due to the specialized nature of complex financial problems and the high cost of expert annotation, we observe certain limitations in existing datasets (Chen et al., 2021c; Zhu et al., 2021; Zhao et al., 2024), including ambiguous questions, oversimplified processes, incorrect answers (*e.g.,* the phrasing "the range of" confuses the LLMs, as it is not clear whether to output a range like 70.18-81.05 or a specific difference of 10.87), and lenient evaluation criteria. Statistical details of these issues are presented in Table 1.

Specifically, we perform updates on the test sets of CodeFinQA, CodeTAT-QA, and FinCode test sets (Krumdick et al., 2024), as well as the validation set of FinanceMATH (Zhao et al., 2024). The annotators are instructed to examine each example and perform three types of **Update Actions**: *disambiguation*, *elaboration*, and *correction*.

- **Disambiguation**: For problems that are unsolvable due to insufficient contextual conditions or unclear target results, minimally modifies the question to eliminate potential ambiguities.

Financial Function

```python
def calc_net_return(init_investment: float,
     growth: float, fee_rate: float, inc_rate:
     float, hurdle: float) -> float:
    """
    Calculate the net return for an investor in
    a hedge fund given various parameters.

    Args:
    initial_investment (float): The initial
    amount invested in the hedge fund...

    Returns:
    net_return (float): The net return for the
    investor after fees, in millions.
    """
    end_value = init_investment * (1 + growth)
    fee = end_value * fee_rate
    net_value = end_value - fee
    hurdle_value = init_investment * (1 + hurdle
        )
    inc_fee = max(0, (net_value - hurdle_value)
    * inc_rate)
    net_return = end_value - (fee + inc_fee) -
    init_investment
    return round(net_return, 2)
```

Figure 2: An example of financial function from the constructed function library to calculate net return.

- **Elaboration**: For Python programs with missing or skipped computational steps, supplement the code and add detailed annotations.
- **Correction**: For problems with incorrect ground truth, the solution and answer are revised.

Additionally, existing evaluation standards are relatively lenient: BizBench allows an error margin 1% (Krumdick et al., 2024), while FinanceMATH disregards units and signs (Zhao et al., 2024). We refine the evaluation criteria by specifying units, percentage formats, signs, and decimal places, and strictly enforce a 0.2% error margin, enhancing the rigor, challenge, and relevance to real-world scenarios. Detailed examples are illustrated in the Appendix C.1.

2.2 Function Library Construction

For LRMs, the challenge lies not in extracting numerical values from relevant texts but in applying domain-specific knowledge to perform complex multi-step numerical computations (Plaat et al., 2024; Chen et al., 2023c). Although LLMs have already acquired a solid understanding of conceptual knowledge in the financial domain, to further refine reasoning capabilities, we collect and annotate a financial function library comprising 3,133 structured Python functions for financial calculations, aimed at improving models' reasoning knowledge.

We begin by collecting 6,138 financial encyclopedia articles from Investopedia, a platform renowned for its extensive expertise in financial knowledge[2]. Each article provides a detailed introduction to a specific financial term, covering fundamental concepts, application scenarios, and potential limitations, some including relevant calculation formulas and practical examples. To distill dense, structured financial reasoning knowledge while reducing annotation costs, we instruct GPT-4o to extract potential financial calculation functions from each article according to a predefined format, as illustrated in Appendix D.1. Each function is required to include a semantically meaningful signature, a concise and clear docstring (functionality, parameters, return values, applicable constraints, and other notes), and step-by-step implementation code with appropriate annotations. Finally, we organize financial experts to rigorously review and revise the generated functions, ensuring their professional expression and logical correctness.

2.3 Expansion of Data Annotation

Existing financial question-answering datasets (*e.g.,* CodeFinQA, CodeTAT-QA) focus primarily on evaluating models' basic concept understanding, precise numerical extraction, and simple calculation abilities within given contexts. Problem-solving processes in these datasets typically involve fewer reasoning steps (*e.g.,* calculate the difference in net profit over two years). These datasets often suffer from redundancy in simple questions and a lack of complex questions, failing to adequately assess the reasoning capabilities of LRMs, such as knowledge application, constraint emphasis and long thought (*e.g.,* compute the net return rate of a fund in Figure 1). As a result, the true reasoning capabilities of LRMs cannot be evaluated comprehensively and objectively. Therefore, optimizing data construction methods, rigorously verifying data quality, and building more challenging datasets have become crucial to improve the evaluation of financial reasoning tasks.

During the data expansion process, we leveraged the structured financial functions to guide GPT-4o in generating new financial numerical reasoning problems and Python solutions. Then, experts rigorously reviewed and corrected them, resulting in 908 high-quality problems with varying reasoning difficulties and a wide knowledge coverage. The data annotation process is as follows:

Seed Function Selection We selected 1,250 financial functions from the library based on operators, arguments, code lines, and long-tail knowledge, prioritizing those with complex computation.

[2]https://www.investopedia.com

Property	**Value**
Function Library	
# Total Functions	3313
# Operators (Avg)	2.85
# Arguments (Avg)	2.64
# Lines of Code (Avg)	3.45
# Financial Concepts Involved	1864
FinanceReasoning Dataset	
# Operators (Easy/Medium/Hard)	1.77/3.79/**10.12**
# Lines of Code (Easy/Medium/Hard)	3.13/4.27/**9.49**
# Parentheses (Easy/Medium/Hard)	0.80/3.28/**11.21**
# Difficulty (Easy/Medium/Hard)	1.69/3.00/**4.88**

Table 2: Statistics of the function library and FinanceReasoning dataset (Avg values of three subsets).

Question and Solution Generation For each seed function, GPT-4o was prompted to generate the complex reasoning problem with the necessary financial tabular data, using the financial terms and the computational processes of the function. The generated Python solutions were required to have clear reasoning paths and be executable to acquire numerical answers, taking into account units, percentages, and decimal precision requirements. Details of prompts are given in Appendix D.3.

Expert Verification The experts are required to review and correct all problems, solutions, and answers to ensure the absence of ambiguities, detailed processes, and correct answers.

2.4 Data Quality Assurance

To ensure the high quality of FinanceReasoning, we implemented a rigorous annotation process. Specifically, we organized a team of 8 graduate students with interdisciplinary backgrounds in finance and computer science, along with 2 experts holding CFA licenses, to participate in the dataset verification. Each financial function and problem were initially reviewed by two graduate students, who provided reasons for errors and suggested modifications. Consistent suggestions were adopted directly. For cases with conflicting opinions, the final modification plan was determined through a discussion between the two experts. With the help of LLMs, the entire annotation process lasted for three months. More annotation examples are provided in the Appendix D.4.

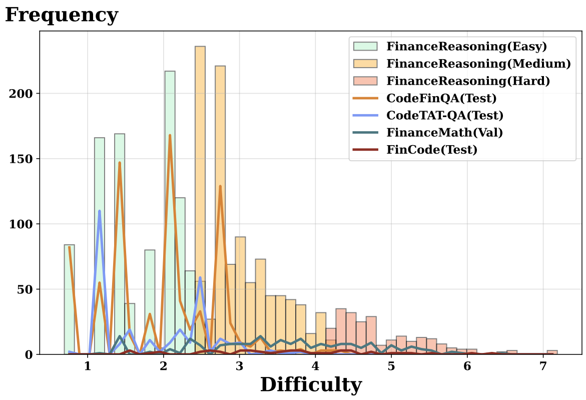

Figure 3: The difficulty distribution of FinanceReasoning, compared with four existing datasets, shows a notably higher proportion of medium and hard problems, presenting greater challenges for complex reasoning.

2.5 Data Grading and Statistics

To evaluate the performance of LRMs in financial numerical reasoning problems of varying difficulty levels, we designed a heuristic algorithm for the first time to assess the difficulty of reasoning for each problem based on the number of operators, pairs of parentheses, and lines of code in the Python program. Specifically, the difficulty of reasoning rc of a problem is defined as:

$$rc = \ln(\max(o, 1)) + \ln(\max(l + p, 1))$$

where o is the number of operators, p is the number of pairs of parentheses, and l is the number of code lines in the Python program.

As illustrated in Table 2 and Figure 3, based on the difficulty of reasoning, we divided the problems into three subsets: *Easy* (1,000 examples), *Medium* (1,000 examples), and *Hard* (238 examples). More analyses are in Appendix C.2. To promote the evaluation of LRMs' reasoning capabilities in the financial domain, we have made **all problems publicly available**, along with the **complete financial function library**.

3 Evaluation System

We developed an evaluation system for complex reasoning on FinanceReasoning, where all evaluations of LLMs were conducted by calling their official API interfaces. Table 18 in the Appendix illustrates exact model version we used.

3.1 Large Language Models

We focused on evaluating six of the most powerful large reasoning models currently available.

- **OpenAI-o1** (OpenAI, 2024d) is trained using large-scale reinforcement learning and employs chain-of-thought reasoning, excelling in general knowledge tasks and code reasoning tasks.
- **OpenAI-o1-mini** (OpenAI, 2024c) is a cost-effective alternative to OpenAI-o1, designed for high performance in STEM fields, particularly in mathematics and coding.
- **OpenAI-o3-mini** (OpenAI, 2025) is OpenAI's latest small reasoning model, providing faster response times while maintaining comparable performance to OpenAI-o1.
- **DeepSeek-R1** (Guo et al., 2025) enhances its reasoning capabilities through multi-stage training with reinforcement learning, using a minimal amount of supervised fine-tuning (SFT) data.
- **Gemini 2.0 Flash Thinking Experimental** (Gemini, 2025) introduces a 1M token context window to deeply understand long texts and incorporates self-correction mechanisms in reasoning tasks.
- **QwQ-32B-Preview** (Team, 2024) is an experimental model of the Qwen team, approaching problems with curiosity, self-questioning, and reflection, striving for a deeper understanding.

Gemini 2.0 Pro Experimental (Gemini, 2025), GPT-4o (OpenAI, 2024a), Claude 3.5 Sonnet (Anthropic, 2024), DeepSeek-V3 (DeepSeek-AI et al., 2024), Llama 3.3 (AI@Meta, 2024b), Llama 3.1 (AI@Meta, 2024a), and Qwen2.5-Max (Qwen et al., 2025) are also evaluated for comparison, providing a baseline to assess the performance between LRMs and traditional LLMs.

3.2 Evaluation Methods

Prompting Methods Following Zhao et al. (2024), we evaluated LLMs with Chain-of-Thought (Wei et al., 2022) and Program-of-Thought (Chen et al., 2023b) to achieve optimal performance and make comparisons. Detailed prompts are illustrated in Figure 14 and Figure 15 in the Appendix, respectively.

Answer Extraction and Evaluation We adopt the answer extraction pipeline from Zhao et al. (2024), using GPT-4o-mini to extract numerical results from the output under the CoT setting, and executing the program from the output under the PoT setting. Finally, we perform a strict accuracy evaluation comparing the numerical results with the ground truth within 0.2% error margin.

Model	Size	Notes	Hard		Medium		Easy		Avg.	
			CoT	PoT	CoT	PoT	CoT	PoT	CoT	PoT
Large Reasoning Models (LRMs)										
DeepSeek-R1	671B	MoE	**83.2**	85.3	**91.1**	**89.8**	**89.8**	**89.2**	**88.0**	**88.1**
OpenAI-o1			81.1	**89.1**	89.7	–	88.0	–	86.3	–
OpenAI-o3-mini			77.3	84.0	87.8	88.6	88.8	88.1	84.6	86.9
OpenAI-o1-mini			71.4	83.6	86.2	86.9	85.6	87.0	81.1	85.8
Gemini 2.0 Flash Thinking Experimental			70.6	81.5	85.2	87.2	88.8	86.6	81.5	85.1
QwQ-32B-Preview	32B		63.5	61.8	81.1	72.8	83.5	74.9	76.0	69.8
Large Language Models (LLMs)										
Gemini 2.0 Pro Experimental			72.3	83.6	88.3	87.4	87.3	87.8	82.6	86.3
GPT-4o			65.6	83.6	84.6	87.9	86.8	88.1	79.0	86.5
Claude 3.5 Sonnet			68.5	83.6	85.7	88.2	87.7	88.4	80.6	86.7
DeepSeek-V3	671B	MoE	66.8	75.6	85.2	87.3	87.2	86.9	79.7	80.7
Qwen2.5-Max		MoE	65.1	82.4	87.2	86.5	89.6	89.1	80.6	86.0
Llama 3.1	405B		51.7	70.2	81.7	87.7	84.1	85.8	72.5	81.2
Llama 3.3	70B		50.4	71.4	79.2	85.9	83.3	84.8	71.0	80.7

Table 3: Results of different models using CoT and PoT prompting methods on the different subsets of FinanceReasoning. The results underscore the superior performance of LRMs (*i.e.,*OpenAI-o1 and Deepseek-R1) with PoT in financial numerical reasoning task.

4 Experiments

We answer the following research questions (RQs): **RQ1**: Do LRMs outperform other LLMs in financial reasoning tasks? **RQ2**: What are the main shortcomings of LRMs? **RQ3**: Does refined knowledge augmentation improve LRMs' performance? **RQ4**: Does model collaboration enhance LRMs' performance? **RQ5**: Does PoT outperform CoT in complex numerical reasoning tasks?

4.1 Main Result (RQ1)

The performance of the evaluated LRMs and LLMs using two prompting methods on the FinanceReasoning are shown in Table 3.

The results demonstrate that the powerful LRM (*i.e.,*OpenAI-o1) using PoT prompting method achieves the best performance, with an accuracy of 89.1% on the *Hard* subset, significantly outperforming other LRMs and LLMs. On the *Easy* and *Medium* subsets, the evaluated LLMs achieve accuracy above 87%, except for the Llama models, where the advantage of LRMs is less pronounced. This further validates that simpler datasets have largely been solved by LLMs, making it difficult to assess the real reasoning capabilities of LRMs. On the *Hard* subset, LRMs exhibit a clear advantage over LLMs in CoT, further confirming the superiority of LRMs in complex reasoning tasks.

In particular, on the *hard* subset, the current superior LRMs (*i.e.,*OpenAI-o1 and DeepSeek-R1) exhibit distinct performance contrasts in CoT and PoT settings. In the CoT setting, DeepSeek-R1 achieves a 2.1% higher accuracy than OpenAI-o1 (**83.2% vs. 81.1%**). However, the PoT prompting method significantly enhances OpenAI-o1's performance, allowing it to surpass DeepSeek-R1 (**89.1% vs. 85.3%**). This suggests that DeepSeek-R1 excels at text-based step-by-step reasoning, while OpenAI-o1 outperforms in programming capabilities. This discrepancy may be due to differences in their training methods and training data.

Setting	GPT-4o (PoT)	Deepseek-R1 (CoT)
wo. knowledge augmentation	83.19	83.19
Passage Retrieval ($n = 10$)		
Vanilla Retrieval	81.93 (-1.26)	82.77 (-0.42)
Function Retrieval ($n = 3$)		
Vanilla Retrieval	90.76 (+7.57)	85.29 (+2.10)
LLM as Judge	89.08 (+5.89)	84.87 (+1.68)
LLM-instructed Retrieval & LLM as Judge	**91.60** (+8.41)	**86.97** (+3.78)

Table 4: Results of different knowledge augmentation methods on the *Hard* set of FinanceReasoning. GPT-4o with refined knowledge augmentation, outperforming OpenAI-o1 **(91.6% vs 89.1%)** under PoT setting.

4.2 Error Analysis (RQ2)

To better analyze the capabilities and limitations of LRMs on difficult problems in our dataset, we conduct a thorough and comprehensive error analysis. This analysis is based on 40 DeepSeek-R1 failure cases from the *Hard* set. We summarize four types of error in the current LRMs on challenging domain-specific reasoning problems, some of which involve compound errors. More details of error cases are provided in Appendix B.

- **Misunderstanding of Problem** (8/40): The model incorrectly interprets the question and context due to a lack of financial knowledge.
- **Formula Application Errors** (14/40): Owing to inexperience in financial reasoning, the model uses an incorrect formula that does not correspond to the specified conditions of the problem.
- **Numerical Extraction Errors** (3/40): The model extracts incorrect variables, especially when processing structured tabular data, despite the fact that the reasoning process and the selected formula are correct.
- **Numerical Calculation Errors** (15/40): When multiple calculation steps are involved, the model produces significant precision differences from the correct answer due to rounding and hallucination during the computation process.

4.3 Knowledge Augmentation (RQ3)

To enhance the understanding and application capabilities of complex formulas of LLMs in financial reasoning tasks, we explored and compared two formats of knowledge and various methods of enhancing knowledge to improve the performance of LLM in domain-specific complex reasoning tasks.

Knowledge Augmentation Settings We use Contriever (Izacard et al., 2022) to retrieve relevant financial knowledge passages or financial Python functions based on the question, following the prompt templates provided in Appendix E.1.

- **Function Retrieval**: We use the question as a query to compute semantic similarity with the function descriptions, retrieving the Top-3 financial functions as relevant knowledge.
- **Passage Retrieval**: For comparison with function retrieval, we segment each collected financial article into passages based on markdown hierarchical structures and retrieved the Top-10 passages for knowledge enhancement.
- **LLM as Retrieval Judge**: Recent studies have shown that models are capable of judging the relevance of candidates retrieved for the question (Guan et al., 2024). In this setting, we first retrieved the Top-30 financial functions and then prompted the LLM to select the Top-3 functions most useful to answer the question, if any.
- **LLM-Instructed Knowledge Retrieval**: In financial problems with hybrid contexts, using short questions or full contexts for retrieval often fails to retrieve directly relevant knowledge (Chen et al., 2023a; Peng et al., 2023). We observed that powerful LLMs (*e.g.,* GPT-4o) can effectively summarize rich semantic information from contexts. Therefore, we prompt the LLM to generate precise retrieval queries based on the context (Li et al., 2025; Verma et al., 2025).

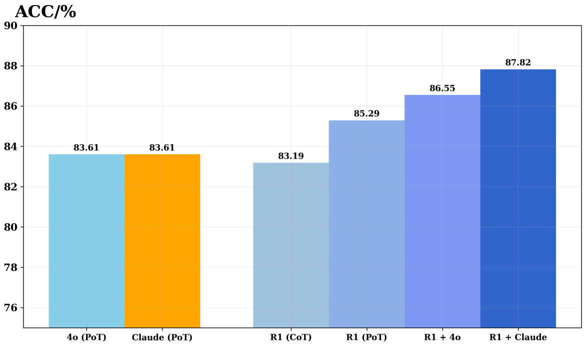

Figure 4: Result of different model combinations and individual models.

Knowledge Augmentation Results As shown in Table 4, the format and method of knowledge augmentation significantly affect the performance of the model reasoning. Specifically, LLMs enhanced with financial function knowledge significantly outperform those enhanced with passage knowledge, as financial functions serve as refined reasoning knowledge. Excessive and intricate passages can disrupt the model's reasoning abilities, resulting in diminished performance for both LLMs and LRMs. Taking advantage of the improved retrieval efficiency caused by *LLM-Instructed Knowledge Retrieval*, the combination approach achieves the best performance, improving the accuracy of GPT-4o to 91.6% with PoT. More analyses are shown in the Appendix E.2.

4.4 Reasoner with Programmer (RQ4)

To address the issue of imprecise numerical calculations in LRMs, we instruct the LRM to act as the **Reasoner**, responsible for carefully reasoning through the problem-solving path, while disregarding its generated numerical results. Then, a code-specialized LLM acts as the **Programmer**, strictly following the reasoning path provided by the Reasoner to generate executable Python programs, which are ultimately executed to obtain precise numerical results. The detailed prompts are provided in the Appendix F.1. Specifically, we employ **DeepSeek-R1** as Reasoner, and the best-performing models under PoT settings, **Claude**

3.5 Sonnet and **GPT-4o**, as Programmers, respectively. As shown in Figure 4, the combination of models achieves significant improvements compared to individual models. Compared to GPT-4o, Claude 3.5 Sonnet demonstrates a stronger ability to follow the given reasoning logic and generate precise code without introducing noise, owing to its programming advantages. The combination of DeepSeek-R1 and Claude 3.5 Sonnet achieves an accuracy of 87.82%, correcting 91.7% of the numerical calculation errors. Furthermore, we isolate the knowledge reasoning capabilities of DeepSeek-R1 in complex financial reasoning tasks, which outperform other LLMs. More analyses are shown in the Appendix F.2.

4.5 Comparing PoT and CoT (RQ5)

Based on an analysis of the hard subset, we observe that the PoT exhibits a markedly stronger performance than the CoT in multi-step and complex numerical reasoning tasks. Specifically, PoT leverages structured code generation to reduce token consumption. Under similar or lower token usage, PoT achieves greater accuracy (Table 3). Moreover, its performance is on par with certain LRMs that utilize test-time scaling strategies. For example, GPT-4o, when prompted with PoT, consumes only 54k tokens to solve the hard subset, whereas CoT requires 173k tokens. Detailed statistics on token consumption are provided in Table 19 in the Appendix.

Figure 16 indicates that LLMs with the PoT prompting method not only significantly reduce token overhead during inference, but can also approach or match the performance of LRMs. DeepSeek-R1 with CoT achieves the similar accuracy on the hard set as GPT-4o with PoT, but consumes much more tokens (742k vs. 54k). This "token-for-accuracy" test-time scaling strategy allows LRMs to maintain high correctness by repeatedly verifying outcomes from multiple perspectives. However, the associated inference cost is prohibitively high. In contrast, PoT achieves performance comparable to LRMs for complex financial calculations, while offering notably better cost-effectiveness.

5 Related work

The emergence of reasoning models such as OpenAI-o1 (OpenAI, 2024d) and DeepSeek-R1 (Guo et al., 2025) has significantly improved the performance of LLMs in complex reasoning tasks in domains such as code (Jain et al., 2024; Chen et al., 2021a), math (Mao et al., 2024; Lightman et al., 2023), and science (Lu et al., 2024; Yue et al., 2024; Wang et al., 2024). Among these large reasoning models (OpenAI, 2024d,c, 2025; Guo et al., 2025; Team, 2024; Team et al., 2025; Gemini, 2025), OpenAI-o1 and DeepSeek-R1 have achieved competitive optimal performance. However, there currently exists a gap between the evaluation of LRMs and real-world domain-specific reasoning tasks, with a lack of evaluation and research on the model's ability to flexibly apply domain knowledge in complex multi-step numerical reasoning. In the financial domain, the difficulty of questions and the quality of annotations become key limitations in evaluating the real reasoning capabilities of LRMs. For example, CodeFinQA and CodeTAT-QA (Krumdick et al., 2024), which are derived from the classic financial question answering datasets FinQA (Chen et al., 2021b) and TAT-QA (Zhu et al., 2021), rely on tabular data extraction and simple arithmetic operations that cannot accurately assess the improvements in reasoning ability of LRMs compared to LLMs. For datasets such as FinCode (Krumdick et al., 2024) and FinanceMath (Zhao et al., 2024), limited complex problems, ambiguous questions, and lenient evaluation criteria hinder the accurate assessment.

6 Conclusion

This paper introduces FinanceReasoning, a credible, comprehensive, and challenging benchmark designed to evaluate the financial numerical reasoning capabilities of LRMs. We update existing numerical reasoning financial question answering datasets, rigorously refine evaluation standards, and explore methods to build complex reasoning datasets tailored for LRM evaluation. We comprehensively evaluated the six most advanced LRMs in subsets of varying difficulty levels. Compared to LLMs, we validated the leading performance of OpenAI-o1 and DeepSeek-R1, while highlighting the need for further improvement in the precise numerical reasoning capabilities among LRMs. Our experiments on knowledge augmentation and the combination of models demonstrate that integrating structured, refined reasoning knowledge and enabling model collaboration can further enhance the complex reasoning performance of LRMs in expert domains.

7 Limitations

In this work, we introduce FinanceReasoning, a benchmark dataset designed to evaluate and enhance large language models in complex financial numerical reasoning tasks that require multi-step quantitative analysis, precise formula application, and hybrid contextual understanding. However, there are still some limitations: (1) We process tabular content as text, whereas in real-world scenarios, tables may also appear as images, requiring additional processing steps. In such cases, datasets such as MathVista (Lu et al., 2024) and MMMU (Yue et al., 2024), which focus on reasoning over image-based questions, serve as valuable complements to our benchmark. We believe that incorporating elements from these datasets into FinanceReasoning could help bridge the gap between text-based and multimodal financial reasoning, enabling a more comprehensive assessment of LLMs' real-world applicability. (2) Due to limited resources, we do not conduct an evaluation of OpenAI-o1 with PoT on the *Easy* and *Medium* subsets, as preliminary experiments suggest they are less challenging, and existing LLMs already demonstrate strong performance on these levels. (3) While we systematically verified and updated numerical answers and program solutions for multiple published datasets, we were unable to perform the same verification for the 1,000-problem test subset of FinanceMath (Zhao et al., 2024), as it does not publicly provide ground-truth references or Python solutions, limiting our ability to ensure consistency in result validation.

References

AI@Meta. 2024a. Llama 3.1 model card.

AI@Meta. 2024b. Llama 3.3 model card.

Anthropic. 2024. Claude 3.5 sonnet.

Liang Chen, Yang Deng, Yatao Bian, Zeyu Qin, Bingzhe Wu, Tat-Seng Chua, and Kam-Fai Wong. 2023a. Beyond factuality: A comprehensive evaluation of large language models as knowledge generators. In *The 2023 Conference on Empirical Methods in Natural Language Processing*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021a. Evaluating large language models trained on code. *Preprint*, arXiv:2107.03374.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023b. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research*.

Wenhu Chen, Ming Yin, Max Ku, Pan Lu, Yixin Wan, Xueguang Ma, Jianyu Xu, Xinyi Wang, and Tony Xia. 2023c. TheoremQA: A theorem-driven question answering dataset. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7889–7901, Singapore. Association for Computational Linguistics.

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. 2021b. 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, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Zhiyu Chen, Wenhu Chen, Charese Smiley, Sameena Shah, Iana Borova, Dylan Langdon, Reema Moussa, Matt Beane, Ting-Hao Huang, Bryan R Routledge, et al. 2021c. 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.

Zhiyu Chen, Shiyang Li, Charese Smiley, Zhiqiang Ma, Sameena Shah, and William Yang Wang. 2022. 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.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jiawei Wang, Jin Chen, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, Junxiao Song, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Litong Wang, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runxin Xu, Ruoyu Zhang, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping

Yu, Shunfeng Zhou, Shuting Pan, T. Wang, Tao Yun, Tian Pei, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wanjia Zhao, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaokang Zhang, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingkai Yu, Xinnan Song, Xinxia Shan, Xinyi Zhou, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. X. Zhu, Yang Zhang, Yanhong Xu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Yu, Yi Zheng, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Ying Tang, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yu Wu, Yuan Ou, Yuchen Zhu, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yukun Zha, Yunfan Xiong, Yunxian Ma, Yuting Yan, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Z. F. Wu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhipeng Xu, Zhiyu Wu, Zhongyu Zhang, Zhuoshu Li, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Ziyi Gao, and Zizheng Pan. 2024. Deepseek-v3 technical report. *Preprint*, arXiv:2412.19437.

Gemini. 2025. Gemini 2.0: Flash, flash-lite and pro.

Jian Guan, Wei Wu, zujie wen, Peng Xu, Hongning Wang, and Minlie Huang. 2024. AMOR: A recipe for building adaptable modular knowledge agents through process feedback. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

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. *Preprint*, arXiv:2501.12948.

Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. 2022. Unsupervised dense information retrieval with contrastive learning. *Transactions on Machine Learning Research*.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2024. Livecodebench: Holistic and contamination free evaluation of large language models for code. *Preprint*, arXiv:2403.07974.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *Preprint*, arXiv:2001.08361.

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, Bangkok, Thailand. Association for Computational Linguistics.

Xiaoxi Li, Guanting Dong, Jiajie Jin, Yuyao Zhang, Yujia Zhou, Yutao Zhu, Peitian Zhang, and Zhicheng Dou. 2025. Search-o1: Agentic search-enhanced large reasoning models. *Preprint*, arXiv:2501.05366.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. *Preprint*, arXiv:2305.20050.

Pan Lu, Hritik Bansal, Tony Xia, Jiacheng Liu, Chunyuan Li, Hannaneh Hajishirzi, Hao Cheng, Kai-Wei Chang, Michel Galley, and Jianfeng Gao. 2024. Mathvista: Evaluating math reasoning in visual contexts with gpt-4v, bard, and other large multimodal models. In *The Twelfth International Conference on Learning Representations*.

Yujun Mao, Yoon Kim, and Yilun Zhou. 2024. CHAMP: A competition-level dataset for fine-grained analyses of LLMs' mathematical reasoning capabilities. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 13256–13274, Bangkok, Thailand. Association for Computational Linguistics.

OpenAI. 2024a. Hello gpt-4o.

OpenAI. 2024b. Learning to reason with llms.

OpenAI. 2024c. Openai o1-mini.

OpenAI. 2024d. Openai o1 system card.

OpenAI. 2025. Openai o3-mini system card.

Baolin Peng, Michel Galley, Pengcheng He, Hao Cheng, Yujia Xie, Yu Hu, Qiuyuan Huang, Lars Liden, Zhou Yu, Weizhu Chen, and Jianfeng Gao. 2023. Check your facts and try again: Improving large language models with external knowledge and automated feedback. *Preprint*, arXiv:2302.12813.

Aske Plaat, Annie Wong, Suzan Verberne, Joost Broekens, Niki van Stein, and Thomas H.W. Back. 2024. Reasoning with large language models, a survey. *ArXiv*, abs/2407.11511.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng

Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. 2024. Mathematical discoveries from program search with large language models. *Nat.*, 625(7995):468–475.

Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, et al. 2025. Kimi k1. 5: Scaling reinforcement learning with llms. *Preprint*, arXiv:2501.12599.

Qwen Team. 2024. Qwq: Reflect deeply on the boundaries of the unknown.

Prakhar Verma, Sukruta Prakash Midigeshi, Gaurav Sinha, Arno Solin, Nagarajan Natarajan, and Amit Sharma. 2025. Plan*rag: Efficient test-time planning for retrieval augmented generation. *Preprint*, arXiv:2410.20753.

Xiaoxuan Wang, Ziniu Hu, Pan Lu, Yanqiao Zhu, Jieyu Zhang, Satyen Subramaniam, Arjun R Loomba, Shichang Zhang, Yizhou Sun, and Wei Wang. 2024. Scibench: Evaluating college-level scientific problem-solving abilities of large language models. In *Forty-first International Conference on Machine Learning*.

Yuqing Wang and Yun Zhao. 2024. Metacognitive prompting improves understanding in large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 1914–1926, Mexico City, Mexico. Association for Computational Linguistics.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*.

Fengli Xu, Qianyue Hao, Zefang Zong, Jingwei Wang, Yunke Zhang, Jingyi Wang, Xiaochong Lan, Jiahui Gong, Tianjian Ouyang, Fanjin Meng, et al. 2025. Towards large reasoning models: A survey of reinforced reasoning with large language models. *Preprint*, arXiv:2501.09686.

Xiang Yue, Yuansheng Ni, Kai Zhang, Tianyu Zheng, Ruoqi Liu, Ge Zhang, Samuel Stevens, Dongfu Jiang, Weiming Ren, Yuxuan Sun, et al. 2024. Mmmu: A massive multi-discipline multimodal understanding and reasoning benchmark for expert agi. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9556–9567.

Yilun Zhao, Hongjun Liu, Yitao Long, Rui Zhang, Chen Zhao, and Arman Cohan. 2024. 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, Bangkok, Thailand. Association for Computational Linguistics.

Fengbin Zhu, Wenqiang Lei, Youcheng Huang, Chao Wang, Shuo Zhang, Jiancheng Lv, Fuli Feng, and Tat-Seng Chua. 2021. TAT-QA: A question answering benchmark on a hybrid of tabular and textual content in finance. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3277–3287, Online. Association for Computational Linguistics.

A Construction of the FinanceReasoning Benchmark

To rigorously assess financial numerical reasoning in LLMs and LRMs, we constructed FinanceReasoning, a benchmark comprising **2,238 problems**. This dataset integrates two key sources:

1. **1,420 problems** carefully reviewed and refined from existing datasets, involving disambiguation, elaboration and correction.

2. **908 newly generated problems**, synthesized by LLMs(*e.g.*,GPT-4o) using a financial function library that we collected and open-sourced, containing 3,133 Python-formatted financial functions. Each generated problem is accompanied by an executable Python solution and precise numerical answer, all expert-verified to ensure accuracy and robustness.

To systematically evaluate model capabilities, we introduce a tiered classification of difficulty based on computational complexity and reasoning depth:

- **Easy level**: Direct data extraction, minimal computation.

- **Medium level**: Basic percentage calculations, application of financial formulas without structural adjustments.

- **Hard level**: Multi-step computations, weighted calculations, and financial concept reasoning.

This construction process, illustrated in Figure 5, underscores FinanceReasoning's credibility, comprehensiveness, and challenge, setting a new standard for evaluating financial reasoning in AI models.

B Error Cases

We present a representative example for each of the four common error types — misunderstanding of the problem, formula application errors, numerical extraction errors and numerical calculation errors in Table 7, Table 6, Table 8, and Table 5, respectively, as made by DeepSeek-R1.

C Details of Public Dataset Updates

In this section, we present five common issues encountered in CodeFinQA (Test), CodeTAT-QA (Test), FinanceMATH (Val), and Fincode (Test), along with the corresponding updating approaches. Besides, we analyze the distribution of difficulty levels among the public datasets and our dataset.

C.1 Cases of Dataset Update

- **Unsolvable Questions**: Problems that either lack sufficient data or are inherently unsolvable. To address this, the case in Table 9 was updated by applying the disambiguation strategy to refine the question.
- **Ambiguous Statements** Questions or statements that are unclear or open to multiple interpretations. In the case of Table 10, we updated the question by applying a disambiguation strategy.
- **Unclear Processes** Inadequate or missing steps in problem-solving processes can make calculations or conclusions difficult to follow. To address this, we enhanced the Python solution in Table 11 by applying an elaboration approach, ensuring a more detailed and structured problem-solving process.
- **Incorrect Answers** Answers that are mathematically or conceptually incorrect, leading to misleading conclusions. In Table 12, the answer was refined by applying a correction process.
- **Inconsistent Evaluation** Issues with grading or assessment standards, where answers that differ in precision (e.g., 0.01, 10 and 1%) are considered equivalent. We refined the answer in Table 13 by applying a correction approach.

C.2 Difficulty Level Distribution Analysis

The distribution of difficulty among public dataset and ours is shown in **??**. As can be seen, the difficulty distribution of FinanceReasoning (Ours) surpasses previous datasets with respect to the number of medium and difficult-level questions, thereby facilitating a more accurate reflection of the model's reasoning ability.

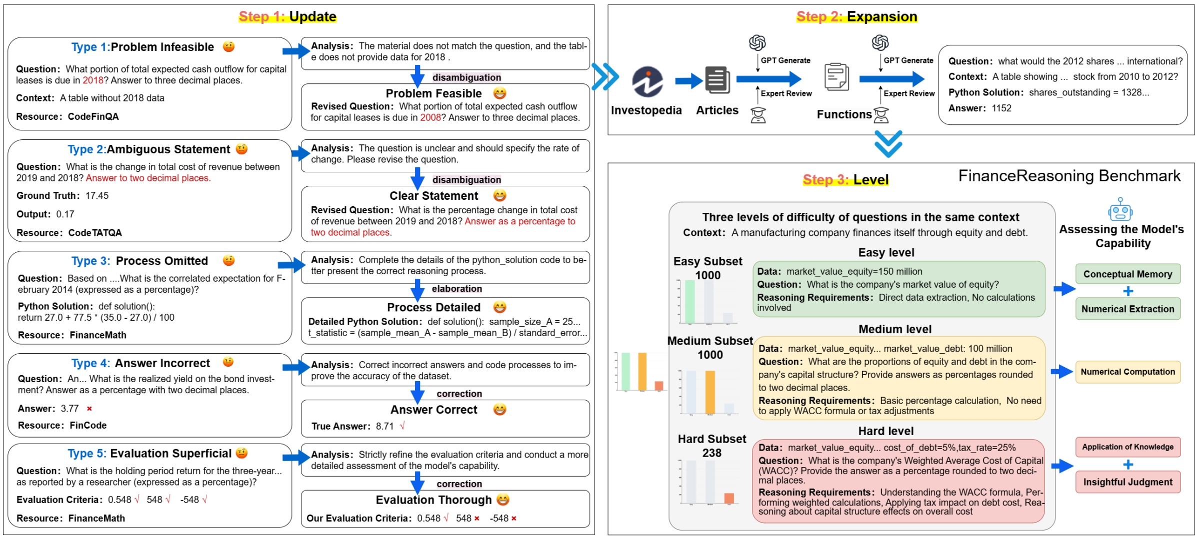

Figure 5: Over Review of Dataset Annotation

D Details of Dataset Expansion

D.1 Financial Function Extraction Prompt

The financial function extraction prompt is designed to guide the model in identifying and extracting relevant financial functions from raw data. The System Input and User Input are provided in Figure 6 and Figure 7, respectively.

D.2 Financial Function Example

The constructed financial functions primarily consist of detailed computational code accompanied by comprehensive annotations. These annotations include explanations of function calculations, descriptions of input parameters and output variables, as well as specifications of application scope and constraints. This enhances the readability of the functions and strengthens the applicability of large models through well-structured documentation. For detailed information, refer to Table 14.

D.3 Automated Question Construction Prompt

The automated question construction prompt is used to generate questions based on the financial functions extracted from the data. This prompt enables the model to formulate questions that test understanding of financial concepts and the application of financial formulas. The System Input and User Input are provided in Figure 8 and Figure 9, respectively.

D.4 Expansion of Data Annotation Case

This section presents examples of the types of questions that can be automatically generated using the automated question construction prompt. These questions are based on the financial functions and concepts extracted earlier, and they help to illustrate the variety of questions that can be created. The goal is to ensure that the questions are both relevant to the domain and challenging enough to test deep understanding. For detailed information, refer to Table 15.

E Details of Knowledge Augmentation Experiment

E.1 RAG prompt templates

LLM Instruct Prompt The LLM Instruct prompt allows the model to generate its own retrieval queries based on the task at hand. The LLM (Large Language Model) is prompted to automatically propose relevant questions or search terms that would retrieve the necessary information to solve the problem. More comprehensive details are given in Figure 11.

Judge Useful Prompt This prompt is used to evaluate whether the retrieved information or external knowledge is useful for answering the given question. The model judges the relevance and reliability of the retrieved data, ensuring that only useful information is used for further processing. For detailed information, refer to Figure 12.

RAG Prompt The RAG prompt combines both the retrieval and generation components. It instructs the model to retrieve information from external sources and generate an answer based on that information. This prompt enables the model to make use of both stored knowledge and newly retrieved data, enhancing the model's ability to produce accurate answers. For detailed information, refer to Figure 10.

E.2 Case study

In case Table 16, we note the combination of R1 and RAG improves accuracy. When using R1 alone, errors may occur due to a lack of sufficient information or context. However, when the RAG method is applied, the retrieval of external knowledge provides additional context and enables the model to generate the correct answer.

F Details of Reasoner with Programmer Experiment

F.1 Summary Program Prompt

The summary program prompt (More details in Figure 13) is designed to generate a Python program that answers a given financial question while strictly adhering to the reasoning process provided by another LRM. This approach enhances transparency and verification of the reasoning model's output by preventing direct reliance on precomputed values.

F.2 Case Study

the comparison between the performance of R1 combined with Claude and R1 alone are shown in Table 17. The task involves solving a problem that, when attempted using R1 alone, results in errors. However, when R1 is combined with Claude, the solution is correctly generated. This comparison highlights the advantage of using reasoning (via Claude) in combination with R1 to improve the accuracy and reliability of the model's output.

G More Experimental Details

The details of the evaluated models are provided in Table 18. The CoT prompt is presented in Figure 14, while the PoT prompt is shown in Figure 15. The token consumption of LLMs and LRMs under CoT and PoT prompting is provided in Table 19.Additionally, the relationship between token consumption and accuracy across the evaluated models is illustrated in Figure 16.

Question:
What is the fair price of the European put option? Answer in USD, rounded to two decimal places.

question_id: test-2208
ground_truth: 5.46
python_solution:
```
def solution():
...
  # Define variables
  S = 150 # Current stock price in USD
  K = 140 # Strike price in USD
  r = 0.02 # Risk-free interest rate (as a decimal)
  T = 0.5 # Time to maturity in years
  volatility = 0.25 # Volatility as a decimal
  # Calculate d1 and d2 using the Black-Scholes formula
  d1 = (log(S / K) + (r + 0.5 * volatility ** 2) * T) / (volatility * sqrt(T))
  d2 = d1 - volatility * sqrt(T)
  # Calculate the put option price
  put_price = K * exp(-r * T) * norm.cdf(-d2) - S * norm.cdf(-d1)
  # Return the put price rounded to two decimal places
  return round(put_price, 2)
...
```
context:
A professional investor is considering purchasing a European put option for a technology stock currently trading at 150 USD. The put option has the following parameters: Current stock price, $S = 150$ USD
Strike price, $K = 140$ USD
Time to maturity, $T = 0.5$ years
Risk-free interest rate, $r = 2\%$ per annum
Volatility, $\sigma = 25\%$ per annum
The investor wants to determine the fair price of the put option to make an informed investment decision.

R1:

...
2. Calculate d_1 and d_2:
We use the following formulas for d_1 and d_2:

$$d_1 = \frac{\ln(S/K) + (r + \frac{\sigma^2}{2})T}{\sigma\sqrt{T}} = \frac{\ln(150/140) + (0.02 + 0.25^2/2) \cdot 0.5}{0.25\sqrt{0.5}}$$

$\ln(150/140) \approx 0.06908$
$(0.02 + 0.03125) \cdot 0.5 = 0.025625$
Numerator: $0.06908 + 0.025625 = 0.094705$
Denominator:

$$d_2 = d_1 - \sigma\sqrt{T} = 0.5357 - 0.1767767 \approx 0.3589$$

...
After refining calculations with precise approximations and standard normal CDF values, the fair price of the European put option rounds to 5.44 USD.

Table 5: Example of numerical calculating error made by DeepSeek-r1.The model applied excessive rounding during the calculation steps. The parts highlighted in red are incorrect,and the parts highlighted in green are correct.

Question:
What is the approximate FICO score of this individual based on the given factors? Answer to the nearest integer.

question_id: test-2144
ground_truth: 74
function:

```
...
    # Calculate the FICO score
    fico_score = (payment_history * payment_history_weight +
         accounts_owed * accounts_owed_weight +
         length_of_credit_history * length_of_credit_history_weight +
         credit_mix * credit_mix_weight +
         new_credit * new_credit_weight)
    return fico_score
```

context:
A young professional is evaluating his creditworthiness to apply for a mortgage. His financial advisor collects data on his credit behavior and assigns a percentage rating to several aspects of his credit profile: a payment history of 90%, accounts owed at 65%, a length of credit history of 70%, a credit mix at 80%, and new credit at 50%. These factors help approximate a FICO score, which weighs each aspect differently: 35% for payment history, 30% for accounts owed, 15% for the length of credit history, 10% for credit mix, and 10% for new credit.

R1:
To calculate the approximate FICO score, each percentage rating is multiplied by its corresponding weight, summed, and then scaled to the FICO range ($300 - 850$).
1. Payment History (35% weight):
$90\% \times 0.35 = 31.5\%$
2. Accounts Owed (30% weight):
$65\% \times 0.30 = 19.5\%$
3. Length of Credit History (15% weight):
$70\% \times 0.15 = 10.5\%$
4. Credit Mix (10% weight):
$80\% \times 0.10 = 8\%$
5. New Credit (10% weight):
$50\% \times 0.10 = 5\%$
Total weighted percentage:
$31.5\% + 19.5\% + 10.5\% + 8\% + 5\% = 74.5\%$
Convert to FICO score (scaled to 850):
$74.5\% \times 850 = 0.745 \times 850 = 633.25$
Rounded to the nearest integer: **633**.
Therefore, the answer is 633.

Table 6: Example of formula application error made by DeepSeek-r1. The model lacks knowledge of the FICO score, leading to an incorrect formula. The parts highlighted in red are incorrect,and the parts highlighted in green are correct.

Question:
A risk manager specializing in market risk is looking to determine the cost of a 2-year zero-coupon bond. The current 1-year interest rate is 8.0%. There's a chance of 50% that the 1-year interest rate will reach 10.0% after one year and an equal probability of its falling to 6.0% in the same period. Suppose the yearly risk premium for duration risk is 40 bps, and the bond faces a value of EUR 1,000. What should be the zero-coupon bond's valuation? Answer to three decimal places.

question_id: test-2164
ground_truth: 854.466
function:

```
def solution():
    bond_face_value = 1000
    risk_free_rate = 0.08
    risk_premium = 0.004
    up_interest_rate = 0.10
    down_interest_rate = 0.06
    probability_up = 0.5
    probability_down = 0.5

    up_price_date1 = bond_face_value / (1 + up_interest_rate + risk_premium)
    down_price_date1 = bond_face_value / (1 + down_interest_rate + risk_premium)

     expected_price_date1 = probability_up * up_price_date1 + probability_down *
down_price_date1
    zero_coupon_bond_price = expected_price_date1 / (1 + risk_free_rate)

    return zero_coupon_bond_price
```

R1:
To determine the valuation of the 2-year zero-coupon bond, we account for the uncertain future interest rates and the yearly risk premium of 40 basis points (bps). The steps are as follows: ...
3. Discounting to Present Value:
The expected value at $t = 1$ is discounted back to $t = 0$ using the first-year rate adjusted for the risk premium $(8.0\% + 0.4\% = 8.4\%)$:

$$\text{PV}_0 = \frac{922.8234}{1 + 0.084} \approx 851.2993$$

Therefore, the answer is 851.299.

Table 7: Example of misunderstanding of the problem error made by DeepSeek-r1. The model misunderstood the concept of a zero-coupon bond and incorrectly included the coupon bond. The parts highlighted in red are incorrect,and the parts highlighted in green are correct.

Question:
What is the taxable income from Jane's ESPP transaction? Answer to the nearest dollar.

question_id: test-2017
ground_truth: 4200

```
def solution():
  # Define the given variables
  purchase_price = 40.0 # Purchase price per share in dollars
  sale_price = 55.0 # Sale price per share in dollars
  shares = 200 # Number of shares
  discount = 0.15 # ESPP discount
  holding_period = 1.5 # Holding period in years (less than 2)

  # Calculate the discounted purchase price per share
  discounted_purchase_price = purchase_price * (1 - discount)

  # Calculate the total gain from selling the shares
  total_gain = (sale_price - discounted_purchase_price) * shares

  # Since the holding period is less than 2 years, treat the entire gain as ordinary
income
  taxable_income = total_gain

  # Return the answer rounded to the nearest dollar
  return round(taxable_income)

# Testing the solution function
print(solution())
```

context:
Jane participates in her company's Employee Stock Purchase Plan (ESPP), which offers a 15% discount on stock purchases. She bought 200 shares at a purchase price of $40 per share. After holding the shares for 18 months, she decides to sell them at $55 per share. The IRS mandates that any share held less than two years must have its gain treated entirely as ordinary income. Jane needs to calculate the taxable income from this transaction to determine her tax liability.

R1:
Jane's taxable income from the ESPP transaction involves two components: the discount received at purchase and the gain from selling the shares, both treated as ordinary income due to the holding period being less than two years.
1. Discount Calculation (Compensation Income):
- The purchase price per share ($40) reflects a 15% discount. The fair market value (FMV) at purchase is calculated as:

$$\text{FMV at purchase} = \frac{\$40}{1 - 0.15} = \frac{\$40}{0.85} \approx \$47.06 \text{ per share.}$$

- Discount per share:

$$\$47.06 - \$40 = \$7.06.$$

...
Therefore, the answer is 3000.

Table 8: Example of numerical extraction error made by DeepSeek-r1. The model treats the purchase price (pre-discount price) as the post-discount price. The parts highlighted in red are incorrect,and the parts highlighted in green are correct.

Question: as of december 31 , 2006 , what was the total total cash obligations aggregate carrying value of long-term debt due in 2006 Answer to three decimal places.

The revised question: As of December 31, 2007, what was the total cash obligations aggregate carrying value of long-term debt due in 2007? Answer to three decimal places.

question_id: test_114
ground_truth: 9.562
program: notes_2005 = 338501
total_obligations = 3540009
percent_2005 = notes_2005 / total_obligations
answer = percent_2005 * 100
context:
...

Maturities 2014

As of December 31, 2006, the aggregate carrying value of long-term debt, including capital leases, for the next five years and thereafter are estimated to be (in thousands):

Year Ending December 31	Amount ($)
2007	253,907
2008	1,278
2009	654
2010	1,833,416
2011	338,501
Thereafter	1,112,253
Total Cash Obligations	3,540,009
Accreted Value of the Discount and Premium of 3.00% Notes and 7.125% Notes	3,007
Balance as of December 31, 2006	3,543,016

The holders of the company's 5.0% (5.0%) notes have the right to require the company to repurchase their notes on specified dates prior to the maturity date in 2010, but the company may pay the purchase price by issuing shares of Class A common stock, subject to certain conditions. Obligations with respect to the right of the holders to put the 5.0% (5.0%) notes have been included in the table above as if such notes mature the date on which the put rights become exercisable in 2007. In February 2007, the company conducted a cash tender offer for its outstanding 5.0% (5.0%) notes to enable note holders to exercise their right to require the company to purchase their notes. (See Note 19.)
...

Table 9: A case where a question from CodefinQA is answered, but because the question itself is flawed (the table lacks 2006 data), it cannot be solved, rendering the entire output meaningless. The parts highlighted in red are incorrect,and the parts highlighted in green are the revised ones.

Question: What is the change in total cost of revenue between 2019 and 2018? Answer to two decimal places.

The revised question: What is the percentage change in total cost of revenue between 2019 and 2018? Answer as a percentage to two decimal places.

question_id: test_142
ground_truth: -5.81
program:

```
total_cost_of_revenue_2019 = df["Cost of revenue: – Total cost of revenue"]["2019"]
total_cost_of_revenue_2018 = df["Cost of revenue: – Total cost of revenue"]["2018"]
answer = ( total_cost_of_revenue_2019 - total_cost_of_revenue_2018 ) / total_cost_of_revenue_2018 * 100.0
```

context:

Category	**2019**	**2018**	**Amount**	**Percent**
Cost of revenue: – Products	29816	34066	-4250	-12%
Cost of revenue: – Services	19065	17830	1235	7%
Cost of revenue: – Total cost of revenue	48881	51896	-3015	-6%

Table 10: A case where answers a question from CodeTAT-QA is answered, but due to the ambiguous phrasing of the question, multiple possible answers exist simultaneously(For example, the question in the table does not specify whether to use percentages or decimals). The parts highlighted in red are incorrect, and the parts highlighted in green are the revised ones.

Question: Assuming that the variances of the underlying populations are equal, independent samples taken from normally distributed groups display the following features: The combined estimation of the common variance is 2,678.05. What is the suitable t-test statistic to verify the assumption that the average of the two populations are the same?

question_id: validation-24
ground_truth: 0.938
topic: Quantitative Analysis & Valuation
program:

```
return (200 - 185) / (2678.05 / 25 + 2678.05 / 18)**0.5
```

The revised program:

```
# Given values
sample_size_A = 25
sample_size_B = 18
sample_mean_A = 200
sample_mean_B = 185
combined_variance = 2678.05

# Standard error of the difference in means
standard_error = (combined_variance / sample_size_A + combined_variance / sample_size_B)**0.5

# t-test statistic calculation
t_statistic = (sample_mean_A - sample_mean_B) / standard_error

return t_statistic
```

context:

Sample Size	Sample Mean	Sample Standard Deviation
A 25	200	45
B 18	185	60

Table 11: A case where a question from FinanceMATH is answered, and although the answer is correct, the process is unclear, and the generated code has missing steps. The parts highlighted in red represent the original code, while the green parts represent the improved code.

Question: An investor purchases a five-year, 6% annual-coupon payment bond at 89.4535 and sells it in four years. Following the purchase of the bond and prior to the receipt of the first coupon, interest rates go down to 8.5%. What is the realized yield on the bond investment? Answer as a percentage with two decimal places.

Before the modification
answer: 3.7699999809
question_id: test-8
program:

```
coupon_rate = 0.06
initial_price = 89.4535
years_til_maturity = 5
interest_rate = 0.085
coupon = coupon_rate * initial_price
discount = initial_price - coupon
final_price = discount * (1 + interest_rate) ** (years_til_maturity - 4)
realized_yield = (final_price / initial_price) - 1
realized_yield_per = 100.0 * realized_yield
round(realized_yield_per, 2)
```

After the modification
answer: 8.71
question_id: test-8
program:

```
face_value = 100
coupon_rate = 0.06
initial_price = 89.4535
years_til_maturity = 5
interest_rate = 0.085
holding_period = 4
coupon = coupon_rate * face_value
sell_price = (face_value + coupon) / ((1 + interest_rate) ** (years_til_maturity -
holding_period))
coupon_received = coupon + coupon * (1 + interest_rate) + coupon * (1 + interest_rate)
** 2 + coupon * (1 + interest_rate) ** 3
realized_yield = ((sell_price + coupon_received) / initial_price) ** (1 /
holding_period) - 1
realized_yield_per = 100.0 * realized_yield
round(realized_yield_per, 2)
```

Table 12: A case where a question from Fincode is answered with an incorrect answer, which we have corrected. The parts highlighted in red represent the original, while the parts highlighted in green represent the corrected version.

Question: What is the holding period return for the three-year timeframe, given the following annual rates of return for a mutual fund as reported by a researcher (expressed as a percentage)?

The revised question: What is the holding period return for the three-year timeframe, given the following annual rates of return for a mutual fund as reported by a researcher (expressed as a percentage)? Answer to three decimal places.

question_id: validation-68
ground_truth: 0.548
program:
```
return ((1+0.14)*(1-0.10)*(1-0.02)-1)*100
```
context:

Year	Return(%)
2008	14
2009	-10
2010	-2

Table 13: A case where a question from FinanceMATH is answered with an evaluation that is not rigorous, and there are multiple correct answers.The parts highlighted in red are incorrect, and the parts highlighted in green are the revised ones.

Financial Function Extraction Prompt

```
SYSTEM_INPUT = '''You are an expert in financial analysis and Python programming. Your task is to analyze financial
↪  articles and extract generalized Python functions that can be reused for similar financial problems. These functions
↪  should be based on financial variables, calculations, or examples mentioned in the article.

The functions you create should include:
1. A clear and concise function signature.
2. A detailed docstring that includes:
   - A brief description of the function's purpose.
   - A description of each parameter.
   - A description of the return value.
   - Notes on applicability, constraints, and important considerations.

If the article does not contain any extractable financial functions, return `[EMPTY]`. If the article contains multiple
↪  functions, extract and output all of them, each wrapped in ```python ``` blocks.

Here is an example of a well-structured financial Python function:

```python
def calculate_present_value(cash_flows: list, discount_rate: float) -> float:
    """
    Calculate the present value of a series of cash flows.

    This function computes the present value (PV) of a list of future cash flows
    using a specified discount rate. It is commonly used in financial analysis
    to evaluate investments or projects.

    Args:
        cash_flows (list): A list of future cash flows (positive or negative).
        discount_rate (float): The discount rate (e.g., interest rate) used to
                               discount future cash flows. Must be greater than -1.

    Returns:
        present_value (float): The present value of the cash flows.

    Notes:
        - This function assumes that cash flows occur at the end of each period.
        - The discount rate must be greater than -1 to ensure a valid result.
        - Cash flows should be provided in chronological order.
        - This function does not account for inflation or taxes.
    """
    if discount_rate <= -1:
        raise ValueError("Discount rate must be greater than -1.")

    present_value = sum(cf / ((1 + discount_rate) ** (i + 1)) for i, cf in enumerate(cash_flows))

    return present_value
```
Your goal is to extract and construct such functions from financial articles. Ensure that the function signature,
↪ docstring, and implementation are clear, concise, and free of problem-specific values.

Important: Only output the function definition (signature, docstring, and implementation). Do not include example usage,
↪ explanations, or any additional text.
'''
```

Figure 6: System Input of Financial Function Extraction Prompt

Financial Function Extraction Prompt

```
USER_INPUT='''I have a financial article. Please analyze the article and extract any generalized Python functions that
↪  can be reused for similar problems. The functions should be based on financial variables, calculations, or examples
↪  mentioned in the article and include a clear function signature, a detailed docstring, and an implementation that
↪  avoids problem-specific values.

Here is the financial article:
{financial_artical}

Please create a generalized Python function with a meaningful name (e.g., calculate_xxx), a clear function signature, and
↪  a detailed docstring. Here is an example of the desired output format:
```python
def calculate_xxx(param1: type, param2: type, ..., paramN: type) -> return_type:
    """
    Brief description of the function's purpose.

    Args:
        param1 (type): Description of param1.
        param2 (type): Description of param2.
        ...
        paramN (type): Description of paramN.

    Returns:
        xxx (return_type): Description of the return value.

    Notes:
        - Applicability: Describe where this function is applicable.
        - Constraints: List any constraints or limitations.
        - Considerations: Mention any important considerations or potential issues.
    """
    # Do math calculation to get the answer
    pass

    # return answer
    return xxx
```

Continue your output:
```python
def calculate_xxx(param1: type, param2: type, ..., paramN: type) -> return_type:
    """
    Brief description of the function's purpose.

Important: Only output the function definition (signature, docstring, and implementation). Do not include example usage,
↪  explanations, or any additional text.
'''
```

Figure 7: User Input of Financial Function Extraction Prompt
```

function:

```
def calculate_hedge_fund_net_return(initial_investment: float, gross_return_rate:
float,
  management_fee_rate: float, incentive_fee_rate: float,
  hurdle_rate: float) -> float:
  """
    Calculate the net return for an investor in a hedge fund with fees and hurdle
rates.
    This function computes the net return in dollar terms for an investor after
    accounting for management fees and incentive fees, based on a hurdle rate.
    Args:
      initial_investment (float): The initial investment amount in millions.
      gross_return_rate (float): The gross return rate achieved by the hedge fund.
      management_fee_rate (float): The management fee rate as a decimal.
      incentive_fee_rate (float): The incentive fee rate as a decimal.
      hurdle_rate (float): The hurdle rate as a decimal percentage of initial
investment.
    Returns:
      net_return (float): The net return for the investor in millions to two decimal
places.
    Notes:
      - Applicability: Suitable for hedge funds using a similar fee structure.
      - Constraints: Assumes management fee is based on year-end assets and incentive
fee
        is calculated net of management fee, based on returns over the hurdle rate.
      - Considerations: Ensure all rates are provided as decimals (e.g., 2% as 0.02).
  """
  year_end_assets = initial_investment * (1 + gross_return_rate)
  management_fee = year_end_assets * management_fee_rate
  net_year_end_assets = year_end_assets - management_fee
  hurdle_amount = initial_investment * (1 + hurdle_rate)
  excess_return = max(0, net_year_end_assets - hurdle_amount)
  incentive_fee = excess_return * incentive_fee_rate
  total_fees = management_fee + incentive_fee
  net_return = year_end_assets - total_fees - initial_investment
  return round(net_return, 2)
```

Table 14: Example of Financial Function From Function Library

Automated Question Construction Prompt

````
SYSTEM_INPUT = '''You are a financial expert with deep knowledge of financial concepts and numerical reasoning. Your task
↪  is to generate a financial numerical reasoning problem based on a given financial function. The problem should
↪  require the application of financial knowledge to solve, and it should include specific numerical values and a clear
↪  financial context.

### Problem Design Requirements:
1. **Specific Numerical Values**: The problem must contain specific numerical values to ensure it is computationally
↪  solvable.
2. **Real-World Context**: The problem context should provide a real-world financial application case, wrapped in
↪  `<context></context>` tags.
3. **Clear Question**: The question should be clear and wrapped in `<question></question>` tags. It should ask for a
↪  single numerical result, specifying the unit, percentage format (any ratio result should be in percentage format),
↪  and rounding requirements (e.g., "Answer to the nearest integer," "Answer as a percentage to two decimal places”).
4. **Function Logic Alignment**: The problem content should align with the computation logic of the given financial
↪  function but should not explicitly mention the function name or its details.
5. **Encourage Problem-Solving**: The problem should encourage problem-solving and numerical application, making it
↪  suitable for finance-related training or assessments.

### Solution Requirements:
1. **Python Program**: Provide a Python program that solves the problem. The program must follow the format `def
↪  solution():` and return the computed result, wrapped in `<solution></solution>` tags.
2. **Function Logic Consistency**: The program should implement the financial calculation from scratch using the
↪  parameters given in the problem statement. It should not directly call the provided function but should follow the
↪  correct financial logic.
3. **Format and Clarity**: The program should be well-structured, with clear variable definitions, calculations, and a
↪  return statement.

### Example:
Given the function:
```python
def calculate_margin_call_requirement(initial_margin: float, current_value: float, maintenance_margin_rate: float) ->
↪  float:
    """
    Calculate the margin call requirement needed to meet maintenance margin.
    """
    required_maintenance_margin = current_value * maintenance_margin_rate
    margin_call_requirement = max(0, required_maintenance_margin - initial_margin)
    return margin_call_requirement
```

A generated problem and solution might look like this:
<context>
An investor opens a margin account with an initial margin of 50,000 to purchase securities. Due to market fluctuations,
↪  the current value of the securities hasdroppedto60,000. The brokerage requires a maintenance margin rate of 30%.
</context>
<question>
What is the minimum amount the investor must deposit to meet the maintenance margin requirement? Answer to the nearest
↪  integer.
</question>
<solution>
```python
def solution():
    # Define variables and their values
    initial_margin = 50000
    current_value = 60000
    maintenance_margin_rate = 0.30

    # Perform calculations
    required_maintenance_margin = current_value * maintenance_margin_rate
    margin_call_requirement = max(0, required_maintenance_margin - initial_margin)

    # Return the final answer
    return round(margin_call_requirement)
```
</solution>

Your goal is to generate such problems and solutions based on the provided financial function.
'''
````

Figure 8: System Input of Automated Question Construction Prompt

Automated Question Construction Prompt

```
USER_INPUT = '''Given the following financial function:
{financial_function}

Please generate a financial numerical reasoning problem based on this function. The problem should include:

1. A real-world financial context wrapped in <context></context> tags.
2. A clear question wrapped in <question></question> tags, specifying the unit, percentage format (if applicable), and
↪  rounding requirements.
3. Specific numerical values to ensure the problem is computationally solvable.

Additionally, provide a Python program that solves the problem. The program must:

1. Follow the format `def solution():` and return the computed result, wrapped in `<solution></solution>` tags.
2. Implement the financial calculation from scratch using the parameters given in the problem statement.
3. Align with the computation logic of the provided function without directly calling it.

Please generate the problem and solution based on the provided financial function.
'''
```

Figure 9: User Input of Automated Question Construction Prompt(lower part)

Question: What is the company's Weighted Average Cost of Capital (WACC)? Answer as a percentage to two decimal places.
question_id: test-2002
ground_truth: 6.9
context:

A manufacturing company is evaluating its financing strategy and needs to calculate its Weighted Average Cost of Capital (WACC) to optimally structure its capital resources. The company's current market value of equity is 150 million, and the market value of its debt is 100 million. The cost of equity is estimated at 9%, while the cost of debt stands at 5%. Considering the corporate tax rate is 25%, the company wants to determine its WACC to make informed investment decisions.

Function used as instruction:

```
def calculate_wacc(market_value_equity: float, market_value_debt: float,
    cost_of_equity: float, cost_of_debt: float, tax_rate: float) -> float:
    """
    Calculate the Weighted Average Cost of Capital (WACC).

    This function calculates the WACC, which represents a firm's average cost
    of financing from all sources, weighted by their respective usage in the
        overall
    capital structure. It gives an overall measure of the firm's cost of capital.

    Args:
        market_value_equity (float): Market value of the firm's equity.
        market_value_debt (float): Market value of the firm's debt.
        cost_of_equity (float): Cost of equity (Re) represented as a decimal (e.g.,
             0.08 for 8%).
        cost_of_debt (float): Cost of debt (Rd) represented as a decimal (e.g.,
            0.04 for 4%).
        tax_rate (float): Corporate tax rate (Tc) represented as a decimal (e.g.,
            0.30 for 30%).

    ...
    """
    total_value = market_value_equity + market_value_debt
    equity_weight = market_value_equity / total_value
    debt_weight = market_value_debt / total_value
    wacc = (equity_weight * cost_of_equity) + (debt_weight * cost_of_debt * (1 -
        tax_rate))

    return wacc
```

Table 15: The newly added generated data in FinanceReasoning, combined with the structured financial function library built as an instruct.

RAG Prompt

```
SYSTEM_INPUT = '''You are a financial expert, you are supposed to answer the given question. You need to first think
↪  through the problem step by step, identifying the exact variables and values, and documenting each necessary step.
↪  Then you are required to conclude your response with the final answer in your last sentence as 'Therefore, the answer
↪  is {final answer}'. The final answer should be a numeric value.'''

USER_INPUT = '''The following question context is provided for your reference:
{question_context}
Question:
{question_question}

The following are the financial functions for your reference:
{financial_function}
'''
```

Figure 10: RAG Prompt

LLM Instruct Prompt

```
SYSTEM_INPUT = '''You are an expert in financial analysis and Python programming. Your task is to analyze a given
↪  financial problem and its context, and generate a concise and precise retrieval query that can be used to search a
↪  financial function library for relevant functions.

The retrieval query should be based on the following principles:
1. Intent Recognition: Identify the core intent of the problem (e.g., calculating present value, estimating risk,
↪  optimizing portfolio).
2. Applicability: Consider the scope and applicability of the function (e.g., time period, asset type, market conditions).
3. Constraints: Include any constraints or limitations relevant to the problem (e.g., input data format, computational
↪  complexity).
4. Generalization: Ensure the query is generalized enough to match multiple potential functions but specific enough to
↪  exclude irrelevant ones.

Your output should be a single, well-structured retrieval query that captures the essence of the problem and its
↪  requirements. The query should be concise, clear, and suitable for vector-based similarity search against a financial
↪  function library.

Here is an example of a retrieval query for a financial problem:
Question: A company wants to evaluate the profitability of a potential investment project. The project involves an initial
↪  investment of $100,000 and is expected to generate annual cash flows of $30,000 for the next 5 years. The company
↪  uses a discount rate of 8% to evaluate such projects. The cash flows are assumed to occur at the end of each year?
Retrieval Query: What function calculates the net present value of a series of annual cash flows with a fixed discount
↪  rate, supports a single initial investment, and handles positive cash flows?
'''

USER_INPUT = '''I have a financial problem and its context. Please analyze the problem and generate a retrieval query
↪  that can be used to search a financial function library for relevant functions. The retrieval query should capture
↪  the core intent of the problem, its applicability, constraints, and generalization.

Here is the financial problem and its context:
{financial_question}

Please generate a retrieval query based on the following guidelines:
1. Intent Recognition: Identify the core intent of the problem.
2. Applicability: Consider the scope and applicability of the function.
3. Constraints: Include any constraints or limitations relevant to the problem.
4. Generalization: Ensure the query is generalized enough to match multiple potential functions but specific enough to
↪  exclude irrelevant ones.

Your output should be a single, well-structured retrieval query. Do not include additional explanations or examples.
Please generate the retrieval query based on the provided financial problem and context.
'''
```

Figure 11: LLM Instruct Prompt

Judge Prompt

```
SYSTEM_INPUT = '''You are a financial expert, you are supposed to judge whether the given financial function is useful
↪  for answering the question.

For each function, follow these guidelines:
1. Determine if the function can directly address the userâĂŹs problem, considering the function's purpose, input
↪  parameters, and return values.
2. Consider the applicability range of the function, assumptions, limitations, and restrictions when evaluating if itâĂŹs
↪  relevant.
3. If the function can effectively contribute to solving the problem or is essential for the calculation or analysis
↪  required, respond with [USEFUL].
4. If the function cannot effectively help in solving the problem, or is irrelevant based on its scope and assumptions,
↪  respond with [USELESS].

Use financial domain knowledge to ensure that each judgment is precise and aligned with common practices for
↪  problem-solving in the finance domain.
'''

USER_INPUT = '''Given a financial question and financial functions, I want you to analyze each of these function to
↪  assess if it can be useful in solving the question.

For each financial function:
1. You need to decide if it is useful based on its fit with the problemâĂŹs requirements and constraints.
2. If the function is relevant to solving my problem, output [USEFUL].
3. If it is not helpful, output [USELESS].
Do not include any additional explanation, just the relevant outputs for each function.

Question:
{financial_question}

Functions:
{financial_function}

Output the results in the following format:
[USEFUL, USELESS, USELESS, ...]
'''
```

Figure 12: Judge Prompt

Question: According to the given table, can you compute the value added from the choice of security (expressed as a percentage)? Answer to a single decimal place.
question_id: test-2193
ground_truth: 5.6
context:

	Portfolio return	Benchmark return	Portfolio weight	Benchmark weight
Domestic	25%	15%	50	35
Bond	9%	5%	30	35
International	15%	18%	20	30

R1:
To compute the value added from the choice of security (selection effect)...
Domestic:

$$(25\% - 15\%) \times 35\% = 3.5\%$$

Bond:

$$(9\% - 5\%) \times 35\% = 1.4\%$$

International:

$$(15\% - 18\%) \times 30\% = -0.9\%$$

Total Selection Effect:

$$3.5\% + 1.4\% - 0.9\% = 4.0\%$$

Therefore, the answer is 4.0.

R1 + LLM instruct(4o):
Instruct from 4o:
What function calculates the security selection effect in portfolio performance attribution...
The function retrieved and considered useful by 4o:

```
def calculate_security_selection_effect(portfolio_returns: list, benchmark_returns:
     list, portfolio_weights: list) -> float:
    ...
    total_security_selection_effect = sum((pr - br) * pw for pr, br, pw in zip(
        portfolio_returns, benchmark_returns, portfolio_weights)

    return round(total_security_selection_effect * 100, 1)
```

Domestic:

$$(25\% - 15\%) \times 50\% = 5.0\%$$

Bond:

$$(9\% - 5\%) \times 30\% = 1.2\%$$

International:

$$(15\% - 18\%) \times 20\% = -0.6\%$$

The final selection effect is:

$$5.0\% + 1.2\% - 0.6\% = 5.6\%$$

Output: 5.6

Table 16: A comparison of Performance between R1 and LLM-instruct Methods in FinanceReasoning.

Summary Program Prompt

```
SYSTEM_INPUT = '''
You are a financial expert and a skilled programmer, you are supposed to generate a Python program to answer the given
↪  question. The returned value of the program is supposed to be the answer. The code should strictly follow the
↪  reasoning process and logic provided by another reasoning model (DeepSeek-R1 or OpenAI-o1), but it must not use any
↪  intermediate results from R1 or o1. Instead, you should implement the calculations from scratch to ensure accuracy.
### Requirements:
1. Follow reasoning model's Reasoning Logic: Carefully analyze the reasoning process provided by reasoning model and
↪  ensure your Python code aligns with it step-by-step.
2. Avoid reasoning model's Intermediate Results: Do not use any intermediate values from reasoning model. Implement all
↪  calculations independently.
3. Precision and Accuracy: Ensure all numerical calculations are precise and free from errors. Use appropriate data types
↪  and rounding rules as needed.

Here is an example of the Python program:
```python
def solution():
    # Define variables name and value
    revenue = 600000
    avg_account_receivable = 50000

    # Do math calculation to get the answer
    receivables_turnover = revenue / avg_account_receivable
    answer = 365 / receivables_turnover

    # return answer
    return answer
```
'''

USER_INPUT = '''
Please generate a Python program to answer the given question. The format of the program should be the following:
```python
def solution():
    # Define variables name and value

    # Do math calculation to get the answer

    # return answer
```

Continue your output:
```python
def solution():
    # Define variables name and value

The following are the summary of reasoning process provided by another reasoning model you should strictly follow.
{summary of reasoning process}
'''
```

Figure 13: Summary Program Prompt
```

Question: What is the total amount of money that John will pay towards the principal in the first 12 months of his mortgage? Answer to the nearest dollar.
question_id: test-2224
ground_truth: 4597
context:

Loan Amount ($)	250,000
Annual Interest Rate (%)	3.75
Loan Term (Years)	30

R1:
To determine the total amount John will pay towards the principal in the first 12 months of his mortgage, we use the following steps:
1. **Calculate the monthly payment (M):**
Using the formula for a fixed-rate mortgage...
2. **Total principal paid after 12 months:**
The formula for the remaining balance B_{12} after 12 payments is:

$$B_{12} = 250,000 \times \frac{(1.003125)^{360} - (1.003125)^{12}}{(1.003125)^{360} - 1} \approx 245,366.50 \qquad (245,403.05)$$

So, total principal paid is:

$$250,000 - 245,366.50 = 4,633.50 \quad \text{(rounded to 4634)}$$

3. **Validation via amortization schedule:**
Summing the principal portions of each monthly payment (rounded to the nearest cent) results in a total principal paid of ...
Therefore, the answer is 4634.

R1 + Claude:
Based on the reasoning process, I'll create a Python program that calculates the total principal paid in the first 12 months using the formula method:

```
def solution():
    # Define variables name and value
    loan_amount = 250000
    annual_rate = 0.0375
    monthly_rate = annual_rate / 12
    #...
    # Calculate remaining balance after 12 months
    remaining_balance = loan_amount * (power_n - power_k) / (power_n - 1)
    # Total principal paid is original loan amount minus remaining balance
    principal_paid = loan_amount - remaining_balance
    # ...

    return answer
```

Program Output: 4597

Table 17: A comparison of the programming approach using R1 alone versus R1 combined with Claude, demonstrated with an example problem. The solution using only R1 is incorrect, whereas the R1+Claude approach yields the correct answer.

Model	Organization	Size	Notes	Source
DeepSeek-R1	DeepSeek	671B	MoE	`deepseek-ai/DeepSeek-R1`
DeepSeek-V3	DeepSeek	671B	MoE	`deepseek-ai/DeepSeek-V3`
Claude 3.5 Sonnet	Anthropic	–		`claude-3.5-sonnet-1022`
Gemini 2.0 Flash Thinking Experimental	Google	–		`gemini-2.0-flash-thinking-exp-01-21`
Gemini 2.0 Pro Experimental	Google	–		`gemini-2.0-pro-exp-02-05`
QwQ-32B-Preview	Alibaba	32B		`Qwen/QwQ-32B-Preview`
Qwen2.5-Max	Alibaba	–	MoE	`qwen-max-2025-01-25`
Llama 3.1	Meta	405B		`meta-llama/Llama-3.1-405B-Instruct`
Llama 3.3	Meta	70B		`meta-llama/Llama-3.3-70B-Instruct`
OpenAI-o1	OpenAI	–		`o1-2024-12-17`
OpenAI-o1-mini	OpenAI	–		`o1-mini-2024-09-12`
OpenAI-o3-mini	OpenAI	–		`o3-mini-2025-01-31`
GPT-4o	OpenAI	–		`gpt-4o-2024-11-20`

Table 18: Details of the organization and model source (*i.e.,* model version for proprietary models, and Huggingface model name for open-source models) for the LLMs evaluated in FinanceReasoning.

COT Prompt

```
SYSTEM_INPUT = '''You are a financial expert, you are supposed to answer the given question.
You need to first think through the problem step by step, identifying the exact variables and values,
and documenting each necessary step. Then you are required to conclude your response with the final
answer in your last sentence as 'Therefore, the answer is {final answer}'. The final answer should
be a numeric value.'''

USER_INPUT = '''The following question context is provided for your reference:
{question_context}
Question:
{question_question}
Let's think step by step to answer the given question.
'''
```

Figure 14: Chain of Thought Prompt

POT Prompt

```
SYSTEM_INPUT = '''You are a financial expert, you are supposed to generate a Python program to answer the given question.
↪ The returned value of the program is supposed to be the answer. Here is an example of the Python program:
```python
def solution():
    # Define variables name and value
    revenue = 600000
    avg_account_receivable = 50000

    # Do math calculation to get the answer
    receivables_turnover = revenue / avg_account_receivable
    answer = 365 / receivables_turnover

    # return answer
    return answer
```
'''

USER_INPUT = '''The following question context is provided for your reference:
{question_context}
Question:
{question_question}
Please generate a Python program to answer the given question. The format of the program should be the following:
```python
def solution():
    # Define variables name and value

    # Do math calculation to get the answer

    # return answer
```

Continue your output:
```python
def solution():
    # Define variables name and value
```
'''
```

Figure 15: Program of Thought Prompt

Model	Tokens (k) for Easy/Medium/Hard	
	CoT	PoT
OpenAI-o1-mini	417 / 1074 / 944	341 / 1015 / 953
OpenAI-o1	695 / 1500 / 1242	505 / - / -
Gemini 2.0 Flash Thinking Experimental	154 / 442 / 311	62 / 164 / 140
QwQ-32B-Preveiw	556 / 1514 / 1307	246 / 609 / 543
DeepSeek-R1	742 / 1499 / 1274	743 / 1593 / 1257
OpenAI-o3-mini	374 / 771 / 712	328 / 664 / 618
Gemini 2.0 Pro Experimental	128 / 295 / 206	60 / 143 / 112
GPT-4o	173 / 428 / 298	54 / 133 / 105
Claude 3.5 Sonnet	90 / 261 / 201	110 / 335 / 275
Deepseek-V3	133 / 322 / 237	56 / 144 / 114
Llama 3.3	145 / 350 / 263	60 / 156 / 130
Llama 3.1	125 / 318 / 235	53 / 139 / 109
Qwen2.5-Max	215 / 526 / 373	58 / 139 / 108

Table 19: Token usage across different models

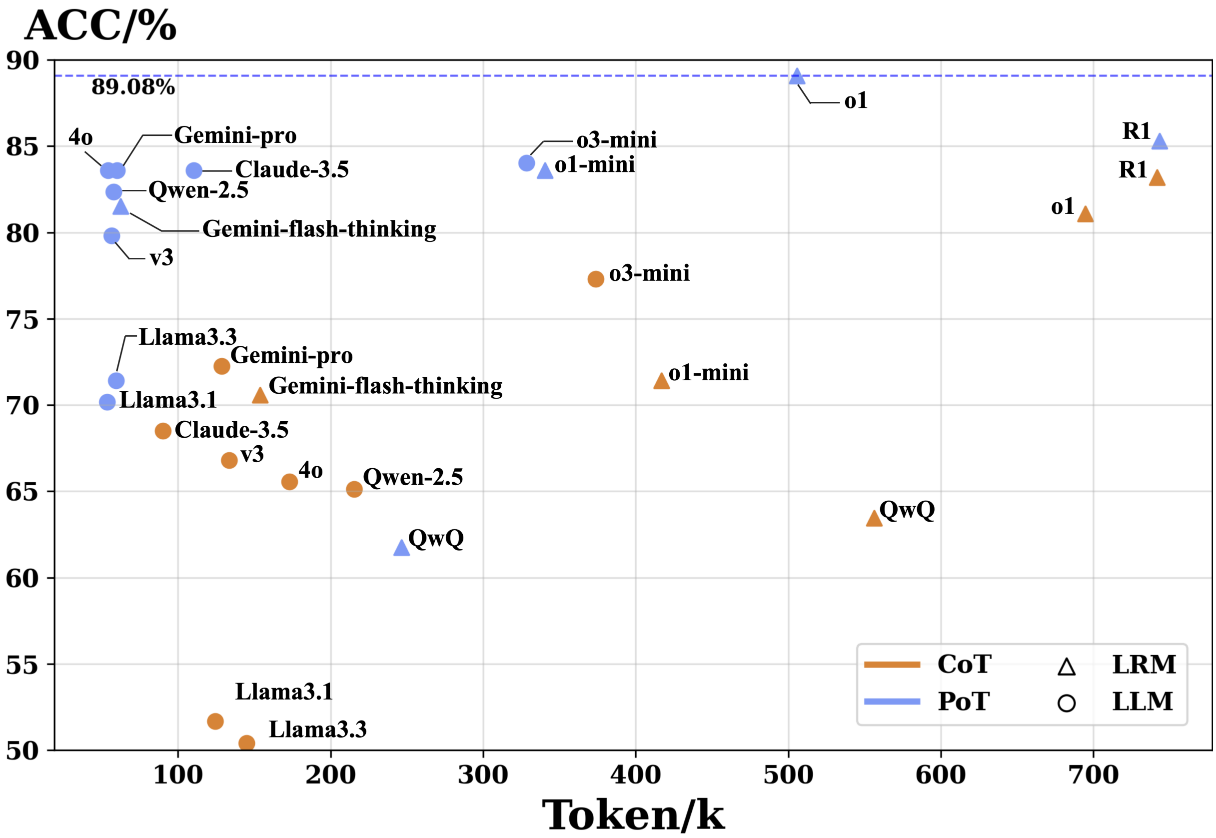

Figure 16: Relationship between token consumption and accuracy for LLMs and LRMs using CoT and PoT Prompting. R1: DeepSeek-R1, o1: OpenAI-o1, o3-mini: OpenAI-o3-mini, Gemini-flash-thinking: Gemini 2.0 Flash Thinking Experimental, QwQ: QwQ-32B-Preview, Gemini-2.0-pro: Gemini 2.0 Pro Experimental, 4o: OpenAI-4o, Claude: Claude 3.5 Sonnet, V3: DeepSeek-V3, Qwen-2.5: Qwen2.5-Max.