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

Zichen Tang Haihong E*[*] Ziyan Ma Haoyang He Jiacheng Liu
Zhongjun Yang Zihua Rong Rongjin Li Kun Ji Qing Huang Xinyang Hu
Yang Liu Qianhe Zheng

Beijing University of Posts and Telecommunications

`bupt-reasoning-lab.github.io/FinanceReasoning`
`BUPT-Reasoning-Lab/FinanceReasoning` 🤗 `BUPT-Reasoning-Lab/FinanceReasoning`

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.

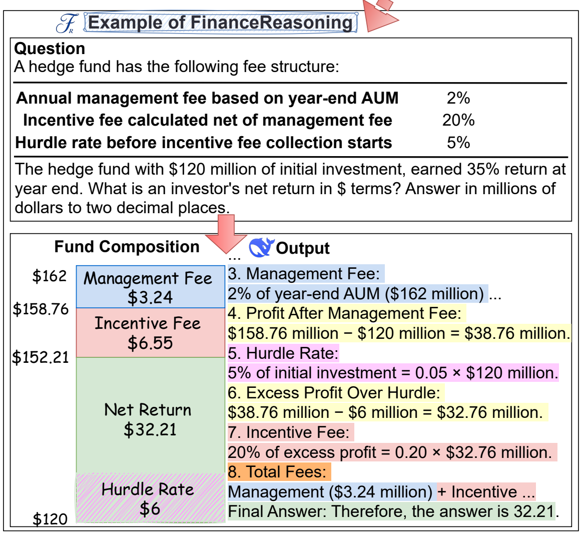
1 Introduction

> "*If you cannot measure it, you cannot improve it.*"
> — Lord Kelvin

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; DeepSeek-AI 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., 2025; Chen et al., 2021a), math (Mao et al., 2024; Lightman et al., 2024), and science (Lu et al., 2024; Yue et al., 2024; Wang et al., 2024).

Dataset Difficulty Distribution and Knowledge Coverage

Dataset	Subset	Easy	Medium	Hard	Knowledge Coverage
CodeFinQA	Test (795)	559	239	6	45.70%
CodeTAT-QA	Test (288)	189	99	0	14.66%
FinCode	Test (47)	6	27	14	15.46%
FinanceMath	Val (200)	34	104	62	18.89%
FinanceReasoning	**All (2,238)**	**1,000**	**1,000**	238	**67.76%**



Question
A hedge fund has the following fee structure:

Annual management fee based on year-end AUM	2%
Incentive fee calculated net of management fee	20%
Hurdle rate before incentive fee collection starts	5%

The hedge fund with $120 million of initial investment, earned 35% return at year end. What is an investor's net return in $ terms? Answer in millions of dollars to two decimal places.

...
3. Management Fee:
2% of year-end AUM ($162 million) ...
4. Profit After Management Fee:
$158.76 million − $120 million = $38.76 million.
5. Hurdle Rate:
5% of initial investment = 0.05 × $120 million.
6. Excess Profit Over Hurdle:
$38.76 million − $6 million = $32.76 million.
7. Incentive Fee:
20% of excess profit = 0.20 × $32.76 million.
8. Total Fees:
Management ($3.24 million) + Incentive ...
Final Answer: Therefore, the answer is 32.21.

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.

However, as illustrated in Figure 1, more real-world domain-specific numerical reasoning tasks

---

[*] Corresponding author.

Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 15721–15749
July 27 - August 1, 2025 ©2025 Association for Computational Linguistics

(a) Re-annotation

Dataset	Subset	Answer Corrections	Question Disambiguations	Total
CodeFinQA	Test (795)	55 (6.92%)	58 (7.30%)	113 (14.21%)
CodeTAT-QA	Test (288)	19 (6.60%)	9 (3.13%)	28 (9.72%)
FinCode	Test (47)	6 (12.77%)	1 (2.13%)	7 (14.89%)
FinanceMath	Val (200)	15 (7.50%)	45 (22.50%)	60 (30.00%)

(b) Re-evaluation

Dataset (Criteria)	DeepSeek-V3	DeepSeek-R1	△ (R1 - V3)	% Relative Increase (R1 vs. V3)
CodeFinQA (Silver)	**61.76**	60.88	-0.88	-1.42%
CodeFinQA (Gold)	85.41	**87.42**	2.01	2.35%
CodeTAT-QA (Silver)	89.24	**89.58**	0.34	0.38%
CodeTAT-QA (Gold)	91.67	**93.75**	2.08	2.27%
FinCode (Silver)	80.85	**82.98**	2.13	2.63%
FinCode (Gold)	87.72	**95.74**	8.02	9.14%
FinanceMath (Silver)	58.50	**71.00**	12.50	21.37%
FinanceMath (Gold)	59.50	**83.50**	24.00	40.34%

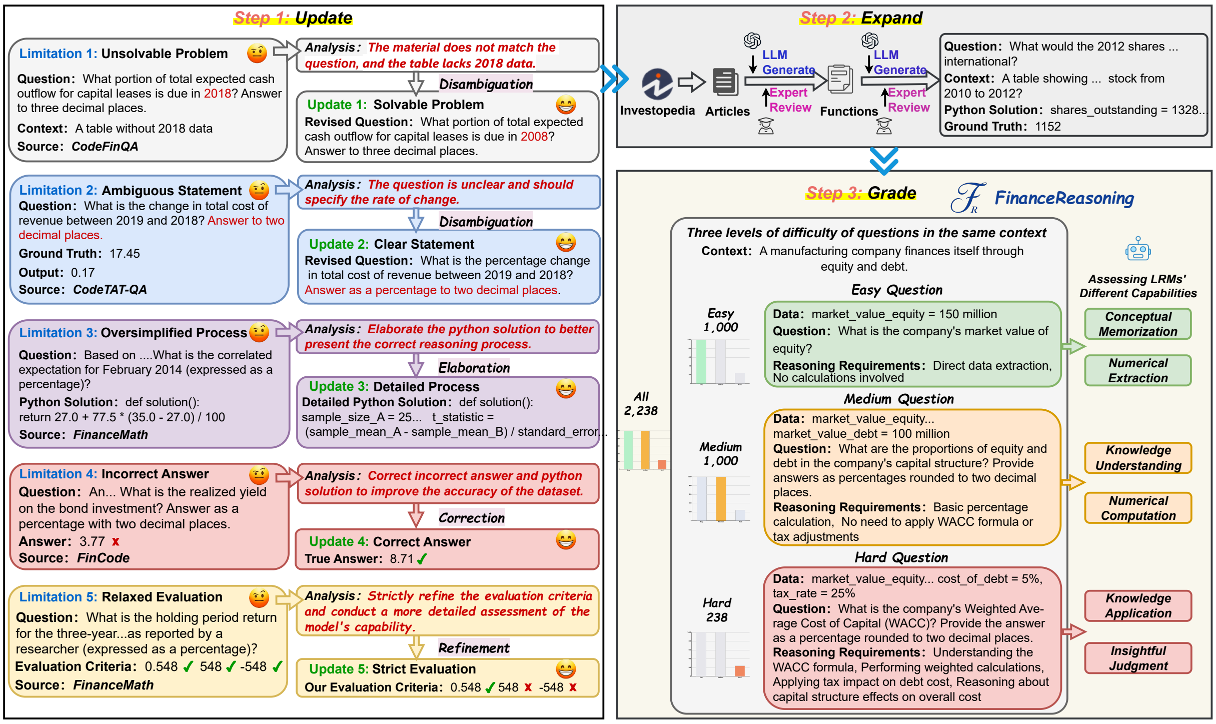
Table 1: (a) **Re-annotation**: For the *test* or *validation* sets of four datasets, the proportion of updated examples ranges from 9.72% to 30.00%. (b) **Re-evaluation**: **Silver** denotes the Accuracy on the original dataset under relaxed criteria, while **Gold** represents the results on the re-annotated dataset under strict criteria. Rigorous revision and evaluation reveal LRMs' actual performance and DeepSeek-R1's significant improvement over DeepSeek-V3.

(*e.g.,* financial quantitative analysis) challenge LRMs to deeply understand and apply domain-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; DeepSeek-AI 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). We also evaluate seven LLMs without reasoning-specific enhancement (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



Limitation 1: **Unsolvable Problem**
Question: What portion of total expected cash outflow for capital leases is due in 2018? Answer to three decimal places.
Context: A table without 2018 data
Source: ***CodeFinQA***

Analysis: ***The material does not match the question, and the table lacks 2018 data.***

Update 1: **Solvable Problem**
Revised Question: What portion of total expected cash outflow for capital leases is due in 2008? Answer to three decimal places.

Limitation 2: **Ambiguous Statement**
Question: What is the change in total cost of revenue between 2019 and 2018? Answer to two decimal places.
Ground Truth: 17.45
Output: 0.17
Source: ***CodeTAT-QA***

Analysis: ***The question is unclear and should specify the rate of change.***

Update 2: **Clear Statement**
Revised Question: What is the percentage change in total cost of revenue between 2019 and 2018? Answer as a percentage to two decimal places.

Limitation 3: **Oversimplified Process**
Question: Based on ....What is the correlated expectation for February 2014 (expressed as a percentage)?
Python Solution: def solution():
return 27.0 + 77.5 * (35.0 - 27.0) / 100
Source: ***FinanceMath***

Analysis: ***Elaborate the python solution to better present the correct reasoning process.***

Update 3: **Detailed Process**
Detailed Python Solution: def solution():
sample_size_A = 25... t_statistic =
(sample_mean_A - sample_mean_B) / standard_error...

Limitation 4: **Incorrect Answer**
Question: An... What is the realized yield on the bond investment? Answer as a percentage with two decimal places.
Answer: 3.77 ✗
Source: ***FinCode***

Analysis: ***Correct incorrect answer and python solution to improve the accuracy of the dataset.***

Update 4: **Correct Answer**
True Answer: 8.71 ✓

Limitation 5: **Relaxed Evaluation**
Question: What is the holding period return for the three-year...as reported by a researcher (expressed as a percentage)?
Evaluation Criteria: 0.548 ✓ 548 ✓ -548 ✓
Source: ***FinanceMath***

Analysis: ***Strictly refine the evaluation criteria and conduct a more detailed assessment of the model's capability.***

Update 5: **Strict Evaluation**
Our Evaluation Criteria: 0.548 ✓ 548 ✗ -548 ✗

Question: What would the 2012 shares ... international?
Context: A table showing ... stock from 2010 to 2012?
Python Solution: shares_outstanding = 1328...
Ground Truth: 1152

Three levels of difficulty of questions in the same context
Context: A manufacturing company finances itself through equity and debt.

Easy Question
Data: market_value_equity = 150 million
Question: What is the company's market value of equity?
Reasoning Requirements: Direct data extraction, No calculations involved

Medium Question
Data: market_value_equity...
market_value_debt = 100 million
Question: What are the proportions of equity and debt in the company's capital structure? Provide answers as percentages rounded to two decimal places.
Reasoning Requirements: Basic percentage calculation, No need to apply WACC formula or tax adjustments

Hard Question
Data: market_value_equity... cost_of_debt = 5%, tax_rate = 25%
Question: What is the company's Weighted Average Cost of Capital (WACC)? Provide the answer as a percentage rounded to two decimal places.
Reasoning Requirements: Understanding the WACC formula, Performing weighted calculations, Applying tax impact on debt cost, Reasoning about capital structure effects on overall cost

Figure 2: Overview of FinanceReasoning's construction.

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

As illustrated in Figure 2, 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 problems.

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., 2021b; Zhu et al., 2021; Zhao et al., 2024), including unsolvable problems, ambiguous statements (*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), oversimplified processes, incorrect answers, and relaxed 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 (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.

- **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 relaxed: 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 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[1]. 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. 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. Detailed examples of financial functions are illustrated in Figure 3 and Table 15.

Financial Function

```
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 3: An example of financial function from the constructed function library to calculate net return.

2.3 Expansion of Data Annotation

Existing financial question-answering datasets (*e.g.,* CodeFinQA, CodeTAT-QA (Krumdick et al., 2024)) 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 real 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

---

[1] `https://www.investopedia.com`

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.

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.

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. The annotation example is provided in Appendix D.2.

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:

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

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.

Property	**Value**
Financial Function Library	
# Total Functions	3,313
# Operators (Avg)	2.85
# Arguments (Avg)	2.64
# Lines of Code (Avg)	3.45
# Financial Concepts Involved	1,864
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 financial function library and FinanceReasoning dataset (Avg values of three subsets).

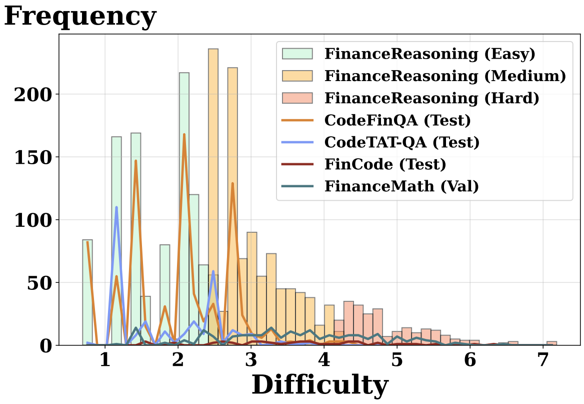


Figure 4: The difficulty distribution of FinanceReasoning, compared with four existing datasets, shows a notably higher proportion of medium and hard problems, presenting greater challenges for complex reasoning.

As illustrated in Table 2 and Figure 4, 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 19 illustrates the exact model versions we used.

Model	Size	Notes	Hard		Medium		Easy		Avg.	
			CoT	PoT	CoT	PoT	CoT	PoT	CoT	PoT
Large Reasoning Models (LRMs)										
OpenAI o1			81.1	**89.1**	89.7	–	88.0	–	86.3	–
DeepSeek-R1	671B	MoE	**83.2**	85.3	**91.1**	**89.8**	**89.8**	**89.2**	**88.0**	**88.1**
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
Claude 3.5 Sonnet			68.5	83.6	85.7	88.2	87.7	88.4	80.6	86.7
GPT-4o			65.6	83.6	84.6	87.9	86.8	88.1	79.0	86.5
Qwen2.5-Max		MoE	65.1	82.4	87.2	86.5	89.6	89.1	80.6	86.0
DeepSeek-V3	671B	MoE	66.8	75.6	85.2	87.3	87.2	86.9	79.7	80.7
Llama 3.3	70B		50.4	71.4	79.2	85.9	83.3	84.8	71.0	80.7
Llama 3.1	405B		51.7	70.2	81.7	87.7	84.1	85.8	72.5	81.2

Table 3: Results of different models using CoT and PoT prompting methods on the different subsets of FinanceReasoning. We use Accuracy of *Hard* subset using PoT prompting as the ranking indicator of model performance. The results underscore the superior performance of LRMs (*i.e.,* OpenAI o1 and DeepSeek-R1) with PoT.

3.1 Large Language Models

We focused on evaluating six of the most powerful LRMs 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** (DeepSeek-AI et al., 2025) enhances its reasoning capabilities through multi-stage training with reinforcement learning, using a minimal amount of supervised fine-tuning 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.
- **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 CoT (Wei et al., 2022) and PoT (Chen et al., 2023b) to achieve optimal performance and make comparisons.

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 in CoT setting, and executing the program from the output in PoT setting. Finally, we perform a strict accuracy evaluation comparing the numerical results with the ground truth within 0.2% error margin.

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 Results (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 with 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 CoT, 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.

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 80 DeepSeek-R1 failure cases with PoT, with stratified sampling (20 *Easy* /20 *Medium* /40 *Hard*). We summarize four types of error in the current LRMs on challenging domain-specific reasoning problems, some of which involve compound errors. The detailed error distribution is shown in Table 4. More details of error cases are provided in Appendix B.

- **Misunderstanding of Problem**: The model incorrectly interprets the question and context due to a lack of financial knowledge.
- **Formula Application Errors**: Owing to inexperience in financial reasoning, the model uses an incorrect formula that does not correspond to the specified conditions of the problem.

Subset	M	F	E	C
Easy	35% (7/20)	5% (1/20)	15% (3/20)	45% (9/20)
Medium	25% (5/20)	30% (6/20)	5% (1/20)	40% (8/20)
Hard	20% (8/40)	35% (14/40)	7.5% (3/40)	37.5% (15/40)

Table 4: Error distribution across difficulty subsets. M means Misunderstanding of Problem, F means Formula Application Errors, E means Numerical Extraction Errors, C means Numerical Calculation Errors.

- **Numerical Extraction Errors**: 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**: 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.

As shown in Table 4, we observe that:

- **Formula Application Errors** and **Numerical Calculation Errors** increase with difficulty, especially the former, which supports the effectiveness of our difficulty categorization. In contrast, **Numerical Extraction Errors** remain consistently low across all subsets. These findings suggest that the model's domain-specific knowledge comprehension and application capabilities should be enhanced through retrieval-augmented generation (RAG) or reinforcement learning (RL), thereby improving its performance on complex reasoning tasks in specialized domains.
- **Numerical Calculation Errors** occur at similar rates across all difficulty levels. Since **Formula Application Errors** are less frequent in easier subsets, improving the handling of calculation-related issues, such as unit conversions, percentage formats, significant digits, and sign correctness, becomes key to boosting accuracy on *Easy* and *Medium* problems. The results indicate the necessity of incorporating external computational tools or model collaboration to strengthen numerical computation capabilities, which could mitigate errors caused by calculation inaccuracies on relatively simple problems.

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 LLMs in domain-specific 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.

- **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).

Knowledge Augmentation Results As shown in Table 5, 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 Appendix E.

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

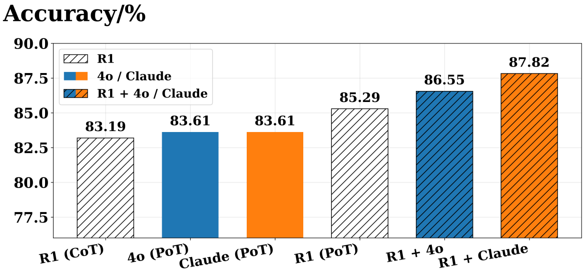


Figure 5: Results of different model combinations and individual models. R1 means DeepSeek-R1, 4o means GPT-4o, Claude means Claude 3.5 Sonnet.

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.

Specifically, we employ **DeepSeek-R1** as Reasoner, and the best-performing models in PoT setting, **Claude 3.5 Sonnet** and **GPT-4o**, as Programmers, respectively. As shown in Figure 5, 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 Appendix F.

4.5 PoT vs. CoT (RQ5)

Based on an analysis of the *Hard* subset, we observe that PoT exhibits a markedly stronger performance than 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 20.

As shown in Figure 6, LLMs using 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* subset 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 (DeepSeek-AI et al., 2025) has significantly improved the performance of LLMs in complex reasoning tasks in domains such as code (Jain et al., 2025; Chen et al., 2021a), math (Mao et al., 2024; Lightman et al., 2024), and science (Lu et al., 2024; Yue et al., 2024; Wang et al., 2024). Among these LRMs (OpenAI, 2024d,c, 2025; DeepSeek-AI 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 relaxed 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.

Limitations

In this work, we introduce FinanceReasoning, a benchmark dataset designed to evaluate and enhance LLMs 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. We commit to incorporating additional FinanceMath *test* subset should it become publicly available in the future, subject to the same rigorous validation process. (4) While FinanceReasoning provides comprehensive evaluation criteria, its long-term utility depends on adapting to evolving challenges. To address this, we commit to maintaining and periodically updating the benchmark through versioned releases on Hugging Face Datasets. (5) We clarify that FinanceReasoning evaluates *perfect-information* scenarios where all solution prerequisites are explicitly provided, thus focusing on the upper bounds of LRMs' precision in deterministic calculations. This design intentionally distinguishes reasoning errors (*e.g.,* outputting -3.1 vs. 310,000 given clear constraints) from ambiguity resolution capabilities. While the important direction of modeling LRMs' proactive clarification-seeking behavior under insufficient conditions falls outside this work's scope, we will pursue it in future research through controlled ambiguity injection and interactive evaluation protocols.

Ethical Considerations

This work complies with ACL ethics guidelines. FinanceReasoning is released under the Creative Commons Attribution 4.0 International License (CC BY 4.0), permitting free use with proper attribution. The codebase is distributed under the Apache License 2.0, ensuring compatibility with commercial and open-source ecosystems.

Acknowledgements

This work is supported by the National Natural Science Foundation of China (Grant Nos. 62176026, 62473271), the Beijing Natural Science Foundation (Grant No. QY24214), and the BUPT Innovation and Entrepreneurship Support Program (Grant Nos. 2025-YC-A033, 2025-YC-A042). This work is also supported by the Engineering Research Center of Information Networks, Ministry of Education, China. We would also like to thank the anonymous reviewers and area chairs for constructive discussions and feedback.

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 *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 6325–6341, Singapore. Association for Computational Linguistics.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 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, 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, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, 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, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:2501.12948.

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*.

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. 2025. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations*.

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, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2024. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*.

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 mathematical reasoning of foundation models in visual contexts. 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 Back. 2024. Reasoning with large language models, a survey. *Preprint*, arXiv: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. *Nature*, 625(7995):468–475.

Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, Chuning Tang, Congcong Wang, Dehao Zhang, Enming Yuan, Enzhe Lu, Fengxiang Tang, Flood Sung, Guangda Wei, Guokun Lai, Haiqing Guo, Han Zhu, Hao Ding, Hao Hu, Hao Yang, Hao Zhang, Haotian Yao, Haotian Zhao, Haoyu Lu, Haoze Li, Haozhen Yu, Hongcheng Gao, Huabin Zheng, Huan Yuan, Jia

Chen, Jianhang Guo, Jianlin Su, Jianzhou Wang, Jie Zhao, Jin Zhang, Jingyuan Liu, Junjie Yan, Junyan Wu, Lidong Shi, Ling Ye, Longhui Yu, Mengnan Dong, Neo Zhang, Ningchen Ma, Qiwei Pan, Qucheng Gong, Shaowei Liu, Shengling Ma, Shupeng Wei, Sihan Cao, Siying Huang, Tao Jiang, Weihao Gao, Weimin Xiong, Weiran He, Weixiao Huang, Wenhao Wu, Wenyang He, Xianghui Wei, Xianqing Jia, Xingzhe Wu, Xinran Xu, Xinxing Zu, Xinyu Zhou, Xuehai Pan, Y. Charles, Yang Li, Yangyang Hu, Yangyang Liu, Yanru Chen, Yejie Wang, Yibo Liu, Yidao Qin, Yifeng Liu, Ying Yang, Yiping Bao, Yulun Du, Yuxin Wu, Yuzhi Wang, Zaida Zhou, Zhaoji Wang, Zhaowei Li, Zhen Zhu, Zheng Zhang, Zhexu Wang, Zhilin Yang, Zhiqi Huang, Zihao Huang, Ziyao Xu, Zonghan Yang, and Zongyu Lin. 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. In *Workshop on Reasoning and Planning for Large Language Models*.

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, Chenyang Shao, Yuwei Yan, Qinglong Yang, Yiwen Song, Sijian Ren, Xinyuan Hu, Yu Li, Jie Feng, Chen Gao, and Yong Li. 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 FinanceReasoning

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:

- **1,420 updated existing problems**, carefully reviewed and refined from existing datasets, involving **Disambiguation**, **Elaboration** and **Correction**.
- **908 newly generated problems**, synthesized by LLMs (*i.e.,* 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 models' capabilities, we introduce a tiered classification of difficulty based on computational complexity and reasoning depth:

- *Easy*: Direct data extraction and minimal computation, involving simple value retrieval with $\leq$2-step reasoning (*e.g.,* YoY growth rate calculation).
- *Medium*: Basic percentage operations and standard financial formula application without structural modifications, requiring $\geq$3-step computations with multiple variables (*e.g.,* Lorenz curve area calculation needing 35 precise steps across 5 segments with 4 boundary checks).
- *Hard*: Multi-stage weighted computations and complex financial concept integration, demanding $\sim$8-step reasoning with 10+ variables (*e.g.,* 80-10-10 mortgage calculations requiring 22+ chained operations).

This construction process, illustrated in Figure 2, underscores FinanceReasoning's **credibility**, **comprehensiveness**, and **challenge**, setting a new standard for evaluating financial numerical reasoning in AI models.

B Error Cases

For each of the four error types, we provide representative DeepSeek-R1 examples in Table 6, Table 7, Table 8, and Table 9, respectively.

C Details of Public Dataset Updates

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

C.1 Cases of Dataset Update

- **Unsolvable Problem**: Problems that either lack sufficient data or are inherently unsolvable. To address this, the case in Table 10 was updated by applying the **Disambiguation** strategy to refine the question.
- **Ambiguous Statement**: Questions or statements that are unclear or open to multiple interpretations. In the case of Table 11, we updated the question by applying the **Disambiguation** strategy.
- **Oversimplified Process**: 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 12 by applying an **Elaboration** approach, ensuring a more detailed and structured problem-solving process.
- **Incorrect Answer**: Answers that are mathematically or conceptually incorrect, leading to misleading conclusions. In Table 13, the answer was refined by applying a **Correction** process.
- **Relaxed Evaluation**: Issues with evaluation standards, where answers that differ in precision (*e.g.,* 0.01, 10 and 1%) are considered equivalent. We refined the answer in Table 14 by adopting a strict standard.

C.2 Difficulty Distribution Analysis

The distribution of difficulty among public datasets and FinanceReasoning is shown in Figure 4. As can be seen, the difficulty distribution of FinanceReasoning surpasses previous datasets with respect to the number of medium and hard-level questions, thereby facilitating a more accurate reflection of the model's reasoning ability.

D Details of Dataset Expansion

D.1 Example of Financial Function

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 LLMs through well-structured documentation. For detailed information, refer to Table 15.

D.2 Example of Financial Problem

The expanded questions are based on the financial functions and concepts extracted earlier, ensuring that the questions are both relevant to the domain and challenging enough to test deep reasoning. For detailed information, refer to Table 16.

E Case of Knowledge Augmentation

The comparison between the performance of **DeepSeek-R1** and **knowledge augmentation** are shown in Table 17. When using DeepSeek-R1 alone, errors may occur due to a lack of sufficient financial knowledge. However, when the knowledge augmentation is applied, the retrieval of external knowledge provides additional context and enables the model to generate the correct answer.

F Case of Reasoner with Programmer

The comparison between the performance of **DeepSeek-R1 combined with Claude 3.5 Sonnet** and **DeepSeek-R1 alone** are shown in Table 18. The task involves solving a problem that, when attempted using DeepSeek-R1 alone, results in errors. However, when DeepSeek-R1 is combined with Claude 3.5 Sonnet, the solution is correctly generated. This comparison highlights the advantage of using reasoning-enhanced model in combination with code-specialized model 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 19. The token consumption of LLMs and LRMs in CoT and PoT settings is provided in Table 20. Additionally, the relationship between token consumption and accuracy across the evaluated models is illustrated in Figure 6.

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
Python Solution:

```
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
```

DeepSeek-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 ($8.0\% + 0.4\% = 8.4\%$):

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

Therefore, the answer is 851.299.

Table 6: Misunderstanding of Problem Example by DeepSeek-R1. The wrong answer is highlighted in red, the ground truth is highlighted in green, while the incorrectly generated content is shown in red. At $t = 0$, there is no need for risk premium adjustment other than that related to interest rate uncertainty, while the model incorrectly adjusted the discount rate at $t = 0$ (by incorporating a risk premium).

Question: Smith demonstrates to Harmonica the total return of a recent transaction performed by the Zenith Fund. Smith explains that the Zenith Fund entered a fully collateralized long position in nearby soybean futures contracts at the quoted futures price of 865.0 (US cents/bushel). Three months afterward, the whole futures position was rolled when the short-term futures price was 877.0 and the long-term futures price was 883.0. What is the Zenith FundâĂŹs three-month total return on the soybean futures transaction, factoring in that the collateral earned an annualized rate of 0.60% during the three-month period between when the initial long position was established and the rolling of the contract (expressed as a percentage)? Answer to three decimal places.

Question ID: test-2162
Ground Truth: 0.853
Python Solution:

```
def solution():
    ...
    # Calculating each return
    price_return = (current_price_short_term - previous_price) / previous_price
    roll_return = (current_price_short_term - current_price_long_term) /
current_price_short_term
    collateral_return = (months / 12) * (collateral_rate / 100)

    # Calculating total return
    total_return = price_return + roll_return + collateral_return

    return total_return * 100
```

DeepSeek-R1:
The Zenith Fund's total return consists of the gain from the futures position and the interest earned on the collateral.
1. Futures Return:
The futures price increased from 865.0 to 877.0 over three months.

$$\text{Profit} = 877.0 - 865.0 = 12.0 \text{ cents/bushel}$$

$$\text{Return} = \frac{12.0}{865.0} = 0.013872 \text{ (or } 1.3872\%)$$

2. Collateral Return:
The annualized interest rate is 0.60%, so the three-month rate is:

$$0.60\% \times \frac{3}{12} = 0.15\%$$

3. Sum of futures return and collateral return:

$$1.3872\% + 0.15\% = 1.5372\%$$

Rounded to three decimal places, the total return is **1.537%**.
Therefore, the answer is 1.537.

Table 7: Formula Application Error Example by DeepSeek-R1. The wrong answer is highlighted in red, the ground truth is highlighted in green, while the incorrectly generated content is shown in red. The model failed to account for the roll yield component in the total return calculation, resulting in an overestimated final answer.

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

Question ID: test-2017
Ground Truth: 4200
Python Solution:

```
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)
```

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.

DeepSeek-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: Numerical Extraction Error Example by DeepSeek-R1. The wrong answer is highlighted in red, the ground truth is highlighted in green, while the incorrectly generated content is shown in red. The model treated the purchase price (pre-discount price) as the post-discount price.

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.

DeepSeek-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: $0.25 \cdot \sqrt{0.5} \approx 0.1767767$ $d_1 \approx \frac{0.094705}{0.1767767} \approx 0.5357$
Next, calculate d_2:

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

...

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 9: Numerical Calculation Error Example by DeepSeek-R1. The wrong answer is highlighted in red, the ground truth is highlighted in green, while the incorrectly generated content is shown in red. The model applied excessive rounding during the calculation steps.

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, 2011, what was the total cash obligations aggregate carrying value of long-term debt due in 2011? Answer as a percentage to three decimal places.

Question ID: test-114
Ground Truth: 9.562
Python Solution:

```
def solution():
    notes_2011 = 338501
    total_obligations = 3540009
    percent_2011 = notes_2011 / total_obligations
    answer = percent_2011 * 100
    return round(answer, 3)
```

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 10: Example of Unsolvable Problem in CodeFinQA. The parts highlighted in red are unsolvable , and the parts highlighted in green are the revised ones. Since the question is flawed due to missing 2006 data in the table, it cannot be solved, making the entire output meaningless.

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
Python Solution:

```
def solution():
    ...
  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
    return round(answer, 2)
```

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 11: Example of Ambiguous Statement in CodeTAT-QA. The parts highlighted in red are ambiguous, and the parts highlighted in green are the revised ones. The ambiguous phrasing of the question leads to multiple possible answers.

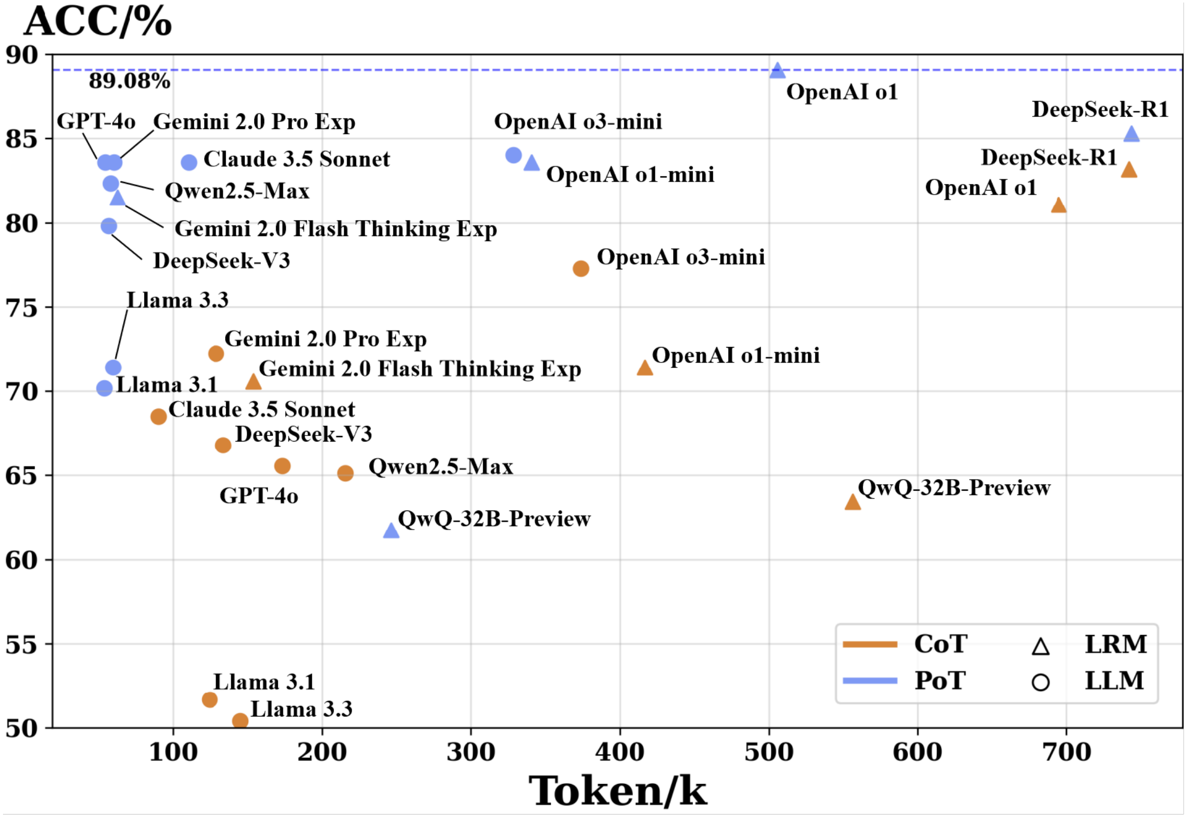


Figure 6: Relationship between token consumption and accuracy across different LLMs and LRMs using CoT and PoT prompting.

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
Python Solution:

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

The Revised Python Solution:

```
def solution():
    # 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 round(t_statistic, 3)
```

Context:

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

Table 12: Example of Oversimplified Process in FinanceMath. The parts highlighted in red are oversimplified, and the parts highlighted in green are the revised ones. While the answer is correct, the solution process lacks clarity, and the code omits some steps.

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 Update
Question ID: test-8
Answer: 3.7699999809
Python Solution:

```
def solution():
    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
    return round(realized_yield_per, 2)
```

After Update
Question ID: test-8
Answer: 8.71
Python Solution:

```
def solution():
    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
    return round(realized_yield_per, 2)
```

Table 13: Example of Incorrect Answer in FinCode. The case had an incorrect answer and incorrect python solution, which we have corrected. The parts highlighted in red are incorrect, and the parts highlighted in green are the revised ones.

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
Python Solution:

```
def solution():
    return round(((1+0.14)*(1-0.10)*(1-0.02)-1)*100, 3)
```

Context:

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

Table 14: Example of Relaxed Evaluation in FinanceMath. The parts highlighted in green are the revised ones. The question is answered with an evaluation that lacks rigor, and multiple correct answers are possible.

Financial 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 15: Example of financial function from function library.

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 an 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 16: A generated problem in FinanceReasoning, combined with the structured financial function as an instruction. The ground truth is highlighted in green.

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

DeepSeek-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.

DeepSeek-R1 + Knowledge Augmentation:
Instruct from GPT-4o:
What function calculates the security selection effect in portfolio performance attribution...
The function retrieved and considered useful by GPT-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 17: A comparison of **DeepSeek-R1** and **DeepSeek-R1 with Knowledge Augmentation** in FinanceReasoning. The ground truth is highlighted in green. Segments highlighted in red or red indicate errors made by **DeepSeek-R1**, while those shown in green indicate correct answers produced by **DeepSeek-R1 with Knowledge Augmentation**.

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

DeepSeek-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.

DeepSeek-R1 + Claude 3.5 Sonnet:
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 18: A comparison of **DeepSeek-R1 alone** and **DeepSeek-R1 combined with Claude 3.5 Sonnet** in FinanceReasoning. The ground truth is highlighted in green. Segments highlighted in red or red indicate errors by **DeepSeek-R1**, while those shown in green indicate correct answers from **DeepSeek-R1 combined with Claude 3.5 Sonnet**.

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 19: Details of the model organization and model source (*i.e.,* model version for proprietary models, and Hugging Face model name for open-source models) for the LLMs and LRMs evaluated in FinanceReasoning.

Model	**Tokens (k) for Easy/Medium/Hard**	
	CoT	**PoT**
OpenAI o1-mini	417 / 1,074 / 944	341 / 1,015 / 953
OpenAI o1	695 / 1,500 / 1,242	505 / - / -
Gemini 2.0 Flash Thinking Experimental	154 / 442 / 311	62 / 164 / 140
QwQ-32B-Preveiw	556 / 1,514 / 1,307	246 / 609 / 543
DeepSeek-R1	742 / 1,499 / 1,274	743 / 1,593 / 1,257
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 20: Token usage across different models.