

Gold-Guided Programmatic Distillation for Financial Reasoning over Hybrid Tables and Text

Stanford CS224N Custom Project

Elana Chen*, **Erica Zhao*** & **Yun Dong***
Department of Computer Science,
Department of Management Science & Engineering,
Department of Computer Science
Stanford, CA 94305, USA
{elanac25, erica117, yund2025}@stanford.edu

Abstract

We propose Gold-Guided Programmatic Distillation to improve the numerical reasoning of compact language models on hybrid tabular and textual financial data. Financial QA is challenging because models must combine structured and unstructured evidence while performing precise multi-step computation, and natural-language rationales are often unreliable for arithmetic-heavy problems. On TAT-QA, a Qwen2.5-72B-Instruct teacher synthesizes executable Python programs guided by ground-truth derivations, replacing error-prone natural language rationales with verifiable programmatic supervision. After filtering, these verified programs are used to fine-tune a Qwen2.5-7B-Instruct student with LoRA, which is further improved through a second round of recovery-based training on teacher-failed examples. Our best student model achieves 87.00 EM and 87.18 F1 on the test set, outperforming both the TAGOP baseline and the teacher model prompted with derivation hints.

1 Key Information to include

- Mentor: Caroline Choi
- External Collaborators (if you have any):
- Sharing project: No
- Late Days: 1 day

Team member	Late days remaining
Elana Chen	6
Erica Zhao	4
Yun Dong	4
Total pooled	3
Team size	3
Late days applied for the report	1

2 Introduction

Financial question answering over hybrid tabular and textual data requires models to aggregate multi-source evidence and perform complex, multi-step numerical reasoning. While Large Language Models (LLMs) can perform step-by-step reasoning using Chain-of-Thought (CoT) prompting [1],

*These authors contributed equally.

they are highly prone to arithmetic calculation errors, particularly when dealing with large numbers, complex equations, and iterative processes [2].

To address this, Program of Thoughts (PoT) disentangles reasoning from computation by having the language model express reasoning steps as executable code, delegating the actual math to an external Python interpreter [2]. Building upon this paradigm, we propose Gold-Guided Programmatic Distillation to transfer robust numerical reasoning capabilities from a massive teacher to a compact student model. In our framework, a Qwen2.5-72B-Instruct teacher synthesizes Python programs guided by the ground-truth derivations of the TAT-QA dataset. By strictly filtering for programs that execute without errors and perfectly match the gold numerical answer and scale, we construct a highly reliable distillation dataset.

We then fine-tune a compact Qwen2.5-7B-Instruct student model on these verified traces using Low-Rank Adaptation (LoRA), replacing error-prone natural language rationales with verifiable programmatic supervision. Furthermore, we extend this framework with an iterative self-improvement phase. By sampling candidate programs from the first-round student on examples the teacher failed to solve, and retaining those that execute successfully, we recover high-quality traces that expand the training distribution beyond the teacher’s original coverage.

3 Related Work

Question answering over hybrid tabular and textual contexts, such as the TAT-QA financial benchmark [3], requires complex numerical reasoning and multi-source evidence aggregation. Early approaches like TAGOP relied on rigid extraction pipelines and predefined symbolic operations, which struggle to bridge the gap to human performance [3]. Consequently, the field has shifted toward Large Language Models (LLMs). Approaches like TAT-LLM demonstrate that LLMs can successfully tackle discrete tabular-text reasoning by decomposing inference into step-wise stages [4, 5], highlighting the potential of LLM-based financial reasoning.

While explicit intermediate reasoning, such as Chain-of-Thought (CoT) prompting [1], helps models decompose complex tasks, natural language rationales remain highly prone to arithmetic hallucinations. Program-of-Thought (PoT) addresses this by representing reasoning as executable programs and delegating computation to an external interpreter, ensuring the precise calculation strictly required for financial QA [2, 6].

Crucially, these reasoning capabilities can be transferred from large teacher models to smaller students. While distilling CoT rationales improves student performance [7, 8], the reliability of these distilled traces dictates downstream success [9]. Frameworks like Program-aided Distillation (PaD) improve trace quality by replacing text rationales with automatically verifiable code [10]. Furthermore, iterative methods like STaR [11] and Rejection Sampling Fine-Tuning (RFT) [12] show that models can autonomously self-improve by generating, verifying, and training on multiple candidate reasoning paths.

Building on these foundations, our work targets hybrid financial QA through highly reliable, expandable programmatic supervision. We distill execution-verified PoT traces rather than error-prone CoT, utilizing gold derivations strictly to guide teacher-side program synthesis. Finally, we introduce an iterative recovery stage that leverages execution-verified student exploration to solve teacher-failed examples, enabling a recursive expansion of high-quality reasoning traces.

4 Approach

4.1 Baseline Models

We evaluate our method against both task-specific and large language model baselines. As a strong traditional baseline for hybrid table-text numerical reasoning, we include TaGOP [3], a model designed specifically for the TAT-QA dataset that performs multi-step reasoning using structured operators over table and paragraph representations.

In addition, we include large language model baselines to isolate the contribution of our programmatic distillation framework. Specifically, we evaluate the base student model (Qwen2.5-7B-Instruct) under

standard prompting conditions without programmatic distillation. We also report the performance of the teacher model (Qwen2.5-72B-Instruct) as a reference upper bound.

4.2 Our Framework

Our framework improves the student model by distilling executable programmatic reasoning traces generated by a teacher model that is prompted with ground-truth derivations, followed by iterative self-improvement. The overall architecture of our two-phase distillation and refinement pipeline is illustrated in Fig. 1.

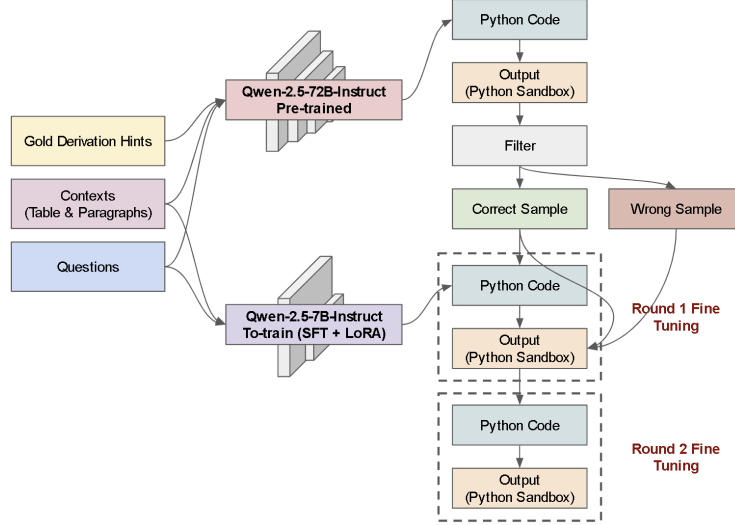


Figure 1: Overview of our two-phase programmatic distillation and iterative fine-tuning framework.

4.2.1 Phase 1: Teacher-Driven Distillation

In the first phase, we extract reasoning paths from the teacher model. Given a financial context C (comprising a table T and paragraphs P) and a query q , the teacher generates a Python program π . To ensure accurate execution traces and guide the teacher through complex multi-step calculations, we inject the TAT-QA ground-truth derivation d into the teacher’s prompt. We further apply DSPy [13] for prompt optimization to format the output strictly as executable code. The generation is defined as:

$$\pi = \text{Teacher}(C, q, d)$$

To construct a high-quality distillation dataset, D_T , we execute generated programs π in a sandboxed Python interpreter. Only programs that execute without compilation errors and yield the exact gold numerical answer ($EM = 1$) and scale are retained. The student model is then trained on D_T via supervised fine-tuning utilizing Low-Rank Adaptation (LoRA)[14]. The student optimizes the log-likelihood of generating π conditioned solely on C and q , without access to the ground-truth derivation d :

$$\mathcal{L}_{\text{Phase1}} = - \sum_{(C, q, \pi) \in D_T} \log P_{\text{Student}}(\pi \mid C, q)$$

4.2.2 Phase 2: Iterative Rejection Sampling and Self-Improvement

In the first round, we retain only teacher-generated programs that execute correctly and match the gold answer, leaving teacher-failed examples in a failure set D_{wrong} . This constrains the student to the teacher’s successful coverage. To recover these harder cases, we introduce a second round of training inspired by Rejection Sampling Fine-Tuning (RFT) and STaR.

Starting from the Phase-1 student model, we revisit each $(C, q) \in D_{\text{wrong}}$ and sample five candidate programs $\hat{\pi}_1, \dots, \hat{\pi}_5$ at a higher decoding temperature ($T = 0.7$):

$$\hat{\pi}_k \sim P_{\text{Student}_{v1}}(\cdot \mid C, q)$$

Each candidate is evaluated using the same sandboxed execution filter as in Phase 1. If a candidate executes successfully and produces the gold answer, it is added to a recovery set D_{rec} . We then perform a second round of LoRA fine-tuning on the augmented dataset $D_{Total} = D_T \cup D_{rec}$ using the same objective as in Phase 1. This stage allows the student to learn from execution-verified self-generated programs beyond the teacher’s original coverage.

4.2.3 Phase 3: Inference Pipeline

At inference, the fully fine-tuned student model generates a programmatic reasoning path $\hat{\pi}$ from unseen contexts and queries. An external Python interpreter executes $\hat{\pi}$ to compute the final numerical answer, eliminating native arithmetic hallucinations.

5 Experiments

5.1 Data

We utilize the TAT-QA dataset [3] for question answering over hybrid financial tables and text. The student model input comprises a table, associated paragraphs, and a natural-language question. A teacher setting additionally incorporates the dataset’s gold derivation as an explicit reasoning hint. The models generate executable Python code to predict a structured answer and scale. Because our approach requires executable code, we exclude examples lacking derivations. This yields an approximate 8:1:1 split of 6,603 training, 834 development, and 831 test examples. From the dataset’s raw structure, we extract the 2D table array, paragraph text, and question text as inputs, reserving the gold answer and scale for evaluation. As illustrated in Fig. 2, the distributions of answer types and sources remain consistent across splits, dominated by arithmetic questions answered exclusively from tables (57%) and those requiring both tables and text (25%).

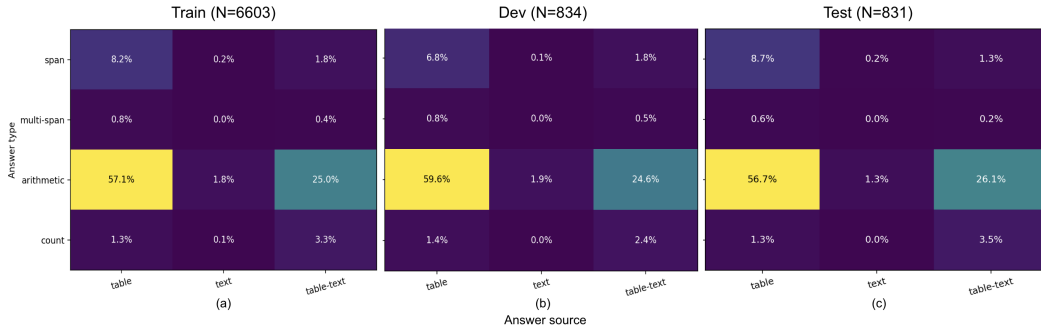


Figure 2: Answer type $\times$ answer source heatmaps across (a) train (b) dev (c) test. The distributions of answer types and sources remain consistent across splits, dominated by arithmetic questions answered exclusively from tables (57%) and those requiring both tables and text (25%).

5.2 Evaluation method

We evaluate using the official TAT-QA evaluation metrics, which take both the predicted answer and scale into account. For span and multi-span questions, we normalize answers (lowercasing and punctuation removal) and compute Exact Match (EM) and token-overlap F1[15]. For multi-span answers, F1 is computed with order-invariant Hungarian matching[16]. For numeric answers, the evaluator converts numbers to a string after applying the predicted scale, where the supported scale labels are "", *thousand*, *million*, *billion*, *percent*. If code execution or output parsing fails, we mark the prediction as *null*, which receives zero EM and F1.

5.3 Experimental Details

We employ Qwen2.5-72B-Instruct as the teacher model for data synthesis and Qwen2.5-7B-Instruct as the student model for fine-tuning. All code and data are available at https://github.com/Erica-zya/ProgrammaticDistillation_Finance.

To minimize execution failures caused by formatting inconsistencies, we utilize DSPy (MIPROv2) alongside manual refinement to enforce strict structural constraints on the teacher prompt. This requires the model to output executable Python code that prints a final JSON object formatted as `{"ans": ..., "scale": ...}`.

During Phase 1 student fine-tuning, we apply Low-Rank Adaptation (LoRA) for one epoch using a learning rate of 2×10^{-4} , rank $r = 8$, $\alpha = 16$, a dropout rate of 0.05, a per-device batch size of 1, and 4 gradient accumulation steps. We evaluate two LoRA configurations: an attention-only variant targeting `q_proj`, `k_proj`, `v_proj`, and `o_proj`, and an attention+MLP variant that additionally targets `gate_proj`, `up_proj`, and `down_proj`.

In Phase 2, we sample five candidate programs per teacher-failed example. Successfully executing samples are merged with the initial dataset for a second round of fine-tuning under identical hyperparameters. The top-performing Round-2 models on the development set are then selected for final test evaluation.

5.4 Results

Main Results: Table 1 presents our main results on the dev and test sets. We compare the baseline model, the teacher model, the Round-1 student, and the Round-2 student under both LoRA settings.

For the attention-only setting, the best final model uses the Round-1 training setup, with learning rate $2e-4$ and 1 epoch. For the attention+MLP setting, the best final model uses a learning rate of $5e-5$, warmup ratio 0.05, cosine learning-rate scheduling, max gradient norm 1.0, and 0.5 training epoch. We select the best Round-2 variant for each setting based on dev performance and report its corresponding test performance.

Model	Fine-tuning Round	Split	EM	F1
TAGOP (Baseline)	–	Test	47.05	47.60
Qwen2.5-7B-Instruct (Student, Baseline)	Without tuning	Test	46.33	46.75
Qwen2.5-72B-Instruct (Teacher, no derivation)	Without tuning	Dev	76.86	77.40
Qwen2.5-72B-Instruct (Teacher, no derivation)	Without tuning	Test	69.55	70.02
Qwen2.5-72B-Instruct (Teacher, with derivation)	Without tuning	Dev	79.62	80.17
Qwen2.5-72B-Instruct (Teacher, with derivation)	Without tuning	Test	78.46	78.73
Qwen2.5-7B-Instruct (Student, Attention-only)	Round 1	Dev	82.73	82.78
Qwen2.5-7B-Instruct (Student, Attention-only)	Round 1	Test	84.12	84.27
Qwen2.5-7B-Instruct (Student, Attention-only)	Round 2	Dev	85.13	85.18
Qwen2.5-7B-Instruct (Student, Attention-only)	Round 2	Test	85.08	85.29
Qwen2.5-7B-Instruct (Student, Attention+MLP)	Round 1	Dev	84.14	84.22
Qwen2.5-7B-Instruct (Student, Attention+MLP)	Round 1	Test	84.72	84.80
Qwen2.5-7B-Instruct (Student, Attention+MLP)	Round 2	Dev	86.33	86.38
Qwen2.5-7B-Instruct (Student, Attention+MLP)	Round 2	Test	87.00	87.18

Table 1: Main results across models and fine-tuning rounds.

Phase 1 Results: The final teacher prompts, with and without derivation hints, were obtained through manual refinement and are shown in Fig. 7. After filtering, 5190 of 6603 training examples are retained. This dataset yields strong Round 1 performance on both dev and test, with attention+MLP outperforming attention-only. Specifically, the attention+MLP model achieves a test EM of 84.72 and F1 of 84.80, surpassing the attention-only model (EM 84.12, F1 84.27). Notably, both Round-1 student models significantly outperform the TAGOP baseline (EM 47.05) and the 72B teacher model provided with gold derivations (EM 78.46).

Phase 2 Results: On the teacher-failed subset, the attention-only model recovers 781 additional examples, while the attention+MLP model recovers 979. Round 2 improves over Round 1 for both variants on dev and test. For attention-only, EM increases from 82.73 to 85.13 on dev and from 84.12 to 85.08 on test. For attention+MLP, EM increases from 84.14 to 86.33 on dev and from 84.72 to 87.00 on test, achieving the highest overall test F1 of 87.18.

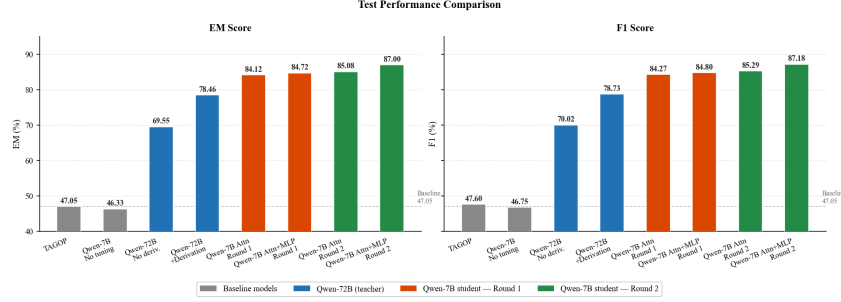


Figure 3: Test set Exact Match (EM) and F1 performance.

6 Analysis

6.1 Performance Improvements from Different Techniques

As shown in Fig. 4, we isolate the drivers of the student model’s performance improvements by evaluating the impact of teacher prompting, derivation inclusion, LoRA module selection, and our recursive training strategy.

First, we found that optimizing the teacher’s input is critical for generating high-quality distillation data. As shown in Table 1, feeding ground-truth derivations to the 72B teacher improves its baseline test Exact Match (EM) from 69.55 to 78.46. Combining this derivation-augmented generation with prompt engineering and strict execution-based filtering allows us to retain 78.6% of the training examples. This creates a highly reliable supervision set that directly enables the strong Round-1 performance of the student models.

Second, we evaluated the effect of expanding the Low-Rank Adaptation (LoRA) modules. Adapting both the attention and MLP projections consistently outperforms adapting the attention weights alone. In the initial fine-tuning phase, the attention+MLP student achieves a test EM of 84.72 and F1 of 84.80, surpassing the attention-only variant (84.12 EM / 84.27 F1). This structural advantage is maintained throughout the subsequent training phase.

Finally, we observe significant gains from our recursive training strategy (Phase 2). By recovering teacher-failed examples via student sampling and executing a second round of fine-tuning, performance improves for both architectural variants. The attention+MLP model reaches a peak test EM of 87.00 and F1 of 87.18. However, an analysis of the training dynamics reveals that these Phase-2 gains are highly sensitive to hyperparameters. While baseline configurations yielded only marginal improvements over Round 1, optimally tuned settings drove the primary performance increase.

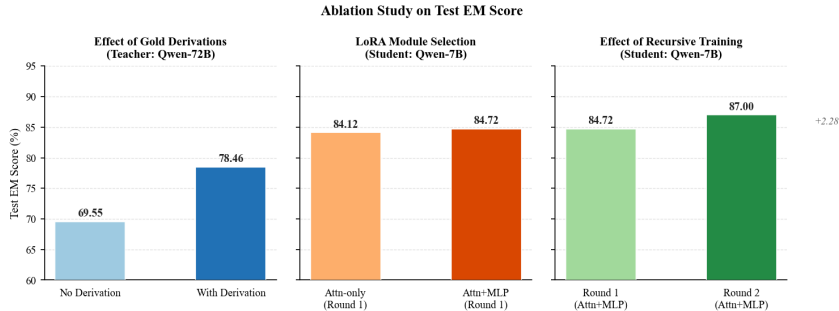


Figure 4: Ablation analysis evaluating the impact of core framework components on test EM scores.

6.2 Subgroup Performance

Figure 5, by answer type and source, shows that the largest gains occur in categories requiring structured numerical reasoning over tables and mixed evidence. The student achieves strong EM across all

question types over TABLE-TEXT contexts, significantly outperforming the TAGOP baseline. Notably, it reaches perfect EM on TABLE-TEXT COUNT, MULTI-SPAN, and SPAN queries, demonstrating the efficacy of programmatic supervision for integrating mixed evidence.

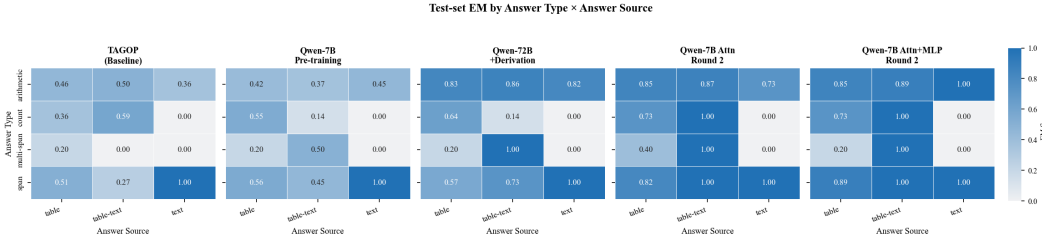


Figure 5: Test-set Exact Match (EM) scores disaggregated by answer type and answer source.

The breakdown reveals profound improvements in ARITHMETIC queries, historically the hardest subset of financial QA. While the baseline struggles to reach 50% EM, programmatic distillation pushes the student over 85% EM on TABLE and TABLE-TEXT domains, proving the value of offloading calculations to an interpreter. Additionally, Round 2 self-improvement enables the student to surpass the teacher across all TABLE-TEXT contexts, confirming that execution-verified exploration successfully discovers reasoning paths the teacher missed. Finally, expanding LoRA parameters to include MLP layers yields uniform gains over Attention-only setups—notably recovering pure TEXT arithmetic from 0.73 to 1.00, indicating feed-forward networks are pivotal for aligning textual semantics with code generation.

6.3 Error Composition

The error decomposition (Figure 6) shows that most of the remaining mistakes of the fine-tuned student arise from *answer prediction errors* rather than scale prediction errors. Compared with the untuned Qwen-7B baseline, round-2 training reduces the total number of errors dramatically (from 445 to 124 and 108). Both answer errors (from 221 to 83 and 76) and scale errors (from 126 to 65 and 54) decrease substantially, indicating that programmatic distillation improves not only reasoning correctness but also alignment with the dataset’s output format. Notably, the number of scale-only errors drops from 122 in the untuned model to 24 and 22 after training, suggesting that exposure to executable reasoning traces also indirectly helps the model learn consistent output scaling. Overall, compared with the untuned Qwen-7B baseline, both fine-tuned Qwen-7B models substantially reduce all three error types, and even outperform the Qwen-72B teacher model.

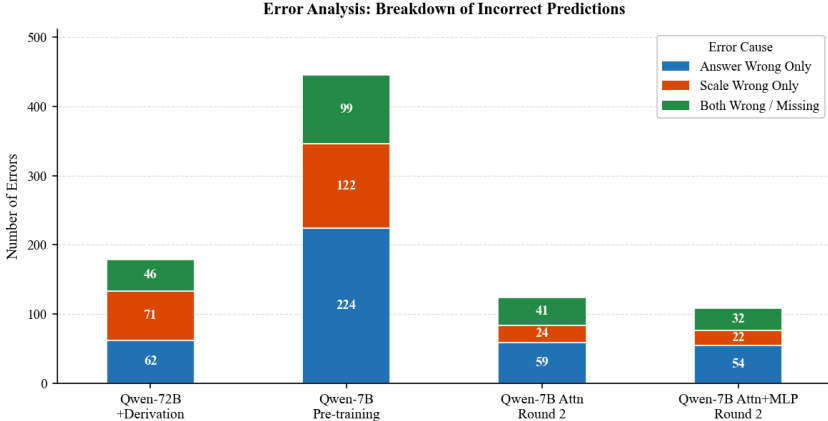


Figure 6: Breakdown of incorrect predictions with three categories: (1) answer wrong only, (2) scale wrong only, and (3) both wrong/missing.

6.4 Limitations

Though our approach leads to a significant improvement in most of the tasks, several systematic limitations remain. First, performance on MULTI-SPAN and COUNT questions for text are less stable and shows smaller improvements, because the teacher model also performed poorly on those questions, and these questions rely more on extraction fidelity than numerical reasoning. Second, although scale errors decrease after training, they remain a non-trivial portion of the remaining failures. This suggests our method largely resolves numerical computation errors, leaving scale grounding and non-programmatic extraction as the primary remaining bottlenecks. Finally, a few subgroup results (e.g., perfect scores on small subsets such as COUNT in TABLE-TEXT) are influenced by the small sample sizes of those categories and should be interpreted cautiously. Overall, the failure analysis suggests that Gold-Guided Programmatic Distillation primarily addresses the core difficulty of structured numerical reasoning, while the remaining errors are increasingly concentrated in extraction-heavy question types and residual scale alignment issues.

7 Conclusion

In this work, we propose Gold-Guided Programmatic Distillation to enhance the numerical reasoning capabilities of compact language models on hybrid table-text financial contexts. By distilling execution-verified programs from a 72B teacher model and recursively self-improving the 7B student on teacher-failed examples, our approach achieves strong performance on the TAT-QA dataset. Our best student model, utilizing both attention and MLP LoRA fine-tuning, reaches 87.00 EM and 87.18 F1 on the test set. This substantially outperforms the TAGOP baseline (47.05 EM) and the derivation-hinted teacher model (78.46 EM).

Our analysis reveals that this framework successfully mitigates arithmetic hallucinations and drastically reduces both mathematical computation and scale prediction errors. The performance gains are most pronounced in complex arithmetic questions that require integrating mixed tabular and textual evidence. However, our error decomposition and subgroup analyses indicate that while programmatic distillation resolves the core difficulty of numerical reasoning, it is less effective for purely extractive tasks, such as multi-span queries. Additionally, residual scale grounding errors remain a primary bottleneck, indicating that scale alignment is still partially decoupled from the execution logic.

Future work will focus on addressing these remaining extraction and scaling limitations. To improve scale grounding, we plan to explore targeted scale-alignment fine-tuning or decoupled prediction heads that handle unit normalization independently from the generated Python logic. To enhance performance on purely extractive queries, future frameworks could integrate dynamic routing pathways, allowing the model to switch between programmatic execution for numerical tasks and direct extraction for non-arithmetic text queries. Finally, applying this programmatic distillation pipeline to broader financial benchmarks, such as FinQA and ConvFinQA, will help validate the generalizability of our framework across different structural domains.

Team contributions

All team members contributed to discussions of project design, model selection, evaluation strategy, result interpretation, report writing, and presentation preparation.

- **Elana Chen:** Proposed a financial arithmetic reasoning focus, developed the DSPy-based prompt optimization pipeline to systematically refine teacher prompt, curated fine-tuning datasets for student training, helped with training and evaluation workflow, tuned LoRA parameters and initiated post-training student evaluation, ran performance comparison to past fine-tuned model implemented by the author of the TAT-QA dataset.
- **Erica Zhao:** Designed the two-phase programmatic distillation pipeline, conducted the primary literature review, implemented the infrastructure for model code generation and sandboxed execution, and was responsible for the training and data recovery for both phases of the attention-only LoRA experimental batch.
- **Yun Dong:** Conducted data exploration, implemented the Hugging Face and Modal-based starter pipeline for model loading, training, and inference, developed the best-performing

teacher prompt through manual refinement, and was responsible for the full attention+MLP experimental branch.

References

- [1] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.
- [2] Wenhui Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*, 2022.
- [3] Fengbin Zhu, Wenqiang Lei, Youcheng Huang, Chao Wang, Shuo Zhang, Jiancheng Lv, Fuli Feng, and Tat-Seng Chua. 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, 2021.
- [4] Fengbin Zhu, Ziyang Liu, Fuli Feng, Chao Wang, Moxin Li, and Tat Seng Chua. Tat-llm: A specialized language model for discrete reasoning over financial tabular and textual data. In *Proceedings of the 5th ACM International Conference on AI in Finance*, pages 310–318, 2024.
- [5] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.
- [6] Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models. In *International conference on machine learning*, pages 10764–10799. PMLR, 2023.
- [7] Lucie Charlotte Magister, Jonathan Mallinson, Jakub Adamek, Eric Malmi, and Aliaksei Severyn. Teaching small language models to reason. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1773–1781, 2023.
- [8] Cheng-Yu Hsieh, Chun-Liang Li, Chih-Kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alex Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 8003–8017, 2023.
- [9] Xiaoyu Tian, Yunjie Ji, Haotian Wang, Shuaiting Chen, Sitong Zhao, Yiping Peng, Han Zhao, and Xiangang Li. Not all correct answers are equal: Why your distillation source matters. *arXiv preprint arXiv:2505.14464*, 2025.
- [10] Xuekai Zhu, Biqing Qi, Kaiyan Zhang, Xinwei Long, Zhouhan Lin, and Bowen Zhou. Pad: Program-aided distillation can teach small models reasoning better than chain-of-thought fine-tuning. 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 2571–2597, 2024.
- [11] Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488, 2022.
- [12] Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. Scaling relationship on learning mathematical reasoning with large language models. *arXiv preprint arXiv:2308.01825*, 2023.
- [13] Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. Dspy: Compiling declarative language model calls into self-improving pipelines, 2023.

- [14] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Liang Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *Iclr*, 1(2):3, 2022.
- [15] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 conference on empirical methods in natural language processing*, pages 2383–2392, 2016.
- [16] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

Appendix

A Prompt

(a)

```
PROMPT_TMPL = """Write a Python program that answers the QUESTION.

Sources of truth:
- TABLE and PARAGRAPHS are the ONLY sources of facts and numbers.
- You MAY use GOLD_DERIVATION only as a hint for the reasoning steps.
- Determine the unit/scale ONLY from explicit cues in TABLE, QUESTION, or PARAGRAPHS.

Strict output rules:
- Output MUST be raw Python code only. Do NOT use Markdown fences.
- Do NOT print anything except the final answer.
- End with EXACTLY ONE print(...) statement (the very last line).

Answer requirements:
1) Compute the final answer and store it in a variable named ans.
2) ans MUST be either:
  - a string, OR
  - a list of strings (for questions with multiple valid answers).
3) If the QUESTION can have multiple answers (e.g., "which years", "list all", filtering conditions), set ans to a list of strings containing ALL matching answers.
4) Do not include duplicates in list answers.

Scale requirements (must match exactly one of these 5):
- The JSON field "scale" MUST be exactly one of: "", "thousand", "million", "billion", "percent".

Final output format:
- Create a dictionary named out with exactly these keys:
  out = {"ans": ..., "scale": ...}
- Print EXACTLY one line of valid JSON.
- The LAST line of the program MUST be:
  print(json.dumps(out, ensure_ascii=False))

TABLE:
{table}

PARAGRAPHS:
{paras}

QUESTION:
{question}

GOLD_DERIVATION (hint only):
{derivation}
"""
```

(b)

```
PROMPT_TMPL = """Write a Python program that answers the QUESTION.

Sources of truth:
- TABLE and PARAGRAPHS are the ONLY sources of facts and numbers.
- Determine the unit/scale ONLY from explicit cues in TABLE, QUESTION, or PARAGRAPHS.

Strict output rules:
- Output MUST be raw Python code only. Do NOT use Markdown fences.
- Do NOT print anything except the final answer.
- End with EXACTLY ONE print(...) statement (the very last line).

Answer requirements:
1) Compute the final answer and store it in a variable named ans.
2) ans MUST be either:
  - a string, OR
  - a list of strings (for questions with multiple valid answers).
3) If the QUESTION can have multiple answers (e.g., "which years", "list all", filtering conditions), set ans to a list of strings containing ALL matching answers.
4) Do not include duplicates in list answers.

Scale requirements (must match exactly one of these 5):
- The JSON field "scale" MUST be exactly one of: "", "thousand", "million", "billion", "percent".

Final output format:
- Create a dictionary named out with exactly these keys:
  out = {"ans": ..., "scale": ...}
- Print EXACTLY one line of valid JSON.
- The LAST line of the program MUST be:
  print(json.dumps(out, ensure_ascii=False))

TABLE:
{table}

PARAGRAPHS:
{paras}

QUESTION:
{question}
"""
```

Figure 7: Prompts used in our pipeline. (a) Teacher prompt with derivation hints. (b) Shared prompt without derivation hints for teacher and student.

B Examples

```
"answer_type=arithmetic|answer_from=table": {
  "source_split": "tatqa_dataset_train_filtered.json",
  "table_uid": "e78f8b29-6085-43de-b32f-be1a68641be3",
  "paragraph_uids": [
    "62be4f5a-1693-4e6b-8bb4-0a4e1e40b409",
    "c63e6ed5-8fe5-46e4-a02a-f923e90e8067",
    "b4093fd4-43ea-4b31-9975-13c0012a0b18",
    "9f6ecb32-9e2c-4036-8209-8905855145c0"
  ],
  "question": {
    "uid": "dbee8930-0110-41c7-95fe-0178ddf4f841",
    "order": 4,
    "question": "What is the 2019 average rate of inflation?",
    "answer": 2.9,
    "derivation": "(2.9+2.9)/2",
    "answer_type": "arithmetic",
    "answer_from": "table",
    "rel_paragraphs": [],
    "req_comparison": false,
    "scale": "percent"
  }
},
"answer_type=span|answer_from=table-text": {
  "source_split": "tatqa_dataset_train_filtered.json",
  "table_uid": "1085435b-dbed-4791-8f29-425c766135e2",
  "paragraph_uids": [
    "4cef88c9-a9da-4cac-a43b-65297effb02d",
    "0701f7d5-62bb-4d24-ae27-422d5b08ada3",
    "0a2d3383-a9ed-4bd8-80df-984eaffac78f",
    "e6b8c7be-3c05-49cf-b463-45c7d67c1fdf"
  ],
  "question": {
    "uid": "f8d70551-f40e-4888-8688-ab207af1679e",
    "order": 4,
    "question": "In which year is the value of finished products higher?",
    "answer": [
      "2019"
    ],
    "derivation": "10,905>7,099",
    "answer_type": "span",
    "answer_from": "table-text",
    "rel_paragraphs": [
      "2"
    ],
    "req_comparison": true,
    "scale": ""
  }
}
```

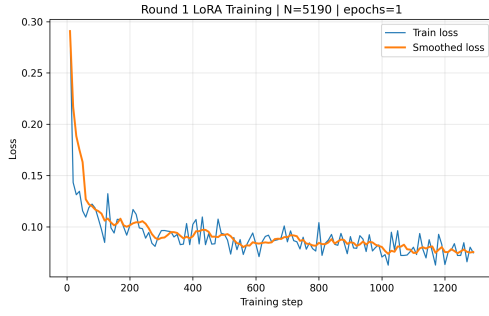
Figure 8: Examples of TAT-QA dataset.

C Hyperparameter Optimization for Attention + MLP Variant

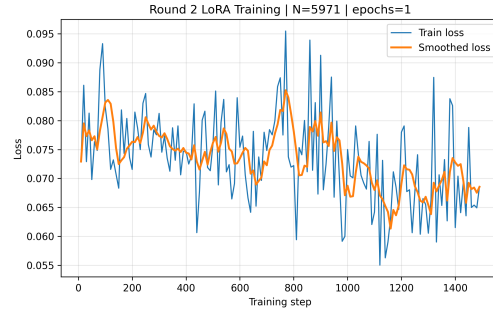
Variant	LR	Epoch	Warmup	Scheduler	Dev EM/F1
v1	2e-4	1.0	–	linear	84.89/84.94
v2	1e-4	1e-4	–	linear	84.89/84.94
v3	5e-5	0.5	0.05	cosine	86.33/86.38
v4	2e-5	0.25	0.05	cosine	84.41/84.46

Table 2: Phase-2 attention+MLP variants under different second-round training settings.

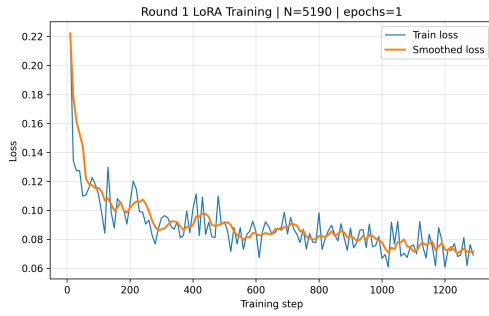
D Loss Curve During Training



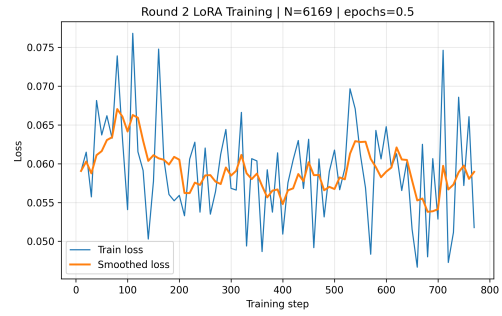
(a) Attention Only, Phase 1



(b) Attention Only, Round 2



(c) Attention + MLP, Phase 1



(d) Attention + MLP, Phase 2

Figure 9: Training dynamics across different experiments and fine-tuning rounds.