LLM Reasoners and Agents for Financial Workflows

Aalok Patwa
University of Pennsylvania
apatwa@upenn.edu

Dan Roth
University of Pennsylvania
danroth@seas.upenn.edu

Abstract

There has been much interest in using large language models (LLMs) in financial institutions to analyze financial documents, create models, and prepare business reports such as memos and slides. LLMs have been shown to have deep domain knowledge, agentic capabilities, and, recently, the ability to reason step-by-step to perform complicated calculations. However, to be useful in a high-stakes domain like finance, models need to be excellent at all three. In this work, we aim to increase the utility of LLMs in finance by improving their *ability to reason about finance* and developing *customized agents for financial workflows.* First, we develop a new synthetic data method for generating financial reasoning datasets, and release one such dataset, called `FinReason`. Second, we show that training on `FinReason` through either supervised fine-tuning or reinforcement learning can improve financial reasoning abilities in small models. Third, we build an LLM assistant to extract information from and answer questions from diverse sources such as SEC filings, earnings call transcripts, and slides. Fourth, we create a tool for financial professionals to audit financial filings that they prepare. Finally, we develop an LLM agent that can automatically synthesize financial models. Our results highlight the struggles of current models in grasping fundamental financial principles, but show that these struggles can be overcome to make LLMs suitable for real-world workflows. Overall, our work presents a path toward finance-specific LLMs that meet the rigorous demands of professional environments.

1 Introduction

In recent years, large language models (LLMs) such as GPT-4, Claude, and LLaMa have demonstrated remarkable capabilities across a wide range of natural language processing tasks, including question answering, summarization, and writing. These models achieved strong performance on a variety of benchmarks, including general-knowledge benchmarks such as MMLU and domain-specific benchmarks like MATH and MedQA. [1] [2][3]. More recently, models have been trained using reinforcement learning to be explicit reasoners, achieving unprecedented success on domains that require complex thinking, such as math and code.

One domain where LLMs have the potential to be transformative is finance. Financial professionals, especially those in industries such as investment banking and private equity, are tasked with analyzing vast amounts of structured and unstructured information, building models based on this data, and preparing reports with decisions and recommendations. The large amount of information to process, combined with tight deadlines, lead to high workload. The promise of using AI, especially LLMs, to automate or assist in these workflows is compelling, but AI has so far had limited adoption in high finance roles.

There are multiple reasons for this. First, while language models exhibit general domain knowledge, they often fall short when tasks require precise numerical reasoning, multi-step logic, and consistency across multiple documents or data sources, each of which could be hundreds of pages of dense financial filings. Second, financial workflows are highly specialized and require tools that can interact

CIS 4980: Senior Capstone.

with a wide range of data types, including tables and charts, and extract deep insights. Third, mistakes are extremely costly in finance, raising the bar for the accuracy and reliability required of AI systems. As a result, finance firms are reluctant to adopt even cutting-edge technologies such as ChatGPT because they are simply not reliable enough out-of-the-box.

However, by integrating domain-specific reasoning, retrieval from diverse data sources, and tool use, it becomes possible to create agents that augment the capabilities of out-of-the-box models and greatly improve performance.

In this work, we improve models' understanding of finance and develop finance-specific applications. First, we develop an extensible synthetic data generation method that can create financial question-answer datasets along with synthetic chains-of-thought, meaning they could be applied to both reinforcement learning or supervised fine-tuning stages of models. We release a sample dataset, `FinReason`, to the public. Second, we train models on `FinReason` using multiple methods, and show that the dataset improves performance on external, human-generated financial benchmarks, demonstrating the utility of this synthetic approach. Finally, we build and evaluate several applications that tackle high-impact financial workflows: long-context question answering across multiple financial data sources, automated audit of financial reports, and one-shot generation of Excel models complete with formulas and formatting.

Overall, our findings suggest that with the right training data and workflow-specific design, LLMs can offer significant value to financial professionals, potentially saving hundreds of hours per month per analyst.

2 Technical background

We first review high-level concepts relating to large language models. More context about the specific approaches we use in our work, including fine-tuning, LLM agents, synthetic data generation, and reinforcement learning, is given later.

2.1 Large language models

LLMs are neural networks trained to understand and generate text. Currently, the dominant neural architecture for LLMs is the transformer [4], which uses a layer called self-attention to allow the representation of previous text in a sequence to autoregressively influence the next piece of text that is generated. Formally, an LLM models the conditional probability:

$$P(x) = \prod_{t=1}^{n} P(x_t \mid x_{<t}; \theta) \tag{1}$$

where each x_i is a piece of text, known in the field as a "token". A token might be a word, a single character, or somewhere in between. When processing an input sequence, a tokenizer first encodes raw text into tokens, producing a sequence of length T. Each token in the sequence is then assigned a trainable embedding vector of dimension D, which thereby represents the input sequence as a $T \times D$ tensor.

Each subsequent layer in a transformer simply updates the embedding representation of each token. The most important layer is self-attention. The attention layer first projects each token's input vector into three separate vectors, called the query, key, and value. Then, for a given token x_t, attention takes the dot product of its query vector q_t with the key vectors of all previous tokens $k_1 ... k_t$. The softmax operation is applied to form a probability distribution, representing the attention weight $w_{t,i}$ that x_t gives to the tokens before it. Finally, the representation of x_t is updated to $\sum_{i=1}^{t} w_{t,i} v_i$, the attention-weighted sum of previous token's value vectors.

Self-attention improved upon existing language model architectures, such as LSTMs, by being easily parallelizable and better modeling long-range dependencies. The entire operation can be done to update the representations of all tokens at once by packaging the queries, keys, and values into tensors and using pure matrix multiplication operations, which are extremely fast on GPUs. These improvements enabled transformer-based models to be scaled to massive parameter counts (hence the "large" in the name).

Transformers were also trained on increasingly large datasets. Researchers discovered "scaling laws" that relate parameter counts to the optimal number of tokens to train on. [5] A trend emerged where larger models trained on larger datasets outperformed smaller models trained on smaller datasets. Most notably, larger models had "emergent capabilities," such as the ability to perform calculations, follow instructions given in their prompts, and learn from in-context examples. [6]

Today, large language models are "pretrained" on huge amounts of data, including much of the Internet. The data mixture includes textbooks, blogs, human-written code, and more, imbuing LLMs with knowledge about almost every topic that can be imagined. However, while pretrained LLMs exhibit some implicit financial knowledge, they struggle with more nuanced tasks requiring precise interpretation of financial data, numerical reasoning, and inference. As a result, additional approaches are required to adapt pretrained models to specific workflows.

2.2 Multimodal models

Multimodal models are an extension of language models that can take inputs from different modalities, including text, images, and audio, rather than being limited to purely textual input. Traditional LLMs operate over tokenized sequences of text alone, but are limited in their ability to understand data visually. This is a critical ability for models to be able to understand data used in the financial domain, including charts, scanned documents, and financial slides.

Modern multimodal architectures introduce modality-specific encoders to handle non-textual data. These encoders transform the data into embeddings, which can then be added or fused with text tokens. For example, in Vision-Language Models (VLMs), a trained image encoder tokenizes input images into patches and embeds them. In early VLMs such as Flamingo [7], image tokens are concatenated with text tokens, allowing text tokens to attend to the image tokens and incorporate them into the output.

Recently, multimodal models have been made widely accessible to developers through the APIs of prominent AI companies. Models like OpenAI's gpt-4o and Gemini 2.0 are pretrained on combined vision-language data to perform tasks that require joint reasoning across both modalities, such as asking questions about a filing that contains text and figures. Although these models still struggle to capture nuance in certain images, as their capabilities grow, their utility in financial workflows improves.

3 Related Work

The application of LLMs and VLMs to finance has garnered significant interest from both academia and industry.

3.1 LLMs for financial information processing and analysis

LLMs have demonstrated strong capabilities in processing and understanding the large volumes of data inherent in finance, coming from sources such as SEC filings, earnings call transcripts, news articles, and research reports. Early applications trained encoder-only LLMs models like BERT for sentiment analysis [8], text classification, and named entity recognition [9]. As LLMs became more powerful, applications expanded to include summarization and question answering of lengthy financial documents. One of the most crucial developments in these models was pretraining on longer context lengths, which allowed models to reason across potentially hundreds of pages of financial documents. Even more recently, research has attempted to enhance the capabilities of VLMs for Visual Question Answering (VQA) on financial charts, since models need to be able to precisely extract data from charts in order to be useful in the domain.[10] [11]

Several LLMs have been trained on financial data corpora in order to bake-in financial reasoning abilities. BloombergGPT [12] was a famous example, where Bloomberg trained a 50B parameter model on their proprietary data. After the release of open-source reasoning models such as DeepSeek-R1, researchers began distilling financial reasoning into smaller models. [13] This has increased the models' knowledge of financial terminology and the steps required to perform numerical workflows.

3.2 Financial benchmarks

The need to evaluate numerical reasoning and information processing capabilities of LLMs in finance has led to the creation of several specialized benchmarks. FinQA [14] measures financial numerical reasoning by asking questions that require multi-step arithmetic to be performed on raw values from tables in financial filings. FinanceBench [15] measures long-context information analysis by asking specific questions from long financial reports.

Nevertheless, frontier LLMs continue to fall short of the accuracy thresholds necessary for production deployment. [16] Models need to get better at interpreting data, including data stored in tables and charts, and reasoning about the steps required to perform complicated financial analysis tasks.

Our work builds upon existing research by improving upon the performance of frontier models out of the box. We tackle the bottleneck of acquiring large-scale financial reasoning data using a synthetic method. We demonstrate that training on this dataset improves model reasoning. We create practical applications for multi-document Q&A, automated financial report auditing, and financial model generation, turning raw LLMs and VLMs into agents that can perform work with higher accuracy.

4 Methods

4.1 Overview

Our work presents a set of contributions aimed at addressing these gaps. First, we develop an extensible synthetic data generation method for financial reasoning datasets. Second, we train models on our synthetic dataset using multiple post-training approaches to teach models reasoning. Finally, we build and evaluate several applications that tackle high-impact financial workflows. An overview of our work is shown in Figure 1.

4.2 Synthetic generation of financial reasoning data

Reasoning models are trained to solve problems by utilizing chains of thought to think through the required intermediate steps. [17]. Even simply asking non-reasoning base models to emit chains-of-thoughts can improve their reasoning abilities. [18] More recently, research has shown that models can be trained to become reasoners through pure reinforcement learning.

Reasoning models show great promise in the financial domain. Much of the job of an analyst requires taking in raw data, e.g. from a filing, and calculating derived metrics that reveal deeper insights about the company. However, calculating these metrics requires not only general mathematical ability, but also intuition about *how* those metrics should be calculated. Therefore, there is a need for finance-specific reasoning datasets that can teach models how to "think like an analyst" and perform complicated financial calculations.

Creating these datasets, however, is difficult. The default method requires human annotators, which are scarce and expensive. Instead, we create a financial reasoning dataset synthetically.

Synthetic data generation has become an increasingly popular methodology for refining LLMs. Recent work has demonstrated that carefully constructed synthetic datasets can significantly enhance model capabilities across a range of tasks. Approaches such as Self-Instruct [19] introduce bootstrapped data generation pipelines, where a pretrained model is used to autonomously create and iteratively refine instruction-tuning datasets for downstream training. Using strong, pre-trained models to generate or verify data has emerged as a popular strategy. In our case, however, we generate a dataset from a fixed set of rules that describe how calculations should be performed. This allows us to verifiably generate correct ground-truths.

The advantages of synthetic data are twofold: it substantially reduces the cost and latency associated with dataset construction, and it affords precise control over the statistical and semantic properties of the training set. This control is particularly critical in specialized domains where the availability of high-quality, annotated data is limited, and where domain-specific reasoning skills are required.

In general, synthetic data generation comes with challenges. For synthetic methods that utilize a strong model to generate examples, there is a risk that the model itself will hallucinate during generation, which can lead to compounding errors in the model that is ultimately trained on the

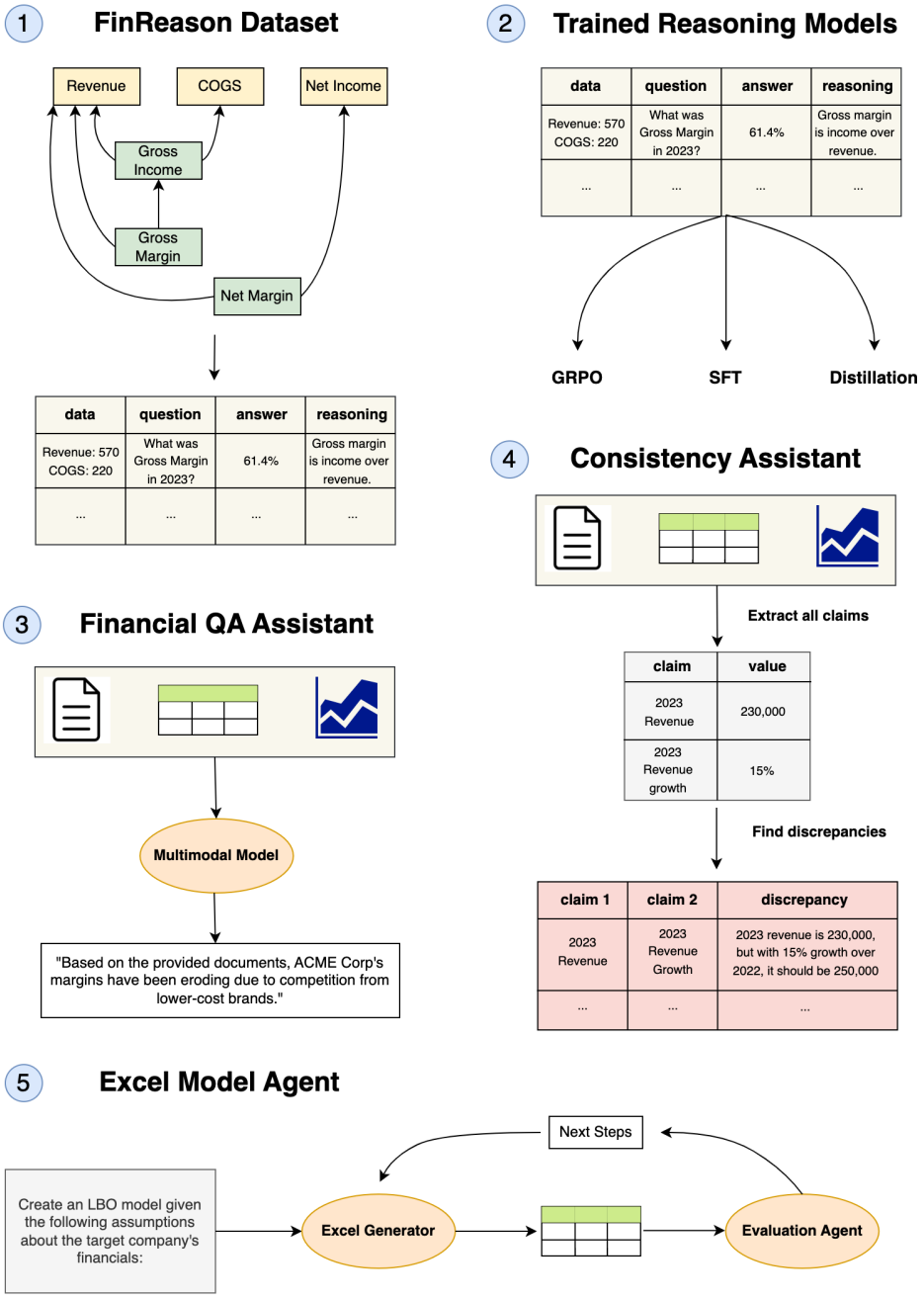

Figure 1: Overview of contributions.

dataset. As a result, synthetic data pipelines generally involve quality control mechanisms, such as rule-based verification, model-based self-critiques, or even manual review. In our case, the rules are guaranteed to create correct ground truths, but the distribution of questions that we can produce is limited by the rules that we define. As a result, we implement mechanisms to increase the variety of the dataset.

4.2.1 Metric dependency graph

The first component of our system is a dependency graph of financial metrics, formalized through two primary classes: Basic Metrics (β) and Derived Metrics (δ). Basic metrics are atomic financial

data points that serve as fundamental inputs: revenue, net income, accounts receivable, etc. These are line items that you would find in a typical public company's earnings report. Derived metrics are composite metrics defined by functions over $\beta \cup \delta$. Therefore, the graph G is defined as:

$$G = \{\beta \cup \delta \mid \forall d \in \delta, \exists f : \beta^n \times \delta^m \to \mathbb{R}\} \tag{2}$$

where f is the computation function for derived metric i. Using these formulas, given any input data containing values for basic metrics, we can obtain a ground-truth value for the derived metric.

For example, consider the derived financial metric called the current ratio, defined as the ratio between current assets and current liabilities. The quantity of total current assets and current liabilities, however, are themselves derived metrics, defined as the sum of several basic metrics. This creates a very convenient recursive relationship we can use to calculate any derived metric that originates from the basic metrics. We can grow a tree, starting from the simplest basic metrics, progressing to first-degree derived metrics (which only depend on basic metrics), then second-degree (dependent on some first-degree derived metrics), and so on. We can simply express the computation function for a k-th degree metric in terms of the values of its direct inputs, as long as each of those inputs (which must be at most $(k-1)$ degree metrics) has a defined computation function as well. Then, given some input values for the basic metrics, we can simply execute the computation function for any derived metric, which may recursively call the computation functions for its dependent metrics.

4.2.2 Recursively synthesizing reasoning traces

This recursiveness also crucially enables us to develop synthetic reasoning traces for any derived metric. Along with the computation function, we define a reasoning trace function. Let $\rho_m(d, y)$ be the reasoning function for metric m with basic metric data d in year y. The reasoning function can be defined recursively as:

$$\rho_m(d, y) = \begin{cases} s_m \circ \bigcirc_{i=1}^{n} \rho_{c_i}(d, y) \circ v_m & \text{if } m \in \delta \\ f_b(m, d, y) & \text{if } m \in \beta \end{cases} \tag{3}$$

where:

- s_m is the static explanation string for metric m
- $\circ$ is the string concatenation operator
- $\bigcirc$ represents repeated string concatenation
- $c_i \in \text{deps}(m)$ are the dependency metrics of m
- v_m is the value of metric m in year y
- $f_b(m, d, y) =$ "Year y m is v_m"

In words, the steps are:

1. Retrieve the static explanation string for the derived metric.
2. Recursively generate reasoning traces for each dependent metric.
3. Concatenate these reasoning traces into a full explanation.
4. Insert the computed value for the original metric into the trace.
5. Return the full reasoning string.

As an example, consider generating a reasoning trace for the metric called net working capital, defined as the difference between operating current assets and operating current liabilities. As the static explanation string, we write, "Net working capital is calculated by subtracting operating current liabilities from operating current assets." We then recursively generate the reasoning traces for these two dependent metrics. Operating current liabilities, for example, has the static string "Operating current liabilities is calculated by adding accounts payable, accrued salaries, and deferred revenue." Then, since these dependent metrics are basic metrics, the recursive calls simply return "Accounts payable is X". The recursive call works its way back up, until we are ready to report the final answer for the original metric, net working capital.

The final reasoning trace may look like this:

> Net working capital is calculated by subtracting operating current liabilities from operating current assets. Operating current liabilities is calculated by adding accounts payable, accrued salaries, and deferred revenue. Accounts payable is 567,392. ... Therefore, operating current liabilities is 1,490,103. Operating current assets is calculated by adding working cash, inventory, accounts receivable, and prepaid assets. ... Therefore, operating current assets is 2,293,103. Therefore, net working capital is 803,000.

Therefore, we can use the same recursive dependency graph that we used to calculate the metric to generate full, natural-language reasoning traces that demonstrate how the metric was calculated.

4.2.3 Adding randomness and variety to generations

The above method is easily extensible, as additional metrics can simply be added along with their formulas. However, we would ideally like to add even more variety to the examples so that any model trained on this dataset would learn to generalize.

First, we designate certain basic metrics as optional. These metrics may or may not be included in the initial raw input data. By varying the included metrics, we introduce diversity across different examples.

Then, any derived metric that depends on an optional metric simply checks whether or not the optional metric is included. If not, it ignores it in its computation function and excludes it from its synthetic reasoning trace. This ensures that any derived metric's ground-truth value and reasoning trace reflect only the data that was actually provided as input.

Second, for each derived metric, we define a set of strings that serve as question templates. These templates represent different ways that the derived metric could be framed as a question. This allows diversity in the prompts in our dataset: we might ask "Calculate 2024 current assets," or "What were total current assets in 2024?"

4.2.4 Dataset generation algorithm

We use the dependency graph to generate examples of tuples `(data, question, answer, reasoning)`.

For each example, we first obtain the set of basic metrics to include in the data. Optional basic metrics are selected with probability $1/2$. Then, we generate the time horizon that the data will cover by randomly choosing a start year and a data period between two and six years. For each year, and each basic metric, we randomly generate a value between 10 and 1000 (or 1 and 100 for basic metrics that are percentages, such as the tax rate). This serves as the `data` in our tuple.

Next, we randomly sample a derived metric and the year for which we want to calculate the metric. We construct the appropriate question string for this metric, which serves as the `question` in the tuple.

Finally, we use the dependency graph to calculate the ground truth `answer` value for this metric and year, and generate the recursive `reasoning`.

We use this algorithm to generate a dataset called `FinReason`, comprised of 3,000 examples on 22 basic metrics and 19 derived metrics. The dataset is publicly available at `https://huggingface.co/datasets/aalokpatwa/financial-reasoning`.

4.3 Supervised fine-tuning and reinforcement learning on `FinReason`

We train small models on `FinReason` to test the dataset's ability to develop reasoning capabilities. We specifically experiment with three different training approaches to evaluate their relative efficacy: supervised fine-tuning on the reasoning traces in `FinReason`, distillation from large, powerful reasoners such as o1, and pure reinforcement learning on the answers in `FinReason` using GRPO.

Algorithm 1 Dataset Generation

Require: n_{train}, n_{test}, D (dependency graph)
Ensure: D_{train}, D_{test} (datasets)
  for $i = 1$ to $(n_{\text{train}} + n_{\text{test}})$ **do**
    $\beta_s \leftarrow \text{SelectBasicMetrics}(\beta)$
    $Y \leftarrow \text{GenerateTimeHorizon}(2, 6)$
    $F \leftarrow \text{GenerateFinancialStatements}(\beta_s, Y)$
    $m \leftarrow \text{SampleMetric}(\delta)$
    $q \leftarrow \text{GenerateQuestion}(m, Y)$
    $a \leftarrow \text{ComputeAnswer}(m, F, D, Y)$
    $r \leftarrow \text{GenerateReasoning}(m, F, D, Y)$
    $s \leftarrow \{F, q, a, r\}$
    $\text{AppendToDataset}(s)$
  end for

4.3.1 Supervised fine-tuning

LLMs are typically pre-trained on vast amounts of general text data, enabling them to acquire broad knowledge and language understanding capabilities. However, these general-purpose models often require further adaptation to excel at specific downstream tasks or specialized domains, such as financial knowledge. Supervised fine-tuning (SFT) is a standard technique used to achieve this adaptation [20].

The core idea behind SFT is to continue the training process of a pre-trained model, but on a smaller, task-specific dataset consisting of input-output examples relevant to the target task. For instance, in our case, the inputs are financial questions, and the desired output is a completion with reasoning and a final answer. During SFT, we slightly modify the parameters of the language model, often using an order of magnitude lower learning rate. However, SFT is a brittle method, as it can lead to catastrophic forgetting where model coherence and knowledge degrades heavily. [21]

The objective function typically minimized during SFT for text generation tasks is the same as during pre-training: the cross entropy loss (or negative log-likelihood) between the model's predicted probability distribution over the next token and the actual next token in the target sequence, conditioned on the input and the preceding tokens in the target sequence. Formally, given a dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$, where x_i is the input and $y_i = (y_{i,1}, \ldots, y_{i,T_i})$ is the target output sequence of length T_i, the objective is to minimize:

$$\mathcal{L}_{SFT}(\theta) = -\sum_{i=1}^{N} \sum_{t=1}^{T_i} \log P(y_{i,t} | x_i, y_{i,1}, \ldots, y_{i,t-1}; \theta) \tag{4}$$

where θ represents the parameters of the model.

Training the full set of parameters in large models can be computationally expensive and memory-intensive. Parameter-Efficient Fine-Tuning (PEFT) methods have been developed to address this challenge. Low-Rank Adaptation (LoRA) [22] is a prominent PEFT technique. LoRA hypothesizes that the change in model weights during adaptation has a low intrinsic rank. Instead of updating the original weight matrix $W_0 \in \mathbb{R}^{d \times k}$, LoRA introduces two low-rank matrices $A \in \mathbb{R}^{d \times r}$ and $B \in \mathbb{R}^{r \times k}$, where the rank $r \ll \min(d, k)$. The update to the weights is represented by the product BA, such that the modified forward pass becomes $h = W_0 x + BAx$. During fine-tuning, only the parameters of A and B are trained, significantly reducing the number of trainable parameters while often achieving performance comparable to full fine-tuning.

For our experiments, we selected the Qwen 2.5-0.5B-Instruct model [23] as our base LLM, chosen for its strong base performance and manageable size. We fine-tuned this model on the reasoning traces present in `FinReason`.

We utilized the LoRA technique for parameter-efficient fine-tuning. Specifically, we applied LoRA to all attention projection and feed-forward linear layers. Our training set consisted of 2700 examples from `FinReason`, while 300 was used for validation during training. Each example consisted of

a prompt that contained both the raw data for basic metrics and a question, and the ground-truth completion was formatted using XML tags to contain both a reasoning trace and a final answer:

<reasoning> {Reasoning trace} </reasoning>
<answer> {Ground-truth final answer} </answer>

The fine-tuning process was implemented using the Hugging Face Transformers and PEFT libraries. Training was conducted on a single NVIDIA A6000 RTX GPU with 48GB of VRAM. The entire SFT process took approximately 23 hours to complete. The resulting fine-tuned model, which we refer to as *FinReason-Qwen-0.5B*, was then evaluated on a held-out test set of `FinReason` examples, as well as FinQA, an external benchmark.

4.3.2 Distillation

Knowledge distillation (KD) is an approach that seeks to transfer the knowledge acquired by a large, computationally expensive model (the "teacher") to a smaller, more efficient model (the "student") [24]. The primary motivation is to bridge the performance gap often observed between large-scale models and their smaller counterparts, enabling the deployment of high-performing models under constraints such as limited computational resources, memory footprint, or latency requirements. In our case, however, we would like to distill the knowledge from a large reasoning model (such as OpenAI's o1 [25]) into a small non-reasoning model.

Originally, KD proposed using the softened outputs of the teacher model as targets for training the student. Instead of only using the hard labels (e.g., one-hot vectors in classification), the student is trained to match the probability distribution generated by the teacher's softmax layer, typically after applying a temperature scaling ($T > 1$). This temperature scaling smooths the distribution, providing richer supervisory signals known as "soft targets," which convey information about the similarities between classes as perceived by the teacher. The final loss function often combines the standard cross-entropy loss with the hard labels and a distillation loss (e.g., Kullback-Leibler divergence) between the student's and teacher's softened probability distributions.

Beyond matching output probabilities, various distillation techniques have emerged. Response-based distillation focuses on matching the final outputs or predictions of the teacher, which is particularly relevant for generative tasks like ours, where the student learns to replicate the teacher's generated text (reasoning traces). Feature-based KD aims to align the intermediate representations or feature maps learned by the student with those of the teacher at specific layers. Relation-based KD goes further by transferring relational knowledge, such as the relationships between different layers or data samples.

We employ response-based distillation from a more powerful teacher model, o1. We prompted o1 to generate answers and reasoning traces for the questions in `FinReason`. To ensure the quality of the distillation data, we filtered these generated traces, retaining only those where o1's final answer was verified as correct. Verification was done by checking for exact matches, in the case of string ground truths, and relative tolerance of 1% in the case of numerical ground truths. These correct reasoning traces, along with the final answer, served as the target outputs for the student model.

We then performed SFT on the Qwen 2.5-0.5B-Instruct student model using this filtered set of high-quality reasoning traces generated by the o1 teacher. The student model was trained using the standard cross-entropy loss to predict the next token in the teacher's reasoning trace and answer, given the question and the preceding tokens. This process effectively distills the reasoning capabilities demonstrated by the larger teacher model into the smaller student model by training the student to replicate the teacher's successful reasoning steps. The training setup (LoRA, hardware, libraries) was kept consistent with the SFT experiment above. This distilled model is referred to as *FinReason-Qwen-0.5B-Distilled*.

4.3.3 Reinforcement learning using GRPO

Our third approach utilized Reinforcement Learning (RL) to directly optimize the model for generating correct and well-formatted reasoning traces, moving beyond the imitation learning paradigm of SFT.

Reinforcement Learning offers a framework for training agents (in this case, LLMs) to make sequences of decisions (generating tokens) to maximize a cumulative reward signal. Unlike SFT, which requires explicit input-output pairs, RL allows models to learn from scalar feedback signals indicating the quality of their generated outputs. This is particularly useful when trying to optimize for tasks that don't have a single ground-truth completion.

A common application of RL in LLMs is Reinforcement Learning from Human Feedback (RLHF) [26, 27]. In RLHF, human preferences between sets of model generations are collected and used to train a separate reward model. This reward model learns to give LLM completions a scalar score. Subsequently, the LLM policy is fine-tuned using an RL algorithm (often Proximal Policy Optimization, PPO [28]) to maximize the scores assigned by this learned reward model, while typically including a penalty term (e.g., KL divergence) to prevent the policy from deviating too far from the original model.

In our case, however, we can move past RLHF to doing pure reinforcement learning. We do not need expensive human preference annotation, because our tasks involve calculations with a single correct answer. We can assign rewards based on the correctness of the LLM's predicted answer and optimize the total reward.

Several RL algorithms have been adapted for LLM fine-tuning. While PPO is widely used, it can cause instability when performing pure correctness-based RL with it due to its KL divergence penalty. Group Relative Policy Optimization (GRPO) [29] has emerged as the most effective algorithm for reasoning-based RL. GRPO adapts the PPO framework by incorporating relative reward information from a group of generated outputs. Instead of relying solely on the absolute advantage $\hat{A}^{\pi_{ref}}(x, y)$ for a single completion, GRPO modifies the advantage calculation based on the rewards within a sampled group. The GRPO objective function, as presented in [29], is:

$$\mathcal{J}_{GRPO}(\theta) = \mathbb{E}_{q\sim P(Q),\{o_i\}_{i=1}^{G}\sim\pi_{\theta_{old}}(o|q)}$$
$$\left[\frac{1}{G}\sum_{i=1}^{G}\left(\min\left(\frac{\pi_\theta(o_i|q)}{\pi_{\theta_{old}}(o_i|q)}A_i, \text{clip}\left(\frac{\pi_\theta(o_i|q)}{\pi_{\theta_{old}}(o_i|q)}, 1-\epsilon, 1+\epsilon\right)A_i\right) - \beta\mathbb{D}_{KL}(\pi_\theta||\pi_{ref})\right)\right] \quad (5)$$

where q is the prompt (question), o_i represents the i-th output (completion) in a group of size G sampled from the old policy $\pi_{\theta_{old}}$, π_{ref} is a reference policy (the original model), ϵ and β are hyperparameters, and $\mathbb{D}_{KL}$ is the Kullback-Leibler divergence used as a regularization term to prevent large policy shifts. The key difference lies in the calculation of the advantage A_i for each output o_i within the group, which incorporates relative reward information:

$$A_i = \frac{r_i - \text{mean}(\{r_1, r_2, \ldots, r_G\})}{\text{std}(\{r_1, r_2, \ldots, r_G\})} \quad (6)$$

where $\{r_1, r_2, \ldots, r_G\}$ are the rewards corresponding to the outputs $\{o_1, o_2, \ldots, o_G\}$ within the group. By normalizing the reward r_i relative to the mean and standard deviation of rewards within the group, the advantage A_i reflects how much better or worse a particular output is compared to the other outputs generated for the same prompt under the current policy.

We applied GRPO to fine-tune the base Qwen 2.5-0.5B-Instruct model. We started from the pre-trained weights without any prior SFT on financial data for this specific experiment run. For each question in our `FinReason` training set, we prompted the model to generate $K = 8$ distinct completions (reasoning traces and answers).

We defined an automated reward function to evaluate these completions. Each completion y received a reward based on two criteria:

- **Formatting Correctness:** A reward component of +0.5 was given if the completion correctly used the required XML-style tags, specifically enclosing the reasoning steps within '<reasoning>...</reasoning>' tags and the final numerical answer within '<answer>...</answer>' tags.
- **Answer Correctness:** An additional reward component of +2.0 was given if the numerical value extracted from the '<answer>' tag was correct.

Thus, a completion could receive a total reward $R(x, y)$ of 0 (incorrect format, incorrect answer), 0.5 (correct format, incorrect answer), or 2.5 (correct format, correct answer).

The GRPO algorithm was then used to update the model's policy π_θ by maximizing the objective in Equation 5, based on the relative rewards assigned to the 8 completions generated for each question. The objective was to increase the probability of generating the completion(s) that received the highest reward within each group of 8. The training setup (LoRA, hardware, libraries) was kept consistent with the previous experiments where applicable to RL training. The resulting model is referred to as *FinReason-Qwen-0.5B-GRPO*.

4.3.4 Evaluation

After these three training experiments, we evaluated the resulting models on a held-out test set from `FinReason`, as well as on questions from FinQA [14], a numerical reasoning dataset that asks questions of data from public companies' documents.

4.4 Long-context question-answer assistant

We developed a question-answering assistant capable of processing and reasoning over multiple large financial documents simultaneously. We leverage recent advances in long-context language models to provide financial professionals with quick and accurate answers based on sources like SEC filings, earnings call transcripts, and investor presentations.

The core of our application is the Gemini 2.0 Flash model. This model was selected primarily for its exceptionally long context window (up to 1 million tokens), which enables ingestion of several long financial documents, such as SEC filings and private company CIMs, which can sometimes exceed 400 pages in length.

Furthermore, the Gemini series of models are natively multimodal, enabling us to ingest a variety of financial document modalities:

- **PDF Documents:** Text is extracted directly from PDF files (e.g., SEC filings, transcripts) using standard parsing libraries. This allows the model to process the textual content natively. Table text is also extracted. Charts in the PDF are extracted and passed to Gemini as images, enabling visual reasoning.
- **Slides and Spreadsheets:** We render these documents page-by-page or sheet-by-sheet into images. These images are then provided alongside the extracted text (if any) to the model, enabling interpretation.

By combining extracted text and rendered images from various sources, we construct a comprehensive multimodal context that the model can process to answer user queries.

4.4.1 How models expand their context windows

Early LLMs had limited context windows, which hampered their utility in many real-world question-answer use cases. Models like Gemini 2.0 use a combination of architectural innovations and training techniques to handle long context windows of millions of tokens. While Gemini's specific architectural details are unknown, we can infer based on open-source innovations.

In a typical transformer generating text autoregressively, the calculation for each new token requires attending to all previous tokens in the sequence. The self-attention mechanism computes query, key, and value vectors for each token. To generate the $(n+1)^{th}$ token, the model needs the key and value vectors for all preceding n tokens. Recomputing these vectors for every previous token at each generation step would be prohibitively expensive.

The KV cache is the mechanism designed to prevent this redundant computation. It stores the computed Key and Value vectors for each token in the sequence across all attention layers. When generating the $(n+1)^{th}$ token, the model computes only the Q vector for the n^{th} token and retrieves the pre-computed K and V vectors for tokens 1 to n from the cache. This makes the generation of subsequent tokens much faster.

There are two main hindrances to long-context inference that need to be overcome. First is that the memory required during inference grows with context length. For a sequence of length L, L keys and values from the KV cache need to be loaded into memory to generate a new token. This can quickly

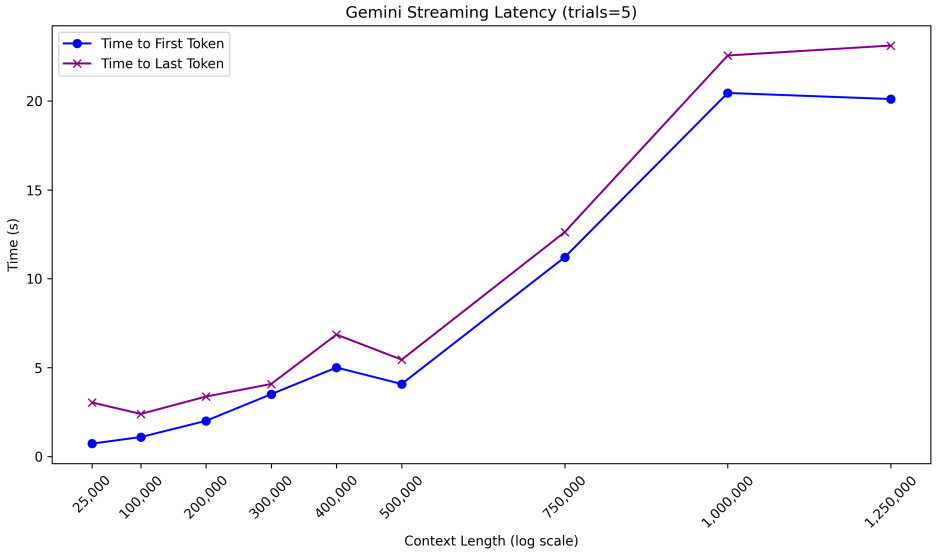

Figure 2: Gemini 2.0 Flash's first-token and last-token latency (normalized for response length) grow as context length increases.

grow out of control. Second is the need for long-context training: the model needs to see extremely long sequences to understand how to pull information from them.

One innovation for the memory issue is InfiniAttention [30], which overcomes the quadratic complexity of attention by modifying the layer to make use of a global memory. In InfiniAttention, each token can only actually attend to a fixed sliding window behind it, such as the previous 256 tokens. However, there is also a fixed global memory that stores some previous keys and values, which are updated dynamically for each new generated token. So, during inference time, a new token first attends to the previous S tokens in the sliding window range, and also the M key/values that are in global memory. That token's own key and value are added to the sliding window KV cache, and then incorporated into the global memory. This turns attention into a linear operation and enables it to scale to potentially infinite context lengths – instead of needing to store all L keys and values in memory, we only need to store $S + M$, which need not grow with L.

Another possible innovation is a new form of positional encoding called LongRoPE [31]. Traditional positional encodings break down at long context lengths because the sinusoidal curves used to assign positions begin to collide. LongRoPE extends these encodings, allowing them to continue providing distinct positional signals even at longer contexts.

Gemini was also likely trained on a large quantity of long-sequence data, such as books or financial filings.

4.4.2 Addressing Long-Context Latency

A significant challenge when working with very long context windows is the increased latency, particularly the time-to-first-token (TTFT). The increased sequence length adds latency due to more computation steps and also more time to load tensors in and out of GPU memory.

Before the first new token can be generated, the KV cache must be populated for the entire input prompt sequence. This initial computation phase is known as the prefill stage. During prefill, the K and V vectors for every token in the input prompt must be calculated and stored in the cache. For very long contexts (e.g., hundreds of thousands or a million tokens), this involves a massive amount of computation, as the attention mechanism (even optimized variants) still needs to process the entire input sequence. Consequently, the prefill stage for long contexts can take a considerable amount of time, leading to high TTFT latency, even though subsequent token generation is relatively fast.

We benchmark the TTFT and time-to-last-token (TTLT) latencies for Gemini 2.0 Flash. These latencies are shown in Figure 2, indicating that both latencies grow considerably with context length.

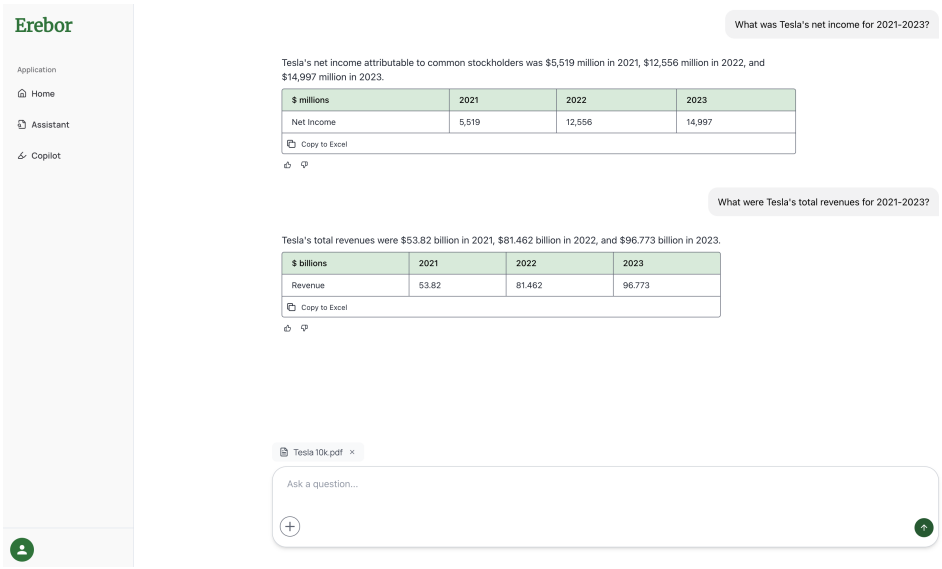

Figure 3: User interface for the financial question-answer assistant.

To mitigate this latency and ensure a responsive user experience, we implemented two key strategies:

1. **Intelligent preprocessing:** Before sending documents to the model, we implemented a preprocessing step to identify and select only the most relevant pages or sections based on the user's query or anticipated information needs. First, for each page in an uploaded PDF or text document, we check the first few words (usually a title or section header). We use parallel LLM calls to decide which of these pages are likely to be relevant for the query. We further check whether certain pages have tables or charts, and include all such pages.
2. **Prompt caching:** In scenarios where the user asks multiple follow-up questions, we want to avoid repeating the pre-fill stage for each new user message, since the same documents are still in context. We can utilize prompt caching, available from the Gemini API, to store the computed KV cache for all of the tokens that are in-common between successive calls. This allows us to re-use stored KVs for the initial context. Subsequent queries only require processing the new tokens (the user's new message), drastically reducing prefill computation time and significantly improving TTFT.

These optimizations allow the application to harness the power of the long context window while maintaining acceptable performance for an interactive assistant.

We also build a user interface for the assistant that enables users to upload documents and ask questions about them (Figure 3).

Finally, we evaluate the assistant on FinanceBench [15], a question-answer dataset on public company SEC filings.

4.5 Financial report audit copilot

We developed an automated financial document auditing application to improve the consistency review process. The primary goal of this framework is to identify numerical discrepancies and inconsistencies within a given document, such as a financial filing or report. This includes verifying explicit calculations (e.g., growth rates) and ensuring that related figures presented throughout the document (in tables, charts, or free text) align logically. For example, if a draft filing states that 2022 revenue was 500, 2023 revenue was 550, and 2023 revenue growth is 20%, this would be a discrepancy that might not be caught until an auditor manually reads through the document. Our application is intended to catch such a discrepancy automatically. The framework operates in two main stages: claim extraction and discrepancy identification. A visual overview is provided in Figure 4.

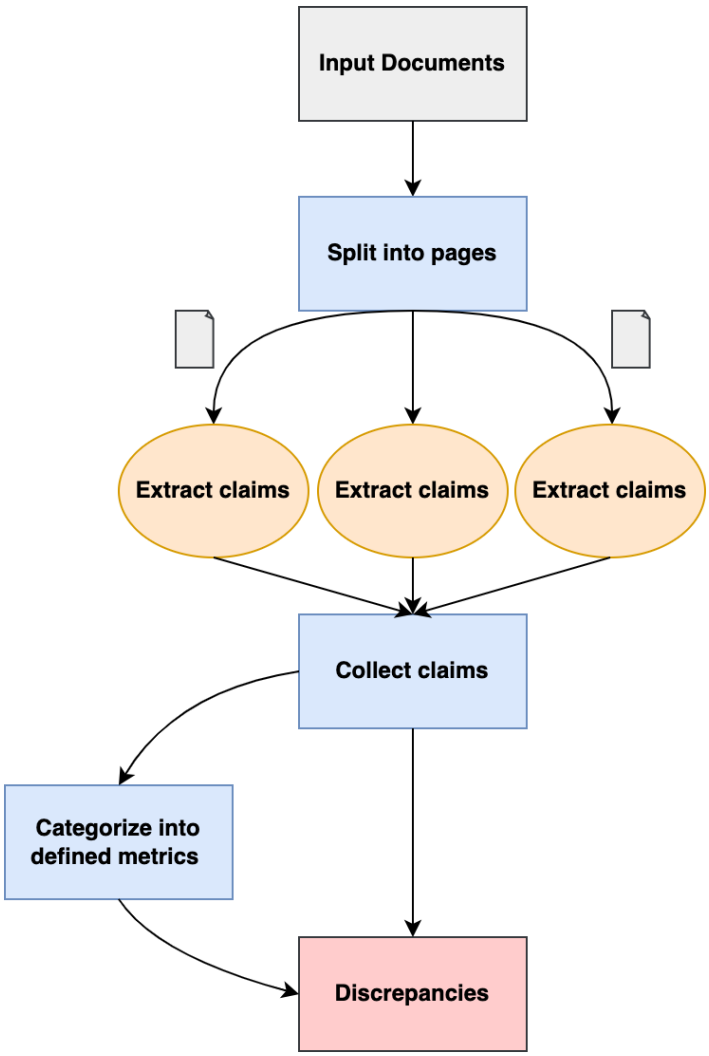

Figure 4: Algorithm used to find discrepancies in numerical claims made in documents.

4.5.1 Parallel claim extraction

The initial stage involves identifying and extracting all quantitative claims made within the input document. A "claim" is defined as any statement asserting a specific numerical value for a financial metric, potentially associated with a specific entity, time period, or unit (e.g., "Company X reported $550 million in revenue for 2023," or "2023 YoY growth was 10%").

We employ Gemini 2.0 Flash for this extraction task. The input document is first processed to extract its raw text content on a page-by-page basis. The LLM is then tasked with identifying and structuring quantitative claims found within the text of each page. The prompt guides the model to extract key components of each claim, such as:

- The metric being reported (e.g., Revenue, Net Income, Growth Rate).
- The numerical value associated with the metric.
- The time period (e.g., Q4 2023, FY2022).
- Units (e.g., Millions USD, %).
- The source sentence or context within the page.

This approach allows the extraction of claims from both structured formats (like tables) and unstructured free text narratives.

Processing potentially lengthy financial documents page-by-page with an LLM can introduce significant latency, making the application slow for users submitting large filings. To mitigate this, we parallelize the claim extraction process. The document is split into individual pages (or small batches of pages), and multiple instances of the LLM extraction task are run concurrently using multithreading. Each thread processes a different page/batch, and the extracted claims are collected centrally once all threads complete. This significantly reduces the wall-clock time required for claim extraction compared to sequential processing, making the initial analysis faster for the end-user.

4.5.2 Discrepancy identification

Once a comprehensive list of quantitative claims has been extracted from the entire document, the second stage focuses on identifying potential inconsistencies or errors among these claims. We employ a hybrid approach combining rule-based formulaic checks with LLM-based reasoning.

First, we use the same dependency graph created in the synthetic data generation step to assess metric consistency. We first ask the LLM to classify extracted claims into metrics in our dependency graph. Then, for each claim classified as a derived metrics, we re-calculate the derived metric using the values from other relevant extracted claims (treating them as the necessary basic metrics). The calculated result is then compared to the explicitly stated value for that derived metric. For example, if we extracted claims for 2022 Revenue ($500M) and 2023 Revenue ($550M), the system calculates the growth rate (10%). If another claim explicitly states "2023 Revenue Growth was 20%," a discrepancy is flagged.

However, many potential inconsistencies may not fit simple, predefined formulas in our limited dependency graph, or may involve more nuanced relationships described in the text. Furthermore, formulaic checks might miss inconsistencies if one of the input values is itself incorrect but consistently used. To address this, we employ a second LLM-based verification step. Related claims (e.g., all claims about revenue across different years, claims about related profitability metrics, claims mentioned in proximity within the text) are grouped together. We prompt Gemini 2.0 Flash with these groups of related claims, asking the model to review the claims and identify any numerical or logical inconsistencies. If the model identifies a potential inconsistency, it returns all of the claims involved in the discrepancy and provides a reasoning.

We also build a user interface that enables users to upload their own documents and clearly view the discrepancies present. After identifying discrepancies, we also surface the source texts for the claims involved in each discrepancy, allowing the user to click on any claim and see the exact text for it in the document. This enables quick human confirmation of any reported discrepancies. The user interface is shown in Figure 5.

We furthermore measure the accuracy of both the claim extraction and discrepancy identification steps to assess the performance of our system. Results are presented in the Experimental Result section.

4.6 Excel model generation agent

We develop an LLM-based agent designed to automate the creation of single-sheet financial models directly in Microsoft Excel format. Given a user prompt describing the desired model and potentially a data source (e.g., CSV file, structured text), the agent aims to generate a complete Excel file incorporating accurate data, formulas instead of hardcoded values, and appropriate formatting.

In the context of LLMs, an agent is a system that uses an LLM, or a series of LLMs, as its core reasoning engine to interact with its environment and achieve goals. Unlike a simple LLM call that just processes input and returns output, an agent can perceive the state of its task by observing the environment, plan a series of steps to achieve its goal, and observe the effect of its intermediate actions to correct course. The most prominent agent implementation is a loop where the LLM plans its next steps, takes action, observes the outcome, and then decides the next steps based on those outcomes.

A popular agent approach is ReAct, which stands for reason and act. [32]. The key idea behind ReAct is to explicitly prompt the LLM to interleave reasoning steps with action steps, forming a

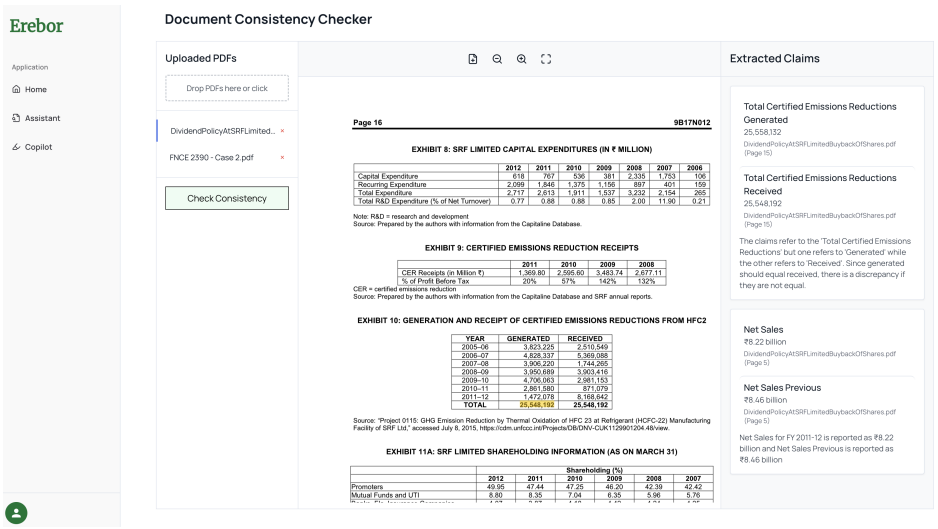

Figure 5: User interface of the document consistency auditor. Users can upload files they want to cross-check, analyze for inconsistencies, and view the exact text that was detected as inconsistent.

cycle of plans, actions, and observations. First, the LLM analyzes the current goal and the state of its environment. It generates a reasoning trace outlining its understanding of what needs to be done next. Then, based on this, the LLM formulates an action. This might involve calling a specific tool, querying a database, generating code, etc. Then, the LLM executes the action and observes the result, feeding it back into another planning prompt to decide the next action. This cycle allows the LLM to break down difficult tasks into tractable subgoals, use external tools as it sees fit, track progress, handle errors, and dynamically adapt its strategy.

Our specific use case of generating structured Excel spreadsheets with intricate formulas and cell formatting presents a unique challenge for LLMs. While we considered structured representations like JSON or XML for representing spreadsheet content, preliminary evaluations suggested that the best approach was asking the LLM to generate code, which could then be executed to generate the spreadsheet.

Therefore, we adopted code generation as the primary tool that the LLM would work with. The agent utilizes the `openpyxl` library within a Python environment. Instead of attempting to generate the entire complex Excel file in one shot, the agent breaks down the model creation task into logical, sequential steps (e.g., setting up headers, inputting historical data, writing projection formulas, applying formatting). For each step, the agent generates Python code using `openpyxl` functions to modify or build upon the Excel workbook.

The agent employs a multi-step, iterative refinement loop, closely following ReAct, involving two primary LLM roles: a Code Generator and an Evaluator.

1. Plan the next sub-task based on the user's goals. We keep track of which sub-task we are currently working on.

2. For the current sub-task, prompt the Generator LLM (we use GPT-o1) to write Python code using `openpyxl`. The prompt includes the specific sub-task and potentially a representation of the current Excel sheet's state (derived from the evaluation step).

3. The generated Python code snippet is executed. This execution results in the creation or modification of the target `.xlsx` file.

4. To assess the progress and correctness of the generated model at the current step, the content of the newly modified Excel sheet is read as a text table. This representation, along with the original sub-task goal and potentially the executed code, is passed to the Evaluator LLM. The Evaluator LLM is prompted to assess the quality, correctness, and completeness of the current Excel state with respect to the sub-task objective. It identifies

errors, missing elements, incorrect formulas (by analyzing the DataFrame structure and values), or formatting issues.

5. The feedback from the Evaluator LLM is used to refine the prompt for the Generator LLM in the next iteration for the *same* sub-task. If the Evaluator identifies issues (e.g., "Revenue growth formula is incorrect in column E," "Historical data was loaded into the wrong rows"), the Generator LLM is asked to correct the previously generated code. This loop (Generate -> Execute -> Evaluate -> Feedback) continues for the current sub-task until the Evaluator LLM confirms that the sub-task is satisfactorily completed or a maximum number of iterations is reached.
6. Once a subtask is deemed complete, the agent moves to the next sub-task in the sequence, building incrementally upon the Excel file generated in the previous steps.

This iterative process allows the agent to handle complex model creation by breaking it down and incorporating self-correction based on execution results and LLM-driven evaluation.

We evaluated the quality of the generated models for one specific type of model: the paper LBO. In this type of model, a short set of assumptions is given about a target company and an LBO transaction, and simple models are created to assess the expected return on the transaction.

5 Experimental Results

5.1 Evaluating frontier models on `FinReason`

We evaluated how well current leading LLMs performed on `FinReason`, our synthetically generated financial reasoning dataset. We aimed to show that `FinReason`, despite being created synthetically, would be challenging even for frontier LLMs.

We evaluated several models from top providers in industry, including OpenAI, Google, and Anthropic. Crucially, we evaluated both reasoning and non-reasoning models.

Our evaluation dataset was the test set of `FinReason`. Each model was prompted with the same system prompt, which specifies the output format:

> Respond in the following format: <reasoning> ... </reasoning> <answer> ... </answer>

, as well as a user prompt specifying the task and reiterating the desired output format:

> Given financial data about a company, your task is to answer a question about the company. Reason about the question step-by-step, putting your thinking between <reasoning> and </reasoning> XML tags and then put your final numerical answer between <answer> and </answer> tags.
>
> It is crucial that all text is between either <reasoning> or <answer> tags.
> Financials: {data}
>
> Question: {question}

In total there were 300 test examples. We ran three evaluation trials per model. For ground truths that were numbers, we considered a model's answer correct if it was within 1% of the ground truth. For yes/no ground truths, we checked for absolute correctness.

The results showed that even frontier models struggle with financial reasoning (Figure 6). OpenAI's recent models perform the best, with reasoners such as o1 and o3-mini achieving 76% and 75% accuracy, respectively. However, no model was able to reach a high enough threshold of accuracy that would lead a financial institution to trust it in production workflows.

The difficulty that these models face in answering even simple income statement and balance sheet metrics show that more work is needed to train them in understanding financial terminology, principles, and in performing calculations.

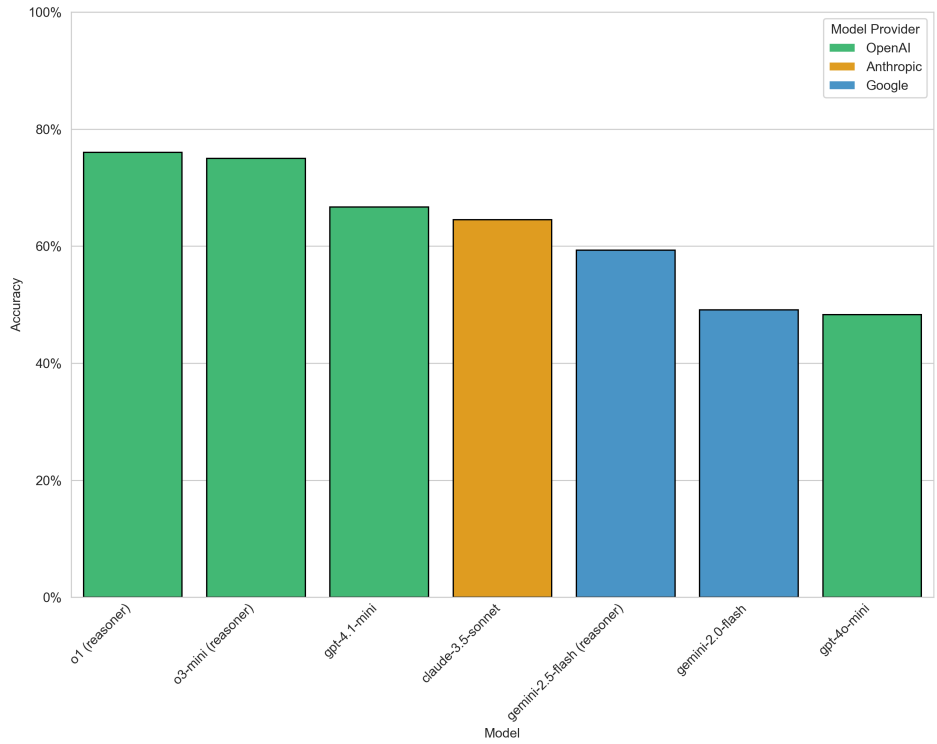

Figure 6: Performance of popular frontier LLMs on the `FinReason` test set.

Furthermore, these results demonstrate the utility of `FinReason` as an evaluation benchmark for frontier LLM's reasoning capabilities. Humans trained in basic financial accounting, such as through an undergraduate curriculum, would not struggle to answer these questions. This highlights the gap between current model capabilities and human expertise. `FinReason` could serve as a reasonable benchmark to assess when models have sufficiently understood basic financial reasoning.

5.2 Training models on `FinReason`

We trained Qwen-2.5-0.5B-Instruct on `FinReason` using three different methods: SFT on synthetic reasoning traces, distillation-based SFT on traces from OpenAI $o1$, and pure reinforcement learning using GRPO. After training, we evaluated the models on a held-out test set from `FinReason`, as well as FinQA, an external financial numerical reasoning dataset. The results are shown in Table 1.

Model	FinReason Test Accuracy (%)	FinQA Accuracy (%)
Base Qwen-2.5-0.5B	22	14
FinReason-SFT	38	18
Distillation-SFT	35	21
GRPO	40	17

Table 1: Accuracies of models trained according to the three techniques.

Our results show that all three methods were able to improve performance of the base model on both test sets. The methods led to larger accuracy gains on `FinReason` test, which could simply be because that test data is more similar to the distribution of the training dataset, while the FinQA questions are slightly different in style and context. Notably, the GRPO method led to the largest gains on `FinReason` test, showing that the model was able to learn its own reasoning patterns for this dataset. However, distillation, led to the largest gains on FinQA, likely because the reasoning traces from o1 were high-quality and general enough to transfer to FinQA.

5.3 Evaluating the long-context question-answer assistant

We evaluated the performance of our long-context financial question-answering assistant, focusing onboth its accuracy in retrieving and reasoning over financial information.

We evaluated the system on the FinanceBench dataset. FinanceBench is specifically designed to test the performance of language models on questions requiring information extraction and reasoning based on public companies' SEC filings (10-K and 10-Q reports). This benchmark aligns well with the intended use case of our assistant.

Our system achieved an accuracy of 83% on the FinanceBench benchmark. This result compares favorably to the performance of models reported in the original FinanceBench paper, demonstrating the effectiveness of utilizing a state-of-the-art long-context, multimodal model for this financial Q&A task. The high accuracy suggests that the model can successfully navigate and synthesize information contained within complex financial disclosures to answer user queries correctly.

5.4 Evaluating the document consistency auditor

We evaluated the performance of the LLM-based financial document auditing framework we developed.

5.4.1 Evaluation Methodology

Evaluating the end-to-end performance, particularly discrepancy identification on real-world documents, is challenging due to the lack of readily available ground-truth datasets with annotated inconsistencies. Therefore, our primary quantitative evaluation focused on the claim extraction component.

To assess claim extraction accuracy, we generated a synthetic dataset. We defined a set of common financial line items and time periods (e.g., "Total net sales," "2023"). For varying numbers of claims ($k \in \{2, 5, 10, 20, 50\}$), we randomly selected k line item/time period pairs and generated random but plausible values (e.g., dollar amounts, percentages) for them. We then used Gemini 2.0 Flash to write realistic-sounding paragraphs incorporating these k claims naturally, mimicking language found in financial filings. This process yielded pairs of structured ground-truth claims, with values and time periods, and corresponding paragraphs containing those claims embedded within narrative text. We generated $N = 100$ such examples for each value of k.

Using this synthetic dataset, we ran our claim extraction pipeline on the generated paragraphs and compared the extracted claims against the ground-truth structured claims used to generate the text. We calculated standard information retrieval metrics such as precision, recall, and F1-score for claim identification.

5.4.2 Claim extraction performance

The accuracy of the claim extraction stage is crucial, as missed or incorrectly extracted claims cannot be used in the subsequent discrepancy analysis. We evaluated the performance as a function of the number of claims (k) embedded within the synthetic paragraphs.

The results, summarized in Figure 7, indicate that the LLM-based extraction performs reasonably well, particularly when the density of claims within the text is lower (smaller k). However, as the number of distinct claims within a given text passage increases, performance tends to degrade, particularly recall (i.e., the model misses some claims). This suggests that while LLMs can effectively extract quantitative information from narrative text, achieving comprehensive extraction in dense, number-heavy financial reporting remains a challenge. Despite this, the achieved accuracy demonstrates the feasibility of using LLMs for this task as a core component of the auditing framework, providing a substantial base of claims for the subsequent analysis stage.

5.5 Discrepancy Identification

Evaluating the discrepancy identification stage quantitatively is complex. However, qualitative analysis using real-world document snippets demonstrates the potential of the hybrid approach.

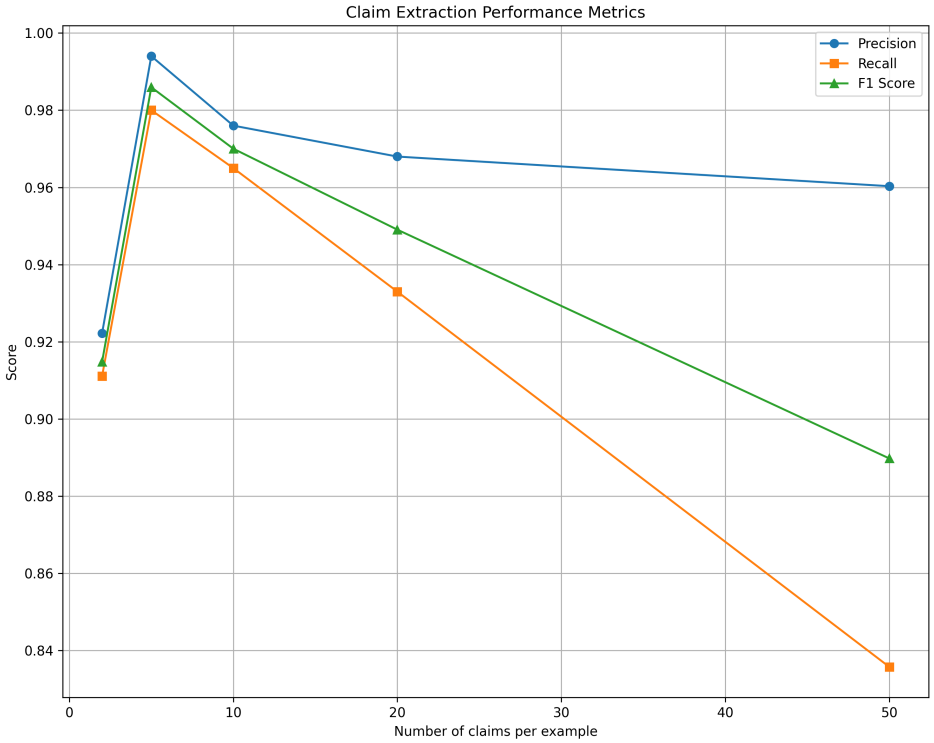

Figure 7: Claim extraction performance as a function of the number of ground-truth claims present in the synthetic text passages.

We observed that the LLM-based consistency reasoning approach proved capable of identifying more nuanced contradictions and inconsistencies that were not covered by explicit formulas. For instance, it could flag inconsistencies between growth rates stated in text and values implied by tables on different pages. While not exhaustive, this LLM-based check acts as a valuable complement to the rule-based system. The user interface, which surfaces the source text for conflicting claims (Figure 5), allows users to quickly validate flagged discrepancies. Further research could involve creating specific benchmarks to quantify the recall and precision of different types of discrepancy detection.

5.6 Evaluating the paper LBO Excel generator

To create an evaluation dataset, we scraped the Internet for sample paper LBO assumptions and augmented the dataset synthetically by replacing certain numerical assumptions with other numbers. We used our agent to generate Excel models for each set of assumptions (Figure 8). We then showed each generated model, along with the assumptions that were inputted into it, to a group of three Wharton students who rated the generated models on three axes: correctness, formatting, and comprehensiveness. Each axis was scored on a scale of 0-2, with 0 meaning no marks (wildly incorrect, wildly misformatted and lacking key parts of the model), 1 meaning partial marks (some mistakes, some formatting issues, some parts missing) and 2 meaning full marks (comparable to the quality of an actual paper LBO model).

After generating $n = 53$ different models and showing them to our team of student evaluators, we obtained average scores of 1.1 in correctness, 1.4 in formatting, and 1.3 in comprehensiveness, showing that our model was successful in producing outputs that were correctly created, well formatted, and comprehensive.

ENTRY ASSUMPTIONS	
Year 1 Revenue	100
EBITDA Margin	40.0%
Entry Multiple	5.0
Year 1 EBITDA	40
Purchase Price	200
FORECAST ASSUMPTIONS	
Revenue YoY Growth	10.0%
Capex as % of Revenue	15.0%
Tax Rate	40.0%
Annual Working Capital Increase	5
Depreciation	20

FORECAST (in millions)	Year 1	Year 2	Year 3	Year 4	Year 5
Revenue	100	110	121	133.1	146.41
EBITDA	40	44	48.4	53.24	58.564
EBIT	20	24	28.4	33.24	38.564
Taxes	8	9.6	11.36	13.296	15.4256
NOPAT	12	14.4	17.04	19.944	23.1384
Add: Depreciation	20	20	20	20	20
Less: Capex	15	16.5	18.15	19.965	21.9615
Less: Increase in Working Capital	5	5	5	5	5
Free Cash Flow	12	12.9	13.89	14.979	16.1769

Figure 8: Sample Excel model produced by our Excel agent.

6 Discussion and Limitations

Our results demonstrate that large language models (LLMs) can be suitable for financial reasoning and workflows through a combination of synthetic data, specialized training, and domain-specific application engineering. In particular, the synthetic data generation methodology underlying `FinReason` provides an effective and scalable solution to the scarcity of annotated financial reasoning data. Moreover, the development of task-specific agents for long-context question answering, automated document auditing, and Excel model generation illustrates that LLMs can transcend simple text generation and begin to act as credible knowledge workers in high-stakes financial environments.

6.1 Impact

First, the creation of `FinReason` represents a meaningful step forward in scaling reasoning capabilities in financial domains. Our results show that even frontier models, despite extensive pretraining, underperform on financial reasoning tasks without domain-specific adaptation (Figure 6). This underscores the necessity for task-specific fine-tuning. We also showed that training on `FinReason` with SFT, distillation, and GRPO improved performance of the base model, demonstrating that domain-specific reasoning training is powerful in making models specialized for certain ways of thinking.

Second, the architecture of our long-context question-answering assistant reveals that multimodal and multi-document reasoning is feasible at scale. Achieving high accuracy on FinanceBench demonstrates that with the appropriate infrastructure for document ingestion, pre-processing, and prompt engineering, long-context LLMs can synthesize complex financial information with a high degree of reliability. This result is particularly impactful given the centrality of SEC filings and earnings materials in financial analysis workflows.

Third, the document auditing agent showcases a practical application where LLMs serve not as black-box generators but as verifiers of consistency across complex documents. By combining symbolic reasoning (through the dependency graph) with language-model-based reasoning, we bridge the gap between rule-based validation and flexible inferential reasoning. This hybrid approach is crucial for financial applications, where even minor inconsistencies can have substantial consequences.

Finally, the Excel model generation agent illustrates that with sufficient planning, action evaluation, and iteration (via the ReAct framework), LLMs can construct non-trivial artifacts like fully-formulated financial models. Although prior work has demonstrated agents that interact with code and APIs, the high precision and formatting demands of financial spreadsheets make this domain an especially strong testbed for agentic reasoning. Our agent was successfully able to generate models for paper LBOs, and obtained high ratings from student evaluators.

6.2 Broader Implications

Our findings have implications beyond finance. Many other domains—such as healthcare, law, and engineering—share characteristics with finance: high-stakes decision-making, structured and semi-structured data, and the need for verifiable reasoning. The methodologies we propose for synthetic dataset creation, multimodal long-context reasoning, hybrid validation systems, and artifact-generating agents could be adapted to workflows in these fields.

Moreover, our results emphasize that frontier LLMs, while impressive, remain insufficiently specialized to be trusted in critical decision-making pipelines without domain-specific fine-tuning and workflow adaptation. As such, the future of LLM deployment in industry likely hinges not merely on scaling model size, but on advances in specialization, tool-use, and verification.

6.3 Limitations

First, although `FinReason` provides a broad set of financial reasoning tasks, it necessarily reflects the biases and coverage limitations of its underlying dependency graph. Certain complex, nuanced financial analyses (e.g., those involving conditional valuation assumptions or scenario modeling) are not captured. Future work should aim to expand the diversity and realism of synthetic financial reasoning datasets, potentially incorporating adversarial examples or semi-synthetic examples derived from real filings. Additionally, certain tasks in finance simply do not have fixed correct answers, as different professionals would disagree on what steps to take in the task. This remains difficult to model with synthetic data.

Second, in our training experiments on `FinReason`, we experimented with a small model of only 0.5B parameters. Although our results are promising, these training experiments need to be scaled up to show that even larger models truly benefit from training on the dataset. It might be the case that larger models, trained on larger corpora, have domain knowledge of finance baked into the model, and therefore need less fine-tuning to achieve good performance. However, we still showed that frontier models still struggle on our dataset, `FinReason`, so we believe there is still considerable room for these models to be improved through finance-specific training.

Third, the long-context question-answer assistant, despite achieving 83% accuracy on FinanceBench, is not infallible. Errors in information extraction or reasoning can still occur, which could have significant consequences in finance. Its reliance on the underlying capabilities of Gemini 2.0 Flash means it inherits any limitations of that model in understanding highly complex tables, intricate chart types, or subtle linguistic nuances. Latency mitigation techniques improve responsiveness but do not eliminate the base latency associated with processing million-token contexts, especially for the initial query.

Fourth, the **audit copilot's** effectiveness is contingent on the accuracy of the claim extraction stage, which we observed degrades as claim density increases (Figure 7). The discrepancy identification relies partly on a predefined dependency graph, limiting its ability to catch inconsistencies involving metrics outside this graph or those based on complex, unstated assumptions. The LLM-based consistency check helps mitigate this but lacks guarantees and was not quantitatively evaluated for recall or precision on real-world errors.

Fifth, while the Excel model generation agent demonstrates impressive capabilities in generating simple financial models such as paper LBOs, scaling this to fully production-grade, multi-sheet, multi-scenario financial models remains a substantial challenge. We are excited to continue to work on this problem further to introduce mechanisms that debug generated models, verify the correctness of formulas in those models, and better follow user instructions.

7 Future Directions

First, future work should enhance the realism and scope of training data and evaluation benchmarks for financial reasoning. This includes expanding datasets like `FinReason` to encompass a broader range of financial concepts, accounting nuances, and real-world data complexities. This might include more complicated financial calculations such as DCF valuation, or more involved data sources from real filings.

Second, more work should go into enhancing the capabilities and robustness of the financial applications presented. This includes improving the multimodal understanding of the Q&A assistant to reliably interpret complex tables and charts, extending the audit copilot to perform consistency checks across more nuanced metrics and time periods, and advancing the Excel generation agent to handle intricate, multi-sheet financial models with dynamic linkages and customized formatting, thereby addressing more complex real-world financial workflows.

Finally, an important future direction is building trust into these models and applications through explainability. This would allow users to understand the reasoning behind model outputs, whether an answer to a question or a generated model. Additionally, uncertainty quantification would help provide users with reliable confidence estimates, alerting them when the model might be hallucinating. This is very important in finance, where incorrect outputs can potentially costs millions of dollars.

References

[1] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding, 2021. URL https://arxiv.org/abs/2009.03300.

[2] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset, 2021. URL https://arxiv.org/abs/2103.03874.

[3] Di Jin, Eileen Pan, Nassim Oufattole, Wei-Hung Weng, Hanyi Fang, and Peter Szolovits. What disease does this patient have? A large-scale open domain question answering dataset from medical exams, 2020. URL https://arxiv.org/abs/2009.13081.

[4] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017. URL https://arxiv.org/abs/1706.03762.

[5] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models, 2020. URL https://arxiv.org/abs/2001.08361.

[6] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. Emergent abilities of large language models, 2022. URL https://arxiv.org/abs/2206.07682.

[7] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katie Millican, Malcolm Reynolds, Roman Ring, Eliza Rutherford, Serkan Cabi, Tengda Han, Zhitao Gong, Sina Samangooei, Marianne Monteiro, Jacob Menick, Sebastian Borgeaud, Andrew Brock, Aida Nematzadeh, Sahand Sharifzadeh, Mikolaj Binkowski, Ricardo Barreira, Oriol Vinyals, Andrew Zisserman, and Karen Simonyan. Flamingo: a visual language model for few-shot learning, 2022. URL https://arxiv.org/abs/2204.14198.

[8] Yi Yang, Mark Christopher Siy UY, and Allen Huang. Finbert: A pretrained language model for financial communications, 2020. URL https://arxiv.org/abs/2006.08097.

[9] Jean Lee, Nicholas Stevens, and Soyeon Caren Han. Large language models in finance (finllms). *Neural Computing and Applications*, January 2025. ISSN 1433-3058. doi: 10.1007/s00521-024-10495-6. URL http://dx.doi.org/10.1007/s00521-024-10495-6.

[10] Srija Mukhopadhyay, Adnan Qidwai, Aparna Garimella, Pritika Ramu, Vivek Gupta, and Dan Roth. Unraveling the truth: Do vlms really understand charts? a deep dive into consistency and robustness, 2024. URL https://arxiv.org/abs/2407.11229.

[11] Archita Srivastava, Abhas Kumar, Rajesh Kumar, and Prabhakar Srinivasan. Enhancing financial vqa in vision language models using intermediate structured representations, 2025. URL https://arxiv.org/abs/2501.04675.

[12] Shijie Wu, Ozan Irsoy, Steven Lu, Vadim Dabravolski, Mark Dredze, Sebastian Gehrmann, Prabhanjan Kambadur, David Rosenberg, and Gideon Mann. Bloomberggpt: A large language model for finance, 2023. URL https://arxiv.org/abs/2303.17564.

[13] Zhaowei Liu, Xin Guo, Fangqi Lou, Lingfeng Zeng, Jinyi Niu, Zixuan Wang, Jiajie Xu, Weige Cai, Ziwei Yang, Xueqian Zhao, Chao Li, Sheng Xu, Dezhi Chen, Yun Chen, Zuo Bai, and Liwen Zhang. Fin-r1: A large language model for financial reasoning through reinforcement learning, 2025. URL https://arxiv.org/abs/2503.16252.

[14] 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. Finqa: A dataset of numerical reasoning over financial data, 2022. URL https://arxiv.org/abs/2109.00122.

[15] Pranab Islam, Anand Kannappan, Douwe Kiela, Rebecca Qian, Nino Scherrer, and Bertie Vidgen. Financebench: A new benchmark for financial question answering, 2023. URL https://arxiv.org/abs/2311.11944.

[16] Qianqian Xie, Weiguang Han, Zhengyu Chen, Ruoyu Xiang, Xiao Zhang, Yueru He, Mengxi Xiao, Dong Li, Yongfu Dai, Duanyu Feng, Yijing Xu, Haoqiang Kang, Ziyan Kuang, Chenhan Yuan, Kailai Yang, Zheheng Luo, Tianlin Zhang, Zhiwei Liu, Guojun Xiong, Zhiyang Deng, Yuechen Jiang, Zhiyuan Yao, Haohang Li, Yangyang Yu, Gang Hu, Jiajia Huang, Xiao-Yang Liu, Alejandro Lopez-Lira, Benyou Wang, Yanzhao Lai, Hao Wang, Min Peng, Sophia Ananiadou, and Jimin Huang. Finben: A holistic financial benchmark for large language models, 2024. URL https://arxiv.org/abs/2402.12659.

[17] DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

[18] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023. URL https://arxiv.org/abs/2201.11903.

[19] Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions, 2023. URL https://arxiv.org/abs/2212.10560.

[20] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners, 2019.

[21] Yun Luo, Zhen Yang, Fandong Meng, Yafu Li, Jie Zhou, and Yue Zhang. An empirical study of catastrophic forgetting in large language models during continual fine-tuning, 2025. URL https://arxiv.org/abs/2308.08747.

[22] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021. URL https://arxiv.org/abs/2106.09685.

[23] 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. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

[24] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015. URL https://arxiv.org/abs/1503.02531.

[25] OpenAI. Openai o1 system card, 2024. URL https://arxiv.org/abs/2412.16720.

[26] Paul Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences, 2023. URL https://arxiv.org/abs/1706.03741.

[27] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback, 2022. URL https://arxiv.org/abs/2203.02155.

[28] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

[29] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

[30] Tsendsuren Munkhdalai, Manaal Faruqui, and Siddharth Gopal. Leave no context behind: Efficient infinite context transformers with infini-attention, 2024. URL https://arxiv.org/abs/2404.07143.

[31] Yiran Ding, Li Lyna Zhang, Chengruidong Zhang, Yuanyuan Xu, Ning Shang, Jiahang Xu, Fan Yang, and Mao Yang. Longrope: Extending llm context window beyond 2 million tokens, 2024. URL https://arxiv.org/abs/2402.13753.

[32] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models, 2023. URL https://arxiv.org/abs/2210.03629.