FINQA: A Dataset of Numerical Reasoning over Financial Data

Anonymous ACL submission

Abstract

The sheer volume of financial statements makes it difficult for humans to access and analyze a business's financials. Robust numerical reasoning likewise faces unique challenges in this domain. In this work, we focus on answering deep questions over financial data, aiming to automate the analysis of a large corpus of financial documents. In contrast to existing tasks on general domain, the finance domain includes complex numerical reasoning and understanding of heterogeneous representations. To facilitate analytical progress, we propose a new large-scale dataset, **FINQA**, with **Q**uestion-**A**nswering pairs over **Fin**ancial reports, written by financial experts. We also annotate the gold reasoning programs to ensure full explainability. We further introduce baselines and conduct comprehensive experiments in our dataset. The results demonstrate that popular, large, pre-trained models fall far short of expert humans in acquiring finance knowledge and in complex multi-step numerical reasoning on that knowledge. Our dataset — the first of its kind — should therefore enable significant, new community research into complex application domains[1].

1 Introduction

Financial analysis is a critical means of assessing business performance, and the consequences of poor analysis can involve costs of billions of dollars (Jerven, 2013; MacKenzie, 2008). To facilitate high quality, timely decision making, professionals — such as analysts or investors — perform complex quantitative analysis to select information from financial reports. Such analysis demands advanced expertise in reasoning among heterogeneous (structured and unstructured) data sources and performing complex numerical reasoning, for example, comparing financial ratios of profitability or growth. These challenges are compounded by an exponentially expanding collection of company financial documents (MacKenzie et al., 2012; Lange et al., 2016) such that it is genuinely unclear whether dedicated human effort can produce fiscal analysis of sufficient quality for current decision making. This poses an interesting question: can we automate such deep analysis of financial data?

A few NLP studies in Question Answering (QA) explored the numerical reasoning capabilities needed to answer questions correctly. For example, the DROP dataset (Dua et al., 2019) focused on Wikipedia-based questions that require numerical reasoning, *e.g.,* "Where did Charles travel to first, Castile or Barcelona?" needs a comparison between the times of two events. However, most prior work only targeted the general domain, where the questions involve much less calculation (mostly one-step calculation) than that of the financial domain. Financial QA is more challenging than classic QA (Rajpurkar et al., 2018; Yang et al., 2018) because it requires the system to spot relevant information across heterogeneous sources, such as tables and unstructured texts, and then create a numerical reasoning path to connect all the information. It also takes substantial knowledge to ask meaningful financial questions. It is not clear how well the large language models, which performed well for general-domain QA, can be adapted to answer realistic, complex financial questions.

This paper introduces **FINQA**, a **expert-annotated** dataset that contains 8,281 financial QA pairs, along with their numerical reasoning processes. Eleven finance professionals collectively constructed FINQA based on the earnings reports of S&P 500 companies (Zheng et al., 2021). The questions in FINQA, such as "Considering the weighted average fair value of options, what was the change of shares vested from 2005 to 2006?" (Figure 1) and "What was the net change in tax

[1]We currently release the dataset at the anonymous site: https://anonymous.4open.science/r/FinQA-INIT

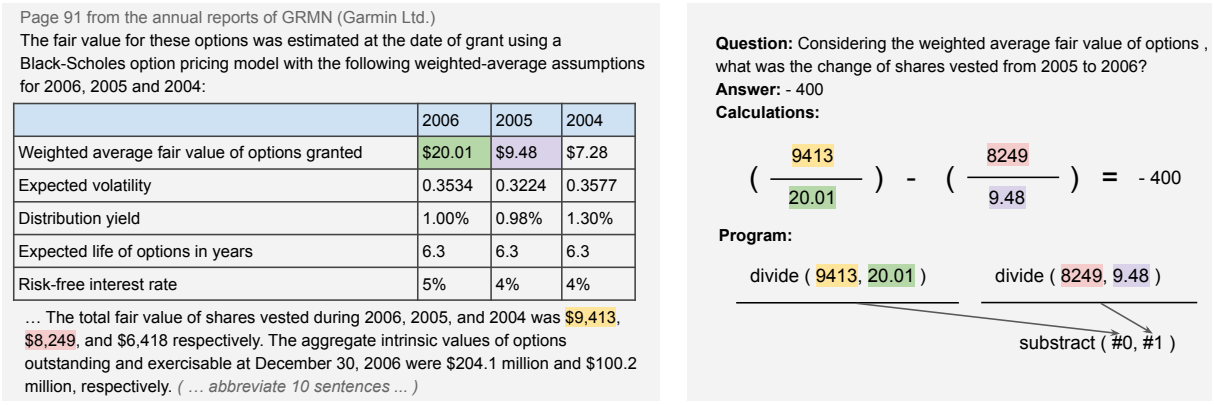

Figure 1: An example from FINQA: The system needs to learn how to calculate the number of shares, then select relevant numbers from both the table and the text to generate the reasoning program to get the answer.

positions in 2014?", require information from both tables and unstructured texts to answer. The reasoning processes answering these questions are made of many common calculations in financial analysis, such as addition, comparison, and table aggregation. To the best of our knowledge, FINQA is the first dataset of its kind to tackle complicated QA tasks based on the real-world financial documents.

We propose a retriever-generator QA framework to first retrieve supporting facts from financial reports, then to generate executable reasoning programs to answer the questions. Equipped with pre-trained language models, such as BERT (Devlin et al., 2019) and RoBERTa (Liu et al., 2019), our proposed approach outperforms all other baselines and achieves an execution accuracy of 65.05%. Although our system outperforms the non-expert crowd (50.68%), the significant accuracy gap between the model and human experts (91.16%) motivates the need for future research.

The main contribution of this work is three-fold:

- We propose the task of QA over financial data to assist financial analysis. The task emphasizes an important phenomenon for the NLP community to study and analyze how the current pre-trained models perform on complex and specialized domains.
- We construct a new large-scale dataset, FINQA, with 8,281 examples written by financial experts, with fully annotated numerical reasoning programs. The dataset and code will be made publicly available.
- We experiment on various baselines and find that the models are still far behind expert performance, strongly motivating future research.

2 Task Definition

Problem Formulation. Presented with a financial report consisting of textual contents E and structured table T, given a question Q, the task is to generate the reasoning program $G = \{w_0, w_1, ...w_n\}$, where w_i is the program tokens defined by domain specific language (DSL), then it is executed to get the answer A:

$$P(A|T, E, Q) = \sum P(G_i|T, E, Q) \quad (1)$$

Where $\{G_i\}$ is all the correct programs to evaluate to the answer. For financial tables, there is typically a description header (blue header in Figure 1), which often gives the timing information; and each row has its name on the left. Some of the financial tables may demonstrate more complicated layouts, *e.g.*, nested structures. As a first step for this direction, in this paper we only focus on the regular layout cases for simplicity.

Domain Specific Language. In this work, we use DSL consisting of mathematical operations and table operations as executable programs. The program consists of a sequence of operations:

$$op_1[\boldsymbol{args_1}], op_2[\boldsymbol{args_2}]..., op_n[\boldsymbol{args_n}] \quad (2)$$

Each operation takes a list of arguments $\boldsymbol{args_n}$. On consulting with financial experts, as most of the accounting and financial valuation theory primarily include linear algebra, we include 10 common types of operations in our dataset. There are 6 mathematical operations: `add`, `subtract`, `multiply`, `divide`, `greater`, `exp`, and 4 table aggregation operations `table-max`, `table-min`, `table-sum`, `table-average`, that apply aggregation operations on table rows.

The mathematical operations take arguments of either numbers from the given reports, or a numerical result from a previous step; The table operations take arguments of table row names. We use the special token $\#n$ to denote the result from the nth step. For example, in Figure 1, the program consists of 3 steps; The first and the second division steps take arguments from the table and the text, respectively, then the third step subtracts the results from the two previous steps. Refer to Appendix A for more details of the operations and the grammars.

Evaluations. Previous studies on QA with numerical reasoning only evaluate the execution accuracy, i.e., the final results from the generated programs, such as DROP (Dua et al., 2019) and MathQA (Amini et al., 2019). However, the applications for the finance domain generally pose much higher requirements of explainability and transparency. Therefore, we also provide the gold programs for our dataset. Besides execution accuracy, we also propose to evaluate the accuracy of the generated programs. Specifically, we replace all the arguments in a program with symbols, and then we evaluate if two symbolic programs are *mathematically equivalent*. For example, the following two programs are equivalent programs:

$$add(a_1, a_2), add(a_3, a_4), subtract(\#0, \#1)$$
$$add(a_4, a_3), add(a_1, a_2), subtract(\#1, \#0)$$

Note that execution accuracy tends to overestimate the performance because sometimes the model just hit the correct answer by chance; While program accuracy tends to produce false negatives since some questions may have multiple correct programs.

3 The FINQA Dataset

3.1 Data Preparation

Data Source. We develop FINQA based on the publicly available earnings reports of S&P 500 companies from 1999 to 2019, collected in the FinTabNet dataset (Zheng et al., 2021). An earnings report is a set of pages in a PDF file that outlines the financials of a company, which usually contains tables and texts. The FinTabNet dataset has annotated the tables in each report.

Data Filtering. Realistic earnings reports contain many tables not suitable for numerical reasoning tasks. Equipped with the table annotations in FinTabNet, we filter the data as follows: First, we extract the pages in earnings reports with at most one table. Second, we exclude the tables with over 20 rows, over 2 description headers, or with other complex nested structures. We also exclude the tables with tedious contents, such as catalogs, which is common in FinTabNet. As stated in §2, these over-complicated tables are out of the scope of this work. Finally, for the tables with 2 description headers, we merge them into a single header to simplify the representations. As a result, a total of 12,719 pages were selected for further annotation.

3.2 Annotation Procedure

Recruiting Expert Annotators. We post job ads on UpWork[2] and hire eleven US-based experts with professional finance backgrounds (CPAs, MBAs, etc.) Each hire is interviewed using four example report pages and asked to compose example Q&A pairs. After hiring, each annotator first goes through a training session to learn the task and the annotation interface (Appendix D). When the workers fully master the annotation process, we launch the official batches for them to work on.

An annotator can compose up to two questions for each given report page or skip if it is hard to compose any meaningful question. We pay around $2.0 for each question, which leads to an average hourly wage of $35.0. The whole data collection took around eight weeks.

We do not use popular micro-task platforms, such as Amazon Mechanical Turk (MTurk), because our preliminary studies show that many MTurk workers can not perform this task effectively. Our experiment with MTurk workers in § 3.3 further echo this observation. As most existing QA datasets were constructed by MTurk workers (Yang et al., 2018; Dua et al., 2019; Chen et al., 2020c), it requires substantial domain-specific knowledge to compose meaningful questions that are hard for computers to answer.

Annotation Task Design. For each page selected in §3.1, the annotators are asked to *(i)* write a meaningful financial question, *(ii)* compose a reasoning program to answer the question, and *(iii)* to annotate the supporting fact. Each page is assigned to one or two experts for annotation. We detail each part as follows. **(I) Financial question:** For a given page of earnings reports, the annotators are asked first to compose a question that is "meaningful for financial analysis or learning insights of the

[2]UpWork (www.upwork.com) is a platform where requesters can recruit skilled freelancers.

company financial reports" and require numerical calculations to answer. We encourage the experts to write questions that require the information from both the text and the table to answer. **(II) Reasoning program:** After providing the question, the annotators are then asked to elaborate the operation steps to answer the question. Specifically, they compose a maximum of 5 steps of operation, where each operation has four slots: "operation", "argument1", "argument2", and "result". The "operation" is one of the ten predefined operations described in §2. An "argument" is a number or a table's row name, either from the report or a previous step's result. For operations that only use one argument, such as table aggregation, workers can leave argument2 blank. The annotation interface (see Appendix D) automatically validates the inputs to ensure correctness. **(III) Supporting fact:** We also ask the annotators to mark all the sentences in the text and the table rows that contain the information needed to answer the question.

3.3 Data Quality Assessment

External experts answer FINQA questions with a high accuracy and a high inter-annotator agreement. To validate the quality of the annotations, as well as to set up human expert performance upper bound, we hire another two financial professionals on UpWork. We randomly sample 200 examples from our dataset, and ask the professionals to answer the questions as well as write the operation steps, following the same procedure as in the dataset construction. The payment is $2.0 per question. For execution accuracy, they reach 92.25% and 90.06%, respectively (mean = 91.16%). For program accuracy, they reach 89.44% and 85.53% (mean = 87.49%). The agreements between the two annotators are 92.65% for execution accuracy, and 86.76% for program accuracy.

Non-expert crowd workers answer FINQA questions a low accuracy. We also test how well non-expert MTurk workers can answer FINQA questions. We distribute the samples to MTurk[3] and take the similar process to distribute each example to two workers. We end up with an average execution accuracy of 50.68% and a program accuracy of 48.17%, which is far below the expert performance; the agreement rate is only around 60%. These results echo our preliminary study's observations for MTurk workers in §3.2.

[3]Three built-in worker qualifications are used: HIT Approval Rate (≥95%), Number of Approved HITs (≥ 3000), and Locale (US Only) Qualification. We do not select any profession constraints. We pay $2.0 for each question.

Examples (Q&A pairs with program, fact)	8,281
Report pages	2,789
Vocabulary	22.3k
Avg. # sentences in input text	24.32
Avg. # tokens in input text	628.11
Avg. # rows in input table	6.36
Avg. # tokens in input table	59.42
Avg. # tokens in all inputs (text & table)	687.53
Max. # tokens in all inputs (text & table)	2,679
Avg. question length	16.63

Table 1: Statistics of FINQA.

3.4 Data Analysis

FINQA contains 8,281 examples. The data is released as training (6,251), validation (883), and test (1,147) following an 75%/10%/15% split. The three sets do not have overlapping input reports. We quantitatively analyze some key properties of FINQA. Table 1 shows the general statistics.

Statistics of Supporting Facts. In FINQA, 23.42% of the questions only require the information in the text to answer; 62.43% of the questions only require the information in the table to answer; and 14.15% need both the text and table to answer. Meanwhile, 46.30% of the examples have one sentence or one table row as the fact; 42.63% has two pieces of facts; and 11.07% has more than two pieces of facts. For the examples with more than one piece of fact, we also calculate the maximum distances between all the same example's facts. 55.48% has a maximum distance of 3 or less sentences[4]; 24.35% has a maximum distance of 4-6 sentences; and 20.17% has over 6 sentences.

Statistics of Reasoning Programs. In the programs, the most frequent operations, `add`, `subtract`, `multiply`, and `divide`, have the distributions of 14.98%, 28.20%, 5.82%, and 45.29%, respectively. The operation `division` has the highest frequency, as calculating ratios is common in financial analysis. In FINQA, 59.10% of the programs have 1 step, 32.71% have 2 steps, and the rest 8.19% have 3 or more steps.

4 Baseline Systems

In this section, we first describe our main baseline framework **FinQANet** in §4.1, and then we

[4]For tables, we consider one row as one "sentence".

introduce other baselines in §4.2.

4.1 The FinQANet Framework

As a preliminary attempt on FINQA, we propose **FinQANet**, with a retriever to first retrieve the supporting facts from the input financial report, then a generator to generate the program to get the answer.

Retriever The full page of the financial report can go beyond 2,000 tokens, which cannot be coped with the current popular QA models (Devlin et al., 2019). Therefore we first retrieve the supporting facts from the input report. For the tables, we use templates to turn each row into sentences. For example, the last row of the table in Figure 1 is represented as 'the risk-free interest rate of 2006 is 5%; ...'. We concatenate each supporting fact with the question and train a classifier using pre-trained LMs like BERT (Devlin et al., 2019). Then we take the top n retrieved facts, reordered as they appear in the input report. This set of retriever results will serve as the input to the second phase. Figure 2 illustrates the retrieving procedure. Another common strategy is sliding window (Alberti et al., 2019). We take the sliding window of a fixed size with a stride to go through the report, then the windows containing all the supporting facts are marked as positive. However, we observe in the experiments that the length of the input to the program generator in the second phase greatly influences the performance. The performance of using sliding window falls far behind the previous method.

Program Generator. Given the retrieved supporting facts from the retriever, the program generator aims to generate the executable program to answer the question. Figure 3 gives an overview of the program generator. The generated tokens come from 3 sources: 1) The input passage (retriever output) and the question tokens $\{e_i\}$, like the numbers or the table row names. 2) The special tokens $\{s_i\}$ from the DSL, like the function names, predefined constants, etc. 3) The step memory tokens $\{m_i\}$ to denote the results from previous steps, like $\#0$, $\#1$, etc. We first use pre-trained LMs to encode $\{e_i\}$, denote the output embeddings as $\{h_i^e\}$. The embeddings of the special tokens and the step memory tokens are randomly initialized and denoted as $\{h_i^s\}$ and $\{h_i^m\}$ respectively. Denote all the token embeddings $H = [h_i^e; h_i^s; h_i^m]$.

An LSTM is used for decoding. At each decoding step T, the program token embeddings H are fed as the input; The decoder output h_T is used

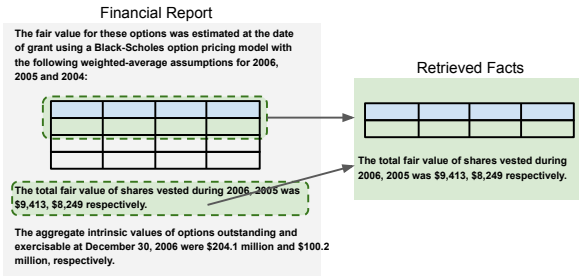

Figure 2: The retriever retrieves supporting facts (text sentences or table rows) from the input financial report.

to calculate the attention vector att_p and att_h over the input and the decoding history. Then a context vector c_T combines all the contextual information:

$$c_T = W_c[att_p; att_h; h_T] \quad (3)$$

Meanwhile, another attention vector $att_p^{'}$ over the input is applied to all the token embeddings:

$$H_T^{'} = W_h[H; H \circ att_p^{'}] \quad (4)$$

Different from other program tokens, the step memory tokens $\{m_i\}$ imply the reasoning path of the program. To make use of such structure information, at each decoding step indicating the end of one $operation[args]$ unit, i.e., the step to generate the ending parentheses in our DSL, we compute another context vector a_T:

$$a_T = W_a[att_p; att_h; h_T] \quad (5)$$

Then the step memory token embedding corresponding to the current step is updated as a_T.

The final prediction is calculated with:

$$w_T = softmax(H_T^{'} \cdot c_T) \quad (6)$$

During inference time, based on the grammar of the DSL, we use masks at each decoding step to ensure the structural correctness of the generated programs. In the retriever phase, we take the top n retrieved results as the input to the program generator. Therefore, for the training of the program generator, we use the retriever result on the training set (combined with the gold facts if there is any wrong prediction) as the input.

4.2 Other Baselines

TF-IDF + Single Op. We use TF-IDF to retrieve the top 2 sentences from the input report. Since the most common case in our dataset is one-step program and the most common operation is division, we take the first number from each sentence and apply the division operation.

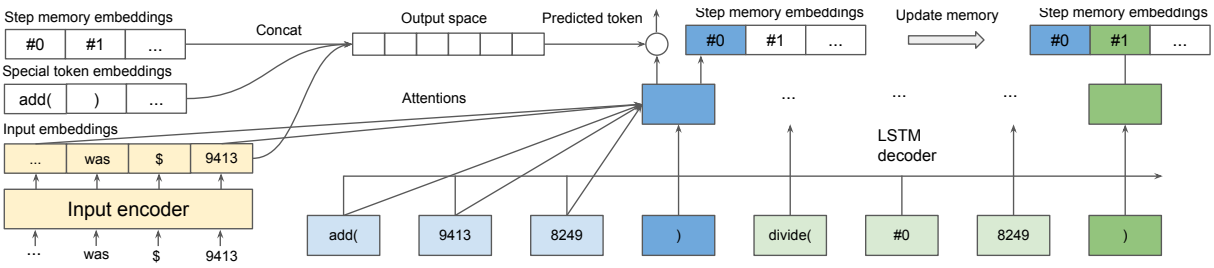

Figure 3: The program generator. The retriever results and the question are first encoded using pre-trained LMs. At each decoding step, the model can generate from the numbers or table row names from the input, the special tokens in the DSL, or the step memory tokens. At the end of the generation of each operation step, we update the step memory token embeddings.

Retriever + Direct Generation. To demonstrate the necessity of generating the reasoning programs, we keep the architecture the same as our model, but directly generating the final results.

Retriever + Seq2seq. We use a Seq2seq architecture for the generator, similar to the Seq2seq baseline in the MathQA dataset (Amini et al., 2019). A bi-LSTM is used for encoding the input, and then an LSTM is used for decoding with attention.

Retriever + NeRd. The Neural Symbolic Reader(NeRd) (Chen et al., 2020d) is also a pointer-generator based model for program generation, with the state of the art results on the MathQA dataset (Amini et al., 2019). Different from ours, it directly learns the program with nested format as a sequence, i.e., without the step memory tokens. This way the model is able to learn the program structures as patterns from very large-scale data (~40k for MathQA), but may fail on learning the reasoning paths. We keep the retriever part the same and compare with the generator part to demonstrate the usefulness of structure learning.

Pre-Trained Longformer. There are also works on modeling very long documents with thousands of characters, with the attention mechanism that scales linearly with sequence length, like the Longformer (Beltagy et al., 2020). To demonstrate the necessity of breaking up into the pipeline of retriever and program generator, we remove the retriever and directly use the pre-trained Longformer as the input encoder in the program generator, and encode the whole report. The table rows are linearized similar as in §4.1.

5 Experimental Results

Experiment Setups. For the retriever, we use BERT-base as the classifier (other pre-trained models perform similarly). Since most of the examples in our dataset have 1 or 2 facts, and we find that longer inputs lower the performance of the program generator, we take the top 3 ranked facts as the retriever results. For the program generator, we experiment on using BERT (Devlin et al., 2019), RoBERTa (Liu et al., 2019), and FinBert (Araci, 2019) as the encoder, to test the performances of popular large pre-trained models. For all models, we use the Adam optimizer (Kingma and Ba, 2015). Check Appendix B for more details of training and parameter settings.

5.1 QA Model Performance

Table 2 presents the results for all the baseline systems. We evaluate the execution accuracy (exe acc) and program accuracy (prog acc) as explained in §2. For the BERT-based retriever, we have 92.98% recall for the top 3 retrieved facts and 94.96% recall for the top 5. Using TF-IDF results in 82.91% recall for the top 5 facts. We use the same retriever results for all retriever-generator based models.

Directly generating the execution results gives near-zero scores, which indicates the necessity of generating the reasoning programs. If without using the retriever-generator pipeline, but directly applying an end-to-end pre-trained Longformer model, the performance falls far behind. Because longer inputs have more numbers which put more confusions on the program generator and thus make it harder to learn. Generally, the program generators using pre-trained models perform much better than the Seq2seq baseline, as there is language modeling knowledge that can also be used for the finance domain. And larger pre-trained models give better performance, as they tend to see more financial corpus during their pre-training. FinBert (Araci, 2019) is a pre-trained model for the finance domain; its main downstream tasks are sentiment analysis. The performance of using FinBert is no better than BERT-large, mostly because its pre-

Baselines	Exe Acc	Prog Acc
TF-IDF + Single Op	1.01	0.90
Retriever + Direct Generation	0.30	-
Pre-Trained Longformer (base)	21.90	20.48
Retriever + Seq2seq	20.40	18.29
Retriever + NeRd (BERT-base)	52.48	49.90
FinQANet (FinBert)	53.71	51.71
FinQANet (BERT-base)	54.95	53.52
FinQANet (BERT-large)	57.43	55.52
FinQANet (RoBERTa-base)	60.10	58.38
FinQANet (RoBERTa-large)	**65.05**	**63.52**
Human Expert Performance	91.16	87.49
General Crowd Performance	50.68	48.17

Table 2: The execution accuracy (Exe Acc) and program accuracy (Prog Acc) for all the models. Although our best system (65.05%) outperforms the non-expert crowd (50.68%), the significant accuracy gap between the model and human experts (91.16%) motivates the need for future research.

Methods	Exe Acc	Prog Acc
full results	**65.05**	**63.52**
Necessity of table and text		
table-only inference	41.62	40.48
text-only inference	16.38	15.33
Performances on table and text		
table-only questions	73.48	72.10
text-only questions	53.70	52.92
table-text questions	45.99	42.34
Performances regarding program steps		
1 step programs	70.27	68.77
2 step programs	63.69	61.79
>2 step programs	31.65	31.65
Programs with constants	39.80	39.80

Table 3: Performance breakdown of FinQANet (RoBERTa-large). The model benefits from using both table and text, as inferences on individual source yield much lower performance. FinQANet is better at answering table-only questions, and the questions that require more steps to solve are indeed more challenging to the model.

training corpus is limited (~30M words from news articles). Comparing FinQANet with the retriever + NeRd baseline (Chen et al., 2020d), it shows the improvements from learning the logical structure of the programs. Another interesting observation is the comparisons with human performances. While there is still a large gap from the human expert upper bound, the best performing model already surpasses the general crowd performance.

5.2 Performance Breakdown

We conduct a set of performance breakdowns using the FinQANet (RoBERTa-large) model. Table 3 shows all the results.

Necessity of using both table and text. We run inferences taking facts only from a single source from the retriever. Inferences on individual source (table-only: 41.62%, text-only: 16.38%) are both far behind the full results (65.05%).

The model performs the best on the table-only questions. The model performs the best on table-only questions (73.48%). Tables tend to have more unified structures and might be easier for the model to learn. Table 3 also shows that the questions involving both tables and texts are the most challenging ones for the model (45.99%).

Questions that need more than two steps to answer are challenging. The model has a low accuracy (31.65%) on the questions that need three or more steps. Meanwhile, not surprisingly, the questions that require only one step are the easiest.

Constants in programs. Many programs in FINQA contain constants as arguments. A constant is often used to convert an English number word to another. For example, we need first to use the constant "1,000" to convert "1.5 billion" to "1,500 million" so that it can be added with "50 million". A constant is also used to explicate the implicit numbers hidden in the language. For example, to calculate "the average for the year 2012, 2013, and 2014", the program needs to use the constant "3" as the denominator, which is not mentioned explicitly in the text. As shown in Table 3, the programs with constants yield great challenges for our model, as the performance (39.8%) is much lower than that of the whole set (65.05%).

5.3 Error Analysis

We sample 50 error cases from the results of the FinQANet (RoBERTa-large) model and analyze them manually. 14% of the errors are caused by the retriever, *e.g.*, missing facts. 38% of them are due to the lack of financial knowledge, such as the meaning of some terminology. The rest are primarily numerical reasoning errors, including complex programs with multiple steps, numerical unit con-

<table>
<tr><td>Error case (1)</td><td>Gold supporting facts: text sentence(s)
[1] additionally , we have other committed and uncommitted credit lines of $ 746 million with major international banks and financial institutions to support our general global funding needs , including with respect to bank supported letters of credit, performance bonds and guarantees .
[2] approximately $ 554 million of these credit lines were available for use as of year-end 2016 .</td><td>Question:
what is the amount of credit lines that has been drawn in millions as of year-end 2016?
Gold program: subtract(746, 554)

Predicted program: multiply(554, const_1000000)</td></tr>
<tr><td>Error case (2)</td><td>Gold supporting facts: table row(s)
<table><tr><td></td><td>shares</td><td>weighted average grant-date fair value</td></tr><tr><td>non-vested at may 31 2009</td><td>762</td><td>42</td></tr><tr><td>non-vested at may 31 2010</td><td>713</td><td>42</td></tr></table></td><td>Question: what is the percentage change in the total fair value of non-vested shares from 2009 to 2010?
Gold program:
multiply(762, 42), multiply(713, 42), subtract(#1, #0), divide(#2, #0)

Predicted program: subtract(713, 42), divide(#0, 42)</td></tr>
<tr><td>Error case (3)</td><td>Gold supporting facts: text sentence(s)
[1] we maintained a $ 1.4 billion senior credit facility with various financial institutions , including the $ 420.5 million term loan and a $ 945.5 million revolving credit facility .</td><td>Question: what is the estimated percentage of revolving credit facility in relation with the total senior credit facility in millions?
Gold program: multiply(1.4, const_1000), divide(945.5, #0)
Predicted program: divide(945.5, const_1000)</td></tr>
</table>

Figure 4: Error cases. In these examples, the retriever results all correctly cover the gold facts; thus we only present the gold facts, gold program, and the predicted program to study the errors of the program generator. We give more error cases in Appendix C, including the cases for the retriever errors. **Example 1**: The financial knowledge to calculate the 'credit lines that has been drawn'. **Example 2**: Complex reasoning of 4 steps. **Example 3**: Number unit conversion between 'billion' and 'million'.

versions, or resolving the ordering and matching of the numbers and the years. Many error cases involve both the numerical reasoning problems and misunderstandings of financial knowledge. We show three representative error cases in Figure 4.

6 Related Work

Questions Answering. There have been several QA datasets involving numerical understandings and calculations. The major source is from structured tables or knowledge bases, owning the nature to succinctly organize numerical information. Popular datasets include ComplexWebQuestions (Talmor and Berant, 2018), WikiTableQuestions (Pasupat and Liang, 2015), Spider (Yu et al., 2018), TabFact (Chen et al., 2020b), etc. For reading comprehension, the dataset most related to ours is the DROP dataset (Dua et al., 2019), which applies simple calculations over texts. The top methods on DROP typically use specific prediction heads for each kind of calculation. HybridQA (Chen et al., 2020c) targets QA over both the table and the text, but not with the focus of numerical reasoning. All these existing datasets are built upon the general domain (mostly based on Wikipedia). In contrast, our dataset focus on the finance domain, which demonstrates much more complex nature in numerical reasoning questions, combining both the structured tables and unstructured texts. Another kind of QA datasets related to ours is the math word problem datasets, like MaWPS (Koncel-Kedziorski et al., 2016), MathQA (Amini et al., 2019). The task is to generate the solution programs given a short input math problem. Existing models include (Kim et al., 2020; Chen et al., 2020a,d), etc.

Financial NLP. Financial NLP has become one of the major application domains attracting growing attentions. Previous works in finance domain include risk management to detect fraud (Han et al., 2018; Wang et al., 2019; Nourbakhsh and Bang, 2019), sentiment analysis to assist market prediction (Day and Lee, 2016; Wang et al., 2013; Akhtar et al., 2017), opinionated Question Answering (Liu et al., 2020), such as the FiQA[5] dataset built from forums and social media. Recent works attempt to develop pre-trained models specialized for finance domain (Yang et al., 2020; Araci, 2019), and the downstream tasks are mostly sentiment classifications. To the best of our knowledge, there is no previous work and dataset on building QA systems of numerical reasoning on financial reports.

7 Conclusion and Future Work

This paper introduces FINQA, a new expert-annotated QA dataset that aims to tackle numerical reasoning over real-world financial data. The questions in FINQA pose great challenge for existing models to resolve domain-specific knowledge, as well as to acquire complex numerical reasoning abilities. We propose baseline frameworks and conduct comprehensive experiments and analysis. The results show that current large pre-trained models still fall far behind the human expert performance. This encourages potential future work on developing pre-training tasks for such realistic, complex application domains. We believe FINQA should serve as a valuable resource for the research community.

[5]https://sites.google.com/view/fiqa/home

8 Ethical Considerations

Data Access and Licensing. We develop FINQA based on the publicly available earnings reports of S&P 500 companies from 1999 to 2019, collected in the FinTabNet dataset (Zheng et al., 2021). The FinTabNet dataset is publicly available under the CDLA-Permissive[6] license, which permits us to create additional annotations on top of the data ("Enhanced Data", §1.5 of CDLA) and publish the annotations ("Publish", §1.9 of CDLA).

Dataset Collection Process and Conditions. For the annotation of our FINQA dataset on Upwork, we first launch interviews of the task introduction with 4 example questions, which is paid as $30, for them to try a few examples to get informed and familiar with the task. Then based on their consents to continue working on the large-scale job, we discuss with the workers to reach agreements on the compensation before starting the large-scale job. We pay around $2.0 per question, and the hourly rates are discussed and agreed upon with both sides based on the working speed of different workers. Among all eleven US-based hires, the average hourly rate is $35.0, and the minimum and maximum hourly rates are $20 and $50, respectively. The evaluation tasks follow the similar procedure, and each question is paid as $2.0.

IRB (Institutional Review Board) Approval. This project is approved by our Institutional Review Board (IRB). The systems trained using our dataset are primarily intended to be used as augmenting human decision-making in financial analysis, but not as a replacement of human experts.

[6]CDLA-Permissive: https://cdla.dev/sharing-1-0/

References

Md. Shad Akhtar, Abhishek Kumar, Deepanway Ghosal, Asif Ekbal, and Pushpak Bhattacharyya. 2017. A multilayer perceptron based ensemble technique for fine-grained financial sentiment analysis. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*, pages 540–546. Association for Computational Linguistics.

Chris Alberti, Kenton Lee, and Michael Collins. 2019. A BERT baseline for the natural questions. *CoRR*, abs/1901.08634.

Aida Amini, Saadia Gabriel, Shanchuan Lin, Rik Koncel-Kedziorski, Yejin Choi, and Hannaneh Hajishirzi. 2019. Mathqa: Towards interpretable math word problem solving with operation-based formalisms. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 2357–2367. Association for Computational Linguistics.

Dogu Araci. 2019. Finbert: Financial sentiment analysis with pre-trained language models. *CoRR*, abs/1908.10063.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *CoRR*, abs/2004.05150.

Kezhen Chen, Qiuyuan Huang, Hamid Palangi, Paul Smolensky, Kenneth D. Forbus, and Jianfeng Gao. 2020a. Mapping natural-language problems to formal-language solutions using structured neural representations. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 1566–1575. PMLR.

Wenhu Chen, Hongmin Wang, Jianshu Chen, Yunkai Zhang, Hong Wang, Shiyang Li, Xiyou Zhou, and William Yang Wang. 2020b. Tabfact: A large-scale dataset for table-based fact verification. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Wenhu Chen, Hanwen Zha, Zhiyu Chen, Wenhan Xiong, Hong Wang, and William Yang Wang. 2020c. Hybridqa: A dataset of multi-hop question answering over tabular and textual data. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings, EMNLP 2020, Online Event, 16-20 November 2020*, pages 1026–1036. Association for Computational Linguistics.

Xinyun Chen, Chen Liang, Adams Wei Yu, Denny Zhou, Dawn Song, and Quoc V. Le. 2020d. Neural symbolic reader: Scalable integration of distributed and symbolic representations for reading comprehension. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Min-Yuh Day and Chia-Chou Lee. 2016. Deep learning for financial sentiment analysis on finance news providers. In *2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, ASONAM 2016, San Francisco, CA, USA, August 18-21, 2016*, pages 1127–1134. IEEE Computer Society.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: pre-training of

deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. 2019. DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 2368–2378. Association for Computational Linguistics.

Jingguang Han, Utsab Barman, Jer Hayes, Jinhua Du, Edward Burgin, and Dadong Wan. 2018. Nextgen AML: distributed deep learning based language technologies to augment anti money laundering investigation. In *Proceedings of ACL 2018, Melbourne, Australia, July 15-20, 2018, System Demonstrations*, pages 37–42. Association for Computational Linguistics.

Morten Jerven. 2013. *Poor numbers: how we are misled by African development statistics and what to do about it*. Cornell University Press.

Bugeun Kim, Kyung Seo Ki, Donggeon Lee, and Gahgene Gweon. 2020. Point to the expression: Solving algebraic word problems using the expression-pointer transformer model. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pages 3768–3779. Association for Computational Linguistics.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. 2016. MAWPS: A math word problem repository. In *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016*, pages 1152–1157. The Association for Computational Linguistics.

Ann-Christina Lange, Marc Lenglet, and Robert Seyfert. 2016. Cultures of high-frequency trading: Mapping the landscape of algorithmic developments in contemporary financial markets. *Economy and Society*, 45(2):149–165.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692.

Zhuang Liu, Degen Huang, Kaiyu Huang, Zhuang Li, and Jun Zhao. 2020. Finbert: A pre-trained financial language representation model for financial text mining. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 4513–4519. ijcai.org.

Donald MacKenzie. 2008. *An engine, not a camera: How financial models shape markets*. Mit Press.

Donald MacKenzie, Daniel Beunza, Yuval Millo, and Juan Pablo Pardo-Guerra. 2012. Drilling through the allegheny mountains: Liquidity, materiality and high-frequency trading. *Journal of cultural economy*, 5(3):279–296.

Armineh Nourbakhsh and Grace Bang. 2019. A framework for anomaly detection using language modeling, and its applications to finance. *CoRR*, abs/1908.09156.

Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*, pages 1470–1480. The Association for Computer Linguistics.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for squad. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 2: Short Papers*, pages 784–789. Association for Computational Linguistics.

Alon Talmor and Jonathan Berant. 2018. The web as a knowledge-base for answering complex questions. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*, pages 641–651. Association for Computational Linguistics.

Chuan-Ju Wang, Ming-Feng Tsai, Tse Liu, and Chin-Ting Chang. 2013. Financial sentiment analysis for risk prediction. In *Sixth International Joint Conference on Natural Language Processing, IJCNLP 2013, Nagoya, Japan, October 14-18, 2013*, pages 802–808. Asian Federation of Natural Language Processing / ACL.

Weikang Wang, Jiajun Zhang, Qian Li, Chengqing Zong, and Zhifei Li. 2019. Are you for real? detecting identity fraud via dialogue interactions. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the*

9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019, pages 1762–1771. Association for Computational Linguistics.

Yi Yang, Mark Christopher Siy Uy, and Allen Huang. 2020. Finbert: A pretrained language model for financial communications. *CoRR*, abs/2006.08097.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 2369–2380. Association for Computational Linguistics.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir R. Radev. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 3911–3921. Association for Computational Linguistics.

Xinyi Zheng, Doug Burdick, Lucian Popa, Peter Zhong, and Nancy Xin Ru Wang. 2021. Global table extractor (gte): A framework for joint table identification and cell structure recognition using visual context. *Winter Conference for Applications in Computer Vision (WACV)*.

Appendix A: Operation Definitions

We describe all the operations in Table 4.

Appendix B: Experiment Details

All the validation results of the baselines are shown in Table 5. The trainings of all models are conducted on TITAN RTX GPUs. All the implementation and pre-trained models are based on the huggingface transformers library. We use the Adam optimizer (Kingma and Ba, 2015). The parameter settings are the following:

Retriever The learning rate is set as 3e-5, with batch size of 16.

TF-IDF + Single Op We use the TF-IDF from the Scikit-learn library.

FinQANet The learning rate is set as 1e-5. For Bert-base, Roberta-base, and finBert we use batch size of 32; For Bert-large and RoBerta-large we use batch size of 16 due to GPU memory constraints.

Retriever + Seq2seq A bidirectional LSTM is used for encoding the input, then an LSTM is used for decoding with attention. Learning rate is set as 1e-3, hidden size as 100.

Retriever + NeRd The parameter settings are the same as FinQANet.

Pre-Trained Longformer We truncate the maximum input length as 2,000. The learning rate is set as 2e-5, with batch size of 16 due to GPU memory constraints.

For more modeling details refer to our released code.

Appendix C: Case Studies

Here we provide more case studies with the full input reports. For all the examples the gold evidence is highlighted in blue.

Appendix D: Annotation Interface

We use Turkle[7] to build our annotation platform, which is a Django-based web application that can run in a local server. Figure 8 and Figure 9 show our annotation interface. After the annotators finish one example, they will use the validation check button to automatically check the validity of their inputs.

[7] https://github.com/hltcoe/turkle

Name	Arguments	Output	Description
add	number1, number2	number	add two numbers: $number1 + number2$
subtract	number1, number2	number	subtract two numbers: $number1 - number2$
multiply	number1, number2	number	multiply two numbers: $number1 \cdot number2$
divide	number1, number2	number	multiply two numbers: $number1/number2$
exp	number1, number2	number	exponential: $number1^{number2}$
greater	number1, number2	bool	comparison: $number1 > number2$
table-sum	table header	number	the summation of one table row
table-average	table header	number	the average of one table row
table-max	table header	number	the maximum number of one table row
table-min	table header	number	the minimum number of one table row

Table 4: Definitions of all operations

Baselines	**Execution Accuracy (%)**	**Program Accuracy (%)**
TF-IDF + Single Op	1.65	1.65
Retriever + Direct Generation	0.87	-
Pre-Trained Longformer (base)	23.83	22.56
Retriever + Seq2seq	24.47	22.87
Retriever + NeRd (BERT-base)	53.49	51.33
FinQANet (FinBert)	53.49	50.82
FinQANet (BERT-base)	54.12	51.71
FinQANet (BERT-large)	58.17	55.39
FinQANet (RoBerta-base)	63.12	60.33
FinQANet (RoBerta-large)	67.43	64.64

Table 5: Results on validation set

Input Report AWK/2014/page_121.pdf

… (abbreviate 20 sentences)... the ppaca effectively changes the tax treatment of federal subsidies paid to sponsors of retiree health benefit plans that provide a benefit that is at least actuarially equivalent to the benefits under medicare part d . the acts effectively make the subsidy payments taxable in tax years beginning after december 31 , 2012 and as a result , the company followed its original accounting for the underfunded status of the other postretirement benefits for the medicare part d adjustment and recorded a reduction in deferred tax assets and an increase in its regulatory assets amounting to $ 6348 and $ 6241 at december 31 , 2014 and 2013 , respectively . the following table summarizes the changes in the company 2019s gross liability , excluding interest and penalties , for unrecognized tax benefits: .

balance at january 1 2013	$ 180993
increases in current period tax position	27229
decreases in prior period measurement of tax positions	-30275 (30275)
balance at december 31 2013	$ 177947
increases in current period tax positions	53818
decreases in prior period measurement of tax positions	-36528 (36528)
balance at december 31 2014	$ 195237

the total balance in the table above does not include interest and penalties of $ 157 and $ 242 as of december 31 , 2014 and 2013 , respectively , which is recorded as a component of income tax expense .

Question: **what was the net change in tax positions in 2014?**
Gold program: **add(53818, -36528), add(#0, 157)**

Retrieved evidence:
[1] at december 31 , 2014 and 2013 , the company had state nols of $ 542705 and $ 628049 , respectively , a portion of which are offset by a valuation allowance because the company does not believe these nols are more likely than not to be realized.
[2] table row: increases in current period tax positions; 53818
[3] decreases in prior period measurement of tax positions; -36528 (36528)
Predicted program:
add(53818, -36528)

Figure 5: Error case study 1: The net change in the tax position is the sum of the increase and the decrease plus the penalties and interest. The model lacks this finance knowledge, thus the retriever fails to retrieve the sentence describing the penalties and interest. Another challenging point is the table understanding, since in this case, it's hard to distinguish the retrieved two table rows for the year 2013 or 2014, using our method that regards each table row as basic unit. The model needs to look at the full table to get this global information.

Input Report MKTX/2004/page_99.pdf

… (abbreviate 5 sentences)... cumulative accrued but unpaid dividends were forfeited upon conversion of shares of series b convertible preferred stock into common stock . as such , the company did not accrue dividends , as liquidation of the shares of series b convertible preferred stock was not anticipated . as of december 31 , 2004 , the company had 110000000 authorized shares of common stock and 10000000 authorized shares of non-voting common stock . as of december 31 , 2003 , the company had 120000000 authorized shares of common stock and 450060 authorized shares of non-voting common stock . common stock entitles the holder to one vote per share of common stock held . … (abbreviate 11 sentences)...

year ended december 31	as of december 31 , 2004	as of december 31 , 2003
2005	$ 2014	$ 177973

Question: **as of december 31 , 2004 , how many shares of common stock were outstanding?**
Gold program: **add(110000000, 10000000)**

Retrieved evidence:
[1] as of december 31 , 2004 , the company had 110000000 authorized shares of common stock and 10000000 authorized shares of non-voting common stock .
[2] as of december 31 , 2003 , the company had 120000000 authorized shares of common stock and 450060 authorized shares of non-voting common stock .
[3] in 2004 and 2003 , the company had 1939734 shares and 1937141 shares , respectively , of common stock that were issued to employees .
Predicted program:
subtract(110000000, 11000000)

Figure 6: Error case study 2: The model does not have the financial knowledge of how to calculate the 'common stock outstanding'.

Input Report K/2013/page_23.pdf-1

… (abbreviate 12 sentences)... underlying gross margin declined by 180 basis points in 2012 as a result of cost inflation , net of cost savings , and the lower margin structure of the pringles business . underlying sga% (sga %) was consistent with 2011 . our underlying gross profit , underlying sga , and underlying operating profit measures are reconciled to the most comparable gaap measure as follows:

(dollars in millions)	2013	2012	2011
reported gross profit (a)	$ 6103	$ 5434	$ 5152
… abbreviate 10 rows ...			
underlying operating profit (d)	$ 2098	$ 2014	$ 2109

Question: if 2014 underlying operating profit increases at the same pace as 2013 , what would it be , in millions?
Gold program: **divide(2098, 2014), multiply(2098, #0)**

Retrieved evidence:
[1] underlying gross margin declined by 110 basis points in 2013 due to the impact of inflation , net of productivity savings , lower operating leverage due to lower sales volume , and the impact of the lower margin structure of the pringles business
[2] table row: (dollars in millions) The underlying operating profit (d) of 2013 is $ 2098 ; The underlying operating profit (d) of 2012 is $ 2014 ; The underlying operating profit (d) of 2011 is $ 2109 ;
[3] during 2013 , we recorded $ 42 million of charges associated with cost reduction initiatives .
Predicted program:
divide(2098, 2098), multiply(2098, #0)

Figure 7: Error case study 3: Complex numerical reasoning.

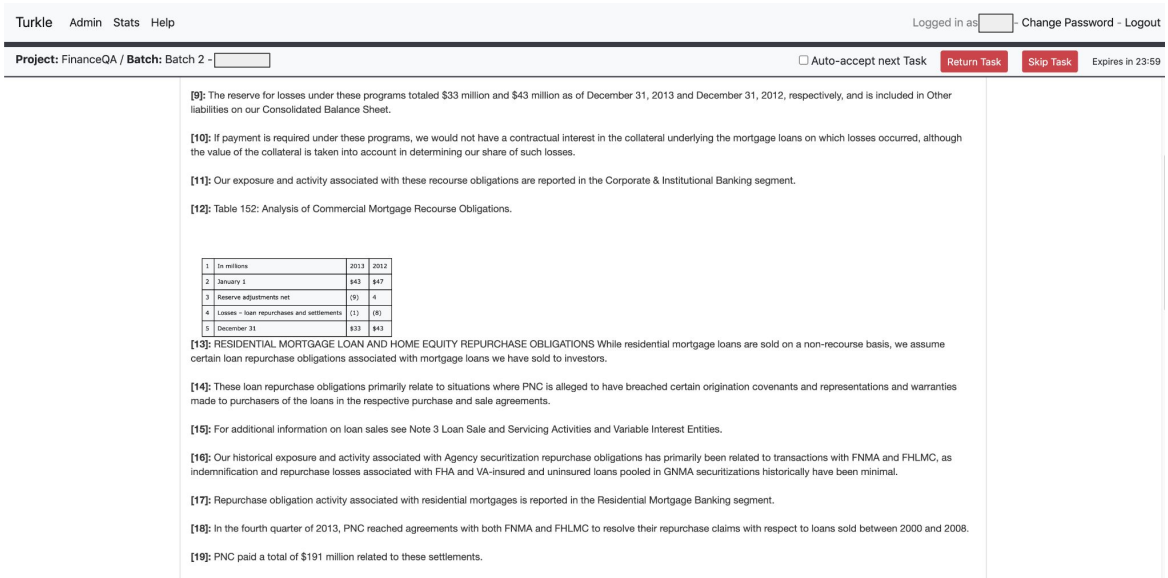

Figure 8: Annotation interface: Display report.

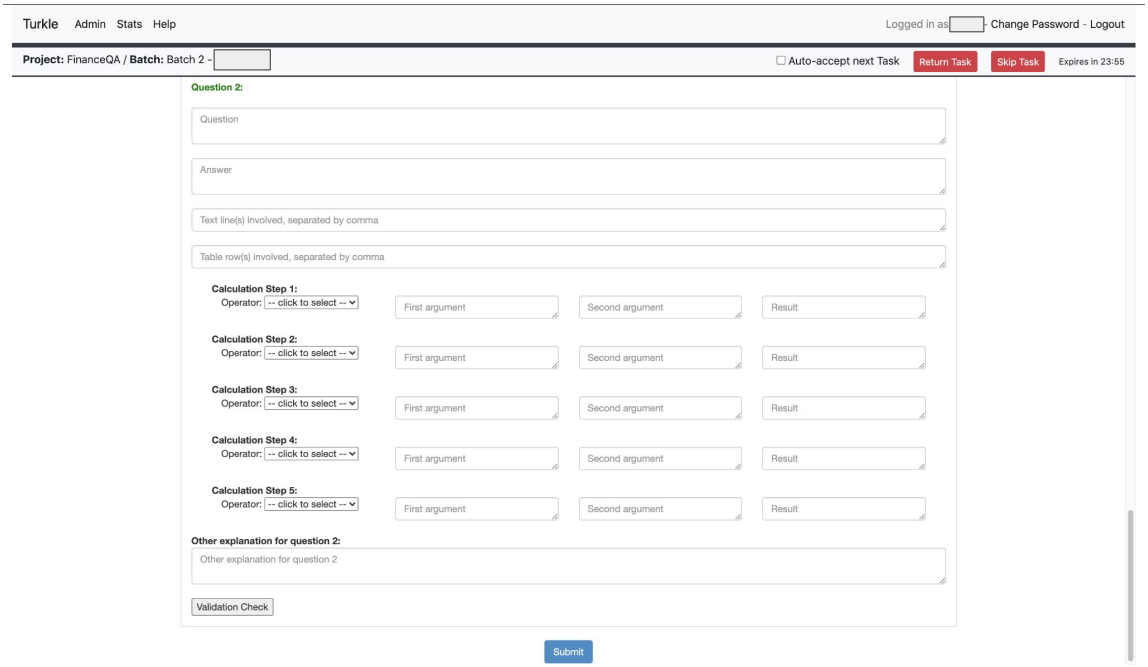

Figure 9: Annotation interface: Annotator input fields.