

# APOLLO: An Optimized Training Approach for Long-form Numerical Reasoning

Jiashuo Sun<sup>1</sup>, Hang Zhang<sup>2</sup>, Chen Lin<sup>1\*</sup>, Yeyun Gong<sup>3</sup>, Jian Guo<sup>2</sup>, Nan Duan<sup>3</sup>

<sup>1</sup> School of Informatics, Xiamen University, China

<sup>2</sup> IDEA Research, China

<sup>3</sup> Microsoft Research Asia

## Abstract

Long-form numerical reasoning aims to generate a reasoning program to calculate the answer for a given question. Previous work followed a retriever-generator framework, where the retriever selects key facts from a long-form document, and the generator generates a reasoning program based on the retrieved facts. However, they treated all facts equally without considering the different contributions of facts with and without numerical information. Furthermore, they ignored program consistency, leading to the wrong punishment of programs that differed from the ground truth. In order to address these issues, we proposed APOLLO (An optimized training aPproach fOr Long-form numerical reasOning), to improve long-form numerical reasoning. APOLLO includes a number-aware negative sampling strategy for the retriever to discriminate key numerical facts, and a consistency-based reinforcement learning with target program augmentation for the generator to ultimately increase the execution accuracy. Experimental results on the FinQA \* and ConvFinQA † leaderboards verify the effectiveness of our proposed methods, achieving the new state-of-the-art. ‡

## 1 Introduction

Long-form numerical reasoning aims to generate an executable program that answers a specific question. Unlike conventional numerical reasoning tasks (MacKenzie, 2008), long-form numerical operates on a long document (e.g., max 2,679 tokens in FinQA (Chen et al., 2021)). Recently, various benchmarks, such as FinQA (Chen et al., 2021) and ConvFinQA (Chen et al., 2022), have been pro-

\*Corresponding author, chenlin@xmu.edu.cn

\*<https://codalab.lisn.upsaclay.fr/competitions/4138#results>

†<https://codalab.lisn.upsaclay.fr/competitions/8582#results>

‡Our code will be publicly available in the future.

## Long-form Document

Operating cash flow from continuing operations for 2017 was \$2.7 billion, a \$191 million, or 8 percent increase compared with 2016, reflecting higher earnings and favorable changes in working capital. **The company has not incurred any borrowing under this or previous facilities.** A total debt of 43.1 million were purchased in 2017 under the authorization. ( .. abbreviate 17 sentences.. )

In Millions	Total Assets	Long-term Debt	Common Equity	abbreviate 3 columns	Interest Coverage Ratio
2015	\$22,088	\$4,289	\$8,081	...	20.2x
2016	\$21,732	\$4,051	\$7,568	...	11.8x
2017	\$19,589	\$3,794	\$8,718	...	12.6x

Total debt, which includes long-term debt, current maturities of long-term debt, was \$ 4.7 billion and \$6.6 billion for 2017 and 2016, respectively. The credit facility contains no financial covenants and is not subject to termination based on change of credit rating or material adverse changes.(.. abbreviate 15 sentences.. )

**Question:** What percentage of total debt was long-term debt in 2017?

**Gold fact:** The “Long-term Debt” column and the “Total debt, which includes long-term debt...” textual fragment.

**Gold Program:** Multiply(4.7, const\_1000), Divide(3794, #0)

**Consistency Program:** Divide(3794, 4.7), Divide(#0, const\_1000)

**Answer:** 0.81

Figure 1: An example of Long-form Numerical Reasoning. The parameters in gold program are directly from the numerical fact (e.g., table column 2 and the textual fragment in green) instead of the non-numerical fact (e.g., the textual fragment in red). The answer can be equally generated from the gold program and the consistent program. Const\_x, #i denotes constant x and the result of the previous  $i - 1^{th}$  operator.

posed to assess the ability of systems to perform long-form numerical reasoning (Figure 1).

A typical framework to solve this task is the retriever-generator question-answering framework, which is firstly introduced by Chen et al. (2021). This framework consists of two stages: training a retriever to identify relevant facts (i.e., a textual fragment or a table column) from the documents and training a generator to generate the executable programs with the retrieved facts. Recently, the use of pre-trained masked language models and ensemble techniques (Wang et al., 2022b; Zhang et al., 2022; Wang et al., 2022a) has led to high accuracy in this task. Zhang and Moshfeghi (2022) and An-

drejczuk et al. (2022) have designed a novel structure for better handling long-form data. Li et al. (2022) has proposed a dynamic retriever-reranker-generator framework to enhance each generation step by a dynamic reranking of retrieved facts.

However, existing studies ignore two critical aspects of long-form numerical reasoning. Firstly, they neglect the importance of numerical facts and treat all facts equally. Numerical facts are more important since they are the direct source of parameters in the generated program (as demonstrated in Figure 1). Secondly, they disregard the issue of program consistency, where programs can have different expressions but produce the same results. As shown in Figure 1, if we use only the gold program as the ground truth in supervised training, the model will wrongly penalize consistent programs.

In this paper, we improve the performance of the retriever-generator framework for long-form numerical reasoning. We propose a number-aware negative sampling approach for retriever training to prioritize and differentiate between numerical facts (Section 3.2). At the generator level, we introduce target program augmentation and consistency-based reinforcement learning to explore the space of consistent programs and improve execution accuracy (Section 3.3).

The contributions of this work are as follows:

- We introduce a novel number-aware negative sampling, demonstrating the effectiveness during retriever training for long-form numerical reasoning tasks.
- We propose consistency-based reinforcement learning and target program augmentation in our generator training to increase execution accuracy.
- Our approaches achieve the current state-of-the-art of **72.47** execution accuracy and **68.01** program accuracy on FinQA, and **78.76** execution accuracy and **77.19** program accuracy on ConvFinQA, respectively.

## 2 Related Work

### 2.1 Retrieval Augmented Generation

A series of previous works explore a retrieval-augmented paradigm for text generation (Amini et al., 2019; Wei et al., 2022; Koncel-Kedziorski et al., 2016; Wang et al., 2022b). Popular datasets include NQ (Kwiatkowski et al., 2019), TriviaQA

(Joshi et al., 2017), FEVER (Thorne et al., 2018), MS-MARCO (Nguyen et al., 2016) and the benchmark KILT (Petroni et al., 2020). The line of this paradigm is, at first, the retriever retrieved amount of passages based on the query. Secondly, the generator takes the retrieved passages as input and generates an answer. It is noteworthy that the retrieval-augmented generation approach focuses on retrieving passages from a vast repository of knowledge (typically Wikipedia) as external information due to the limited input provided (mostly only one query). For long-form numerical reasoning tasks, the model needs to focus on retrieving the most critical facts directly from input since it comprises a significant number of sentences.

### 2.2 Numerical Reasoning

**MWP Numerical Reasoning** MWP (Math Word Problem) numerical reasoning is a challenging task that has been introduced for years. The task is to calculate the answer to a short textual question by generating an arithmetic expression. There are several benchmark datasets including MathQA (Amini et al., 2019) and MaWPS (Koncel-Kedziorski et al., 2016) which all focus on generating target programs for math or calculation problems. The questions in MWP are described in a controlled manner, exhibiting strong regularity, therefore, some previous works use template-based (Wang et al., 2019) or tree-based (Jie et al., 2022) methods. However, the MWP numerical reasoning tasks are mostly general and simple which are far different from the tasks we explore. This work focuses on complex numerical reasoning in the long-form document.

**Long-form Numerical Reasoning** Long-form numerical reasoning is more challenging than MWP. The task is to generate the program for a specific question based on retrieved facts from a long document. Chen et al. (2021) and Chen et al. (2022) have introduced FinQA and ConvFinQA, which are complex question-answering and conversational numerical reasoning tasks for financial reports, respectively. Since long-form numerical reasoning is based on retrieval and generation, therefore some works improve them separately. Wang et al. (2022b) has used DeBERTa (He et al., 2021) to pre-training on financial data. Zhang et al. (2022) has proposed an ensemble approach by developing models with different specialized capabilities and fusing their strengths. Wang et al. (2022a) has de-

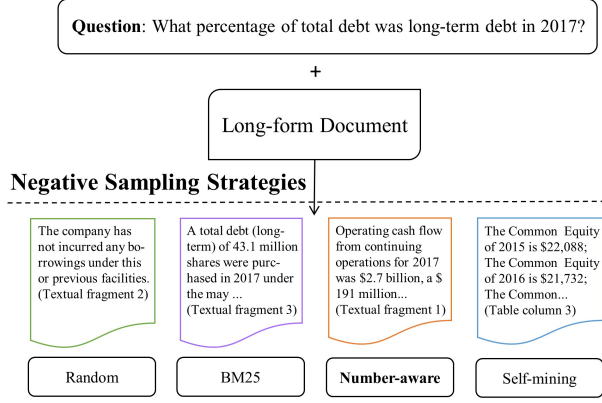


Figure 2: The novel hard negative sampling strategy in APOLLO. The facts for four methods are sampled from the same document in Figure 1. We compared our sampling method with three frequently conventional methods: Random, BM25 and Self-mining.

vised a cell retriever module to retrieve gold cells to avoid bringing unrelated cells to the generator. However, previous works haven’t considered the significance of numerical facts and the consistency of the program, making the performance of the retrievers and generators drop substantially.

### 3 Methodology

In this section, we describe our approach in detail. Similar to [Chen et al. \(2021\)](#), we utilize the retriever-generator framework and train each module independently. However, our approach differs from previous work ([Zhang et al., 2022](#); [Zhang and Moshfeghi, 2022](#)) in two ways: (1) the retriever is trained using number-aware negative sampling, and (2) the generator is trained with target program augmentation and subsequently refined using consistency-based reinforcement learning.

#### 3.1 Task Definition

Given a question  $q$  and a long-form document  $d$  consisting of textual and structured-table facts, long-form numerical reasoning aims to generate a program that can be executed to get the right answer. Since the supporting document is too long to handle, this task can be decomposed into two sub-tasks: 1) retrieving the key facts from document  $d$ ; 2) generating the program based on retrieved facts and  $q$ . The objective can be written as:

$$P(G|q; d) = P(G|q, F; d)P(F|q; d) \quad (1)$$

where  $G = [w_0, w_1, \dots, w_l]$  donate the golden program sequence consisting of  $l$  program tokens

$w, F = \{f_1, f_2, \dots, f_m\}$  donates the key fact set where each fact  $f_i$  potentially contribute to answer the question  $q$ . In a typical retriever-generator framework ([Chen et al., 2021](#)), the retriever learns to maximize  $P(F|q; d)$ , and the generator aims to maximize  $P(G|q; d)$ .

#### 3.2 Retriever

The retriever is based on the sequence-pair classification model [Nogueira and Cho \(2019\)](#). The question  $q$  and each fact  $f_i$  are concatenated to a BERT ([Devlin et al., 2018](#)) transformer. A pooling layer followed by a linear layer is adopted to obtain the score  $s^m$  for each fact:

$$s^m = W_l^T cls(Encoder(concat(q, f_i))) \quad (2)$$

where  $f_i \in F$  and  $cls()$  extracts BERT’s hidden vector at token  $[CLS]$ ,  $W_l$  is a projection vector.

**Number-aware Negative Sampling** The score  $s^m$  is used to rank each fact. To compute the  $s^m$  accurately, the retriever is trained on both positive and negative facts. Previous works ([Chen et al., 2021, 2022](#); [Wang et al., 2022a](#)) randomly sample negative facts in the given long-form document. However, since numerical reasoning is directly related to numbers, intuitively, facts without numbers are less contributing. As shown in Figure 1, the two green-highlighted facts are both numerical facts, which are the primary source of parameters in the gold program. In contrast, the red-highlighted non-numerical fact contributes less to the gold program and answer. We expect the retriever to focus more on numerical facts.

In order to effectively train the retriever, we utilize a number-aware negative sampling strategy which involves extracting numerical facts from negative examples within long-form documents and randomly selecting a subset of these numerical facts.

Figure 2 illustrates the differences between number-aware negative sampling and conventional negative sampling strategies. These conventional strategies include (1) Random: selecting a random fact from the corpus, (2) BM25: selecting the most similar fact returned by BM25 ([Robertson et al., 2009](#)) algorithm, and (3) Self-mining: selecting the highest ranking fact retrieved by a well-trained retriever. It can be observed that random sampling does not guarantee sufficient high-quality negative facts. Although BM25 constructs negative facts that have a high overlap with the characters in the

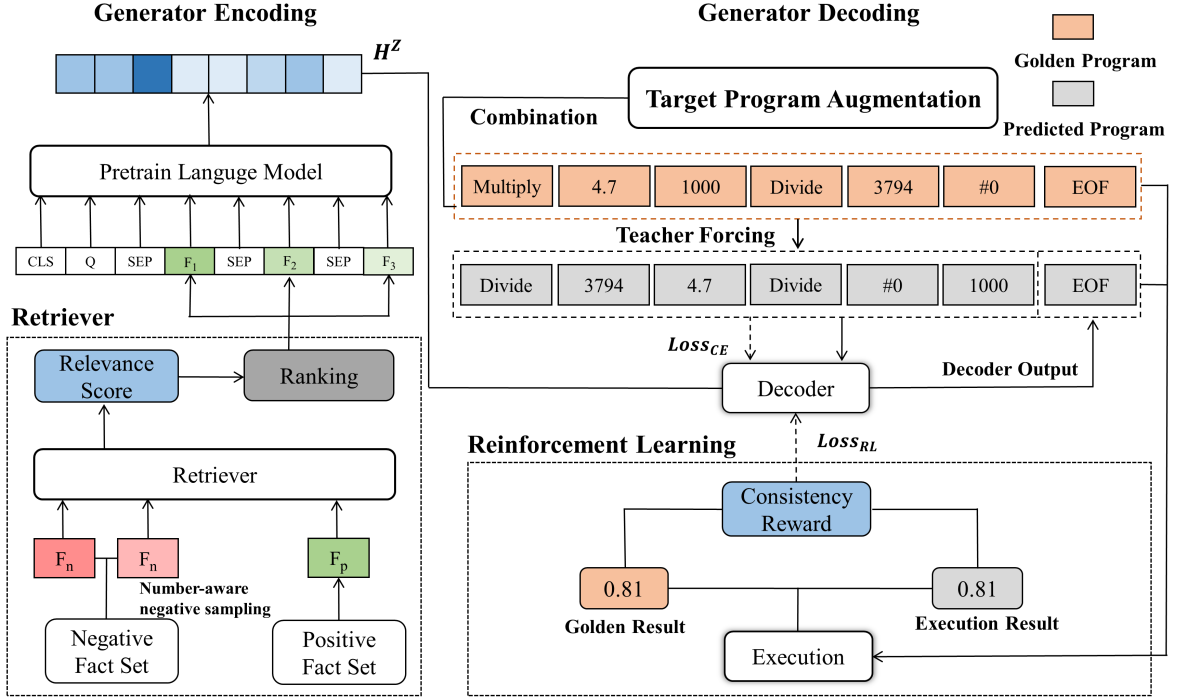


Figure 3: The overall architecture of retriever-generator framework with APOLLO.  $F_n$  and  $F_p$  denotes negative facts and positive facts for training, respectively, and  $F_1, F_2, F_3$  represent the retrieved facts. We use golden program in Figure 1 as an example. The left portion of the figure illustrates the retriever and encoding process for the generator, while the right portion illustrates the complete process of generating the "EOF" token, implementing target program augmentation, and consistency-based reinforcement learning. The generator utilizes cross-entropy to supervise the generation of predicted programs, using both the golden program and programs generated through target program augmentation as reference. Then, APOLLO samples consistent program and executes with golden program to obtain the execution and golden results, which are then used in Equation 6 to calculate the consistent reward. This consistent reward is then employed to update all parameters.

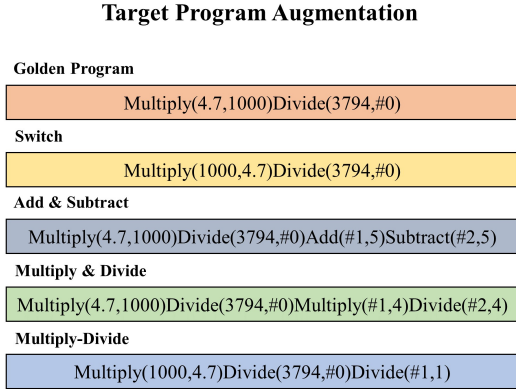


Figure 4: The specific form of four target program augmentation construction. All the results of the programs generated by target program augmentation are the same as the golden program.

question, they tend to contain few numbers and are less supportive of generating programs. Furthermore, the self-mining strategy is prone to sampling duplicate negative facts (e.g., from the same table column), making the negative facts less informative

in discriminating key facts.

### 3.3 Generator

The typical generator consists of a pre-trained encoder and a decoder. First, the question and Top-k retrieved facts are concatenated together to a BERT encoder to produce token-level contextualized embeddings:

$$H^c = \text{Encoder}(\text{concat}(q, Z)) \quad (3)$$

where  $Z = \{f_1, f_2, \dots, f_k\}$  is the Top-K retrieved facts set. Then, the decoder takes the embedding  $H^c$  as input and decodes the numerical reasoning program step by step:

$$P(w_t|q) = \text{Decoder}(w_{t-1}, H^c) \quad (4)$$

where  $w_t$  is generated token in step  $t$ .

Cross-entropy loss is adopted to supervise the generator. In cross-entropy, the expected value at each position is the corresponding token from the ground-truth program. However, due to the



program consistency issue, the supervision is sometimes misleading. For example, in Figure 1, the ground truth is `Multiply(4.7, const_1000)`, `Divide(3794, #0)`. Then, although `Divide(3794, 4.7)`, `Divide(#0, const_1000)` also obtains the correct execution result, since it does not have an exact string match, it will be wrongly penalized using cross entropy loss.

To accurately compute  $P(G|q; d)$  for all consistent programs, we first expand the ground truth set by target program augmentation. However, there are an infinite amount of programs for calculating a correct answer, this paper focuses on regular construction forms for simplicity. Additionally, we also utilize consistency-based reinforcement learning to directly optimize for the expected correctness of the execution result.

**Target Program Augmentation** In generating the program, we have ten program computation operators, including Add, Subtract, Multiply, Divide, and so on. (Other operators and their specific parameters and outputs can be found in Appendix A). We adopt four target program augmentation methods: Switch, Add & Subtract, Multiply & Divide, and Multiply-Divide. These four methods details are the following:

**Switch** For the Add and Multiply operators, the two arguments are interchangeable. Heuristically, for programs with these two operators in the ground truth, we can swap these two parameters to increase the amount of data. Moreover, a program that has  $n$  operators with addition or multiplication operators can create  $2^n - 1$  more new samples for training.

**Add & Subtract** For the same program, the result does not change after adding and subtracting a random constant at the end.

**Multiply & Divide** Similar to the previous construction, the result does not change after multiplying and dividing a random constant at the end.

**Multiply-Divide** The same program multiplies or divides a constant one does not change its result.

We use golden program in Figure 1 as an example to illustrate the specific format of the four target program augmentation demonstrated in Figure 4.

### Consistency-based Reinforcement Learning

Instead of using teacher forcing at each step of program generation, the next token is obtained by sampling from the output distribution. At the end of the generation procedure, the generated program is executed and compared against the correct

answer to determine a reward. Let  $G_g$  denotes the program generated by the generator and  $G_T$  denotes the ground truth program corresponding to the question. We define the reward  $R(G_g, G_T)$  as

$$R(G_g, G_T) = \begin{cases} -2, & \text{U.E.P} \\ -1, & \text{E.P but wrong answer} \\ +1, & \text{E.P and right answer} \end{cases} \quad (5)$$

where *U.E.P* stands for unexecutable program and *E.P* stands for executable program. The reinforcement learning loss is the negative expected reward over possible generated program  $L^{RL} = -\mathbb{E}_w[R(G_g, G_T)]$ . Inspired by Zhong et al. (2017), we derive the policy gradient for  $L^{RL}$ :

$$\begin{aligned} \nabla L_{\Theta}^{RL} &= -\nabla_{\Theta} (\mathbb{E}_{w \sim p_w} [R(G_g, G_T)]) \\ &\approx -R(G_g, G_T) \nabla_{\Theta} \sum_t (\log p_w(w_t; \Theta)) \end{aligned} \quad (6)$$

where  $p_w$  denotes the probability of choosing token  $w_t$  during decoding time step  $t$ . Moreover, we approximate the expected gradient using a single Monte-Carlo sample  $w$ .

The training process with consistency-based reinforcement learning is shown in Figure 3. While the golden program and the generated program differ at the character-level, they are identical at the consistency-level, leading to a positive reward. However, if only used cross-entropy training, the predicted program generated by decoding may be incorrectly penalized.

## 4 Experiment

In this section, we mainly introduce the datasets we evaluate in our long-form numerical reasoning task and the experimental effect of our work to show the advantages of our methods from an experimental comparison view.

### 4.1 Datasets

We conduct evaluation experiments on two datasets: FinQA (Chen et al., 2021) and ConvFinQA (Chen et al., 2022).

**FinQA** FinQA is a dataset of numerical reasoning over long-form financial data, containing 8, 281 financial reports, along with their QA pairs and annotated numerical reasoning processes by eleven finance professionals based on the earnings reports of S&p 500 companies (Zheng et al., 2021). The data is released as training (6, 251), dev (883), and

Model	FinQA(dev)		FinQA(test)		ConvFinQA(dev)		ConvFinQA(test)	
	Exe Acc	Prog Acc	Exe Acc	Prog Acc	Exe Acc	Prog Acc	Exe Acc	Prog Acc
Retriever+NeRd (Ran et al., 2019)	47.53	45.37	48.57	46.76	-	-	-	-
Longformer (Beltagy et al., 2020)	23.83	22.56	21.90	20.48	-	-	-	-
GPT-2 (Radford et al., 2019)	-	-	-	-	59.12	57.52	58.19	57.00
T-5 (Raffel et al., 2020)	-	-	-	-	58.38	56.71	58.66	57.05
FinQANet (Chen et al., 2021)	61.22	58.05	61.24	58.86	68.32	67.87	68.90	68.24
ELASTIC <sup>§</sup> (Zhang and Moshfeghi, 2022)	65.00	61.00	62.16	57.54	-	-	-	-
DyRRen (Li et al., 2022)	66.82	63.87	63.30	61.29	-	-	-	-
TabT5* (Andrejczuk et al., 2022)	-	-	70.79	68.00	-	-	-	-
CellRetriever+UniLM* (Wang et al., 2022a)	-	-	68.00	65.21	-	-	-	-
APOLLO	69.70	65.91	67.99	65.60	76.47	74.14	76.00	74.56
- Ensemble model	<b>72.91</b>	<b>70.83</b>	<b>71.07</b>	<b>68.94</b>	<b>78.46</b>	<b>75.91</b>	<b>78.76</b>	<b>77.19</b>
General Crowd Performance	-	-	50.68	48.17	-	-	46.90	45.52
Human Expert Performance	-	-	91.16	87.49	-	-	89.44	86.34

Table 1: Performance comparisons on the dev set, test set of FinQA and dev set, private test set of ConvFinQA. The pre-trained models utilized in rows 5 to 11 of the table are all RoBERTa-large, except for TabT5 and CellRetriever+UniLM, which use T5 and UniLM (Dong et al., 2019) respectively. The missing data in the table is due to the fact that many works do not report their results. \* denotes the results of ensemble models, since many works only report their ensemble model results.

test (1, 147) following a 75%/10%/15% split. The long-form financial documents contained heterogeneous data (structured and unstructured data such as tables and texts) and compounded many financial terms in questions, e.g., "Shares vested", and "Pre-tax earnings" which is challenging in this long-form numerical reasoning task.

**ConvFinQA** ConvFinQA (Conversational Finance Question Answering) is a dataset of conversational long-form numerical reasoning over financial data, containing 3,892 conversations consisting of 14,115 questions and annotated by expert annotators to compose the question based on the simulated conversing flow. The data has split into 3,037/421/434 for train/dev/test sets. However, the reasoning chains throughout the conversation pose great challenges for the models to learn when to refer to or discard the conversation history and how to assemble the reasoning path.

## 4.2 Baselines

We compare our model with several competitive models. (1) **FinQANet** (Chen et al., 2021), which utilizes a retriever-generator framework to generate programs based on retrieved facts. (2) **NeRd** (Ran et al., 2019), which employs a BERT-based pointer-generator model to generate symbolic nested programs. (3) **Longformer** (Beltagy et al., 2020),

<sup>§</sup>refers to the ELASTIC (Zhang and Moshfeghi, 2022), in which we find a serious data leak and we do their experiment over based on their released code <https://github.com/NeuraSearch/NeurIPS-2022-Submission-3358>. The data leak result of their model and APOLLO are shown in Appendix B.

which inputs the entire contents of long-form documents and generates programs. (4) **GPT-2** (Radford et al., 2019), which uses the GPT-2 model with prompts to generate programs. (5) **T5** (Raffel et al., 2020), which is similar to GPT-2 and utilizes the T5 model with prompts to generate programs. (6) **CellRetriever+UniLM** (Wang et al., 2022a), which employs both cell and row retrievers to retrieve facts and integrates multiple generators to generate programs. (7) **ELASTIC** (Zhang and Moshfeghi, 2022), which utilizes an adaptive symbolic compiler to generate programs. (8) **Ant Risk AI** (Zhang et al., 2022), which develops models with different specialized capabilities and fuses their strengths to retrieve and generate programs. (9) **TabT5** (Zhang et al., 2022), which uses the T5 model pre-trained on Wikipedia tables to generate programs. (10) **DyRRen** (Li et al., 2022), which enhances each generation step through a dynamic reranking of retrieved facts using a retriever-reranker-generator framework. (11) **Human performance**, which includes both experts and non-experts in the FinQA and ConvFinQA datasets. The results are taken from the original paper (Chen et al., 2021, 2022).

## 4.3 Evaluation Metrics

**Retriever** In retriever, we use Recall Top-3 and Recall Top-5 to evaluate our model. This metric evaluates the retrieval result by determining the percentage of correct positive predictions out of all positive predictions. However, since there may be more than one positive prediction in each sample, we assume that the first  $N$  predictions in the recall

Model	Pre-training	FinQA(dev)		FinQA(test)		ConvFinQA(dev)	
		R@3	R@5	R@3	R@5	R@3	R@5
FinQANet (Chen et al., 2021)	RoBERTa	91.30	93.89	89.82	93.22	88.95	92.74
FinQANet (Chen et al., 2021)	DeBERTa-v3	92.03	95.06	90.37	93.78	89.49	92.91
Ant Risk AI (Zhang et al., 2022)	RoBERTa	91.54	95.11	90.16	94.12	-	-
- Ensemble model	RoBERTa	92.63	95.89	90.77	94.33	-	-
APOLLO	RoBERTa	93.58	95.62	91.76	93.95	91.67	94.56
APOLLO	DeBERTa-v3	94.22	96.08	92.37	94.49	92.18	95.01
- Ensemble model	RoBERTa	<b>95.03</b>	<b>96.54</b>	<b>93.31</b>	<b>94.98</b>	<b>92.40</b>	<b>95.15</b>

Table 2: The experimental results of retriever Recall Top-3 and Top-5 on the dev set and test set in FinQA and only dev set on ConvFinQA. ConvFinQA only has private test set available currently which does not have ground truth for retrieved facts. All pre-training models are large-size models.

Type	#N	FinQA		ConvFinQA	
		R@3	R@5	R@3	R@5
Random	3	90.37	94.07	89.49	92.91
BM25	3	88.20	92.32	88.21	91.40
Number-aware	3	92.02	94.19	<b>92.18</b>	<b>95.01</b>
Random	2	89.56	93.63	89.27	92.63
Random	4	89.87	93.74	89.15	92.80
Random	5	89.75	93.67	88.91	92.78
BM25	2	88.86	92.81	86.02	88.12
BM25	4	88.35	91.29	87.57	89.11
BM25	5	88.83	92.36	86.37	89.95
Self-mining & R	3	89.47	94.01	88.62	90.82
Self-mining & B	3	84.28	87.24	86.91	88.08
Self-mining & N	3	91.25	93.73	88.31	90.27
Number-aware	2	91.98	94.04	91.87	94.68
Number-aware	4	<b>92.37</b>	<b>94.49</b>	91.92	94.70
Number-aware	5	92.01	94.23	91.91	94.26

Table 3: The results of different negative sampling strategies on test set in FinQA and dev set in ConvFinQA. Self-mining & R, B, and N denote using Self-mining with Random, BM25, and Number-aware negative sampling, respectively. #N: Ratio of positive and negative samples for training.

top-N are all positive predictions.

**Generator** In generator, we use Execution Accuracy and Program Accuracy to evaluate our model. Execution Accuracy evaluates the model by calculating the accuracy between the predicted program result and the golden executable result. Program Accuracy calculates the accuracy of the operators and operands between the predicted program and the golden program.

#### 4.4 Implementation Details

Our model is implemented using Pytorch (Paszke et al., 2019) and Transformer (Wolf et al., 2020), and then trained on a server with two NVIDIA Tesla A100 GPUs of 40G memory. For retriever,

Model	FinQA		ConvFinQA	
	Exe Acc	Prog Acc	Exe Acc	Prog Acc
Generator	66.95	64.62	75.64	73.13
w. Switch	67.07	64.62	75.78	73.13
w. Add & Sub	67.25	64.95	75.82	73.54
w. Mul & Divide	67.46	65.07	75.95	73.70
w. Mul-Divide	66.95	64.62	75.64	73.13
w. RL	67.36	65.14	76.14	73.61
w. RL & TPA	<b>67.99</b>	<b>65.60</b>	<b>76.47</b>	<b>74.14</b>

Table 4: The performances of APOLLO with consistency-based reinforcement learning and different target program augmentation methods. RL & TPA denotes combine two approaches, which performs best on both datasets.

we use RoBERTa-large and DeBERTa-v3-large as the classifier. We take the top-3 ranked facts as the retriever results. Training epochs are set to 50 and batch size for all datasets is 8. The initial learning rate is set to 9e-6, and we schedule the learning rate to warm up at the beginning and gradually decrease the learning rate during training. For generator, we adopt RoBERTa as the encoder, the initial learning rate is set to 1e-5 and then adopt learning rate scheduler. For consistency-based reinforcement learning, we adopt continual learning to continue train based on our best performance model. For all models, the maximum sequence length is set to 512. Besides, we use Adam as optimizer (Kingma and Ba, 2014) to update the parameters of the models and clip the gradient every iteration to prevent gradient explosion as well as applying weight decay to prevent over-fitting.

#### 4.5 Main Results

Table 1 presents the generator performance of APOLLO and baselines on FinQA and ConvFinQA. APOLLO achieves the highest scores on both datasets, with 71.07 execution accuracy, 68.94 pro-

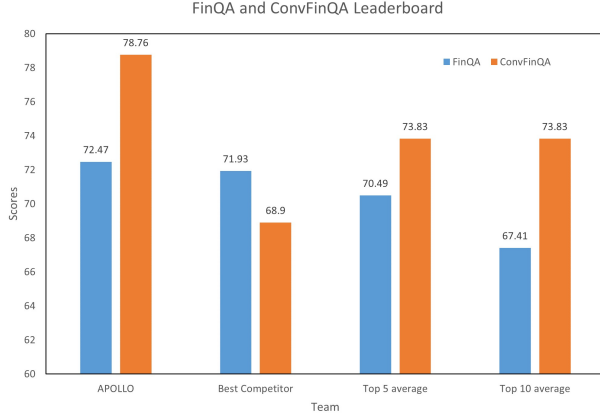


Figure 5: Performance comparisons on the private test set of the FinQA and ConvFinQA. We report APOLLO, best competitor, Top 5 average and Top 10 average scores on both leaderboard. At the time of submission (25 Nov. 2022), APOLLO has achieved state-of-the-art in both leaderboards.

gram accuracy on FinQA test set and 78.46 points execution accuracy, 75.91 program accuracy on ConvFinQA dev set.

Table 2 presents the retriever performances of APOLLO and baselines on FinQA and ConvFinQA. Overall, APOLLO achieves the highest scores on both datasets. A comparison of APOLLO to the baseline FinQANet demonstrates a notable advantage, with 2.00 points higher Recall 3 and 0.71 Recall 5 on FinQA, and 2.69 points higher Recall 3, 2.1 Recall 5 on ConvFinQA. However, some works (Wang et al., 2022b,a; Li et al., 2022) do not report their retriever performance on dev or test sets and do not release their source code, so they are not included in the retriever results table. Additionally, our single model has achieved a significant improvement over previously published ensemble models on both the dev and test sets.

In general, APOLLO achieves the best performance on both FinQA and ConvFinQA leaderboard of the private test set, shown in Figure 5.

#### 4.6 Ablation Study

APOLLO comprises three crucial components: number-aware negative sampling in retriever training; consistency-based reinforcement learning and target program augmentation in generator training. To gain a deeper understanding of our model, we conduct an extensive ablation study to investigate the impact of these three modules individually.

**Results of Number-aware Hard Negative Training** Table 3 illustrates the performance of differ-

ent methods for constructing negative facts. The results indicate that our number-aware negative sampling strategy surpasses random, BM25 and self-mining on both long-form numerical reasoning datasets. Furthermore, we experimentally investigate the impact of the ratio of positive to negative facts on training. We conduct experiments with positive to negative fact ratios of 3, 4 and 5, and find that the ratio has a substantial influence on the results. It demonstrates that the positive and negative fact ratio has a great influence on training. The best results for the random method, BM25 and self-mining are obtained with ratio of 3, where the ratio changes, the performance decreases.

**Results of Consistency-based Reinforcement Learning Training and Target Program Augmentation** Table 4 presents the results of utilizing consistency-based reinforcement learning and target program augmentation in generator training. Noteworthy, among the four target program augmentation methods, the Multiply & Divide methods demonstrated the greatest enhancement on both datasets. We suspect this is due to the operation distributions, where Multiply, and Divide have the total distributions of 51.11%. These two operations contribute largely to dataset, and there is also a high percentage of simultaneous between multiplication and division in the program, leading to more stable model convergence. However, the target program augmentation method in ConvFinQA receives poor performance compared with FinQA, we suspect that many answers in ConvFinQA are directly obtained from the original long-form document, making the target program augmentation ineffective. Additionally, the consistency-based reinforcement learning method also yielded significant performance gains on both datasets. Furthermore, the best results are obtained by combining the two approaches, with target program augmentation followed by reinforcement learning training.

Moreover, APOLLO increases the diversity of the generated programs, shown in Appendix C.

## 5 Conclusion

This work presents an optimized training approach for long-form numerical reasoning, APOLLO. Unlike previous work, APOLLO enhances the retriever with number-aware negative sampling strategy to better classify numerical facts and the generator with consistency-based reinforcement learning as well as target program augmentation to generate



more accurate and diverse programs. As a result, APOLLO achieves the state-of-the-art results on both FinQA and ConvFinQA leaderboards, which significantly outperforms all the other models.

## Limitations

APOLLO mainly has two limitations:

- The negative sampling strategy in our retriever training, which is sensitive to numerical data, is appropriate for tasks that involve a substantial amount of numerical information, particularly in financial domain. However, it is not suitable for tasks that have lower numerical content, as it lacks scalability in those cases.
- The target program augmentation construction form is relatively basic. We will explore more advanced forms of construction in future work.

## References

- Aida Amini, Saadia Gabriel, Peter Lin, Rik Koncel-Kedziorski, Yejin Choi, and Hannaneh Hajishirzi. 2019. Mathqa: Towards interpretable math word problem solving with operation-based formalisms. *arXiv preprint arXiv:1905.13319*.
- Ewa Andrejczuk, Julian Martin Eisenschlos, Francesco Piccinno, Syrine Krichene, and Yasemin Altun. 2022. Table-to-text generation and pre-training with tabt5. *arXiv preprint arXiv:2210.09162*.
- Iz Beltagy, Matthew E Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*.
- Zhiyu Chen, Wenhui Chen, Charese Smiley, Sameena Shah, Iana Borova, Dylan Langdon, Reema Moussa, Matt Beane, Ting-Hao Huang, Bryan Routledge, et al. 2021. Finqa: A dataset of numerical reasoning over financial data. *arXiv preprint arXiv:2109.00122*.
- Zhiyu Chen, Shiyang Li, Charese Smiley, Zhiqiang Ma, Sameena Shah, and William Yang Wang. 2022. Convfinqa: Exploring the chain of numerical reasoning in conversational finance question answering. *arXiv preprint arXiv:2210.03849*.
- Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.
- Li Dong, Nan Yang, Wenhui Wang, Furu Wei, Xiaodong Liu, Yu Wang, Jianfeng Gao, Ming Zhou, and Hsiao-Wuen Hon. 2019. Unified language model pre-training for natural language understanding and generation. *Advances in Neural Information Processing Systems*, 32.
- Pengcheng He, Jianfeng Gao, and Weizhu Chen. 2021. Debertav3: Improving deberta using electra-style pre-training with gradient-disentangled embedding sharing. *arXiv preprint arXiv:2111.09543*.
- Zhanming Jie, Jierui Li, and Wei Lu. 2022. Learning to reason deductively: Math word problem solving as complex relation extraction. *arXiv preprint arXiv:2203.10316*.
- Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. 2017. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *arXiv preprint arXiv:1705.03551*.
- Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
- Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. 2016. Mawps: A math word problem repository. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1152–1157.
- Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. 2019. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466.
- Xiao Li, Yin Zhu, Sichen Liu, Jiangzhou Ju, Yuzhong Qu, and Gong Cheng. 2022. Dyrren: A dynamic retriever-reranker-generator model for numerical reasoning over tabular and textual data. *arXiv preprint arXiv:2211.12668*.
- Donald MacKenzie. 2008. *An engine, not a camera: How financial models shape markets*. Mit Press.
- Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. 2016. Ms marco: A human generated machine reading comprehension dataset. In *CoCo@ NIPS*.
- Rodrigo Nogueira and Kyunghyun Cho. 2019. Passage re-ranking with bert. *arXiv preprint arXiv:1901.04085*.
- Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.
- Fabio Petroni, Aleksandra Piktus, Angela Fan, Patrick Lewis, Majid Yazdani, Nicola De Cao, James Thorne, Yacine Jernite, Vladimir Karpukhin, Jean

- Maillard, et al. 2020. Kilt: a benchmark for knowledge intensive language tasks. *arXiv preprint arXiv:2009.02252*.
- Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.
- Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J Liu, et al. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(140):1–67.
- Qiu Ran, Yankai Lin, Peng Li, Jie Zhou, and Zhiyuan Liu. 2019. Numnet: Machine reading comprehension with numerical reasoning. *arXiv preprint arXiv:1910.06701*.
- Stephen Robertson, Hugo Zaragoza, et al. 2009. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389.
- James Thorne, Andreas Vlachos, Christos Christodoulopoulos, and Arpit Mittal. 2018. Fever: a large-scale dataset for fact extraction and verification. *arXiv preprint arXiv:1803.05355*.
- Bin Wang, Jiangzhou Ju, Yunlin Mao, Xin-Yu Dai, Shujian Huang, and Jiajun Chen. 2022a. A numerical reasoning question answering system with fine-grained retriever and the ensemble of multiple generators for finqa. *arXiv preprint arXiv:2206.08506*.
- Lei Wang, Dongxiang Zhang, Jipeng Zhang, Xing Xu, Lianli Gao, Bing Tian Dai, and Heng Tao Shen. 2019. Template-based math word problem solvers with recursive neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*.
- Yanbo J Wang, Yuming Li, Hui Qin, Yuhang Guan, and Sheng Chen. 2022b. A novel deberta-based model for financial question answering task. *arXiv preprint arXiv:2207.05875*.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*.
- Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45.
- Jiaxin Zhang and Yashar Moshfeghi. 2022. Elastic: numerical reasoning with adaptive symbolic compiler. *arXiv preprint arXiv:2210.10105*.
- Renhui Zhang, Youwei Zhang, and Yao Yu. 2022. A robustly optimized long text to math models for numerical reasoning on finqa. *arXiv preprint arXiv:2207.06490*.
- Xinyi Zheng, Douglas Burdick, Lucian Popa, Xu Zhong, and Nancy Xin Ru Wang. 2021. Global table extractor (gte): A framework for joint table identification and cell structure recognition using visual context. In *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, pages 697–706.
- Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*.

## A Appendix: Operator Definition

Following by Chen et al. (2021), we define all the operations in Table 6, which illustrates the specific operators and their operands. The first four operators account for the largest distribution (about 94.29%) of the entire dataset. The operation division has the highest frequency, as calculating ratios are common in financial analysis. Moreover, *none* is a placeholder indicating that does not input another operand.

## B Appendix: Data leakage

Before May 11<sup>§</sup>, FinQA dataset suffered a serious data leakage. This is due to the fact that the ground-truth is directly utilized in the construction of the test data, resulting in an inflated recall 3 accuracy of the retriever (about 95%) and ridiculously high execution accuracy of the generator. However, when testing on the private dataset of FinQA leaderboard, which doesn't have ground truth, the model performance is very poor. ELASTIC (Zhang and Moshfeghi, 2022) uses this wrong data, so we reported FinQANet, ELASTIC, and APOLLO based on the data leakage, as shown in Table 5.

Model	Exe Acc	Prog Acc
FinQANet	65.05	63.52
ELASTIC	68.96	65.21
APOLLO	<b>72.44</b>	<b>69.48</b>
Gold-Retriever	70.00	68.76

Table 5: The experimental results on test set in FinQA with data leakage. All the model are fine-tuning on RoBERTa-large. Gold-Retriever denotes the retrieved facts directly using ground-truth.

## C Appendix: Examples of Target Program Augmentation

In Section 3.3, we give a few examples of specific construction forms of target program augmentation, and to demonstrate that our method can generate programs more diversity.

**Specific Constructed Form** We take a slightly more complex program as an example to explain exactly how we build target program augmentation. The four target program augmentation constructed forms are illustrated in Table 7.

**Diversity of Program Generation** Most of the models trained by cross-entropy which can only generate programs with fixed templates, but since both FinQA and ConvFinQA are from real finance domain, our target program augmentation and reinforcement learning training is necessary for generate programs with more flexible. We trained several models with different hyper-parameters using our method and compared their outputs to ground-truth. The results are shown in Table 8, which indicates that our model doesn't rely on templates, but generates more diverse and consistent identical programs.

<sup>§</sup><https://github.com/czyssrs/FinQA/commits/main/dataset>

Name	Operands	Output	Description
Add	(number1,number2)	number	add two numbers
Subtract	(number1,number2)	number	subtract two numbers
Multiply	(number1,number2)	number	multiply two numbers
Divide	(number1,number2)	number	divide two numbers
Exp	(number1,number2)	number	calculate exponent:
Greater	(number1,number2)	bool	return number1>number2
Table-sum	(table-header,none)	number	the summation of one table row
Table-average	(table-header,none)	number	the average of one table row
Table-max	(table-header,none)	number	the maximum number of one table row
Table-min	(table-header,none)	number	the minimum number of one table row

Table 6: The operators and operands defined by (Chen et al., 2021)

Methods	Programs
Original	Add(101, 96), Multiply(Const_10, 105), Add(#0, #1), Divide(#2, Const_3)
Switch	Add(96, 101), Multiply(Const_10, 105), Add(#0, #1), Divide(#2, Const_3)
	Add(101, 96), Multiply(105, Const_10), Add(#0, #1), Divide(#2, Const_3)
	Add(96, 101), Multiply(105, Const_10), Add(#0, #1), Divide(#2, Const_3)
	Add(101, 96), Multiply(Const_10, 105), Add(#1, #0), Divide(#2, Const_3)
	Add(96, 101), Multiply(Const_10, 105), Add(#1, #0), Divide(#2, Const_3)
	Add(101, 96), Multiply(105, Const_10), Add(#1, #0), Divide(#2, Const_3)
	Add(96, 101), Multiply(105, Const_10), Add(#1, #0), Divide(#2, Const_3)
Add & Subtract	Add(101, 96), Multiply(Const_10, 105), Add(#0, #1), Divide(#2, Const_3), Add(#3, Const_7), Subtract(#4, Const_7)
Multiply & Divide	Add(101, 96), Multiply(Const_10, 105), Add(#0, #1), Divide(#2, Const_3), Multiply(#3, Const_4), Divide(#4, Const_4)
Multiply-Divide	Add(101, 96), Multiply(Const_10, 105), Add(#0, #1), Divide(#2, Const_3), Multiply(#3, Const_1)

Table 7: The specific form of four Target program augmentation construction.

Programs
Multiply(4.7, 1000), Divide(3794, #0) (Original)
Divide(3794, 4.7), Divide(#0, 1000) (RL)
Multiply(1000, 4.7), Divide(3794, #0) (Switch)
Multiply(4.7, 1000), Divide(3794, #0), Add(#1, Const_6), Subtract(#2, Const_6) (Add & Subtract)
Multiply(4.7, 1000), Divide(3794, #0), Multiply(#2, Const_4), Divide(#3, Const_4) (Multiply & Divide)
Multiply(4.7, 1000), Divide(3794, #0), Multiply(#1, Const_1) (Multiply-Divide)

Table 8: Experiment of the programs generated by our model after reinforcement learning and target program augmentation training. Our model can generate programs with more diversity.