Improving compositional generalization for multi-step quantitative reasoning in question answering

Armineh Nourbakhsh[1,2], **Cathy Jiao**[1], **Sameena Shah**[2], **Carolyn P. Rosé**[1]

[1]Language Technologies Institute, Carnegie Mellon University
[2]J.P. Morgan AI Research
anourbak@cs.cmu.edu

Abstract

Quantitative reasoning is an important aspect of question answering, especially when numeric and verbal cues interact to indicate sophisticated, multi-step programs. In this paper, we demonstrate how modeling the compositional nature of quantitative text can enhance the performance and robustness of QA models, allowing them to capture arithmetic logic that is expressed verbally. Borrowing from the literature on semantic parsing, we propose a method that encourages the QA models to adjust their attention patterns and capture input/output alignments that are meaningful to the reasoning task. We show how this strategy improves program accuracy and renders the models more robust against overfitting as the number of reasoning steps grows. Our approach is designed as a standalone module which can be prepended to many existing models and trained in an end-to-end fashion without the need for additional supervisory signal. As part of this exercise, we also create a unified dataset building on four previously released numerical QA datasets over tabular data [1].

1 Introduction

Any natural language system that processes or interacts with numeric data requires quantitative reasoning to function. This has inspired research in several NLP domains, including reading comprehension (Andor et al., 2019; Mishra et al., 2022), textual entailment (Roy et al., 2015; Ravichander et al., 2019), data-to-text generation (Parikh et al., 2020; Suadaa et al., 2021), and question answering (Chen et al., 2020; Zhang et al., 2021). A major challenge in quantitative reasoning is the interplay between numeric expressions and natural language (Roy et al., 2015). Standard neural approaches rely heavily on lexical matching, leading to overfitting over spurious verbal patterns. In contrast, a purely symbolic approach excels at numerical reasoning, but struggles when sophisticated verbal reasoning is required (Ravichander et al., 2019). In this paper we introduce a novel attention strategy that captures the interplay between numeric and verbal modalities, which improves program accuracy and renders models more robust to overfitting.

Focusing on the question answering task, we show how our proposed method, named **CompAQT** (COMPositional Attention for QuanTitative reasoning), enables the model to attend to relevant parts of text at each reasoning step. CompAQT enhances the performance of state of the art models on several recently released QA datasets, especially for multi-step programs. It is implemented as a plug-and-play module that can be added to existing models with minimal effort and without the need for any additional supervision.

Concretely, we offer the following contributions:

- We propose a compositional attention module equipped with an alignment loss that improves SOTA performance on numeric QA tasks.
- We demonstrate how the proposed approach improves the models' program accuracy and renders them more robust in multi-step reasoning tasks.
- We combine and refine four recently released datasets on QA over tabular data. We unify their annotation schema so that they can be used interchangeably.

2 Related work

Studies that have tackled quantitative reasoning in QA tasks fall into two categories. Some studies have explored quantitative reasoning for answering questions over real-world data such as statistical records (Cheng et al., 2022), Wiki entries (Chen et al., 2020), enterprise documents (Katsis et al., 2021), and financial reports (Zhu et al., 2021).

[1]The dataset and code are available at https://github.com/ArmiNouri/CompAQT

Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, pages 1916 - 1932
December 7-11, 2022

Since numeric data is very often expressed in tabular structures, this category often involves question answering over tabular data, or hybrid table/text passages. Other studies have explored Math Word Problems (Ling et al., 2017), which require modeling abstractions and mapping arithmetic logic between language and math symbols (Kim et al., 2020; Wu et al., 2021).

2.1 QA over tabular data

As with many modern QA models, most tabular QA approaches use a retriever-generator architecture (Jurafsy and Martin, 2021), where the retriever identifies relevant table cells and encodes them using tabular encoding (Yin et al., 2020; Herzig et al., 2020) or verbalization (Chen et al., 2021). The generator produces the program necessary to derive the answer. This provides the opportunity to measure model performance in terms of program accuracy as well as execution accuracy.

Numerous studies have tackled quantitative reasoning in retriever-generator models. Retriever-focused studies have proposed structure and number aware representations that model the magnitude, polarity, or relationship among quantities (Wang et al., 2017; Herzig et al., 2020; Wu et al., 2021)[2]. The need for large-scale in-domain datasets limits the applicability of these methods. Hence generator-focused studies have attempted to enhance quantitative reasoning at generation time, using graph-based reasoning (Zhang et al., 2021), knowledge infusion (Nararatwong et al., 2022), logical programming (Kurosawa and Yanaka, 2022), and causal reasoning (Li et al., 2022).

Despite major improvements, quantitative reasoning remains a challenge (Al-Negheimish et al., 2021). The challenge stems from the memorization of spurious lexical patterns by the model, especially in the absence of large-scale training data (Ravichander et al., 2019). This is reminiscent of the problem of compositional generalization, which has been studied in-depth in numerous NLU fields including semantic parsing (Furrer et al., 2020), visual question answering (Saqur and Narasimhan, 2020), data-to-text generation (Mehta et al., 2022), and learning from instruction (Li et al., 2019). Our study adapts key findings from these domains and extends them to the quantitative QA task.

[2]Please refer to Appendix E for a detailed review of our experiments on creating operator-aware and operand-aware representations. While showing promise, they do not improve multi-step reasoning for the QA task.

2.2 Compositional generalization

Compositional generalization is a model's ability to recognize new structures that are novel, but made up of previously seen components (Montague, 1973). Oren et al. (2020) explore several methods to improve compositional generalization for semantic parsing tasks, including the downsampling of repetitive patterns, using grammar-based decoding, and supervising the attention weights to ensure proper alignments are maintained between input and output terms. A method that consistently outperforms other approaches in text-to-SQL and tabular QA tasks is attention coverage. Coverage is a penalty term that encourages the model not to pay too much attention to familiar (i.e. frequently seen) terms and focus its attention weights on new, unseen terms at test time.

Yin et al. (2021) propose a simple yet effective method to supervise attention weights for a semantic parser using a small number of samples. They first find span-level alignments between the natural language input and the program output using a heuristic algorithm. Next, they encourage attention weights to follow the alignments by adding a supervised attention loss. The loss can be thought of as a regularization term that prevents the model from overfitting to spurious patterns.

In a quantitative QA task, the input is a natural language question and the output is a program made up of arithmetic operators and operands. As such, the problem is similar to the semantic parsing task. Using this intuition, our study adapts some of the key findings from the aforementioned studies and extends them to the quantitative QA setting. We demonstrate how attention supervision and coverage penalty can be combined to improve performance on the tabular QA task, especially for multi-step programs. In contrast to Yin et al. (2021), our approach does not require any additional supervision. To the best of our knowledge, our study is the first to position and address quantitative reasoning in the context of compositional generalization.

3 Problem statement

In the retriever-generator configuration of a QA model, our goal is to improve quantitative reasoning in the generator component. Figure 1 illustrates the typical architecture of a generator as an encoder-decoder model. The encoder uses a contextual representation model such as RoBERTa (Liu

	Question	Evidence	Program
1	What was the net change in ***revenue*** from ***2019*** to ***2020***?	***2019 revenue*** was $*80*M ***2020 revenue*** was $*60*M	`subtract`(*80*, *60*)
2	What was the net change in ***expenses*** between ***2018*** and ***2021***?	***2018 expenses*** were $*20*M ***2021 expenses*** were $*30*M	`subtract`(*20*, *30*)
3	What was the percent change in ***revenue*** from ***2019*** to ***2020***?	***2019 revenue*** was $*80*M ***2020 revenue*** was $*60*M	`subtract`(*80*, *60*) `divide`(#0, *80*) `multiply`(#1, 100)
4	What was the percent change in ***expenses*** between ***2018*** and ***2021***?	***2018 expenses*** were $*20*M ***2021 expenses*** were $*30*M	`subtract`(*20*, *30*) `divide`(#0, *20*) `multiply`(#1, 100)

Table 1: Example of compositional alignments between input questions and output programs in the financial QA task. Blue underlined text indicates terms that relate to arithmetic operators. *Red italicized* text indicates terms that relate to operands. ***Bold italicized text*** indicates terms that are shared between the question and evidence.

et al., 2019). The decoder combines a recurrent module with one or more cross-attention heads between the natural language input and the program output. As the output is generated step by step, it is crucial for the cross-attention module(s) to capture relevant components of the input, otherwise they can simply memorize spurious verbal patterns and fail to generalize, especially as the number of steps grows in the output.

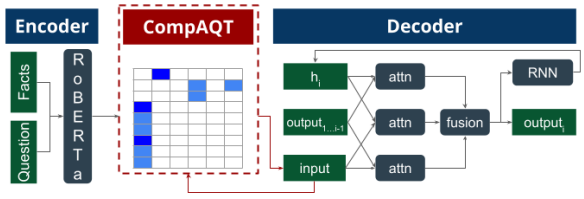

Figure 1: The typical encoder-decoder architecture of a quantitative QA generator. We introduce the compositional attention component (middle, enclosed in dotted line) to enhance the alignments between natural language input and program output.

Table 1 illustrates this phenomenon with four examples from a QA task. Each row displays a natural language question, the set of facts that can be used as evidence to answer the question, and the program to arrive at the correct answer. Presented with the first three examples, it is conceivable that a human would be able to extrapolate that "percent change" is calculated by first measuring the net change (i.e. subtraction) and then scaling the number as a percentage. Humans are able to do this by recognizing components in the question that have been previously encountered (e.g. "percent change" and "expenses") even if they were not encountered in this particular arrangement.

Many neural models struggle to exhibit the same behavior, due to overfitting to spurious patterns in natural language, or in the output. As we will later discuss in Section 5.1, quantitative reasoning datasets can exhibit a long-tail distribution, biased towards simpler patterns. Figure 2 illustrates how this phenomenon takes place in the training split of one such dataset. The figure shows the prominence of the most common sequences of arithmetic operators in the FinQA training set (Chen et al., 2021). As the number of steps grows, the tail grows longer and the sample size smaller, thus providing less information to the model and forcing it to rely on repetitively encountered patterns in the past.

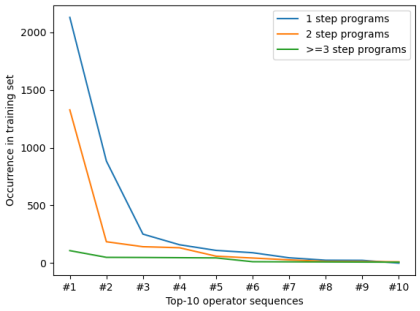

Figure 2: The long-tail effect in multi-step programs in the FinQA training set (Chen et al., 2021).

Our goal is to encourage the model to focus its attention on relevant components of the input during generation. Figure 3 illustrates the expected attention patterns for the fourth example from Table 1. The figure illustrates two key points: 1) During program generation, the terms that overlap between the question and the evidence do not matter as much as non-overlapping terms. 2) When generating operators (such as `subtract` or `divide`) attention should be focused on terms that are exclusive to the question. Whereas when generating the operands (such as `80` or `20`), attention should be focused on terms that are exclusive to the evidence. Constants such as `100` or `#0` may depend on the question, the facts, or the previously generated steps.

Using this insight, we encourage the model to adjust its attention patterns accordingly. Ideally, we would like to implement a cross-attention mechanism that mimics the alignments shown in Figure 3. However, doing so would require additional training data with gold alignments between the input and the output, akin to Yin et al. (2021). To avoid this, instead of using a *cross*-attention module, we propose a *self*-attention module applied to the input alone. The output of this self-attention module will be served as input to any downstream

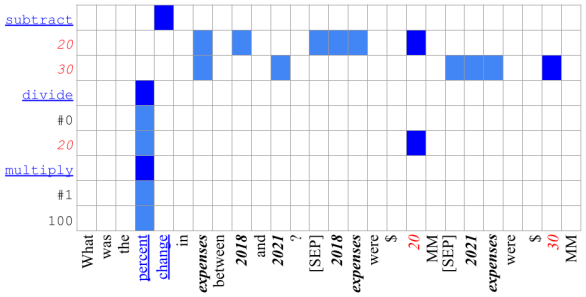

Figure 3: Semantic alignments between the natural language input from a financial QA task, and the corresponding program output.

cross-attention modules.

The self-attention module *primes* the input in such a way that downstream cross-attention can achieve the desired alignments. This means that the self-attention module attends most to tokens in the input that are non-repetitive (in Figure 3 these would be non-bold tokens such as "percent" or "change"). We hypothesize that this would make it easier for any downstream cross-attention module to distinguish between *primed* tokens and *unprimed* tokens, and use this information to distribute its attention mass more accurately, ignoring non-operative tokens.

The remaining sections describe our methodology and experimental results in detail.

4 Methodology

Let Q be a question made up of a sequence of tokens $q_1, \cdots, q_n$. Let F be the evidence obtained by the retriever, made up of a sequence of tokens $f_1, \cdots, f_m$. Note that the evidence can be composed of one or more concatenated facts, as illustrated in Table 1. Consistent with Chen et al. (2021), we represent the output program S as a sequence of steps $s_1, \cdots, s_l$. Each step s_i is composed of an operator o_i (such as `divide` or `subtract`) and exactly two operands, $a_{i,1}$ and $a_{i,2}$[3]. An operand can have one of three values: 1) It can be one of the tokens in F. 2) It can be a constant used in scaling or counting operations, e.g. `const_100`. The list of possible constants is pre-defined. 3) It can be a reference to a previous step, e.g. `#0`. The maximum number of steps is pre-defined.

[3]We follow the notation used by FinQA, where programs are modeled as right-expanding binary trees with each operation having two operands. If necessary, one or more operands are set to `NONE`. Here, `NONE` is a special constant.

Given a retriever-generator model, we prepend a self-attention module to the generator, as illustrated by the red dotted box in Figure 1. First, we encode $Q||F$ using a contextual embedding model such as RoBERTa (Liu et al., 2019) with embedding size d_{enc}. This results in an embedding matrix $\boldsymbol{U} \in \mathbb{R}^{d_{\text{enc}} \times (n+m)}$. At each generation step i, we apply scaled dot-product self-attention (Vaswani et al., 2017) to $\boldsymbol{U}$, resulting in the attention grid $\boldsymbol{A}^{(i)} \in \mathbb{R}^{(n+m)\times(n+m)}$ and the attention output $\boldsymbol{X}^{(i)} \in \mathbb{R}^{d_{\text{enc}} \times (n+m)}$. Our goal is to encourage $\boldsymbol{A}^{(i)}$ to focus its alignments properly, such that $\boldsymbol{X}^{(i)}$ supplies relevant information to the generator.

We follow a similar strategy to Yin et al. (2021), but in the absence of gold alignments, use the heuristics described in Section 3. Concretely, we add the below term to the loss:

$$\mathcal{L}^{(i)}_{\text{align}} = \frac{1}{n+m} \sum_{k=1}^{n+m} \sum_{j=1}^{n+m} (a^{(i)}_{j,k} - p_{\text{prior}}(\boldsymbol{u}^{(i)}_j | \boldsymbol{u}^{(i)}_k))^2 \quad (1)$$

where $a^{(i)}_{j,k}$ is the attention weight between the jth and kth tokens in $\boldsymbol{A}^{(i)}$, and $p_{\text{prior}}(\boldsymbol{u}^{(i)}_j | \boldsymbol{u}^{(i)}_k)$ is defined as:

$$\max\{0, \min_{j' \neq k}(\text{dist}(\boldsymbol{u}^{(i)}_{j'}, \boldsymbol{u}^{(i)}_k)) - a^{(i-1)}_{j,k}\}$$

where dist is the cosine distance between two vectors, scaled between 0 and 1, and $a^{(0)}_{j,k} = 0$ for all j and k.

The term $\min_{j' \neq k}(\text{dist}(\boldsymbol{u}^{(i)}_{j'}, \boldsymbol{u}^{(i)}_k))$ encourages the model to distribute attention to each token based on its closest similarity to any other token in the input. This is balanced against the previous attention distribution $a^{(i-1)}_{j,k}$, leading to the following behavior:

1. For tokens that are repeated more than once (e.g. those tokens shared between the question and the evidence), lower attention is encouraged. This helps the model to disregard tokens such as "expenses" and "2018" illustrated in Figure 3.

2. For terms that are unique to the question or the evidence, high attention is encouraged in early steps. This helps the model to focus on tokens such as "percent" and "20".

3. In later steps, the model is discouraged from focusing on previously well-attended tokens. For instance after the model attends to the word "change" in order to generate `subtract`, it learns to shift its focus away.

(1) and (2) emulate the regularization strategy proposed by Yin et al. (2021), while (3) emulates the concept of coverage proposed by Oren et al. (2020) with the contrast that it tracks tokens seen in previous generation steps for the same sample. The total alignment loss for a sample is calculated as an aggregation over all steps, with linear decay:

$$\mathcal{L}_{\text{align}} = \frac{1}{l}\sum_{i=1}^{l}\mathcal{L}_{\text{align}}^{(i)} - \alpha i \qquad (2)$$

The linear decay term helps the model assign a higher penalty to earlier generation steps. Suppose the gold program is:

- `subtract(20, 30), divide(#0, 20)`

and the output generated by the model is either one of the below options:

- `subtract(20, 30), multiply(#0, 20)`
- `add(20, 30), multiply(#0, 20)`

In the absence of linear decay, both predictions would receive the same penalty. The decay term assigns a lower penalty to the first prediction, since it gets the first operation correct.

Finally, the alignment loss can be added to the default loss of the generator:

$$\mathcal{L}_{\text{total}} = \lambda\mathcal{L}_{\text{align}} + (1-\lambda)\mathcal{L}_{\text{generator}} \qquad (3)$$

5 Experiments

In this section, we describe our experimental set up, including the datasets and the baseline models.

5.1 Datasets

We use four datasets that focus on numerical reasoning over hybrid table/text context, all released within a year of this publication.

FinQA (Chen et al., 2021) is based on a collection of financial reports published by U.S. companies that were released as part of FinTabNet (Zheng et al., 2021). Each passage is composed of a table and a few sentences that surround the table, describing its content. The questions, designed by domain experts, all require numerical reasoning.

TAT-QA (Zhu et al., 2021) is also focused on financial reports, but includes documents from non-U.S. companies. As such, the reports do not conform to a standard format and include a more diverse set of metrics. The dataset includes span-based and multi-span questions as well as questions requiring arithmetic reasoning. In our experiments, we focus on the latter category.

HiTab (Cheng et al., 2022) is a collection of tables that include statistical data, collected from various national agencies. The tables have complex hierarchical or nested structure, and answering them requires spatial as well as numerical reasoning. As with TAT-QA, we discard questions that do not require any arithmetic operations.

MULTIHIERTT (Zhao et al., 2022), which is also based on FinTabNet, combines the challenges of the above-mentioned datasets, bringing together complex tabular structures and hybrid table/text contexts. Again, we filter the dataset down to those samples that require numerical reasoning.

All four datasets provide the reasoning program required to derive the answers, allowing any model to be evaluated on program accuracy. Since our study is focused on multi-step generation, we use program accuracy as our evaluation metric.

5.1.1 Unified dataset

Of the datasets mentioned above, FinQA is exclusively focused on multi-step quantitative reasoning. The remaining datasets tackle additional challenges such as extractive QA, spatial reasoning, and table representation. Therefore we filter TAT-QA, HiTab, and MULTIHIERTT down to those samples that require quantitative reasoning. We also transform each sample so that it conforms to the standard FinQA format. This helps us bypass the challenge of addressing complex tabular structures, which is out of scope for this study. Please refer to Appendix A for further details.

Table 2 shows statistics for each dataset, as well as the distribution of 1 step, 2 step, and 3+ step programs.

[4] TAT-QA samples include a flag to distinguish arithmetic questions from span-based questions. However, this flag is only available in the train and dev sets, but not in the test set. Therefore we split the dev set into 230 dev examples and 307 test examples.

[5] MULTIHIERTT does not include an annotated test set. Therefore we split the validation set into 100 dev examples and 108 test examples.

Dataset	# passages	# QA pairs used			# steps in program		
		Train	Dev	Test	1 step	2 steps	3+ steps
FinQA	2,789	6,251	883	1,147	4,894	2,709	678
TAT-QA[4]	2,479	4,355	230	307	2,721	616	1,555
HiTab	513	879	186	199	1,075	120	69
MULTIHIERTT[5]	2,291	2,083	100	108	560	862	869
Combined	8,071	13,568	1,349	1,761	9,520	4,388	3,171

Table 2: Statistics of the four datasets, and the combined dataset. Note that with the exception of FinQA, only a subset of samples (involving multi-step quantitative reasoning) is used from each dataset.

5.2 Baselines

To establish baselines, we use the following models that have demonstrated SOTA performance on the datasets mentioned in the previous section[6].

FinQANet was proposed by Chen et al. (2021) and applied to the FinQA dataset. The architecture is similar to that illustrated in Figure 1 but missing the compositional self-attention module.

TAGOP was proposed by Zhu et al. (2021) and applied to the TAT-QA dataset. A crucial difference between TAGOP and FinQANet is that the former is not designed to perform multi-step reasoning, but approaches the task as a classification problem. As an example, it may predict that a `change ratio` calculation is required, which implies a subtraction followed by a division. Seven such arithmetic operations are permitted.

In addition to the above, we use a pointer-verbalizer network (**PVN**) as a universal baseline against all four datasets. The model is inspired by the Expression-Pointer Transformer proposed by Kim et al. (2020). The authors argue that generating an arithmetic program as a disjoint sequence of operators and operands is not consistent with how humans approach quantitative reasoning. Instead, they propose the concept of an "Expression Token", which represents a full operation autonomously (e.g. instead of generating `divide`, `20`, `30` as a sequence, they recommend generating `divide(20, 30)` as one token). Following this idea, PVN also generates Expression Tokens, but uses two pointer mechanisms—one to select operators from the list of all possible options, and one to select operands from the list of numbers expressed in the evidence, or a predetermined list of possible constants. In addition, it uses verbalization to map operators from a symbolic space (e.g. +) into the semantic space (e.g. "divide"). Please refer to Appendix B for implementation details.

We use the above three models as baselines, and measure their performance before and after adding CompAQT. To remain as consistent as possible with the initial settings of these models, we use the same hyperparameters and settings described in the original papers. We also use RoBERTa-large (Liu et al., 2019) to encode the input, since all models report best performance on this model. We perform grid search on the development set of FinQA to tune the values for α and λ, which are subsequently both set to 0.1. We use the same values throughout all of our experiments. Please refer to Appendix C for additional details and the full list of hyperparameters for each baseline.

Note that TAGOP has been designed for single-step programs. Therefore we apply it to the original version of the TAT-QA dataset, but analyze the results based on the actual number of steps in each program. When adding CompAQT to TAGOP, we prepend it once, and instead of using the multi-step loss with linear decay, we only calculate the alignment loss once per program. Further, note that since our study is only focused on generation and not retrieval, we use gold facts provided by each dataset. Please refer to Appendix D to see details of experiments using retrieved facts.

6 Results and discussion

In this section, we investigate the effectiveness of CompAQT through four questions:

1. Does CompAQT improve the performance of the baseline models on the four datasets?

2. Does CompAQT encourage compositional generalization by enabling the models to attend to relevant parts of the input?

3. Does each component of CompAQT contribute to enhanced performance?

4. Can CompAQT's performance be attributed merely to added parameters?

Additionally, we examine whether the combined dataset offers an advantage over the largest constituent dataset.

[6]The creators of HiTab and MULTIHIERTT have proposed baseline models that were not included in our experiments. This is because the HiTab model is focused on encoding tabular data rather than quantitative reasoning. The MULTIHIERTT model, named MT2Net, is similar to FinQANet, but includes an additional sub-module that only applies to span-based questions—again, out of the scope of this study.

Model	Dataset	Program accuracy			
		1 step	2 steps	3+ steps	Overall
TAGOP[7]	TAT-QA	45.01	39.56	42.73	43.25
+CompAQT		+1.06	+0.72	+1.00	+0.63
PVN	Combined	68.14	61.33	13.54	56.64
+CompAQT		**+2.64**	**+2.12**	**+3.09**	**+2.57**
FinQANet	FinQA	75.63	65.87	30.36	68.44
+CompAQT		**+3.05**	**+9.25**	**+5.49**	**+5.30**
FinQANet	TAT-QA	73.33	63.76	64.88	70.71
+CompAQT		**-3.33**	+0.00	+1.38	-0.74
FinQANet	HiTab	34.70	25.14	15.91	30.12
+CompAQT		+0.03	**+4.50**	+1.44	**+2.11**
FinQANet	MULTIHIERTT	38.99	40.07	15.01	38.94
+CompAQT		-0.12	**+2.28**	+1.76	**+1.88**
FinQANet	Combined	65.11	62.00	30.76	58.60
+CompAQT		+1.49	**+3.14**	**+3.40**	**+2.28**

Table 3: Program accuracy for program generation using baseline models. Additional performance gain/loss is indicated after CompAQT is added to each model. **Bold** numbers indicate that a gain/loss is significant at $p < 0.005$, based on the paired-bootstrap test proposed by Berg-Kirkpatrick et al. (2012), with $b = 10^3$.

6.1 Model performance

Table 3 shows the program accuracy of each baseline model, before and after adding CompAQT. As the table illustrates, CompAQT significantly contributes to performance on multi-step programs in three of the four datasets. An interesting exception is the TAT-QA dataset, which does not exhibit the long-tail distribution displayed in Figure 2. As Table 2 shows, TAT-QA is biased towards 3-step programs with repeating patterns (e.g. `change ratio` is a common 3-step program). Here, CompAQT offers comparable performance to the baseline, with slightly higher robustness to multi-step programs, sometimes at a slight cost to single-step programs. For datasets that exhibit the long-tail distribution, CompAQT offers improvement on all categories, but especially on multi-step programs. This is especially noteworthy for HiTab, which is the smallest and most skewed collection.

Among the three baselines, FinQANet outperforms others on individual as well as the combined dataset. Adding CompAQT further improves FinQANet's performance on all datasets with the exception of TAT-QA. Therefore we use FinQANet+CompAQT for the remaining analyses presented in this paper.

[7]Note that TAGOP cannot generate multi-step programs and can therefore only be applied to the TAT-QA dataset, where multi-step programs have been collapsed into single-step operations (e.g. `change ratio`).

6.2 Qualitative examples

Table 4 shows four examples from the validation set of the FinQA dataset. The first question asks for a percentage calculation. Percentage calculations are the most common two-step operation in the training set, and the FinQANet model is able to produce the gold program without any additional guidance from CompAQT. In the second example, the model is asked to perform an operation on two metrics that are expressed as percentages. This time, possibly by relying heavily on memorizing the relationship between the word "percentage" and the `subtract-divide` operation, FinQANet mistakenly generates a `subtract-divide` sequence, whereas CompAQT is able to determine that "percentage" refers to an operand. In the third example, FinQANet once again performs a percentage calculation, possibly by associating the word "change" with "percentage change". Once again CompAQT is able to drive attention towards the correct program, and distinguishes between a net and a percent change.

The final example shows a case where CompAQT is not able to improve baseline performance. This challenge here is to understand the relationship between the number of shares and the average price. This requires a level of financial literacy that is not resolved by compositional generalization alone. As demonstrated in Chen et al. (2021) financial expertise plays a major role, even in human performance.

6.3 Attention patterns

To confirm whether CompAQT is assisting the model in detecting key operational terms, we analyze the top-attended tokens within the input. Table 5 lists the top attended tokens throughout the training process for the FinQANet+CompAQT model on the FinQA dataset.

As training progresses, CompAQT encourages the model to attend to key terms that indicate arithmetic operations (such as "net" and "growth"). Swapping CompAQT with a basic self-attention module shows a similar convergence, but the module is not as quick to learn important terms. In fact even at the 50th epoch, the basic self-attention module is still encouraging attention on terms that do not indicate an operation, such as "annual" and "year". This shows that the additional components in CompAQT (alignment and coverage loss) assist the model in converging to more meaningful

Question	Evidence	Gold program	FinQANet	FinQANet + CompAQT
What was the percentage change in the fair value from 2010 to 2011?	1) the fair value of 2011 is $99 2) the fair value of 2010 is $81	`subtract(99, 81), divide(#0, 81)`	`subtract(99, 81), divide(#0, 81881)`	`subtract(99, 81), divide(#0, 81)`
What was the difference in operating profit as a percentage of net sales between 2001 and 2003?	1) the company reported operating profit as a percent of net sales 2) operating profit in 2001 is 19 3) operating profit in 2003 is 26	`subtract(26, 19)`	`subtract(26, 19), divide (#0, 19)`	`subtract(26, 19)`
What is the change in the warranty reserve from 2017 to 2018?	1) balance as of 2017 is $23 2) balance as of 2018 is $24	`subtract(24, 23)`	`subtract(24, 23), divide(#0, 23)`	`subtract(24, 23)`
For the 4th quarter of 2011 approximately how much was spent on stock repurchases?	1) total number of shares purchased is 3915 2) total of average price paid per share is $98	`multiply(3915, 98)`	`add(3915, 98)`	`add(3915, 98)`

Table 4: Four examples from the FinQA dataset, showing CompAQT's success and failure in capturing compositional expressions. Note that some numbers have been truncated to save space.

	Top attended token		
	Epoch #1	**Epoch #25**	**Epoch #50**
Basic self-attention	[CLS], ?, what, and	company, what, year, 2018	percentage, ratio, annual, year
CompAQT	the, of, ?, what	year, company, percentage, annual	percentage, growth lowest, net

Table 5: Top attended tokens throughout the training process for a vanilla self-attention module versus CompAQT. As training progresses, CompAQT learns to attend to tokens closely associated with quantitative operations. The results are based on FinQANet+CompAQT, applied to the FinQA dataset.

Model	Program accuracy			
	1 step	**2 steps**	**3+ steps**	**Overall**
FinQANet	75.63	65.87	30.36	68.44
+self-attention	+2.95	+2.08	+2.00	+2.57
+alignment loss	+0.07	+5.31	+2.01	+1.95
+coverage term	+0.03	+1.97	+0.97	+0.69
+linear decay	+0.00	+0.07	+0.51	+0.06

Table 6: Ablation results on the FinQA dataset using FinQANet as the base model.

attention patterns.

6.4 Ablation study

We perform a series of experiments to examine the impact of each component of CompAQT. Table 6 shows the results after applying FinQANet to the FinQA dataset. "Self-attention" indicates the addition of a plain self-attention module without any compositional guidance. "Alignment loss" indicates the addition of the minimum-distance component in Equation 4. "Coverage term" indicates the addition of $-a_{j,k}^{(i-1)}$ to alignment loss. "Linear decay" indicates replacing a simple average loss with the linear decay term in Equation 2. As the table shows, each component contributes to the program accuracy. The self-attention module offers an improvement that is relatively consistent across all programs, whereas the alignment loss and the coverage term favor multi-step programs, as intended. Lastly, linear decay further improves results for the longest programs by a small margin.

To ensure that the effectiveness of CompAQT is not simply due to added parameters, we also perform a series of experiments that measure the performance of CompAQT with additional parameters in the form of additional layers and attention heads. Each row in Table 7 shows how much program accuracy improves over using FinQANet without CompAQT. As the table shows, additional parameters are not always helpful and can undermine the performance of the model, especially for multi-step programs. This may also indicate that the regularizing effect of CompAQT can be counteracted by larger parameters, leading to overfitting over smaller datasets.

6.5 Pre-training across datasets

Since the combined dataset uses the same format for all constituent datasets, it is easy to investigate the impact of pre-training on larger and more diverse data. Figure 4 shows how FinQANet+CompAQT performs on the combined test set, as more collections are added to its training set. As the Figure illustrates, despite providing a sizeable number of single-step programs, TAT-QA fails to improve the performance substantially on multi-step programs. This is likely due to the fact that all TAT-QA programs fall into seven categories, which allows the model to memorize them. In contrast, despite its small size, adding MULTIHIERTT improves performance on 1 step and 2 step programs. 3+ step programs remain a challenge across all datasets, but the model shows steady progress as the dataset size grows.

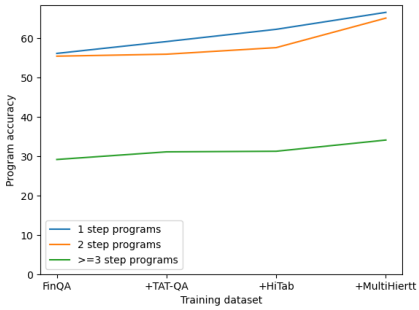

Figure 4: Program accuracy on the combined dataset as training datasets are added iteratively.

# heads	# layers	# params	# Program accuracy (improvement over baseline)		
			1 step	2 steps	3+ steps
1	1	4.2M	+3.05	+9.25	+5.49
4	1	4.3M	+3.97	+7.30	+5.31
1	2	8.4M	+4.22	+6.76	+3.12
4	2	8.6M	+4.26	+5.99	+2.86

Table 7: The performance of FinQANet+CompAQT on the FinQA dataset. As additional parameters are added in the form of multiple heads or more layers, the model's performance does not increase.

7 Conclusion

In this study, we proposed a method to improve multi-step quantitative reasoning for question answering. Our method facilitates compositional generalization by encouraging the model to attend to relevant components of the input at each generation step. We demonstrated the effectiveness of our approach over four recently released tabular QA datasets. Our method, named CompAQT, was able to significantly improve program accuracy on three of the datasets, especially for multi-step programs. We also created a collection of QA samples for multi-step quantitative reasoning, by combining the datasets and unifying their format.

In future studies, we hope to explore data-native approaches to quantitative reasoning such as augmentation and synthesis, as well as approaches embedded in symbolic reasoning (Ravichander et al., 2019; Shi et al., 2020).

8 Limitations

All datasets used in this study were created based on English-language documents, with three of the four datasets focusing on financial reports, and two focusing specifically on regulatory disclosures provided by the United States Securities and Exchange Commission. As such, our unified dataset is biased towards attributes and patterns expected in such reports, including GAAP metrics[8], currency units, and left-to-right orientation for tabular structures.

Since our study is focused on the generation component of a QA model, we have disregarded the challenges involves in retrieval and representation of information such as those explored by Cheng et al. (2022) and Zhao et al. (2022).

Lastly, our approach is not intended for settings where complex or high-level quantitative insights are required (e.g. anomaly or trend detection). In such settings the large space of operations makes it challenging to rely exclusively on soft alignments between the input and output.

References

Hadeel Al-Negheimish, Pranava Madhyastha, and Alessandra Russo. 2021. Numerical reasoning in machine reading comprehension tasks: are we there yet? In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 9643–9649, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Daniel Andor, Luheng He, Kenton Lee, and Emily Pitler. 2019. Giving BERT a calculator: Finding operations and arguments with reading comprehension. 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)*, pages 5947–5952, Hong Kong, China. Association for Computational Linguistics.

Taylor Berg-Kirkpatrick, David Burkett, and Dan Klein. 2012. An empirical investigation of statistical significance in NLP. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 995–1005, Jeju Island, Korea. Association for Computational Linguistics.

Wenhu Chen, Hanwen Zha, Zhiyu Chen, Wenhan Xiong, Hong Wang, and William Yang Wang. 2020. HybridQA: A dataset of multi-hop question answering over tabular and textual data. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1026–1036, Online. Association for Computational Linguistics.

Zhiyu Chen, Wenhu Chen, Charese Smiley, Sameena Shah, Iana Borova, Dylan Langdon, Reema Moussa, Matt Beane, Ting-Hao Huang, Bryan Routledge, and William Yang Wang. 2021. Finqa: A dataset of numerical reasoning over financial data. *Proceedings of EMNLP 2021*.

[8] https://www.cfainstitute.org/en/advocacy/issues/gaap

Zhoujun Cheng, Haoyu Dong, Zhiruo Wang, Ran Jia, Jiaqi Guo, Yan Gao, Shi Han, Jian-Guang Lou, and Dongmei Zhang. 2022. HiTab: A hierarchical table dataset for question answering and natural language generation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1094–1110, Dublin, Ireland. Association for Computational Linguistics.

Daniel Peter Furrer, Marc van Zee, Nathan Scales, and Nathanael Schärli. 2020. Compositional generalization in semantic parsing: Pre-training vs. specialized architectures. *arXiv e-prints*, page arXiv:2007.08970.

Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Eisenschlos. 2020. TaPas: Weakly supervised table parsing via pre-training. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4320–4333, Online. Association for Computational Linguistics.

Dan Jurafsy and James H. Martin. 2021. *Speech and language processing*, 3 edition, chapter 23.

Yannis Katsis, Saneem Chemmengath, Vishwajeet Kumar, Samarth Bharadwaj, Mustafa Canim, Michael Glass, Alfio Gliozzo, Feifei Pan, Jaydeep Sen, Karthik Sankaranarayanan, and Soumen Chakrabarti. 2021. Ait-qa: Question answering dataset over complex tables in the airline industry.

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)*, Online. 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*.

Tomoya Kurosawa and Hitomi Yanaka. 2022. Logical inference for counting on semi-structured tables. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics: Student Research Workshop*, pages 84–96, Dublin, Ireland. Association for Computational Linguistics.

Moxin Li, Fuli Feng, Hanwang Zhang, Xiangnan He, Fengbin Zhu, and Tat-Seng Chua. 2022. Learning to imagine: Integrating counterfactual thinking in neural discrete reasoning. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 57–69, Dublin, Ireland. Association for Computational Linguistics.

Yuanpeng Li, Liang Zhao, Jianyu Wang, and Joel Hestness. 2019. Compositional generalization for primitive substitutions. 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)*, pages 4293–4302, Hong Kong, China. Association for Computational Linguistics.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. 2017. Program induction by rationale generation: Learning to solve and explain algebraic word problems. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 158–167, Vancouver, Canada. Association for Computational Linguistics.

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. *ArXiv*, abs/1907.11692.

Sanket Vaibhav Mehta, Jinfeng Rao, Yi Tay, Mihir Kale, Ankur Parikh, and Emma Strubell. 2022. Improving compositional generalization with self-training for data-to-text generation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4205–4219, Dublin, Ireland. Association for Computational Linguistics.

Swaroop Mishra, Arindam Mitra, Neeraj Varshney, Bhavdeep Singh Sachdeva, Peter Clark, Chitta Baral, and Ashwin Kalyan. 2022. Numglue: A suite of fundamental yet challenging mathematical reasoning tasks. *CoRR*, abs/2204.05660.

Richard Montague. 1973. *The Proper Treatment of Quantification in Ordinary English*, pages 221–242. Springer Netherlands, Dordrecht.

Rungsiman Nararatwong, Natthawut Kertkeidkachorn, and Ryutaro Ichise. 2022. KIQA: Knowledge-infused question answering model for financial table-text data. In *Proceedings of Deep Learning Inside Out (DeeLIO 2022): The 3rd Workshop on Knowledge Extraction and Integration for Deep Learning Architectures*, pages 53–61, Dublin, Ireland and Online. Association for Computational Linguistics.

Inbar Oren, Jonathan Herzig, Nitish Gupta, Matt Gardner, and Jonathan Berant. 2020. Improving compositional generalization in semantic parsing. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 2482–2495, Online. Association for Computational Linguistics.

Ankur Parikh, Xuezhi Wang, Sebastian Gehrmann, Manaal Faruqui, Bhuwan Dhingra, Diyi Yang, and Dipanjan Das. 2020. ToTTo: A controlled table-to-text generation dataset. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1173–1186, Online. Association for Computational Linguistics.

Abhilasha Ravichander, Aakanksha Naik, Carolyn Rose, and Eduard Hovy. 2019. EQUATE: A benchmark evaluation framework for quantitative reasoning in natural language inference. In *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*, pages 349–361, Hong Kong, China. Association for Computational Linguistics.

Subhro Roy, Tim Vieira, and Dan Roth. 2015. Reasoning about Quantities in Natural Language. *Transactions of the Association for Computational Linguistics*, 3:1–13.

Raeid Saqur and Karthik Narasimhan. 2020. Multimodal graph networks for compositional generalization in visual question answering. In *Advances in Neural Information Processing Systems*, volume 33, pages 3070–3081. Curran Associates, Inc.

Qi Shi, Yu Zhang, Qingyu Yin, and Ting Liu. 2020. Learn to combine linguistic and symbolic information for table-based fact verification. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 5335–5346, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Lya Hulliyyatus Suadaa, Hidetaka Kamigaito, Kotaro Funakoshi, Manabu Okumura, and Hiroya Takamura. 2021. Towards table-to-text generation with numerical reasoning. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1451–1465, Online. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. In *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc.

Yan Wang, Xiaojiang Liu, and Shuming Shi. 2017. Deep neural solver for math word problems. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 845–854, Copenhagen, Denmark. Association for Computational Linguistics.

Qinzhuo Wu, Qi Zhang, Zhongyu Wei, and Xuanjing Huang. 2021. Math word problem solving with explicit numerical values. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 5859–5869, Online. Association for Computational Linguistics.

Pengcheng Yin, Hao Fang, Graham Neubig, Adam Pauls, Emmanouil Antonios Platanios, Yu Su, Sam Thomson, and Jacob Andreas. 2021. Compositional generalization for neural semantic parsing via span-level supervised attention. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2810–2823, Online. Association for Computational Linguistics.

Pengcheng Yin, Graham Neubig, Wen tau Yih, and Sebastian Riedel. 2020. TaBERT: Pretraining for joint understanding of textual and tabular data. In *Annual Conference of the Association for Computational Linguistics (ACL)*.

Qiyuan Zhang, Lei Wang, Sicheng Yu, Shuohang Wang, Yang Wang, Jing Jiang, and Ee-Peng Lim. 2021. NOAHQA: Numerical reasoning with interpretable graph question answering dataset. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 4147–4161, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Yilun Zhao, Yunxiang Li, Chenying Li, and Rui Zhang. 2022. MultiHiertt: Numerical reasoning over multi hierarchical tabular and textual data. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6588–6600, Dublin, Ireland. Association for Computational Linguistics.

X. Zheng, D. Burdick, L. Popa, X. Zhong, and N. Wang. 2021. Global table extractor (gte): A framework for joint table identification and cell structure recognition using visual context. In *2021 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 697–706, Los Alamitos, CA, USA. IEEE Computer Society.

Fengbin Zhu, Wenqiang Lei, Youcheng Huang, Chao Wang, Shuo Zhang, Jiancheng Lv, Fuli Feng, and Tat-Seng Chua. 2021. TAT-QA: A question answering benchmark on a hybrid of tabular and textual content in finance. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3277–3287, Online. Association for Computational Linguistics.

A Dataset unification

In order to unify the datasets, we convert each of them into the same format. We choose FinQA as the reference format, since it encodes multi-step programs in a standardized representation. Each program is encoded as a right-expanding binary tree where each operation is guaranteed to have two operands, with one of more operands set to `NONE` if necessary.

FinQA provides two versions of each data-table: one with the raw format, and one where the content has been normalized such that all row headers are merged into one row header, and all column headers are merged into one column header. This removes discrepancies in the way tabular data is represented across different samples. The dataset also includes gold facts from each table and surrounding text. The facts have been tokenized and verbalized such that the model can easily map operands to number expressed in them.

Transforming other datasets to match the FinQA format requires following three steps: 1) Normalizing the tables, 2) Normalizing the programs, and 3) Normalizing the evidence. Below, we describe these steps in detail.

A.0.1 Normalizing the tables

In the TAT-QA and HiTab, each table is represented as a nested array. In MULTIHIERTT, each table is represented by the raw html code, which is easily convertible into a nested array.

The top n rows and the left m columns of each table form the column and row headers, respectively. To find n, we follow Algorithm 1. A similar method is applied to determine m. We then merge the contents of the top n rows and the left m columns to form a singular column header and a singular row header.

A.0.2 Normalizing the programs

MULTIHIERTT represents programs in a format that is compatible with FinQA (e.g. `subtract(20, 30)`, `divide(#0, 20)`). TAT-QA and HiTab represent them in a non-standardized format (e.g. `(20-30)/20` or `1-30/20`). We use an Abstract Syntax Parser[9] to process each expression into a tree. We then programmatically traverse the tree to generate a FinQA-compatible program. Next, we match each operand to the evidence. If not found within the evidence, an operand is replaced by a constant (e.g. `multiply(#0, 100)` → `multiply(#0, const_100)`.

Algorithm 1 Column header finder

```
N ← num_rows
K ← num_cells_in_first_row
non_empty_cells ← []
for i ∈ {1, ..., K} do
    if table[1][i] ≠ empty then
        non_empty_cells[i] = TRUE
    else
        non_empty_cells[i] = FALSE
    end if
end for
n ← 2
while n ≤ N and FALSE ∈ non_empty_cells do
    for i ∈ {1, ..., K} do
        if table[n][i] ≠ empty then
            non_empty_cells[i] = TRUE
        else
            non_empty_cells[i] = FALSE
        end if
    end for
    n := n + 1
end while

return n
```

A.0.3 Normalizing the evidence

We use a tokenizer similar to the FinQA tokenizer to process each sentence and table within each passage. This is to ensure that the operands are guaranteed to match the numbers mentioned in the evidence.

Table 8 shows additional filters applied to each dataset.

Dataset	Configuration
FinQA	no filters
TAT-QA	arithmetic category only
HiTab	no multi-span answers at least one operation required
MULTIHIERTT	arithmetic category only no span-based answers

Table 8: How each dataset was filtered to include in the unified collection.

[9] We use the standard python3.7 ast module. https://docs.python.org/3.7/library/ast.html

B Pointer Verbalizer Network

The programs are composed of two types of tokens: 1) operator tokens, which are sampled from a symbolic space (i.e. math symbols), and 2) operators, which are either numbers mentioned in the facts, or constants such as 100 or 1,000,000 for scaling the output. Models such as FinQANet treat the program as a sequence of tokens, not differentiating between operators and operands. At each step a new token is sampled from the universe of all possible operators and operands, and a mask is used to make sure two operands are generated after each operator. The process stops once the `EOF` operator is generated.

In contrast to FinQANet, our Pointer Verbalizer Network uses a two-pronged approach that accounts for the differences between the operators and operands, and employs verbalization to enhance performance.

B.1 Generating operators

The generator can be regarded as an encoder-decoder model akin to sequence-to-sequence models employed in Neural Machine Translation (NMT). However as opposed to NMT, the operators are sampled from a symbolic space (i.e. math symbols), which does not have the same distribution or compositionality as the input space (i.e. text). A similar problem exists in most semantic parsing tasks, but it is sometimes alleviated by mapping the target domain into a space that is close to the input domain. For example in a Text-to-SQL task, instead of referring to "table_#3.column_#1", the names or descriptions of the table and column are used (e.g. "students.name"). This allows the model to leverage compositional semantics in the target domain as well as the source domain.

We pursue a similar strategy by mapping the program operators into text, i.e. verbalizing them. For example, instead of the categorical symbol `divide`, the token "divide" is used to represent the division operation. As a result, the operator generator produces a sequence of tokens. These tokens can be mapped to the nearest operator based on their cosine similarity. The loss is thus calculated as:

$$\mathcal{L}_{\text{operator}} = \gamma \mathcal{L}_{\text{CE}} + (1-\gamma)\mathcal{L}_{\text{reg}} \quad (4)$$

where γ is a hyperparameter, $\mathcal{L}_{\text{CE}}$ is the cross-entropy loss between the predicted operators and the true operators, and $\mathcal{L}_{\text{reg}}$ is a regression loss defined as the sum of cosine distance and MSE loss between predicted operator token embeddings $\mathbf{o}_i$ and the true operator token embeddings $\mathbf{a}_i$.

$$\mathcal{L}_{\text{reg}} = \frac{1}{N}\sum_{i=1}^{N} \text{cosine}(\mathbf{o}_i, \mathbf{a}_i) + \text{MSE}(\mathbf{o}_i, \mathbf{a}_i) \quad (5)$$

The regression loss functions as a regularizer to ensure that the verbalized predictions do not stray too far from true operator tokens. In our experiments, we set $\gamma = 0.8$.

B.2 Generating operands

As opposed to the operators, the operands are always selected from a list of existing numbers or constants. This, along with the verbalization of operators allows us to approach operand-generation using a pointer network (Vinyals et al., 2015). At each step, the predicted operator token embedding $\mathbf{o}_i$ is used as the hidden state, and the model selects the top two options from the list of possible operands.

Figure 5 illustrates the proposed architecture. The top part of the figure shows how the sequence of operators id generated. The cross-attention mechanism helps augment the question with information from the facts. The output is then attended to by the verbalized vocabulary of operators. The output of this step is used in a recurrent network to predict the operator in each step.

The bottom part of the figure shows how the operands are selected for each predicted operator. This time, attention is used to augment the fact representation with information from the question. Using this, as well as the output of the operator predictor, the model uses a pointer network to select operands from a list of possible numbers and constants.

C Experiment details

Tables 9 lists the settings and parameters for each baseline. The settings used for FinQANet and TAGOP are based on (Chen et al., 2021) and (Zhu et al., 2021), respectively. For all experiments that involved CompAQT, α and λ were both set to 0.1.

All experiments were conducted on a machine with 8 NVIDIA T4 GPUs with 16GBs of memory per GPU.

D Experiments using retrieved facts

Table 10 shows ablation results similar to Table 6 but for retrieved facts (instead of gold facts). Sim-

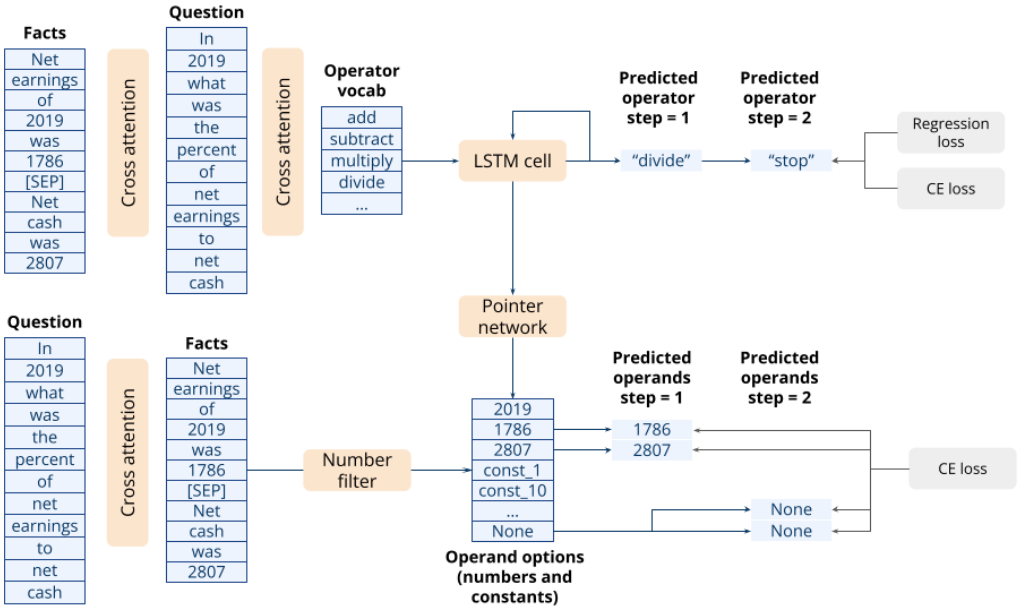

Figure 5: The architecture of the Pointer Verbalizer Network. The top half shows the operator predictor and the bottom half shows the operand predictor.

Parameter	FinQANet	TAGOP	PVN
encoder	RoBERTa-large(Liu et al., 2019)		
batch size	16	32	64
learning rate	$2e-5$	$5e-5$	$2e-5$
optimizer	Adam (Kingma and Ba, 2015) with $\beta_1 = 0.9$ and $\beta_2 = 0.999$		
epochs	100	50	50

Table 9: Settings used for FinQANet experiments.

Model	Program accuracy			
	1 step	2 steps	3+ steps	Overall
FinQANet	64.13	57.03	20.56	58.30
+self-attention	+3.01	+1.97	+1.95	+2.44
+alignment loss	+0.01	+4.66	+1.83	+2.00
+coverage term	+0.00	+2.02	+0.55	+0.31
+linear decay	+0.00	+0.10	+.48	+0.00

Table 10: Ablation results on the FinQANet model, applied to the FinQA dataset with retrieved facts.

ilar trends hold when using retrieved facts, with CompAQT modifications having a larger impact on multi-step programs.

E Operation-aware pre-training

The FinQANet generator performs better on examples with simpler programs which contain 1 or 2 steps compared complex programs with 3 or more steps. We perform masked language modeling (MLM) finetuning on the FinQANet language encoder over FinQA and TAT-QA data to see how it can encourage the generator perform better on complicated programs. First, we take the dev sets of FinQA and TAT-QA, and split both dataset 80/20 for training and testing masked token prediction. The resulting train and test set sizes are 1136 and 284, respectively. Second, we finetune Roberta (Liu et al., 2019) using three masking methods.

- **op**: mask one operator in every program
- **const**: mask one constant/operand in every program
- **op+const**: mask either an operator (50%) or a constant/operand (50%) in every program

For each example in our new dataset, we encode the natural language question, supporting facts, and program in a sequence. For each experiment, we finetune Roberta until it achieves 70-80% accuracy in guessing the masked tokens. Finally, we train the FinQANet program generator with our finetuned Roberta model. During inference we use the finetuned Roberta encoder in the FinQANet generator.

Table 11 shows the experiment results. The baseline program out-performs most of the finetuned program generators except on the FinQA test set. In all experiments, masking the program operators (e.g., add, subtract, divide) leads to better performance compared to masking program operands or masking both operators and operands. A possible explanation may be that there are only 10 program

Finetune	FQA-Valid	FQA-Test	TQA-Valid
baseline	62.29	61.90	71.32
op	61.61	**62.34**	70.35
const	60.02	58.24	71.32
op+const	58.55	57.9	68.60

Table 11: Program accuracy of FinQANet program generator with baseline and finetuned Roberta encoders on the FinQA dev set. FQA-dev: results on the FinQA dev set at 25 epochs. FQA-test: results on the FinQA test set at 300 epochs. TQA-dev: results on the TAT-QA dev set at 50 epochs.

operators compared to many possible operands. Furthermore, given the small size of our data used for finetuning, masking different types of tokens with a large range of values may confuse the finetuned model, and impair its ability to learn ties between text semantics and the desired output program. This intuition is reflected in figure 6 which shows the loss and accuracy curves of the op, const, and op+const experiments compared to the baseline program generator.

We conduct further analysis on the generated programs with respect to program complexity to better understand how finetuning the Roberta encoder affects the generated programs. As shown in figure 7, the baseline program generator performs better than most finetuned program generators in program accuracy in 1-step and 2-step programs. However, the finetuned program generators perform better than the baseline program generator in program accuracy on ≥ 4-step programs on both the FinQA dev and test sets. A possible explanation is that while the baseline program generator is competitive in overall program accuracy, it is biased to performing well on shorter programs due to the distribution of program lengths in the FinQA dataset. Thus, finetuning the program generator can help the model generalize better to complex programs at a slight cost in performance on shorter programs (as indicated by performance of ≥ 4-step programs).

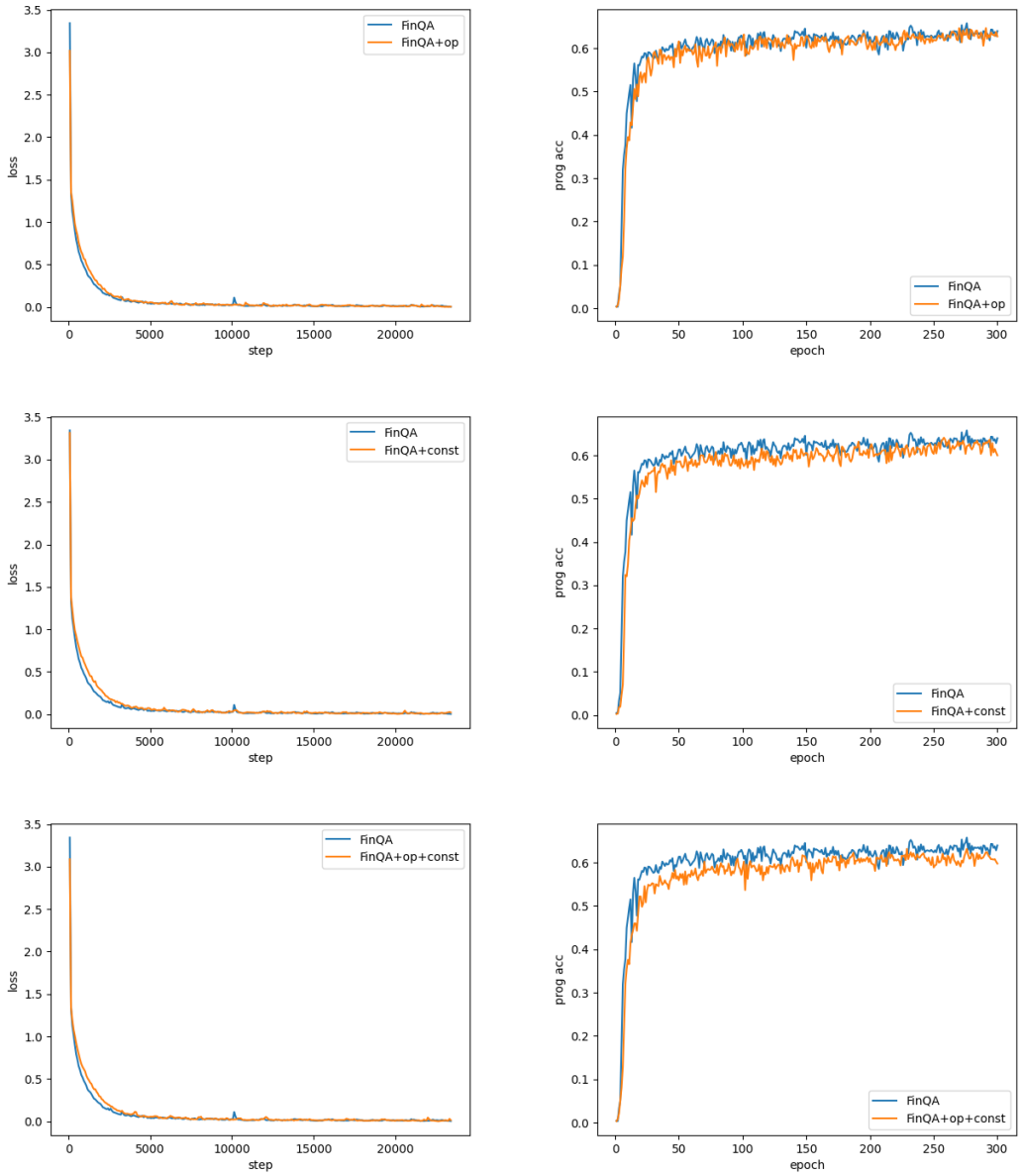

Figure 6: Loss and accuracy curves of finetuned program generators on FinQA on the validation set

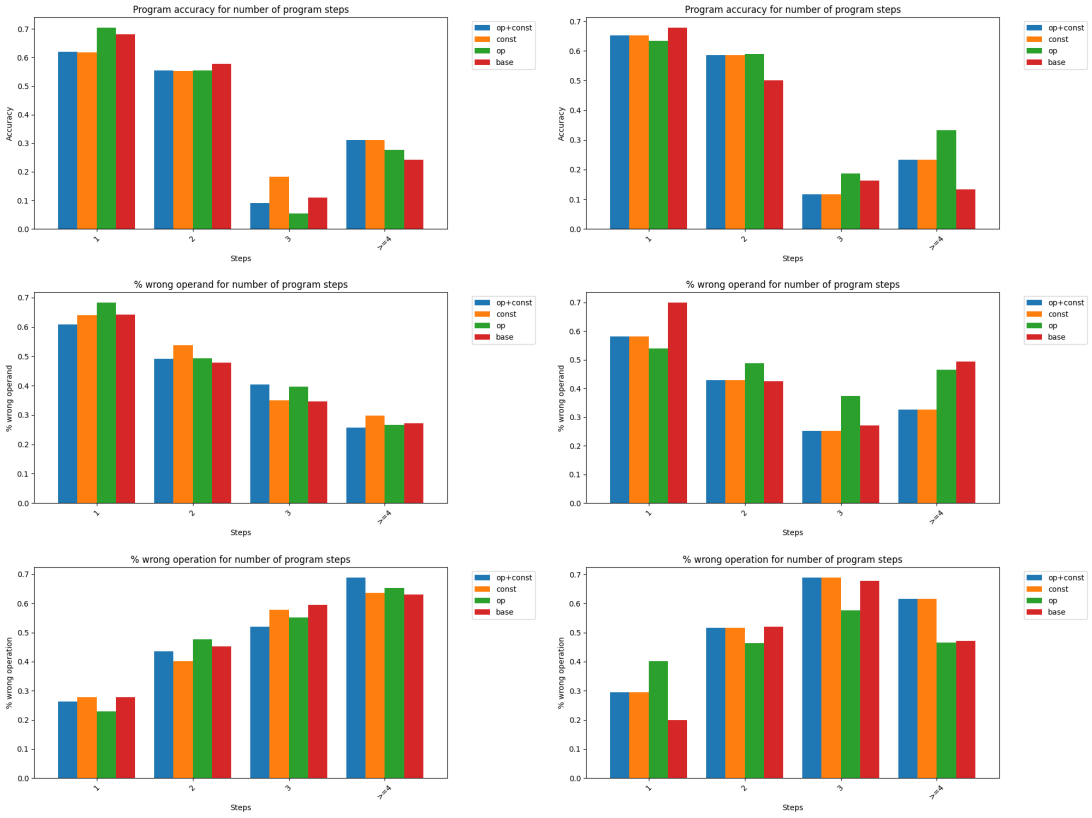

Figure 7: Program accuracy, % wrong operands, and % wrong operations with respect to number of steps of finetuned program generators on the FinQA dev set (right column) and test (left column).