NAPG: Non-Autoregressive Program Generation for Hybrid Tabular-Textual Question Answering

Tengxun Zhang[1]   **Hongfei Xu**[1*] **Josef van Genabith**[2]   **Deyi Xiong**[3]   **Hongying Zan**[1]

[1]Zhengzhou University, Henan, China

[2]DFKI and Saarland University, Informatics Campus, Saarland, Germany

[3]Tianjin University, Tianjin, China

{ztx313,hfxunlp}@foxmail.com, Josef.Van_Genabith@dfki.de,

dyxiong@tju.edu.cn, iehyzan@zzu.edu.cn

Abstract

Hybrid tabular-textual question answering (QA) requires reasoning from heterogeneous information, and the types of reasoning are mainly divided into numerical reasoning and span extraction. Current numerical reasoning methods autoregressively decode program sequences, and each decoding step produces either an operator or an operand. However, the step-by-step decoding suffers from exposure bias, and the accuracy of program generation drops sharply as the decoding steps unfold due to error propagation. In this paper, we propose a non-autoregressive program generation framework, which independently generates complete program tuples containing both operators and operands, can address the error propagation issue while significantly boosting the speed of program generation. Experiments on the ConvFinQA and MultiHiertt datasets show that our non-autoregressive program generation method can bring about substantial improvements over the strong FinQANet (+5.06 Exe Acc and +4.80 Prog Acc points) and MT2Net (+7.97 EM and +6.38 F1 points) baselines, establishing the new state-of-the-art performance, while being much faster (∼21x) in program generation. Finally, with increasing numbers of numerical reasoning steps the performance drop of our method is significantly smaller than that of the baselines. Our code will be publicly available soon.

1 Introduction

Most previous QA studies focus on homogeneous data, such as unstructured text (Hermann et al., 2015; Chen et al., 2017; Yang et al., 2018; Li et al., 2020; Nie et al., 2020) or structured knowledge bases (Yih et al., 2015; Weston et al., 2015; Talmor and Berant, 2018; Zhang et al., 2020b; Zhang and Balog, 2020). In comparison, hybrid tabular-textual QA (Chen et al., 2020a,b; Zhu et al., 2021; Chen et al., 2021; Li et al., 2022; Chen et al., 2022;

---

* Corresponding author.

	EM	F1
1 step	43.62	47.80
2 steps	34.67	37.91
3 steps	22.43	24.57
>3 steps	15.14	17.19
Full Results	36.22	38.43

Table 1: Results of different numerical reasoning steps using MT2Net (RoBERTa-large) on the test set of MultiHiertt.

Zhao et al., 2022) reasons from heterogeneous information and is more challenging as it often requires numerical reasoning to answer the question in addition to span extraction.

To empower hybrid tabular-textual QA models with numerical reasoning ability, TAGOP (Zhu et al., 2021) uses sequence tagging to extract supporting facts, then performs a single arithmetic operation with one of a number of pre-defined operators. To support multi-step reasoning, the encoder-decoder transformer such as T5 (Raffel et al., 2020) or decoder transformer such as GPT-2 (Radford et al., 2019) can be used to autoregressively decode program sequences, but previous work (Chen et al., 2022) has verified that using pre-trained models does not have advantages in numerical reasoning. FinQANet (Chen et al., 2021) and MT2Net (Zhao et al., 2022) use the RoBERTa model (Liu et al., 2019) as the encoder and a specially designed LSTM decoder with structural preservation of the program autoregressively decode program sequences, and each decoding step produces either an operator or an operand, achieving the current state-of-the-art performance on ConvFinQA (Chen et al., 2022) and MultiHiertt (Zhao et al., 2022) datasets, respectively. However, this step-by-step autoregressive decoding process suffers from severe exposure bias. During training, the model uses gold references as decoding history (teacher forcing), and learns to rely on the reference decoding history.

But the decoding history is very likely to be wrong during inference if the model cannot produce high quality predictions, and wrong predictions in early steps may negatively affect subsequent predictions and lead to further errors in following steps (Zhang et al., 2019). Unfortunately, this is the case with hybrid QA, where the prediction performance of current methods are far from good, and as a result of error propagation, program generation accuracy drops heavily as the number of decoding steps increases (Table 1).

Non-autoregressive generation can relieve the dependency on prediction results of earlier steps, addressing the exposure bias issue, while boosting the generation speed with better parallelization. In this paper, we propose a **N**on-**A**utoregressive **P**rogram **G**eneration model (NAPG). Instead on generating the program sequence in a step-by-step manner, we only use the encoder representation (without decoding history), and employ an independent numerical reasoning tuple (operator, operands) generator for each reasoning step to predict the operator and its operands. The numerical reasoning tuple generator contains a soft masking mechanism (Zhang et al., 2020a) to derive its specific input representation from the encoder representation by highlighting the operand representations of the step, followed by an operator generator, an operand generator and an order predictor. We also utilize a length predictor to control the number of numerical reasoning tuples produced. As the numerical reasoning tuple generator does not leverage any previous decoder history steps, our method prevents the generation with the exposure bias problem and greatly improves the generation speed due to parallelization.

Our main contributions are as follows:

- We propose a non-autoregressive program generation model (NAPG), which can generate the full reasoning programs in parallel. Compared to previous autoregressive generators, our method does not suffer from the exposure bias issue and is much faster due to parallelization.

- In our experiments on the ConvFinQA (Chen et al., 2022) and MultiHiertt (Zhao et al., 2022) datasets, the NAPG model can bring about substantial improvements over the strong FinQANet (+5.06 Exe Acc and +4.80 Prog Acc points) and MT2Net (+7.97 EM and +6.38 F1 points) baselines, establishing

... INCENTIVE PLANS Discretionary Annual Incentive Awards Citigroup grants immediate cash bonus payments and ...

In millions of dollars	Cash flow hedges-2	Benefit plans-3	...
...	...	...	...
Balance at December 31, 2017	$-698	$-6,183	...
Change, net of taxes	-30	-74	...
Balance at December 31, 2018	-728	-6257	...
...	...	...	...

The maximum length of time over which forecasted cash flows are hedged is 10 years ...

Maximum potential amount of future payments

In billions of dollars at December 31, 2018, except carrying value in millions	Expire within1 year	Expire after1 year	Total amountout standing	Carrying value(in millions of dollars)
Financial standby letters of credit	$31.80	$65.30	$97.10	$131
Performance guarantees	7.7	4.2	11.9	29

Question: What's the total amount of the Financial standby letters of credit in the years where Benefit plans-3 is less than -6,250 (in million) ?
Program: add(31.8,65.3), add(#0,97.1), add(#1,131)
Answer: 325.2

Figure 1: An example from the MultiHiertt dataset. In the numerical reasoning question, the system needs to locate which year has less than -6,250 Benefit plans-3 from the first table, and then select the relevant numbers from the second hierarchical table as operands to calculate the answer with addition as the operator. Better viewed in color, the supporting facts are in light blue boxes.

the new state-of-the-art performance, while being ∼21 times as fast in program generation. Our further analysis shows that, the performance loss of NAPG is also significantly smaller than the baseline with increasing numbers of numerical reasoning steps.

2 Preliminaries

Task Description Question answering over hybrid tabular textual data requires reasoning from heterogeneous information, involving numerical reasoning or span extraction. As shown in Figure 1, given the question Q, the system is to find its answer from tables T and texts E. For some cases, the model only needs to extract an answer span A from the input. For many other cases involving numerical reasoning, the model has to generate a program sequence $G = \{g_0, g_1, ..., g_n\}$, where g_i stands for the token of the program, which is either extracted from the input, or selected from pre-defined special tokens, including operators and special operands, and the probability of an answer A is calculated by summing over the probabilities
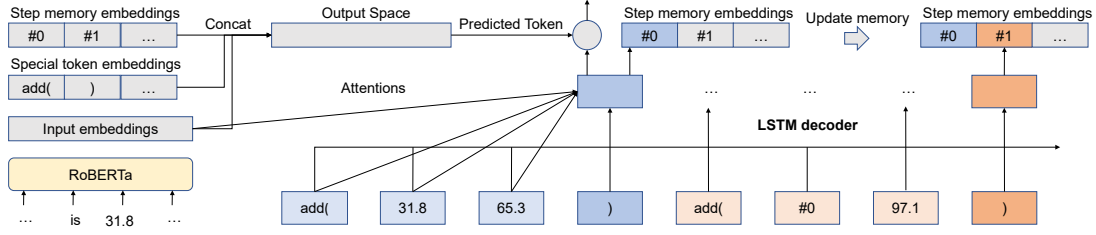
Figure 2: Autoregressive Program Generation for Numerical Reasoning.

of all programs G_i from which the answer A can be obtained:

$$P\left(A|T, E, Q\right) = \sum_i P\left(G_i|T, E, Q\right) \quad (1)$$

Fact Retrieving MT2Net converts data cells of tables into sentences with their row and column headers. Due to the input length limitation of PLMs, MT2Net first concatenates the question with each sentence as input to train a BERT-based binary-classifier (bi-classifier) for supporting fact classification. Next, it takes the top n sentences based on the supporting fact classification prediction as the input for the next stage. Another classifier is used to determine whether the next stage is span extraction or numerical reasoning.

Span Extraction MT2Net uses the T5-base model (Raffel et al., 2020) for span extraction questions, where the model takes the concatenation of the question and the sentences containing supporting facts as input, and generates the answer spans.

Autoregressive Numerical Reasoning MT2Net first uses RoBERTa as an encoder to obtain the context-aware representations of the question and the sentences containing supporting facts, and concatenates them with the embeddings of pre-defined special tokens, such as the function names, predefined constants, etc. Next, it uses an LSTM decoder to generate the program sequence for the deduction of the answer. Each decoding step makes predictions over the concatenated matrix and selects either an operator or an operand, as shown in Figure 2. FinQANet uses the same module for autoregressive numerical reasoning like MT2Net, but it does not use an additional classifier to predict question type during fact retrieving, nor does it have span extraction module.

3 Our Approach

We present our non-autoregressive program generation model, which can generate the full program sequences independently to address the exposure bias issue of the step-by-step autoregressively program generation model, and speed up the generation by supporting better parallelization. The NAPG model framework is shown in Figure 3. It first uses a bi-classifier to retrieve the most relevant facts, and uses another bi-classifier to identify the question type like MT2Net. For span extraction questions, NAPG uses the same T5-base model as MT2Net to generate the answer given the concatenation of the question and the sentences containing supporting facts. But for numerical reasoning, we design a non-autoregressive approach to program generation, which is quite different from the autoregressive LSTM decoder used by MT2Net. Especially, NAPG can also use numerical reasoning modules to solve span extraction questions without the additional question classification and span extraction models.

3.1 Non-Autoregressive Program Generation

Our non-autoregressive program generation is implemented by a length predictor to predict the number of numerical reasoning steps (number of program tuples), a soft-masking operand extractor to identify all operands in the program sequence, and a set of modules for the program generation of all steps, where each reasoning step includes a soft-masking operand generator to select the operands for the operator, an operator generator to predict the operator, and an order predictor to decide the order of the operands. As the program tuples (each containing one operator, two operands and the order of the two operands) of different programming steps are likely to be different, we use the same architectures but independent modules for different steps.

Since operations such as averaging and numerical order conversion may occur during reasoning,
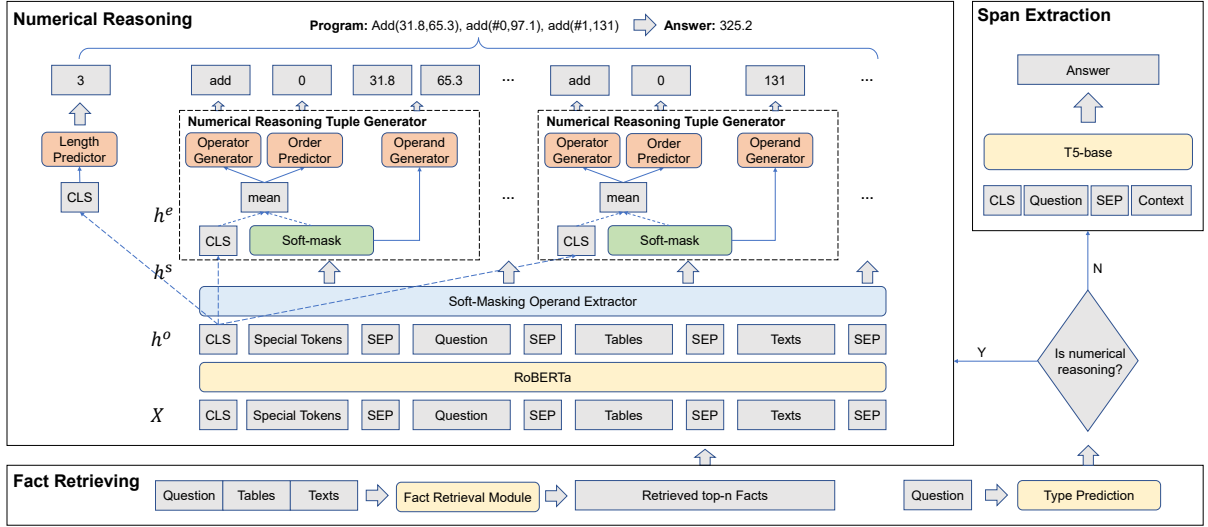
Figure 3: The NAPG Model. For multi-step reasoning, the Soft-Masking Operand Generator is only computed once and extracts all operands of all reasoning steps, there are multiple different Numerical Reasoning Tuple Generators and each Numerical Reasoning Tuple Generator independently picks its operands, operator, order of each reasoning step.

we add constants within 10 and common order values into special tokens following FinQANet and MT2Net, and concatenate special tokens with the question and the sentences containing supporting facts as input to the RoBERTa encoder.

Length Predictor   Numerical reasoning requires a variable number of inference steps, so we employ a multi-class classifier as the length predictor to predict the number of reasoning steps, where each reasoning step contains a complete program tuple (operator, operands, order). The classifier is an FFN layer with the RoBERTa representation of the [CLS] token without soft masking as input.

$$p^{\text{length}} = \text{softmax}\left(\text{FFN}\left([\text{CLS}]\right)\right) \quad (2)$$

where FFN is a 2-layer feed-forward network with GELU (Hendrycks and Gimpel, 2016) as the activation function.

Soft-Masking Operand Extractor   Extracting correct operands is crucial for the inference of the correct answer. We empirically find that extracting **all operands** for the program sequence prior to the Numerical Reasoning Tuple Generator benefits the performance ($ 4.5), and we employ an FFN as the Soft-Masking Operand Extractor layer over the full RoBERTa representation to extracting all operands.

$$p^t = \text{softmax}\left(\text{FFN}\left(\boldsymbol{h}^o\right)\right) \quad (3)$$

where FFN is a 2-layer feed-forward network

with GELU as the activation function. $\boldsymbol{h}^o$ is the RoBERTa representation. p^t represents the probability that the token is an operand.

Then we soft mask (Zhang et al., 2020a) the RoBERTa representation with p^t.

$$\boldsymbol{h}^s = \boldsymbol{h}^o * p^t + \boldsymbol{v}^m * (1 - p^t) \quad (4)$$

where $\boldsymbol{h}^s$ is the soft-masked representation, $\boldsymbol{v}^m$ stands for the mask embedding, and "*" indicates element-wise multiplication.

A large p^t would make the soft masking result close to the original embedding, while a small p^t would turn the result close to the mask embedding. The soft-masking mechanism can thus represent the operands with a higher priority. Compared to using the classification results for hard masking, soft masking is differentiable and can be trained in an end-to-end manner while alleviating the error propagation issue. We use a zero vector with all dimensions set to 0 as the mask embedding, as we empirically find that this works better than the other options (including the embedding of the special [MASK] token) for this task.

Especially, when answering span extraction questions, we predict that the length is 0, and then generate the answer spans from the outputs of the soft masking operand extractor.

Operand Generator   We also utilize the soft masking mechanism to extract the **two operands** of the specific reasoning step from the input in

the Numercical Reasoning Tuple Generator. Compared to the soft-masking Operand Extractor that identifies all operands of the complete program sequences, the Operand Generator only finds the two operands of the specific program step by selecting only two tokens with the highest prediction probabilities from either the numbers in the retrieved sentences or the set of pre-specified tokens representing the computation results of preceding program steps or pre-defined numbers.

$$p^e = \text{softmax}\left(\text{FFN}\left(\boldsymbol{h}^s\right)\right) \tag{5}$$

$$\boldsymbol{h}^e = \boldsymbol{h}^s * p^e + \boldsymbol{v}^m * (1 - p^e) \tag{6}$$

where $\boldsymbol{h}^s$ and $\boldsymbol{h}^e$ are the soft-masked representation from the Operand Extractor and the soft masking result respectively. p^e represents the probability that the token is the operand of the step.

Operator Generator We define six operators following FinQANet and MT2Net: Addition, Subtraction, Multiplication, Division, Exp, Greater. We average the soft-masked representations produced by the operand generator of the reasoning step and the embedding of the [CLS] token as input, and use a multi-classifier as the operator generator to select the operator.

$$p^{\text{op}} = \text{softmax}\left(\text{FFN}\left(\text{mean}\left([\text{CLS}] | \boldsymbol{h}^e\right)\right)\right) \tag{7}$$

Order Predictor The order of the two operands matters when the operator is subtraction, division, exp or greater. We also take the mean pooling of the [CLS] token embedding and the soft-masked embeddings produced by the operand generator of the reasoning step as input, and use a bi-classifier to predict the order of the operands (whether their order is as in the input or in the reverse order).

$$p^{\text{order}} = \text{softmax}\left(\text{FFN}\left(\text{mean}\left([\text{CLS}] | \boldsymbol{h}^e\right)\right)\right) \tag{8}$$

3.2 Training

To jointly optimize all objectives for numerical reasoning, we minimize the weighted sum of the negative log-likelihood losses of individual modules.

$$
\begin{aligned}
\mathcal{L} = & \lambda^t * \text{NLL}\left(\log\left(p^t\right), r^t\right) \\
& + \lambda^{\text{length}} * \text{NLL}\left(\log\left(p^{\text{length}}\right), r^{\text{length}}\right) \\
& + \lambda^e * \sum_{i=0}^{n} \text{NLL}\left(\log\left(p_i^e\right), r_i^e\right) \\
& + \lambda^{\text{op}} * \sum_{i=0}^{n} \text{NLL}\left(\log\left(p_i^{\text{op}}\right), r_i^{\text{op}}\right) \\
& + \lambda^{\text{order}} * \sum_{i=0}^{n} \text{NLL}\left(\log\left(p_i^{\text{order}}\right), r_i^{\text{order}}\right)
\end{aligned}
\tag{9}
$$

where NLL stands for the negative log-likelihood loss function, r indicates the ground truths, λ represents the weight of each module, and n is the maximum number of reasoning steps.

3.3 Discussions

The general design of NAPG only involves element-wise computations, the single FFN for length prediction, and 3 sets of FFNs with different parameters but the same architecture for operand generation, operation generation and order prediction respectively. When generating the program tuple sequence, the length predictor only needs to be computed once, and the element-wise computations can be easily parallelized. As for each set of FFNs with different parameters but the same architecture, their activation function can be easily parallelized, and their linear layers with different parameters can be parallelized with the batch matrix-matrix multiplication function implemented in almost all modern linear algebra libraries.

Compared to autoregressive decoding, non-autoregressive counterparts ignore the decoding history which may have a potential negative impact on its coherence, but in case of program generation for numerical reasoning, coherence may be affected less, while **the autoregressive decoding is very likely to be mislead especially for program generation given the prediction quality is not high** ($< 50\%$**, as shown in Figure 4).**

4 Experiments

4.1 Settings

Dataset We conducted our experiments on the ConvFinQA (Chen et al., 2022) and MultiHiertt (Zhao et al., 2022) datasets. The ConvFinQA dataset poses a great challenge in modeling long-range, complex numerical reasoning paths in real-world conversations (Chen et al., 2022). Compared with existing datasets, each document in MultiHiertt contains multiple hierarchical tables and longer unstructured text. A more complex reasoning process across multiple tables and paragraphs is required to correctly answer questions (Zhao et al., 2022).

Dataset	Avg. # text words	Table types	Avg. # tables	Turn	# Train	# Dev	# Test
ConvFinQA	675.61	Flat	1	Multi	11,104	1,490	1,521
MultiHiertt	1,645.9	Hierarchical	3.89	Single	7,830	1,044	1,566

Table 2: Statistics of ConvFinQA and MultiHiertt datasets.

The test set labels of the ConvFinQA and Multi-Hiertt datasets are not public. Table 2 shows that the ConvFinQA dataset poses great challenge in modeling long-range, complex numerical reasoning paths in real-world conversations. Compared to ConvFinQA, each document in MultiHiertt contains more hierarchical tables and longer unstructured text, so it is more difficult to correctly answer the question.

Evaluation Metrics We evaluated the performance by Exact Matching (EM) and the adopted numeracy-focused F1 (Dua et al., 2019) for MultiHiertt, and execution accuracy (Exe Acc) and program accuracy (Prog Acc) for ConvFinQA following previous work. Exe Acc calculates the accuracy of the execution results of the prediction program. Prog Acc calculates the accuracy of the prediction program (both operators and operands) with the ground program, which is stricter because there may be multiple different programs that can execution the same results.

Baselines GPT-2 (Radford et al., 2019) and T5 (Raffel et al., 2020) are two generative models. TAGOP (Zhu et al., 2021) first uses the sequence tagging method to extract facts, then performs only one arithmetic operation with pre-defined operators. FinQANet (Chen et al., 2021) and MT2Net (Zhao et al., 2022) are able to perform multi-step reasoning, and they both use an autoregressive LSTM decoder to generate the program.

Model Settings We tuned hyper-parameters on the development set (§ 4.7). To fairly compare with existing state-of-the-art results, GPT-2 and T5 use medium and large respectively, the rest of baselines are based on the RoBERTa-large model. To focus on program generation and for fair comparison, we only replace the program generation module of MT2Net with NAPG and leave the other parts unchanged. As FinQANet uses a single LSTM to decode either the program sequence or the span to extract according to question type without having an individual span extraction module, we ask the length predictor of NAPG to predict a length of 0 and extract the span with the highest predic-

	Exe Acc	Prog Acc
GPT-2 (medium)	58.19	57.00
T5 (large)	58.66	57.05
FinQANet (RoBERTa-large)	68.90	68.24
Ours (RoBERTa-base)	69.82	68.84
Ours (RoBERTa-large)	**73.96**	**73.04**

Table 3: Main results on ConvFinQA.

	EM	F1
TAGOP (RoBERTa-large)	17.81	19.35
FinQANet (RoBERTa-large)	31.72	33.60
MT2Net (RoBERTa-large)	36.22	38.43
Ours (RoBERTa-base)	38.19	38.81
Ours (RoBERTa-large)	**44.19**	**44.81**

Table 4: Main results on MultiHiertt.

tion probability directly from the output of the operand extractor in this case to take care of span extraction questions on the ConvFinQA dataset. To fairly compare with existing state-of-the-art results, we used the same hyper-parameters as the SotA model. We train NAPG (RoBERTa-large) with batch size of 6 on a single RTX3090 GPU on both ConvFinQA and MultiHiertt datasets. We set the maximum number of reasoning steps n to 5, 10 for ConvFinQA and MultiHiertt respectively. It takes about 12 hours to train the numerical reasoning model of NAPG (RoBERTa-large) on MultiHiertt, and it takes about 10 hours to train the numerical reasoning model of NAPG (RoBERTa-large) on ConvFinQA. For base model, λ^{op} was set to 2, λ^{order} was set to 1.5, λ^{t} was set to 1.2, λ^{length} was set to 1.1, and λ^{e} was set to 1 for ConvFinQA, λ^{op} was set to 2, the others were all set to 1 for MultiHiertt. For large model, λ^{op} was set to 2, and λ^{order} was set to 1.5, the others were set to 1 for both ConvFinQA and MultiHiertt datasets.

4.2 Main Results

We first compare NAPG (with both base and large settings) with our baselines. Results are shown in Tables 3 and 4.

Tables 3 and 4 show that: 1) using pre-trained models for numerical reasoning program gener-

| | Dev | |
	EM	F1
MT2Net (RoBERTa-large)	41.35	41.35
Ours (RoBERTa-large)	**48.20**	**48.20**

Table 5: Main results of numerical reasoning on Multi-Hiertt.

	Dev EM/F1	Test EM/F1
MT2Net (RoBERTa-base)	35.69/37.81	34.32/36.17
NAPG (RoBERTa-base)	39.27/40.21	38.19/38.81
NAPG* (RoBERTa-base)	**41.37/42.24**	**38.44/39.58**
MT2Net (RoBERTa-large)	37.05/39.96	36.22/38.43
NAPG (RoBERTa-large)	45.79/46.15	44.19/44.81
NAPG* (RoBERTa-large)	**45.88/47.27**	**45.59/47.04**

Table 6: Results of NAPG and NAPG* on MultiHiertt.

	EM	F1
MT2Net	36.22	38.43
+ Soft Masking Operand Extractor	41.51	41.88
NAPG	**44.19**	**44.81**

Table 7: Performance of Soft Masking Operand Extractor on Multihiertt.

ation does not lead to better performance than LSTM. 2) already with the RoBERTa base setting, our method is able to achieve better performances on both datasets. 3) using the RoBERTa large setting can further boost the performance of NAPG, and lead to large improvements on both ConvFinQA (+5.06/+4.80 Exe/Prog Acc points) and MultiHiertt (+7.97/+6.38 EM/F1 points) datasets.

As our approach only modifies the program generation part, we also tested the performance of NAPG and MT2Net on all numerical reasoning questions in the development set of MultiHiertt. Results are shown in Table 5.

Table 5 shows that NAPG can lead to large improvements over the MT2Net baseline (+6.85 EM and F1) in numerical reasoning.

4.3 Performance w.r.t. Reasoning Steps

To verify whether NAPG can really address the error accumulation issue of autoregressive program generation, we analyze the performance of NAPG and MT2Net w.r.t. different numbers of reasoning steps. For fairness, we used RoBERTa-large as the encoder of both NAPG and MT2Net. As the test set is not publicly available, our analysis is performed on the development set of MultiHiertt and the results of MT2Net are from Zhao et al. (2022). Results are shown in Figure 4.

Figure 4 shows that: 1) despite the metrics reporting highest scores with 2 reasoning steps, the general performance trend is descending while increasing the number of reasoning steps. 2) our NAPG approach outperforms the MT2Net in all aspects by a large margin. 3) as the number of reasoning steps increases, the improvements of our method are much larger over the autoregressive MT2Net baseline (+11.41/+11.67 EM/F1 when the number of reasoning steps is 3 and +14.42/+14.43 EM/F1 when it is larger than 3).

The performance drop with increased numbers of reasoning steps with our non-autoregressive method is much smaller than the autoregressive MT2Net, showing the advantage of NAPG in handling questions that require inference with long program sequences. Intuitively, in the generation of longer program sequences, the autoregressive model is more likely to suffer from exposure bias, while the non-autoregressive generation prevents our method from suffering from this issue.

4.4 Performance of NAPG without Span Extraction Model

To focus on program generation and for fair comparison, we only replace the program generation module of MT2Net with NAPG and leave the other parts unchanged. But NAPG can also use numerical reasoning modules to solve span extraction questions without the additional question classification and span extraction models (we call it as NAPG*). We tested the performance of NAPG and NAPG* using the same parameters on the MultiHiertt dataset. Results are shown in Table 6.

Table 6 shows that NAPG* is able to achieve better performances on both base and large settings, perhaps because there are fewer span extraction questions in the MultiHiertt dataset (1524/7830 on the train set), using the two types of problems to train NAPG* model together will enable the model to learn more knowledge.

4.5 Performance of Soft Masking Operand Extractor

To test the effectiveness of the soft masking operand extraction and non-autoregressive decoding separately, we also add the Soft-Masking Operand Extraction module into the original MT2Net with the RoBERTa-large setting. Results are shown in Table 7.
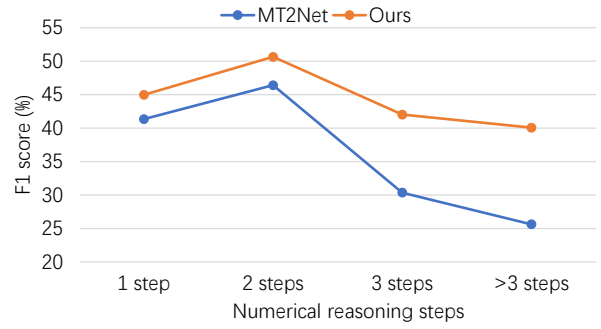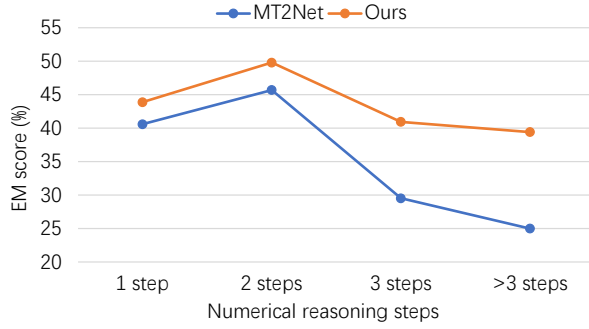
Figure 4: Performance of different numerical reasoning steps on the development set of MultiHiertt.

	MT2Net EM/F1	Ours EM/F1
text-only questions	34.78/34.78	**46.96/46.96**
table-only questions	40.83/41.95	**46.75/47.98**
with ≥ 2 tables	75.86/75.86	**86.21/86.21**
table-text questions	40.27/40.94	**45.01/45.96**
with ≥ 2 tables	71.70/71.70	**72.64/72.64**
Full Results	39.85/40.59	**45.79/46.72**

Table 8: Performance of different question types on the development set of MultiHiertt.

					Dev base EM/F1	large EM/F1
λ^t	λ^l	λ^e	λ^{op}	λ^{od}		
1	1	1	1	1	38.60/39.54	44.35/45.29
2					37.84/38.77	**44.92/45.86**
	2				37.45/38.39	**44.64/45.57**
		2			37.93/38.87	42.72/43.66
			2		**39.27/40.21**	43.77/44.71
				2	**39.18/40.11**	43.87/44.81
2	1.5				37.55/38.49	45.21/46.15
			2	1.5	**37.93/38.87**	**45.79/46.72**

Table 9: Effects of different weights of each module. λ^l and λ^{od} stand for λ^{length} and λ^{order} respectively in Eq. 9.

Model	Time (s)	Speed-up
LSTM	168.86	1x
Ours	8.04	21x

Table 10: Time costs for program generation.

Table 7 shows that the Soft-Masking Operand Extraction module can benefit the performance of MT2Net, and our non-autoregressive decoding method can lead to further improvements.

4.6 Performance Analysis w.r.t. Question Type

We tested the ability of NAPG and MT2Net in handling different sources of supporting facts with the RoBERTa-large setting. Results are shown in Table 8.

Table 8 shows that NAPG outperforms the strong MT2Net baseline in all evaluations. Text-only questions are not span extraction but all of them are numerical reasoning questions in the MultiHiertt dataset, our NAPG method greatly improves the performance of numerical reasoning and leads to such substantial improvements (+12.18 EM and F1). Given that table-only questions normally require numerical reasoning, and that the number of reasoning steps positively co-relates with the number of tables, the large improvements (+10.35 EM and F1) over the strong baseline (75.86 EM and F1) for table-only questions with no less than 2 tables confirm the advantage of NAPG in the numerical reasoning of complex questions.

4.7 Ablation Study of Hyper-Parameters

To study the effects of different components in NAPG on the performance, we explored a number of hyper-parameter values for the combination of training losses (Eq. 9) on the development set of MultiHiertt. Specifically, we first increase only one of all hyper-parameters to 2 while keeping the others set to 1 in each experiment, and then increase all hyper-parameters that lead to improvements for either the base setting or the large setting together, while assigning the hyper-parameter that leads to more improvements a larger value. Results are shown in Table 9.

Table 9 shows that the best performing settings are different with different model settings. The best setting among all tested cases for the base setting is to use a λ^{op} of 2 while setting the others to 1, and for the large setting is to use a λ^{op} of 2, a λ^{order} of 1.5 while setting the others to 1.

Type I	**Question:** How much did the company 2019s valuation allowance decrease from 2011 to 2012? **Reference:** -0.09542 subtract(19520,21579), divide(#0,21579) **MT2Net:** -0.10548 ['subtract(', '19520.0', '21579.0', ')', 'divide(', '#0', '19520.0', ')', 'EOF'] **Ours:** -0.09542 ['subtract(', '19520.0', '21579.0', ')', 'divide(', '#0', '21579.0', ')']
TypeII	**Question:** What's the total amount of the U.S. dollars sold for Pounds sterling in the years where U.S. dollars sold for Pounds sterling is greater than 1? **Reference:** 735.0 add(390,268), add(#0,77) **MT2Net:** 658.0 ['add(', '390.0', '268.0', ')', 'EOF'] **Ours:** 735.0 ['add(', '390.0', '268.0', ')', 'add(', '#0', '77.0', ')']
Type III	**Question:** How much of profit before taxes is there in total (in 2017) without U.S. tax reform impact and Gain on sale of equity investment? (in million) **Reference:** 5639.0 add(4082,1256), add(#0,301) **MT2Net:** 5554.0 ['add(', '4082.0', '1256.0', ')', 'add(', '#0', '301.0', ')', 'subtract(', '#1', '85.0', ')', 'EOF'] **Ours:** 5639.0 ['add(', '4082.0', '1256.0', ')', 'add(', '#0', '301.0', ')']
Type IV	**Question:** What's the average of the Fuel for Amount in the years where Wheelabrator is positive? **Reference:** 626.66667 add(603,649), add(#0,628), divide(#1,3) **MT2Net:** 626.0 ['add(', '603.0', '649.0', ')', 'divide(', '#0', 'const_2', ')', 'EOF'] **Ours:** 626.66667 ['add(', '603.0', '649.0', ')', 'add(', '#0', '628.0', ')', 'divide(', '#1', '3.0', ')']
Type V	**Question:** What's the total amount of U.S. large cap stocks , U.S. small cap stocks, Non-U.S. large cap stocks and Non-U.S. small cap stocks in 2013? (in million) **Reference:** 273.0 add(140,56), add(#0,56), add(#1,21) **MT2Net:** 322.0 ['add(', '140.0', '56.0', ')', 'add(', '#0', '21.0', ')', 'add(', '#1', '21.0', ')', 'add(', '#2', '21.0', ')', 'add(', '#3', '21.0', ' **Ours:** 273.0 ['add(', '140.0', '56.0', ')', 'add(', '#0', '56.0', ')', 'add(', '#1', '21.0', ')']

Table 11: Case study. Type I: MT2Net produces the correct program length but takes a wrong operand. Type II: MT2Net under-generates the program sequence. Type III: MT2Net over-generates the program sequence. Type IV: MT2Net selects the wrong operator and operands and ends the decoding early. Type V: MT2Net takes the wrong operand and is stuck in the loop.

4.8 Program Generation Speed Analysis

Non-autoregressive program generation allows our approach to benefit from parallelization. We compared the program generation speed of NAPG and the LSTM decoder of MT2Net by recording the time costs of the program generation modules on all numerical reasoning questions in the training set of MultiHiertt. Results are shown in Table 10.

Table 10 shows that NAPG is 21 times as fast as MT2Net, showing the substantial advantage of non-autoregressive decoding over autoregressive decoding in terms of speed due to parallelization. And we also test the RoBERTa consumes 67.95 seconds for one epoch on the training set.

4.9 Case Study

We manually inspect a few samples of MT2Net and our approach from the development set for a case study. Results are shown in Table 11.

Table 11 shows that our method can produce a more accurate program than MT2Net when the

number of reasoning steps is correctly predicted (Type I). NAPG can predict the number of reasoning steps correctly (Type II and Type III), and generate the correct program sequences for the interpreting of the answers (Type IV and Type V) when MT2Net fails. We conjecture that the good performance of NAPG might benefit from the weighted combination of individual loss functions (Eq. 9), which allows us to tune the model for specific goals (as shown in Table 9), while the sequence generation of MT2Net fully relies on the single token prediction loss.

5 Related Work

Hybrid tabular-textual QA Chen et al. (2020b) present the first hybrid tabular-textual QA dataset, HybridQA, by linking table cells to Wiki pages via hyperlinks manually, and the answer is usually a span or paragraphs obtained from heterogeneous information. Zhu et al. (2021) and Chen et al. (2021) present TAT-QA and FinQA based on

financial reports, which require numerical reasoning. TAT-HQA (Li et al., 2022) and ConvFinQA (Chen et al., 2022) are extensions of these two datasets respectively. Zhao et al. (2022) present the MultiHiertt dataset, which contains multiple hierarchical tables and longer unstructured text. A more complex reasoning process across multiple tables and paragraphs is required to correctly answer the question.

Numerical Reasoning   Numerical reasoning ability is very important for many NLP tasks (Thawani et al., 2021; Pal and Baral, 2021), especially in QA, such as text QA (Dua et al., 2019; Ran et al., 2019; Hu et al., 2019; Shao et al., 2021; Guo et al., 2021; Zhang et al., 2021), table QA (Iyyer et al., 2017; Herzig et al., 2020; Katsis et al., 2021; Liu et al., 2021; Yang et al., 2022; Pan et al., 2022), and hybrid tabular-textual QA (Chen et al., 2021; Zhu et al., 2021; Li et al., 2021, 2022; Zhao et al., 2022; Chen et al., 2022; Deng et al., 2022). Geva et al. (2020); Berg-Kirkpatrick and Spokoyny (2020); Pi et al. (2022) attempt to inject numerical reasoning ability into pre-trained language models. Zhu et al. (2021); Li et al. (2022); Zhu et al. (2022) perform a single arithmetic operation based on predefined operators. The encoder-decoder transformer such as T5 (Raffel et al., 2020) or decoder transformer such as GPT-2 (Radford et al., 2019) can be used to autoregressively decode program sequences, but previous work (Chen et al., 2022) has verified that they do not have advantages in numerical reasoning. FinQANet (Chen et al., 2021) and MT2Net (Zhao et al., 2022) can perform better for multi-step reasoning, both of them use the LSTM decoder to autoregressively generate the program.

6   Conclusion

Hybrid tabular-textual question answering (QA) requires reasoning from heterogeneous information, and numerical reasoning is its key challenge compared to extractive QA. To address the severe exposure bias issue of current autoregressive methods when program generation performance is far from good, we present a non-autoregressive program generation (NAPG) framework for numerical reasoning, which facilitates program generation in parallel. Our framework independently generates complete program tuples containing both the operator and its operands. Compared to previous autoregressive decoding methods, NAPG does not suffer from exposure bias, and can significantly boost program generation speed.

Our experiments on the ConvFinQA and Multi-Hiertt datasets show that: 1) our proposed model can bring about large improvements over the strong FinQANet ($+5.06/+4.80$ Exe/Prog Acc points) and MT2Net ($+7.97/+6.38$ EM/F1 points) baselines, establishing the new state-of-the-art performance, while being much faster ($\sim$21x) in program generation. 2) the performance drop of our method is also significantly smaller than the autoregressive LSTM decoder of MT2Net with increasing numbers of numerical reasoning steps.

Limitations

We believe that this paper provides useful insights for the numerical reasoning part in hybrid tabular-textual QA. However, there are limitations to our work:

- We only study the program generation of MT2Net and FinQANet while keeping the other parts unchanged. That said, the numerical reasoning is the main challenge of the hybrid textual-tabular QA task compared to traditional extractive QA studies, and how to address the exposure bias issue of program generation is the main focus of this work.

- Our experiments are mainly conducted on the MultiHiertt dataset even though we also tested the effectiveness on the ConvFinQA dataset. However, MultiHiertt is the more challenging compared to the other benchmarks (Zhao et al., 2022), which does not involve either multiple tables and longer texts (Zhu et al., 2021; Chen et al., 2021; Li et al., 2022) or multi-step numerical reasoning (Chen et al., 2020b; Li et al., 2021).

References

Taylor Berg-Kirkpatrick and Daniel Spokoyny. 2020. An empirical investigation of contextualized number prediction. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4754–4764.

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. 2017. Reading wikipedia to answer open-domain questions. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1870–1879.

Wenhu Chen, Ming-Wei Chang, Eva Schlinger, William Wang, and William W Cohen. 2020a. Open question answering over tables and text. *arXiv preprint arXiv:2010.10439*.

Wenhu Chen, Hanwen Zha, Zhiyu Chen, Wenhan Xiong, Hong Wang, and William Yang Wang. 2020b. 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.

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*.

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. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 6279–6292, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Yang Deng, Wenqiang Lei, Wenxuan Zhang, Wai Lam, and Tat-Seng Chua. 2022. PACIFIC: Towards proactive conversational question answering over tabular and textual data in finance. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 6970–6984, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

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

Mor Geva, Ankit Gupta, and Jonathan Berant. 2020. Injecting numerical reasoning skills into language models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 946–958.

Xiao-Yu Guo, Yuan-Fang Li, and Gholamreza Haffari. 2021. Improving numerical reasoning skills in the modular approach for complex question answering on text. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 2713–2718.

Dan Hendrycks and Kevin Gimpel. 2016. Bridging nonlinearities and stochastic regularizers with gaussian error linear units. *CoRR, abs/1606.08415*, 3.

Karl Moritz Hermann, Tomáš Kočiskỳ, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching machines to read and comprehend. In *Proceedings of the 28th International Conference on Neural Information Processing Systems-Volume 1*, pages 1693–1701.

Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Mueller, 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.

Minghao Hu, Yuxing Peng, Zhen Huang, and Dongsheng Li. 2019. A multi-type multi-span network for reading comprehension that requires discrete reasoning. 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 1596–1606.

Mohit Iyyer, Wen-tau Yih, and Ming-Wei Chang. 2017. Search-based neural structured learning for sequential question answering. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1821–1831.

Yannis Katsis, Saneem Chemmengath, Vishwajeet Kumar, Samarth Bharadwaj, Mustafa Canim, Michael Glass, Alfio Gliozzo, Feifei Pan, Jaydeep Sen, Karthik Sankaranarayanan, et al. 2021. Ait-qa: Question answering dataset over complex tables in the airline industry. *arXiv preprint arXiv:2106.12944*.

Jiaqi Li, Ming Liu, Min-Yen Kan, Zihao Zheng, Zekun Wang, Wenqiang Lei, Ting Liu, and Bing Qin. 2020. Molweni: A challenge multiparty dialogues-based machine reading comprehension dataset with discourse structure. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 2642–2652.

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.

Xiao Li, Yawei Sun, and Gong Cheng. 2021. Tsqa: tabular scenario based question answering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 13297–13305.

Qian Liu, Bei Chen, Jiaqi Guo, Morteza Ziyadi, Zeqi Lin, Weizhu Chen, and Jian-Guang Lou. 2021. Tapex: Table pre-training via learning a neural sql executor. In *International Conference on Learning Representations*.

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 preprint arXiv:1907.11692*.

Liqiang Nie, Yongqi Li, Fuli Feng, Xuemeng Song, Meng Wang, and Yinglong Wang. 2020. Large-scale question tagging via joint question-topic embedding learning. *ACM Transactions on Information Systems (TOIS)*, 38(2):1–23.

Kuntal Kumar Pal and Chitta Baral. 2021. Investigating numeracy learning ability of a text-to-text transfer model. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 3095–3101.

Feifei Pan, Mustafa Canim, Michael Glass, Alfio Gliozzo, and James Hendler. 2022. End-to-end table question answering via retrieval-augmented generation. *arXiv preprint arXiv:2203.16714*.

Xinyu Pi, Qian Liu, Bei Chen, Morteza Ziyadi, Zeqi Lin, Yan Gao, Qiang Fu, Jian-Guang Lou, and Weizhu Chen. 2022. Reasoning like program executors. *arXiv preprint arXiv:2201.11473*.

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, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551.

Qiu Ran, Yankai Lin, Peng Li, Jie Zhou, and Zhiyuan Liu. 2019. Numnet: Machine reading comprehension with numerical reasoning. 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 2474–2484.

Zhihong Shao, Lifeng Shang, Qun Liu, and Minlie Huang. 2021. A mutual information maximization approach for the spurious solution problem in weakly supervised question answering. 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 4111–4124.

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

Avijit Thawani, Jay Pujara, Filip Ilievski, and Pedro Szekely. 2021. Representing numbers in nlp: a survey and a vision. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 644–656.

Jason Weston, Antoine Bordes, Sumit Chopra, Alexander M Rush, Bart Van Merriënboer, Armand Joulin, and Tomas Mikolov. 2015. Towards ai-complete question answering: A set of prerequisite toy tasks. *arXiv preprint arXiv:1502.05698*.

Jingfeng Yang, Aditya Gupta, Shyam Upadhyay, Luheng He, Rahul Goel, and Shachi Paul. 2022. Tableformer: Robust transformer modeling for table-text encoding. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 528–537.

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

Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. 2015. Semantic parsing via staged query graph generation: Question answering with knowledge base. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1321–1331.

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.

Shaohua Zhang, Haoran Huang, Jicong Liu, and Hang Li. 2020a. Spelling error correction with soft-masked bert. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 882–890.

Shuo Zhang and Krisztian Balog. 2020. Web table extraction, retrieval, and augmentation: A survey. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 11(2):1–35.

Shuo Zhang, Zhuyun Dai, Krisztian Balog, and Jamie Callan. 2020b. Summarizing and exploring tabular data in conversational search. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1537–1540.

Wen Zhang, Yang Feng, Fandong Meng, Di You, and Qun Liu. 2019. Bridging the gap between training and inference for neural machine translation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4334–4343, Florence, Italy. 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.

Fengbin Zhu, Wenqiang Lei, Fuli Feng, Chao Wang, Haozhou Zhang, and Tat-Seng Chua. 2022. Towards complex document understanding by discrete reasoning. In *Proceedings of the 30th ACM International Conference on Multimedia*, pages 4857–4866.

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.