NAPG: NON-AUTOREGRESSIVE PROGRAM GENERATION FOR HYBRID TABULAR-TEXTUAL QUESTION ANSWERING

Anonymous authors
Paper under double-blind review

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. The current numerical reasoning method uses LSTM to 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 with progressive decoding. In this paper, we propose a non-autoregressive program generation framework, which facilitates program generation in parallel. Our framework, which independently generates complete program tuples containing both operators and operands, can significantly boost the speed of program generation while addressing the error accumulation issue. Our experiments on the MultiHiertt dataset shows that our model can bring about large improvements ($+7.97$ EM and $+6.38$ F1 points) over the strong baseline, establishing the new state-of-the-art performance, while being much faster ($\sim$21x) in program generation. The performance drop of our method is also significantly smaller than the baseline with increasing numbers of numerical reasoning steps.

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 & Berant, 2018; Zhang et al., 2020b; Zhang & Balog, 2020). In comparison, hybrid QA (Chen et al., 2020a;b; Zhu et al., 2021; Chen et al., 2021) reasons from heterogeneous information (such as texts and tables) and is more challenging as it sometimes requires numerical calculation to answer the question in addition to extracting information. The numerical inference procedure is usually implemented by a program generator that generates the programming sequence and an interpreter to execute the program. Compared with existing benchmarks (Chen et al., 2020b; Zhu et al., 2021; Katsis et al., 2021; Chen et al., 2021), the MultiHiertt (Zhao et al., 2022) dataset is based on a wealth of financial reports, its questions may involve multiple hierarchical tables and longer texts, and the reasoning process required to answer the question is more complex.

MT2Net (Zhao et al., 2022) obtains the state-of-the-art performance on the MultiHiertt dataset. It first converts data cells of tables into sentences with their row and column headers. As pre-trained LMs cannot handle very long input sequences, MT2Net uses pre-trained LMs like BERT (Devlin et al., 2019) to compare the question with each sentence to identify the existence of supporting facts. Another classifier is employed to distinguish between questions that require numerical reasoning or span extraction. For numerical reasoning, MT2Net uses RoBERTa (Liu et al., 2019) as the encoder and an LSTM as the decoder to gradually generate the program, where each decoding step produces either an operator or an operand. However, this step-by-step autoregressive decoding process suffers from exposure bias. During training, the model uses gold references as decoding history, but the decoding history might be wrong during inference, and wrong predictions in early steps may lead to further errors in following steps. As a result of error accumulation, program generation accuracy drops heavily with decoding.

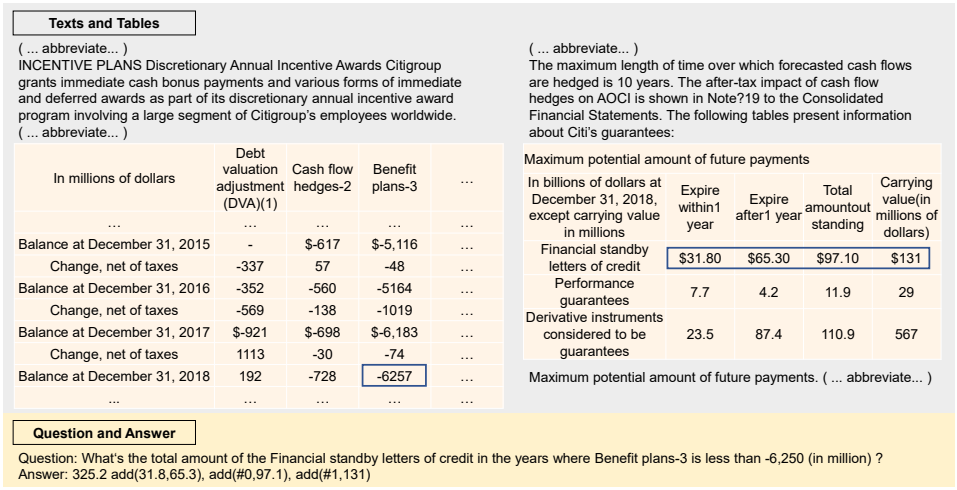

Figure 1: An example of the MultiHiertt dataset. In the numerical reasoning question, the system needs to locate which year has less than -6,250 (in million) Benefit plans-3 from the first table, and then select the relevant numbers from the second hierarchical table as operands to calculate the answer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

To address the exposure bias issue with the autoregressive program generation, and to accelerate the reasoning benefiting from better parallelization with independent decoding, in this paper, we propose a **N**on-**A**utoregressive **P**rogram **G**eneration model (NAPG). Following the design of MT2Net, we also first use the fact retrieval module to obtain the most relevant supporting facts, and then send them to the span extraction module and the numerical reasoning module according to the type of the question. But for numerical reasoning, we first use RoBERTa as an encoder to extract all operands for the program sequence, and soft-mask the encoding representation according to the prediction probability that the token is an operand. Next, we generate the full reasoning program in parallel with a number of generators and the soft-masked representation. Each generator independently generates a complete program tuple including both the operator and its operands at the fixed position. A length predictor is introduced to predict the length of the program. As the program generators do not leverage the prediction of the other generators, the program generations of different steps are independent. This can prevent the generation from the exposure bias problem and greatly improve 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 the previous autoregressive generator, our method does not suffer from the exposure bias issue and is much faster due to parallelization.
- In our experiments on the MultiHiertt dataset, the NAPG model can bring about huge improvements ($+7.97$ EM and $+6.38$ F1 points) over the state-of-the-art MT2Net model while being $\sim$21 times as fast in program generation. Our further analysis shows that, the performance loss of NAPG is also significantly smaller than our 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. While 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-

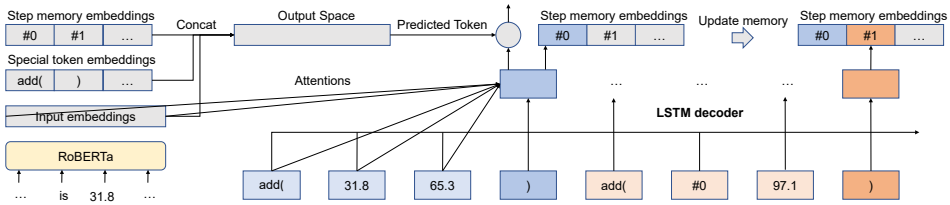

Figure 2: Autoregressive Program Generation for Numerical Reasoning.

defined special tokens, including operators and special operands, and the probability of an answer A is calculated by summing over the probabilities of all programs G_i that compute A:

$$P\left(A|T,E,Q\right)=\sum_{i}P\left(G_i|T,E,Q\right) \tag{1}$$

Fact Retrieving MT2Net converts data cells of tables into sentences with their row and column headers. As the input text in a document of the MultiHiertt dataset may exceed 3,000 tokens and 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 sequence.

Autoregressive Numerical Reasoning Answering the question may require multi-step 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. 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. Since programs are generated autoregressively, their generation suffers from exposure bias: during training, it takes the gold references for step-by-step decoding ("teacher forcing"), but during inference, the predictions of previous steps are used and may mislead the generation of following steps. This leads to error accumulation, and the performance of the MT2Net drops rapidly in practice as the number of inference steps increases.

3 Our Approach

We present the non-autoregressive program generation model that independently generates the full program sequence to address the exposure bias issue of the step-by-step program generation model and to 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 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.

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, 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

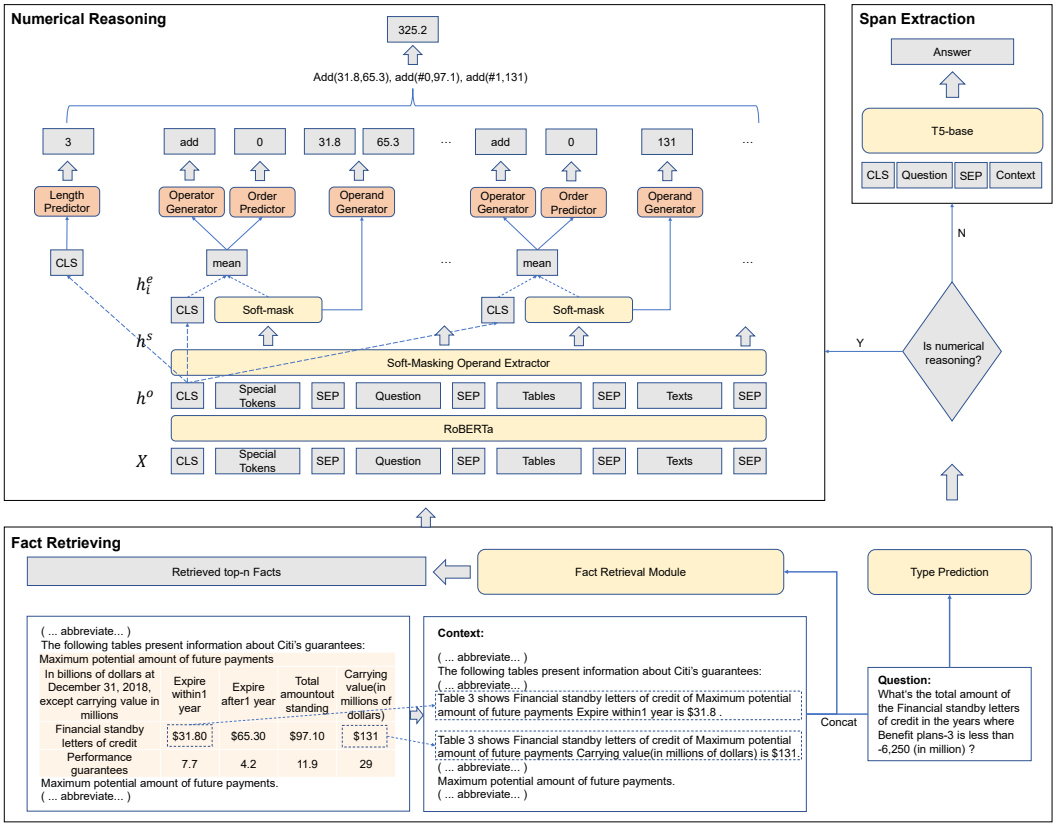

Figure 3: The NAPG Model.

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 and its 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.

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 (including the operator and its operands). The classifier is an FFN layer with the RoBERTa representation of the [CLS] token without soft masking as input.

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

Soft-Masking Operand Extractor Extracting correct operands is crucial for the inference of the correct answer. We empirically find that extracting operands of the full program sequence benefits the performance, and employ a soft-masking operand extractor layer before extracting operands for each reasoning step.

Since operations such as averaging and numerical order conversion may occur during reasoning, we add constants within 10 and common order values into special tokens following MT2Net, and concatenate special tokens with the question and the sentences containing supporting facts as input to the RoBERTa encoder. We train an FFN to identify all operands of the expected reasoning program in the input.

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

where FFN is a 2-layer feed-forward network with GELU (Hendrycks & Gimpel, 2016) 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.

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

where $\boldsymbol{h}^s$ is the soft-masked representation, $\boldsymbol{v}^m$ stands for the mask embedding, and "*" indicates the 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 masking vector. 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 out that this works better than the other options (including the embedding of the special [MASK] token) for this task.

Soft-Masking Operand Generator We also utilize the soft masking mechanism to extract the two operands of the specific reasoning step from the input. Compared to the soft-masking operand extractor that identifies all operands of the input, the soft-masking operand generator only finds the operands of the 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.

Note that the soft-masking mechanism in the operand extractor highlights all operands in the sentences, while the soft-masking mechanism in the operand generator only highlights the operands of the single reasoning step, which helps the following operator generation and order prediction procedures of that step rely more on the two operands.

Operator Generator We define six operators following 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.

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

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 the same as in the input or in the reverse order).

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

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 the 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 compute for 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 (Xu et al., 2021a;b).

Compared to autoregressive decoding, non-autoregressive counterparts ignore the decoding history which may have a potential negative impact on its coherence, but in case with the program generation for numerical reasoning, the coherence may affect less, while the autoregressive decoding is very likely to be mislead 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 MultiHiertt dataset (Zhao et al., 2022). Compared to other hybrid textual-tabular QA benchmarks, questions of the MultiHiertt dataset may involve multiple hierarchical tables and longer texts, and the reasoning process required to answer the question is more complex. The dataset consists of 10,440 questions with 2,513 financial documents, and is split into three parts: training (75%), development (10%), and test (15%). The labels of the test set are not publicly available.

Evaluation Metrics Following MultiHiertt, we evaluated the performance of different approaches with exact matching (EM) and the adopted numeracy-focused F1 (Dua et al., 2019).

Baselines 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. To fairly compare with existing state-of-the-art results, all our baselines are based on the RoBERTa-large model.

Model Settings We find that the good performing hyper-parameters are different for different model settings. We tuned hyper-parameters on the development set (§ 4.6). For base model, λ^{op} was set to 2, the others were all set to 1. For large model, λ^{t} was set to 2, and λ^{length} was set to 1.5, the others were set to 1. We set the maximum number of reasoning steps n to 10.

4.2 Main Results

We first compare NAPG (with both base and large settings) with our baselines. Results are shown in Table 1.

	Dev		Test	
	EM	F1	EM	F1
TAGOP (RoBERTa-large)	19.16	21.08	17.81	19.35
FinQANet (RoBERTa-large)	32.41	35.37	31.72	33.60
MT2Net (RoBERTa-large)	37.05	39.96	36.22	38.43
Ours (RoBERTa-base)	39.27	40.21	38.19	38.81
Ours (RoBERTa-large)	**45.79**	**46.72**	**44.19**	**44.81**

Table 1: Main results.

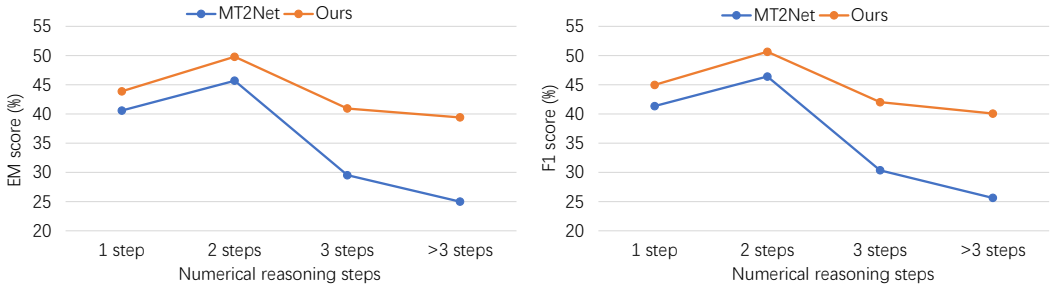

Figure 4: Performance of different numerical reasoning steps on the development set.

Table 1 shows that: 1) already with the RoBERTa base setting, our method is able to achieve better performance than the SOTA MT2Net model on both the development ($+2.22$ EM and $+0.25$ F1) and the test set ($+1.97$ EM and $+0.38$ F1) even using only a much smaller model than baselines. 2) using the RoBERTa large setting can further boost the performance of NAPG, and lead to huge improvements over the MT2Net baseline on both the development set ($+8.74$ EM and $+6.76$ F1) and the test set ($+7.97$ EM and $+6.38$ F1). It's worth noting that our approach only modifies the program generation part of MT2Net, leaving the other parts unchanged. The large performance gain with our method shows the effectiveness and superiority of NAPG over the LSTM decoder used by MT2Net.

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 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 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 ($+11.41$ EM and $+11.67$ F1 when the number of reasoning steps is 3 and $+14.42$ EM and $+14.43$ F1 when the number of reasoning steps is larger than 3).

The performance drop 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, the non-autoregressive generation prevents our method from suffering from this issue and might be the reason for the good performance of NAPG over MT2Net especially on these challenging questions.

4.4 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. Results are shown in Table 2.

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

Table 2: Time costs for program generation.

Table 2 shows that NAPG is 21 times as fast as the MT2Net, showing the great advantage of non-autoregressive decoding over the step-by-step autoregressive decoding in terms of speed due to parallelization.

4.5 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 3.

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

Table 3: Performance of different question types on the development set.

Table 3 shows that NAPG outperforms the strong MT2Net baseline in all evaluations. Given that table-only questions normally require numerical reasoning, and that the number of reasoning steps positively co-relates to the number of tables, the huge 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.

4.6 ABLATION STUDY OF HYPER-PARAMETERS

We explored a number of hyper-parameter values for the combination of training losses (Eq. 9) to study their effects on performance. 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 with a larger value. Results are shown in Table 4.

Table 4 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.

4.7 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 5 (in Appendix).

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

Table 4: Effects of different weights of each module.

Table 5 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 4), while the sequence generation of MT2Net fully relies on the single token prediction loss.

5 Related Work

Question Answering Most previous Question Answering studies focus on homogeneous data (Seonwoo et al., 2020; Ko et al., 2020; Ram et al., 2021; Fajcik et al., 2021; Yang et al., 2021; Eisenschlos et al., 2021; Yang et al., 2022; Cheng et al., 2022a; Katyayan & Joshi, 2022), such as unstructured text datastes (Rajpurkar et al., 2016; Dua et al., 2019), or structured tabular datasets (Iyyer et al., 2017; Katsis et al., 2021; Cheng et al., 2022b). Recently, there are many question answering studies (Chen et al., 2020b; Zhu et al., 2021; Chen et al., 2021; Feng et al., 2022; Li et al., 2022; Zhao et al., 2022) working on heterogeneous datasets, such as textual-tabular datasets.

Numerical Reasoning Numerical reasoning ability is very important for many NLP tasks (Dua et al., 2019; Zhang et al., 2021; Zhu et al., 2021; Chen et al., 2021; Zhao et al., 2022). Hybrid tabular and textual question answering requires numerical reasoning. TAGOP (Zhu et al., 2021) uses the sequence tagging method to extract facts, and performs a single arithmetic operation based on predefined operators. FinQANet (Chen et al., 2021) and MT2Net (Zhao et al., 2022) can perform 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 the types of reasoning can be categorized into numerical reasoning and span extraction. In this paper, 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 the program generation speed.

Our experiments on the MultiHiertt dataset show that: 1) our proposed model can bring about large improvements ($+7.97$ EM and $+6.38$ F1 points) over the strong MT2Net baseline, establishing a 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.

REFERENCES

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

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

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

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

Zhoujun Cheng, Haoyu Dong, Ran Jia, Pengfei Wu, Shi Han, Fan Cheng, and Dongmei Zhang. Fortap: Using formulas for numerical-reasoning-aware table pretraining. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1150–1166, 2022a.

Zhoujun Cheng, Haoyu Dong, Zhiruo Wang, Ran Jia, Jiaqi Guo, Yan Gao, Shi Han, Jian-Guang Lou, and Dongmei Zhang. 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)*, pp. 1094–1110, 2022b.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186, 2019.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. 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)*, pp. 2368–2378, 2019.

Julian Eisenschlos, Maharshi Gor, Thomas Mueller, and William Cohen. Mate: Multi-view attention for table transformer efficiency. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 7606–7619, 2021.

Martin Fajcik, Josef Jon, and Pavel Smrz. Rethinking the objectives of extractive question answering. In *Proceedings of the 3rd Workshop on Machine Reading for Question Answering*, pp. 14–27, 2021.

Yue Feng, Zhen Han, Mingming Sun, and Ping Li. Multi-hop open-domain question answering over structured and unstructured knowledge. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pp. 151–156, 2022.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

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

Mohit Iyyer, Wen-tau Yih, and Ming-Wei Chang. 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)*, pp. 1821–1831, 2017.

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

Pragya Katyayan and Nisheeth Joshi. Design and development of rule-based open-domain question-answering system on squad v2. 0 dataset. *arXiv preprint arXiv:2204.09659*, 2022.

Miyoung Ko, Jinhyuk Lee, Hyunjae Kim, Gangwoo Kim, and Jaewoo Kang. Look at the first sentence: Position bias in question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1109–1121, 2020.

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

Moxin Li, Fuli Feng, Hanwang Zhang, Xiangnan He, Fengbin Zhu, and Tat-Seng Chua. 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)*, pp. 57–69, 2022.

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

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

C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. (140), 2020.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 2383–2392, 2016.

Ori Ram, Yuval Kirstain, Jonathan Berant, Amir Globerson, and Omer Levy. Few-shot question answering by pretraining span selection. 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)*, pp. 3066–3079, 2021.

Yeon Seonwoo, Ji-Hoon Kim, Jung-Woo Ha, and Alice Oh. Context-aware answer extraction in question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 2418–2428, 2020.

Alon Talmor and Jonathan Berant. 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)*, pp. 641–651, 2018.

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

Hongfei Xu, Qiuhui Liu, Josef van Genabith, and Deyi Xiong. Modeling task-aware mimo cardinality for efficient multilingual neural machine translation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pp. 361–367, 2021a.

Hongfei Xu, Qiuhui Liu, Josef van Genabith, Deyi Xiong, and Meng Zhang. Multi-head highly parallelized lstm decoder for neural machine translation. 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)*, pp. 273–282, 2021b.

Jingfeng Yang, Aditya Gupta, Shyam Upadhyay, Luheng He, Rahul Goel, and Shachi Paul. 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)*, pp. 528–537, 2022.

Yinfei Yang, Ning Jin, Kuo Lin, Mandy Guo, and Daniel Cer. Neural retrieval for question answering with cross-attention supervised data augmentation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pp. 263–268, 2021.

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

Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. 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)*, pp. 1321–1331, 2015.

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

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

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

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

Yilun Zhao, Yunxiang Li, Chenying Li, and Rui Zhang. 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)*, pp. 6588–6600, 2022.

Fengbin Zhu, Wenqiang Lei, Youcheng Huang, Chao Wang, Shuo Zhang, Jiancheng Lv, Fuli Feng, and Tat-Seng Chua. 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)*, pp. 3277–3287, 2021.

A Appendix

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 theU.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 5: 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.