Towards Robust Numerical Question Answering: Diagnosing Numerical Capabilities of NLP Systems

Jialiang Xu[1]*, **Mengyu Zhou**[2]†, **Xinyi He**[3]*, **Shi Han**[2], **Dongmei Zhang**[2]
[1] University of Illinois at Urbana-Champaign
[2] Microsoft Research [3] Xi'an Jiaotong University
jx17@illinois.edu, hxyhxy@stu.xjtu.edu.cn,
{mezho, shihan, dongmeiz}@microsoft.com

Abstract

Numerical Question Answering is the task of answering questions that require numerical capabilities. Previous works introduce general adversarial attacks to Numerical Question Answering, while not systematically exploring numerical capabilities specific to the topic. In this paper, we propose to conduct numerical capability diagnosis on a series of Numerical Question Answering systems and datasets. A series of numerical capabilities are highlighted, and corresponding dataset perturbations are designed. Empirical results indicate that existing systems are severely challenged by these perturbations. *E.g.*, Graph2Tree experienced a 53.83% absolute accuracy drop against the "Extra" perturbation on ASDiv-a, and BART experienced 13.80% accuracy drop against the "Language" perturbation on the numerical subset of DROP. As a counteracting approach, we also investigate the effectiveness of applying perturbations as data augmentation to relieve systems' lack of robust numerical capabilities. With experiment analysis and empirical studies, it is demonstrated that Numerical Question Answering with robust numerical capabilities is still to a large extent an open question. We discuss future directions of Numerical Question Answering and summarize guidelines on future dataset collection and system design.

1 Introduction

Numeracy is an essential part for real-world NLP applications (Sundaram et al., 2022; Thawani et al., 2021b; Sundararaman et al., 2020; Spithourakis and Riedel, 2018). **Numerical QA** (Question Answering) is one representative group of such number-dependent NLP tasks. *E.g.*, Math Word Problem Solving (Zhang et al., 2020a; Miao et al., 2020; Koncel-Kedziorski et al., 2016), Discrete Reasoning (Dua et al., 2019; Hu et al., 2019; Al-Negheimish et al., 2021a), Tabular Question Answering (Zhong et al., 2017; Chen et al., 2020b; Zhu et al., 2021; Chen et al., 2021). These Numerical QA tasks require NLP systems to arrive at a numerical answer from the numbers in the question and context. By studying how existing NLP systems perform in these Numerical QA tasks, we could take a glimpse at what capabilities are required for building NLP systems in the future.

In an ad-hoc manner, a line of work revealed robustness issues of handling Numerical QA in existing NLP systems. Through adversarial attacks with designed dataset perturbations, number-related limitations were exposed: *E.g.*, utilizing spurious correlation in datasets (Patel et al., 2021; Kumar et al., 2021; Al-Negheimish et al., 2021b; Pi et al., 2022b), incorrectly representing numbers (Nogueira et al., 2021; Kim et al., 2021) and failing to extrapolate (Kim et al., 2021; Pal and Baral, 2021). This line of work inspires us to ask following questions: 1) What is the overall landscape of *robustness issues of numerical capabilities* in existing NLP systems? Can we find a more systematic way to investigate the number-related limitations? 2) How to *diagnose* each numerical capability and *evaluate* the severity of it not being captured in a system? Can we further develop new adversarial *perturbation* methods on Numerical QA for diagnosis and evaluation? 3) How to *address* the numerical robustness issues? How do *existing solutions* work and what are possible *future directions*?

To answer the above questions, in this paper we propose the **DNC** (**D**iagnosing **N**umerical **C**apbilities) **framework**[1] as shown in Figure 1.

Most existing Numerical QA systems (see §2.1) take a two-stage approach to extract and manipulate numbers. As shown in the **QA Stages** part of Figure 1, systems usually first recognize numbers

* The contributions by Jialiang Xu and Xinyi He have been conducted and completed during their internships at Microsoft Research Asia, Beijing, China.

† Corresponding author.

[1] Our code and datasets used are available in the link: https://github.com/microsoft/NumberDiagnosis

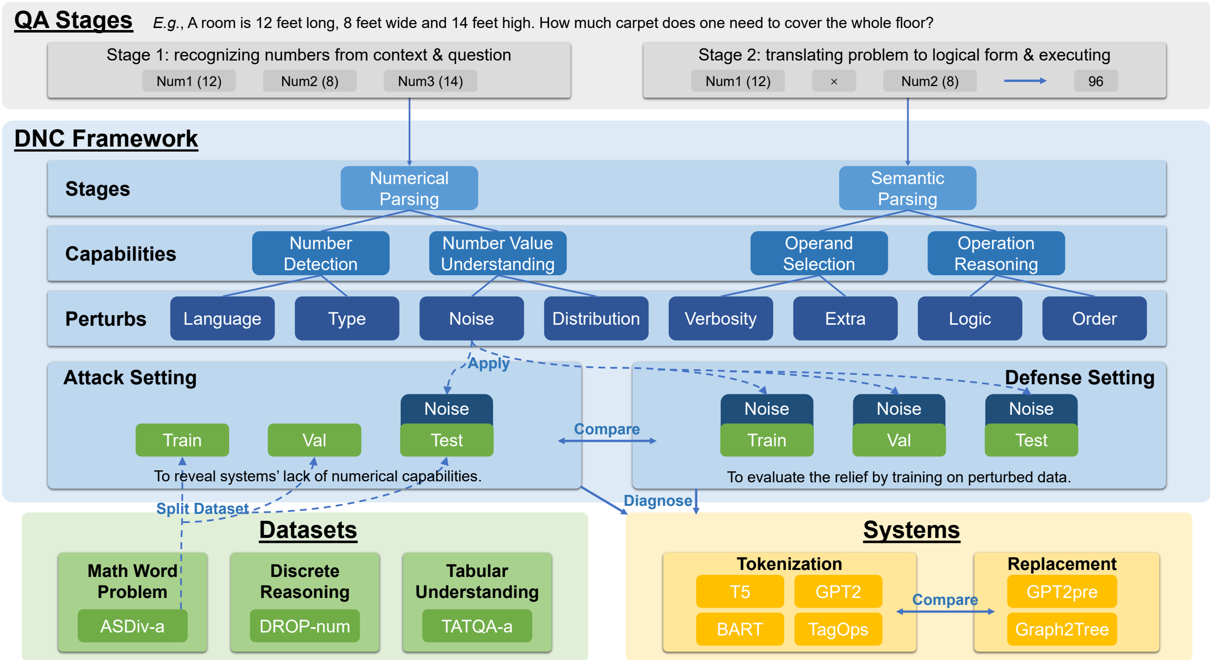

Figure 1: Overview of DNC Framework. The process of Numerical QA solving is divided into two logical stages. Four capabilities are required to complete the stages, each maps to two perturbations. Perturbations can be applied to appropriate train / validation / test splits of Numerical QA datasets under Attack or Defense Setting. Models of the NLP systems are trained and then evaluated on the perturbed datasets as a diagnosis of their numerical capabilities.

in the context and question and treat them as candidate operands. Then, with the understanding of the question semantics, they select corresponding operands, and explicitly generate logical forms or implicitly execute operations to get the final result.

The above two stages correspond to the two groups of numerical capabilities (see §4.1) covered by our **`DNC Framework`** (as shown in Figure 1). In Stage 1, we focus on a system's capabilities to recognize different forms of numbers ("Number Detection"), and to parse and represent number values correctly ("Number Value Understanding"). In Stage 2, we focus on the capabilities to correctly choose operands ("Operand Selection") and operations ("Operation Reasoning") by understanding context and question. For each of these four capabilities, two perturbations (see §4.2) are proposed by us to diagnose the capability. Each perturbation is designed to be trivial to humans and thus cannot easily fool humans, but it could bring down existing NLP systems (under the "Attack" setting), and therefore expose the robustness issue of lacking its corresponding capability.

By applying the above diagnosis to various NLP **`Systems`** and Numerical QA **`Datasets`** (as shown in Figure 1), in §5 we find that existing systems experience significant performance drops, which verifies their lack of robust numerical capabilities. *E.g.*, Graph2Tree experienced a 53.83% absolute accuracy drop against the "Extra" perturbation on ASDiv-a, and BART experienced 13.80% accuracy drop against the "Language" perturbation on the numerical subset of DROP.

From another point of view, the perturbations are also applicable for data augmentation. Under the "Defense" setting (see §4.3), perturbations are applied to all splits of the dataset. A system's performance of the same perturbation under both "Attack" and "Defense" settings are compared (in §5.2) to show if the corresponding robustness issue could be relieved by augmenting the training data. Empirical results indicate that despite the recovery in most cases, the performance still fall lower than the original level.

Finally, based on the "Attack" and "Defense" results in §5 and additional experiments, in §6 we compare some existing design choices in Numerical QA, such as: Is it better to generate logical forms (and then execute the program/expression) or predict answers directly in an end-to-end way? Shall we break numbers into subword tokens or substitute them with a placeholder that can be later re-substituted? We also discuss the open questions and future directions on the robust numerical capabilities of NLP systems, including recent relevant development such as neural program execution and numerical data synthesizing.

In summary, our major contributions are:

- The DNC framework is proposed by us to systematically diagnose the robustness of NLP systems on numerical capabilities. A series of number-related perturbation methods are designed for the capabilities.
- Comprehensive diagnosing experiments on adversarial attacks and data augmentations are conducted by us on five systems over three Numerical QA tasks. We show the overall picture of numerical robustness issues of the systems, and the partial effectiveness of our simple defense mechanism.
- Based on experiments and previous work, we provide guidelines for existing numerically-robust NLP system designs and discussions for future directions on robust Numerical QA.

2 Related Work

2.1 Numerical Question Answering

Previous work has proposed Numerical QA datasets and systems. In this paper we consider as examples the domains of Math Word Problem, Discrete Reasoning and Tabular QA.

Math Word Problem (Kushman et al., 2014; Upadhyay and Chang, 2017; Miao et al., 2020; Qin et al., 2020; Lan et al., 2022) concerns arithmetic questions collected from lower-grade elementary school coursework. Neural network are employed with different architectures such as Seq2Seq (Wang et al., 2017; Chiang and Chen, 2019), Seq2Tree (Xie and Sun, 2019; Liang et al., 2021) and Graph2Tree (Zhang et al., 2020b; Shen and Jin, 2020). Recently, large end-to-end pretrained language models (Chowdhery et al., 2022; Pi et al., 2022a) have also been showing impressive results in Math Word Problem.

Discrete Reasoning (Dua et al., 2019; Al-Negheimish et al., 2021a; Hu et al., 2019) concerns questions requiring logistic and arithmetic operations on real-world paragraphs. Discrete Reasoning Systems are mainly based on Graph Attention Networks (Chen et al., 2020a) or the Transformer architecture (Ran et al., 2019).

Tabular QA and Semantic Parsing (Zhu et al., 2021; Chen et al., 2021; Zhong et al., 2017; Pasupat and Liang, 2015) concerns question answering in the domain of tabular data, which often involves a large amount of numbers and requires arithmetic aggregations to arrive at the final answer. Tabular QA systems (Dong et al., 2022; Liu et al., 2022; Iida et al., 2021; Herzig et al., 2020; Yin et al., 2020) are mainly based on Pretrained Language Models with Transformer backbones. Tabular QA systems mainly aim at converting natural language utterance into executable expressions such as commands in SQL language.

2.2 Numeracy Limitations in NLP Systems

Efforts have been dedicated to reveal numeracy limitations in NLP systems. (Patel et al., 2021; Kumar et al., 2021; Al-Negheimish et al., 2021b; Pi et al., 2022b; Nogueira et al., 2021; Kim et al., 2021; Pal and Baral, 2021). However, previous work mainly focused on borrowing adversarial attack methods from general QA such as re-ordering sentences (Patel et al., 2021; Al-Negheimish et al., 2021b; Kumar et al., 2021), substituting synonyms (Kumar et al., 2021; Pi et al., 2022b), or adding irrelevant information (Patel et al., 2021; Pi et al., 2022b), while having limited exploration into capabilities specific to Numerical QA problems such as understanding different number values, recognizing different number surface forms or selecting related numbers.

3 Preliminaries

A *Numerical Question Answering* problem is defined to consist of a problem prompt (question) $\mathcal{P}$ and a problem body (context) $\mathcal{B}$. Depending on the task type, the problem body takes the form of either a paragraph or a mixture of free-form text paragraphs and structured data such as tables. Let $\mathcal{V}$ be the vocabulary of the textual words, $\mathcal{Q}$ be the set of the numerical values in $\mathcal{P} \cup \mathcal{B}$, and $\mathcal{Q}^+$ be the numerical values that can be arithmetically computed with $\mathcal{Q}$, then the problem prompt and body can be formulated as $\mathcal{P} = \bigoplus_i p_i,\ p_i \in \mathcal{V} \cup \mathcal{Q}$ and $\mathcal{B} = \begin{cases} \bigoplus_j b_j \\ \bigoplus_{i,j} \tau_{i,j} \oplus \bigoplus_k b_k \end{cases},\ \tau_{i,j}, b_k \in \mathcal{V} \cup \mathcal{Q}$. Here $\oplus$ denotes the concatenation operation, $p_\cdot$ and $b_\cdot$ are prompt and body textual words, and $\tau_{\cdot,\cdot}$ are the body tabular cells.

The target output $\mathcal{T}$ of the problem is either a numerical value $\mathcal{T}_{ans}$ that is an element in $\mathcal{Q}^+$ or a mathematical expression $\mathcal{T}_{eq}$ that consists of elements in the concerned numerical values $\mathcal{Q}$ and the simple operators $\mathcal{O} = \{+, -, \times, \div\}$. *I.e.*

$\mathcal{T} = \begin{cases} \mathcal{T}_{ans} : q \in \mathcal{Q}^+ \\ \mathcal{T}_{eq} : \bigoplus_i t_i,\ t_i \in \mathcal{Q} \cup \mathcal{O} \end{cases}$. With $\mathcal{P}$ and $\mathcal{B}$ as input and $\mathcal{T}$ as output, a trained Numerical QA system can be regarded as a mapping f such that

$$f : (\mathcal{P}, \mathcal{B}) \to \mathcal{T} \tag{1}$$

Note that this expression not only describes the Numerical QA tasks, but also generalizes to other numeracy-related NLP tasks such as Tabular Entailment (Chen et al., 2020b) and Timeseries-based Fraudulent Detection (Padhi et al., 2021).

In this paper, we design and apply perturbations to the samples in the dataset to form perturbed prompt $\mathcal{P}^\star$, perturbed body $\mathcal{B}^\star$ and perturbed ground truth target $\mathcal{T}^\star$. We show that existing systems are fragile against numerical pertubation by showing that on a large portion of the dataset, the previous mapping fails to generate correct perturbed target, *i.e.*:

$$f : (\mathcal{P}^\star, \mathcal{B}^\star) \not\to \mathcal{T}^\star \tag{2}$$

4 DNC Framework

Our approach aims at diagnosing the numerical weakness of existing Numerical Question Answering models. We list out and explain a series of **numerical capabilities** that are critical to solving Numerical Question Answering problems in §4.1. We then design numerical **perturbations** targeting these capabilities in §4.2. With the designed perturbations, we examine the weaknesses under two different **perturbations settings** in §4.3.

These three sections are represented in Figure 1. as the "**Capabilities**" stripe, the "**Perturbs**" stripe, and the "**Attack Setting**" and " **Defense Setting**".

4.1 Numerical Capabilities

We classify numerical capabilities into three major categories, concerning different aspects of numerical understanding, as below:

Number Detection is the capability of recognizing numbers of different surface forms. For instance, the English word *"Forty-two"* and the Arabic number *"42.0"* are regarded the same number in Numerical QA and should not affect the final arithmetic answer of a question.

Number Value Understanding is the capability of understanding numbers of different value distributions. Systems are expected to not only apply arithmetic calculation on a specific set of numbers (*e.g.*, integers of values smaller than 500 as included in the BERT tokenizer vocabulary). Robust Numerical QA systems are also expected to handle values such as float-point numbers and numbers larger than 500.

Operand Selection is the capability of deciding which numbers to select as the operands in the arithmetic process. One important aspect of selecting related values is to exclude numbers that are 1) irrelevant to the Numerical QA problem scenario, or 2) relevant to the problem scenario but not essential to the question solving. Systems are expected to select as operands the important values from the unimportant values.

Operation Reasoning is the capability of inferring operations from the logic pattern described in the text. In an arithmetic process, the operation is independent from the operands, therefore different operations can be applied to the same set of selected related numbers in different questions. Systems are expected to decouple operation from operands and select the operation in an operand-agnostic way.

4.2 Perturbations

Perturbations are designed according to each numerical capabilities. In Table 1, an example problem is provided for each of the perturbations. The formal definition of the perturbations is provided in Appendix A.

Language Perturbation targets the Number Detection capability and diagnoses how accurate can systems detect numbers in different surface forms. To perturb a number string n_s, we replace it with its English form of the number with Num2Words.[2] This perturbation changes number surface forms but not their values.

Type Perturbation targets the Number Detection capability and challenges systems to detect numbers in their float-point forms. To perturb a number string n_s, we concatenate it with the string ".0". Similar to Language Perturbation, only the number detection capability is diagnosed with this perturbation. Contrary to the Noise perturbation in the next paragraph, the Type perturbation does not propose additional calculation difficulty by changing number values.

Noise Perturbation targets the Number Value Understanding capability and challenges systems to not only understand arithmetic operations of not only integers but also float-point numbers. To perturb a number n, we randomly attach a one-

[2] https://github.com/savoirfairelinux/num2words

Capability	Perturbation	Example Problem Pair	T5 Prediction
Number Detection	Language	Original: A mailman has to give out 192 pieces of junk mail. If he goes to 4 blocks, how many pieces of junk mail should he give each block? Perturbed: A mailman has to give out one hundred and ninety-two pieces of junk mail. If he goes to four blocks, how many pieces of junk mail should he give each block?	Original: 192 / 4 ✓ Perturbed: 92 / 4 × Expected: 192 / 4
	Type	Original: There were 105 parents in the program and 698 pupils, too. How many people were present in the program? Perturbed: There were 105.0 parents in the program and 698.0 pupils, too. How many people were present in the program?	Original: 105 + 698 ✓ Perturbed: 105 + 688 × Expected: 105 + 698
Number Value Understanding	Noise	Original: Tony had $20. He paid $8 for a ticket to a baseball game. At the game, he bought a hot dog for $3. What amount of money did Tony have then? Perturbed: Tony had $20.2. He paid $8.5 for a ticket to a baseball game. At the game, he bought a hot dog for $3.5. What amount of money did Tony have then?	Original: 20 - 8 - 3 ✓ Perturbed: 208.52 - 3.5 × Expected: 20.2 - 8.5 - 3.5
	Distribution	Original: Frank had $16. After buying some new toys he had $8 left.How much did he spend on toys? Perturbed: Frank had $1281. After buying some new toys he had $478 left.How much did he spend on toys?	Original: 16 - 8 ✓ Perturbed: 1215 - 878 × Expected: 1281 - 478
Operand Selection	Extra	Original: John has twelve shirts. Later he bought four more shirts. How many shirts does John have in total? Perturbed: John has twelve shirts. Later he bought four more shirts. Frank had $16. How many shirts does John have in total?	Original: 12 + 4 ✓ Perturbed: 16 + 12 × Expected: 12 + 4
	Verbosity	Original: The roller coaster at the state fair costs 6 tickets per ride. If 8 friends were going to ride the roller coaster, how many tickets would they need? Perturbed: The roller coaster at the state fair costs 6 (not 30) tickets per ride. If 8 (not 119) friends were going to ride the roller coaster, how many tickets would they need?	Original: 6 * 8 ✓ Perturbed: 8 * 119 × Expected: 6 * 8
Operation Reasoning	Logic	Original: Jack received 8 emails in the morning and 2 emails in the afternoon. How many emails did Jack receive in the day? Perturbed: Jack received 8 emails in the morning and 2 emails in the afternoon. How many more emails did Jack receive in the morning than in the afternoon?	Original: 8 + 2 ✓ Perturbed: 8 + 2 × Expected: 8 - 2
	Order	Original: A DVD book holds 126 DVDs. There are 81 DVDs already in the book. How many more DVDs can be put in the book? Perturbed: There are 81 DVDs already in the book. A DVD book holds 126 DVDs. How many more DVDs can be put in the book?	Original: 126 - 81 ✓ Perturbed: 81 - 126 × Expected: 126 - 81

Table 1: Examples of DNC Perturbations and Corresponding Predictions by T5. For each perturbation an example original and perturbed problem pair is shown. The rightmost column shows some error cases where T5 generates correct equation on the original problem but fails on the perturbed. The ground truth equation of the perturbed problem is also provided after "Expected".

digit fractional part with uniform distribution. This perturbation introduces new float-point numbers and breaks the original number value distribution in the dataset by adding an random variable.

Distribution Perturbation targets the Number Value Understanding capability and challenges systems to conduct arithmetic with larger integers. To perturb a number n, we randomly offset the value with a normal distribution. Based on the observations in Wallace et al. (2019), we choose to perturb the majority of the numbers to larger than 500. This perturbation introduces large numbers and breaks original number value distribution in the dataset.

Verbosity Perturbation targets the Operand Selection capability and challenges systems to select the correct quantity in the problem by adding explicitly irrelevant numbers into the problem. To perturb a number string n_s, we concatenate it with an irrelevant number in parentheses, the irrelevant number is preceded by "not". This perturbation introduces numbers without breaking the distribution of relevant numbers in the dataset.

Extra Perturbation targets the Operand Selection capability and challenges systems to exclude irrelevant numbers. To perturb a problem $(\mathcal{B}, \mathcal{P})$, An irrelvant sentence containing numbers randomly sampled from the corpus is added to the body $\mathcal{B}$. This perturbation breaks the number distribution by introducing extra instances of different numbers for the same problem.

Logic Perturbation targets the Operation Reasoning capability and challenges systems to choose correct operations for the same set of numbers. In this paper, for two datasets described in §5.1, TATQA and ASDiv-a, the Operation perturbation demands additional attention. On TATQA it is based on template matching via SpaCy[3] and automatic conversions, while on ASDiv-a it is based on manual annotation due to the diversity of patterns in the ASDiv-a dataset. This perturbation

[3]https://spacy.io/

introduces extra problems of different operations.

Order Perturbation targets the Operation Reasoning capability and challenges systems to choose correct operations for the same set of numbers. On ASDiv-a, the order of sentences in the problem body is manually altered in a manner that changes the order of number occurrence but not the problem logic. This perturbation does not break the operation distribution within the dataset.

4.3 Perturbing Settings

With the aforementioned perturbations, we construct perturbed datasets under different settings to investigate systems' numerical capabilities and the effectiveness of the perturbations from different perspectives. For a specific dataset with a training / validation / testing split, different splits are perturbed under different settings. In this paper we consider the following two settings of Attack and Defense, as compared in Table 2:

Attack. By applying the perturbations to the testing split of the dataset, we construct a challenge set to evaluate the corresponding numerical capability of existing systems. Systems are trained on the original datasets and evaluated on the perturbed challenge set.

Defense. Under the defense setting, perturbations are applied to all of training, validation, and testing split of the dataset. By comparing systems' performance under the Defense with Attack settings, we investigate to what extent the performance drop can be alleviated by using the perturbations as a data augmentation approach.

Setting	Attack	Defense
Train on	train	train⋆
Validate on	val	val⋆
Test on	test⋆	test⋆

Table 2: The Comparison between Two Settings in DNC. Perturbations (denoted by "⋆") are applied to different dataset splits (train / val / test) under each setting.

To perturb under Attack or Defense setting, suitable samples are first filtered according to a series of conditions. The perturbations are applied only to these filtered samples. The filtered samples in the dataset split(s) are replaced with their perturbed version to form the perturbed dataset. The filtering conditions and the formalized algorithm are provided in Appendix B.

5 Experiments

5.1 Experiment Setup

Datasets. In this paper, we used ASDiv-a (Miao et al., 2020), DROP (Dua et al., 2019), and TATQA (Zhu et al., 2021) as our Numerical Question Answering datasets. For DROP and TATQA, we filtered out DROP-num and TATQA-a, the numerical subsets of them. The statistics of these datasets are shown in Table 4.

Systems. We selected representative systems on each dataset and test their performance against perturbations. For the ASDiv-a dataset, we use Graph2Tree (Patel et al., 2021). For the DROP dataset, we use BART-base and T5-base from Huggingface.[4] For the TATQA dataset, we utilize TagOps with the RoBERTa backbone as described in the original paper.

Compute Environment. All experiments are done on a Linux machine equipped with 4 NVIDIA Tesla V100 16GB GPUs. The average runtime of our experiments ranges from one to three hours.

Hyperparameters. In our experiments, we adopt a general setting of hyperparameters of epoch number = 40, learning rate = $1e-5$ and batch size = 32. It is observed in our exploratory experiments that while the hyperparameters such as learning rate and batchsize do affect the absolute performance of the models, they have a modest effect on the general trend of the models' strengths and weaknesses against the numerical perturbations. The details and analysis are provided in Appendix C.

5.2 Experiment Results and Analysis

The experiment results are provided in Table 3. The metric we report is 1) the metric on original datasets (Original), and 2) the absolute change of the metric on perturbed datasets, denoted by "Δ". We additionally provide the raw metric and relative drop in Table 9 and Table 10 in the Appendix. The calculation details of the observation can be found in Appendix D.2.

Attack. As can be observed in Table 3 and Table 10, most systems were severely challenged under the Attack setting and experienced significant performance drop, ranging from 5% to 50% absolute drop and 5% to 80% relative drop in answer denotation accuracy. Between the two DNC goals, Semantic Parsing causes a more severe challenge, averaging 19.66% absolute drop and 31.79% rela-

[4]https://github.com/huggingface/transformers

Configuration		ASDiv-a								DROP-num		TATQA-a
		T5		BART		GPT2		Graph2Tree		T5	BART	TagOps
Setting	Perturbation	Acc_{eq}	Acc_{ans}	Acc_{eq}	Acc_{ans}	Acc_{eq}	Acc_{ans}	Acc_{eq}	Acc_{ans}	Acc	Acc	Acc
Attack (Δ)	Language	-18.85%	-18.85%	-23.77%	-27.05%	-12.30%	-12.30%	-7.65%	-7.38%	-10.62%	-14.73%	-18.62%
	Type	-37.70%	-11.48%	-32.79%	-15.57%	-17.21%	-10.66%	0.27%	1.09%	-7.70%	-11.06%	-5.34%
	Noise	-36.89%	-36.89%	-18.85%	-21.31%	-9.84%	-9.02%	0.27%	0.55%	-	-	-
	Distribution	-16.39%	-14.75%	-29.51%	-18.03%	-13.11%	-13.11%	-6.56%	-6.56%	-	-	-
	Verbosity	-41.80%	-44.26%	-25.41%	-29.51%	-10.66%	-11.48%	-33.33%	-33.88%	-9.58%	-13.31%	-1.90%
	Extra	-25.41%	-27.87%	-41.80%	-45.90%	-28.69%	-28.69%	-53.83%	-54.64%	-11.79%	-11.67%	-1.21%
	Logic	-29.51%	-27.87%	-36.89%	-35.25%	-25.41%	-23.77%	-28.42%	-21.86%	-	-	-14.29%
	Order	-34.43%	-5.74%	-33.61%	-4.10%	-27.87%	-7.38%	-33.33%	-7.10%	-	-	1.12%
Defense (Δ)	Language	-12.30%	-13.93%	-19.67%	-24.59%	2.46%	2.46%	-7.65%	-7.38%	0.07%	-1.84%	-7.59%
	Type	-11.48%	-12.30%	-4.92%	-6.56%	3.28%	4.10%	1.64%	1.91%	0.46%	-0.95%	2.93%
	Noise	-14.75%	-14.75%	-3.28%	-4.92%	3.28%	4.10%	0.55%	0.27%	-	-	-
	Distribution	-20.49%	-20.49%	-8.20%	-9.84%	-8.20%	-9.02%	-6.83%	-6.01%	-	-	-
	Verbosity	-15.57%	-16.39%	-5.74%	-7.38%	-0.82%	0.00%	-0.27%	1.09%	-5.13%	-1.84%	2.25%
	Extra	0.00%	1.64%	-2.46%	-4.10%	-17.21%	-18.03%	-20.22%	-17.76%	-11.32%	-10.44%	-9.14%
	Logic	-	-	-	-	-	-	-	-	-	-	13.64%
	Order	-25.41%	-4.10%	-27.87%	-7.38%	-1.64%	23.77%	-29.23%	-7.92%	-	-	19.47%
Original	None	68.03%	72.95%	67.21%	72.95%	44.26%	45.08%	66.94%	68.58%	49.42%	50.36%	42.41%

Table 3: The Results of DNC Framework. Five NLP systems are evaluated with three Numerical QA tasks under both Attack and Defense settings. The symbol "Δ" stands for the absolute metric difference between the current setting and the original setting. The color scale represents the distance from the original setting, deeper means further from the original setting. For ASDiv-a, Acc_{eq} and Acc_{ans} refer to the prediction accuracy of ground truth equations and denotation accuracy of answers, respectively. For DROP-num and TATQA-a, Acc refers to the denotation accuracy of the answers. We provide the raw performance and relative change of the metrics *w.r.t.* the original setting in Appendix D.1. "-" denotes that automatic perturbation and automatic data augmentation as described by §4.3 is not applicable here. We provide detailed explanation of the reason why they are not applicable in Appendix E.

Dataset	**# Training**	**# Validation**	**# Testing**
ASDiv-a	974	122	122
DROP-num	42258	5282	5283
TATQA-a	1971	245	247

Table 4: The Statistics of the Datasets Used.

tive drop, as compared to the 13.15% absolute drop and 19.66% relative drop by Numerical Parsing.

Among the considered systems, Transformer-based Seq2Seq systems (T5, BART, GPT2) are more sensitive than the tasks-specific Graph2Tree system against the perturbations stemming from the Numerical Parsing goal. The former resulted in 17.42% absolute drop and 27.06% relative drop, while Graph2Tree only experienced 3.07% absolute drop and 4.48% relative drop. The masking of numbers used by Graph2Tree allows it to remain unaffected against a portion of the perturbations targeting the Numerical Parsing goal.

Defense. As a counteracting approach, the defense mechanism helps alleviate systems' lack of corresponding numerical capabilities by applying automatic perturbations to the training and validation set. Via Defense, the lack according to the Semantic Parsing gets more recovery of (17.96% absolute improvement and 26.95% relative improvement *vs.* 6.52% absolute improvement and 11.42% relative improvement).

Among the considered systems, Transformer-based Seq2Seq systems benefits more from Defense than the Graph2Tree system (12.53% absolute improvement and 20.52% relative improvement *vs.* 11.58% absolute improvement and 16.88% relative improvement).

Despite the recovery from Defense, the challenge is still not solved. As the majority of the defense performance is still more than 10% below the original performance. This observation indicates that the lack of Numerical Capabilities is still an open question.

Summary. Our DNC framework provides insights on two major aspects of the diagnosis to Numerical QA systems:

1) It is demonstrated that severe numerical weaknesses exist in current Numerical QA systems ("Attack"), and they can not be trivially eliminated via, although benefiting from, an automatic data augmentation process ("Defense").

2) The systems' weaknesses are explicitly profiled in a quantitative and interpretable manner through the models' susceptibility difference to

a diversity of perturbations.

6 Guidelines and Open Directions

In this section, phenomena observed on different systems and datasets were summarized to provide comparison for existing methods. Also, recent related efforts corresponding to these phenomena were discussed to point open directions in the domain of Numerical QA.

6.1 Target: Logical Form Generation *vs.* Answer Predicting

One attribute specific to Numerical QA is the reasoning processes leading to the numerical answers, which is usually described by logical forms. On datasets where the ground truth logical forms are provided as an additional supervision (*e.g.*, ASDiv-a and TATQA), the systems have two options for the target: 1) **Logical Form Generation**, where systems generate the logical form which is later input to external symbolic executing systems such as Python scripts or SQL engines, and 2) **Answer Predicting**, where systems directly predict the output answer in an end-to-end manner. On datasets where ground truth logical forms are not provided (*e.g.*, DROP), the latter is the most frequently adopted approach. Logical Form Generation and Answer Predicting differ in the actual object to conduct the executing step of the logical form insinuated by the question (external symbolic systems *vs.* neural systems). With Answer Predicting, systems are expected to possess the capability of executing the logical forms internally.

We investigate to what extent do existing systems possess this execution capability, by comparing the impact of the problem target $\mathcal{T}$ in Numerical QA on ASDiv-a. The systems are trained to predict two different targets: 1) the logical form (*i.e.*, the MWP equation), and 2) the logical form and the execution result. Since most MWP-specific systems are incapable of predicting answers directly, we choose the Transformer-based systems GPT2, BART and T5. Results in Table 5 indicate that: 1) on existing systems, Logical Form Generation is beneficial for higher accuracy, and 2) even though models managed to compose equations with high accuracy, they struggle to faithfully execute an equation to get the correct answer.

Recent work also pays increasing attention to the execution capability. Systems such as TAPEX (Liu et al., 2022) and POET (Pi et al., 2022a) have been leveraging data synthesizing and intermediate pretraining to learn neural program executors and achieved state-of-the-art results over systems leveraging Logical Form Generation. This recent development shows the potential of neural systems with enhanced execution capability on the Numerical QA task.

Model	**Acc**$_{ans}$	**Acc**$_{eq}$
GPT2$_{ans}$	6.56%	-
GPT2$_{eq}$	45.08%	44.26%
BART$_{ans}$	9.02%	-
BART$_{eq}$	72.95%	67.21%
T5$_{ans}$	2.46%	-
T5$_{eq}$	72.95%	68.03%

Table 5: Comparing Models with Different Prediction Targets on ASDiv-a. For a model $\mathcal{M}$, $\mathcal{M}_{eq}$ / $\mathcal{M}_{ans}$ predicts equation / equation and answer, respectively. Acc$_{eq}$ and Acc$_{ans}$ stand for the denotation accuracy of the generated equation and the accuracy of the directly predicted answer, respectively.

6.2 Numbers: Tokenization *vs.* Replacement

We also investigate the impact of different ways of manipulating numbers. There are two mainstream existing methods to process and represent numbers, herein referred to as the **Tokenization** and **Replacement** methods.

Tokenization methods such as WordPiece (Wu et al., 2016) and BPE (Sennrich et al., 2016) adopted by existing Numerical QA systems divides numbers into potentially multiple sub-word level tokens. *E.g.*, The number `768` will be divided into tokens `7` and `68` by T5's tokenizer. This approach stems from the fundamental fact that existing systems' vocabularies are finite while the occurrences of numbers in a Numerical QA dataset can be too diverse to include in a finite vocabulary. Tokenization causes extra representation cost and erases the digit integrity by potentially introducing multiple tokens for a single number.

Replacement substitutes numbers with special tokens in the input (`[NUM1]`, `[NUM2]`, *etc.*), which are later re-substituted with the original number in the output logical forms. This approach avoids multiple tokens by providing exactly one representation for each number, but has its own limitations handling number diversity since the recognition of numbers are usually performed with rule-based matching, which is often non-exhaustive.

In this paper, T5, BART, GPT2 and TagOps adopts Tokenization, while Graph2Tree adopts Re-

Model	Perturbation	Acc_{eq}	Acc_{ans}	Δ Acc_{eq}	Δ Acc_{ans}
GPT2_{token}	Language	31.97%	32.79%	-12.30%	-12.30%
	Type	27.05%	34.43%	-17.21%	-10.66%
	Noise	34.43%	36.07%	-9.84%	-9.02%
	Distribution	31.15%	31.97%	-13.11%	-13.11%
	Verbosity	33.61%	33.61%	-10.66%	-11.48%
	Extra	15.57%	16.39%	-28.69%	-28.69%
	Logic	18.85%	21.31%	-25.41%	-23.77%
	Order	16.39%	37.70%	-27.87%	-7.38%
	Original	44.26%	45.08%	0.00%	0.00%
$\text{GPT2}_{replace}$	Language	46.72%	47.54%	-9.84%	-9.84%
	Type	56.56%	57.38%	0.00%	0.00%
	Noise	56.56%	57.38%	0.00%	0.00%
	Distribution	56.56%	57.38%	0.00%	0.00%
	Verbosity	22.13%	22.13%	-34.43%	-35.25%
	Extra	10.66%	11.48%	-45.90%	-45.90%
	Logic	32.79%	40.16%	-23.77%	-17.21%
	Order	18.03%	36.07%	-38.52%	-21.31%
	Original	56.56%	57.38%	0.00%	0.00%

Table 6: The Results of Tokenization and Replacement on GPT2. GPT2_{token} adopts the Tokenization method and $\text{GPT2}_{replace}$ adopts the Replacement method.

placement. We implement two variations of GPT2: GPT2_{token} and $\text{GPT2}_{replace}$ to compare their robustness against different perturbations on the ASDiv-a dataset.Results in Table 6 indicate that Replacement has an advantage when no perturbation is present or when the perturbation only involves changes in number value. However, when the perturbation changes number values, the Replacement-based system is more severely challenged.

We hypothesize that the Replacement method removes all numerical information such as the format and value of numbers in the problem and lost numeracy capabilities, therefore the system receives only textual signals such as number order or word frequency, which further encouraged systems to learn from spurious correlations as stated in Patel et al. (2021). This hypothesis is consistent with the observations of a recent study (Thawani et al., 2021a) that investigates of the mutual-enhancement between numeracy and literacy.

The respective limitations of Tokenization and Replacement are calling for more numeracy-preserving number representation methods. Some studies have suggested changing number surface forms (Kim et al., 2021) or using dataset-agnostic representation (Sundararaman et al., 2020), however they either create extra token loads or could not generalize well on large-scale real-world dataset. The numeracy-preserving number representation is another bottleneck for Numerical QA.

7 Conclusion

In this paper we aim at diagnosing numerical capabilities in existing NLP systems. We list out a series of numerical capabilities and design corresponding dataset perturbations. Empirical results show that existing systems still lack numerical capabilities to a large extent, and this lack cannot be eliminated in a trivial manner. Analysis into the empirical results, discussion of the the existing practices, and insights for future directions of Numerical QA dataset collection and system design are also provided.

Limitations

Our pipeline has limitations in the following two aspects that we plan to address in the future:

Dependency on ground truth equation. Currently, three of the eight DNC perturbations have strong dependency on the ground truth solving equation, which is missing in datasets such as DROP. We hope to utilize semi-supervised approaches in the future to enlarge the coverage of the DNC perturbations.

Perturbing scalability. Currently our filters cover only a portion of the whole dataset due to DNC filtering and perturbing questions based on manual rules and templates. we hope to develop more automatic filtering and perturbing in the future. Also, DNC can only apply perturbations to numbers provided by the problem, which limits its diagnosing power in questions where an unspecified number is used, *e.g.*, when numerical commonsense knowledge is involved.

Ethical Statements

The model implementation and datasets utilized in this paper are based on publication and open-source repositories. Licenses protocols are followed in the process of our experiments. No new datasets or NLP applications are presented in this paper and no violation of privacy or usage of demographic information was involved in our process of interacting with the datasets. Our experiments do not involve lots of compute time/power as reported in the paper.

References

Hadeel Al-Negheimish, Pranava Madhyastha, and Alessandra Russo. 2021a. Discrete reasoning templates for natural language understanding. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Student Research Workshop*, pages 80–87, Online. Association for Computational Linguistics.

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

Kunlong Chen, Weidi Xu, Xingyi Cheng, Zou Xiaochuan, Yuyu Zhang, Le Song, Taifeng Wang, Yuan Qi, and Wei Chu. 2020a. Question directed graph attention network for numerical reasoning over text. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6759–6768, Online. Association for Computational Linguistics.

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

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. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3697–3711, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Ting-Rui Chiang and Yun-Nung Chen. 2019. Semantically-aligned equation generation for solving and reasoning math word problems. 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 2656–2668, Minneapolis, Minnesota. Association for Computational Linguistics.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek B Rao, Parker Barnes, Yi Tay, Noam M. Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Benton C. Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier García, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Oliveira Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Díaz, Orhan Firat, Michele Catasta, Jason Wei, Kathleen S. Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2022. Palm: Scaling language modeling with pathways. *ArXiv*, abs/2204.02311.

Haoyu Dong, Zhoujun Cheng, Xinyi He, Mengyu Zhou, Anda Zhou, Fan Zhou, Ao Liu, Shi Han, and Dongmei Zhang. 2022. Table pre-training: A survey on model architectures, pre-training objectives, and downstream tasks. In *IJCAI'2022 SURVEY TRACK*.

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, Minneapolis, Minnesota. Association for Computational Linguistics.

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

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, Hong Kong, China. Association for Computational Linguistics.

Hiroshi Iida, Dung Thai, Varun Manjunatha, and Mohit Iyyer. 2021. TABBIE: Pretrained representations of tabular data. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3446–3456, Online. Association for Computational Linguistics.

Jeonghwan Kim, Giwon Hong, Kyung-min Kim, Junmo Kang, and Sung-Hyon Myaeng. 2021. Have you seen that number? investigating extrapolation in question answering models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 7031–7037, Online and

Punta Cana, Dominican Republic. Association for Computational Linguistics.

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

Vivek Kumar, Rishabh Maheshwary, and Vikram Pudi. 2021. Adversarial examples for evaluating math word problem solvers. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 2705–2712, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Nate Kushman, Yoav Artzi, Luke Zettlemoyer, and Regina Barzilay. 2014. Learning to automatically solve algebra word problems. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 271–281, Baltimore, Maryland. Association for Computational Linguistics.

Yihuai Lan, Lei Wang, Qiyuan Zhang, Yunshi Lan, Bing Tian Dai, Yan Wang, Dongxiang Zhang, and Ee-Peng Lim. 2022. Mwptoolkit: An open-source framework for deep learning-based math word problem solvers. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(11):13188–13190.

Zhenwen Liang, Jipeng Zhang, Jie Shao, and Xiangliang Zhang. 2021. MWP-BERT: A strong baseline for math word problems. *CoRR*, abs/2107.13435.

Qian Liu, Bei Chen, Jiaqi Guo, Morteza Ziyadi, Zeqi Lin, Weizhu Chen, and Jian-Guang Lou. 2022. TAPEX: Table pre-training via learning a neural SQL executor. In *International Conference on Learning Representations*.

Shen-yun Miao, Chao-Chun Liang, and Keh-Yih Su. 2020. A diverse corpus for evaluating and developing English math word problem solvers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 975–984, Online. Association for Computational Linguistics.

Rodrigo Nogueira, Zhiying Jiang, and Jimmy Lin. 2021. Investigating the limitations of the transformers with simple arithmetic tasks. *CoRR*, abs/2102.13019.

Inkit Padhi, Yair Schiff, Igor Melnyk, Mattia Rigotti, Youssef Mroueh, Pierre Dognin, Jerret Ross, Ravi Nair, and Erik Altman. 2021. Tabular transformers for modeling multivariate time series. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3565–3569. IEEE.

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, Punta Cana, Dominican Republic. Association for Computational Linguistics.

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

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are NLP models really able to solve simple math word problems? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2080–2094, Online. Association for Computational Linguistics.

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

Xinyu Pi, Bing Wang, Yan Gao, Jiaqi Guo, Zhoujun Li, and Jian-Guang Lou. 2022b. Towards robustness of text-to-SQL models against natural and realistic adversarial table perturbation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2007–2022, Dublin, Ireland. Association for Computational Linguistics.

Jinghui Qin, Lihui Lin, Xiaodan Liang, Rumin Zhang, and Liang Lin. 2020. Semantically-aligned universal tree-structured solver for math word problems. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3780–3789, Online. Association for Computational Linguistics.

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, Hong Kong, China. Association for Computational Linguistics.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Yibin Shen and Cheqing Jin. 2020. Solving math word problems with multi-encoders and multi-decoders. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 2924–2934, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Georgios Spithourakis and Sebastian Riedel. 2018. Numeracy for language models: Evaluating and improving their ability to predict numbers. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2104–2115, Melbourne, Australia. Association for Computational Linguistics.

Sowmya S. Sundaram, Sairam Gurajada, Marco Fisichella, Deepak P, and Savitha Sam Abraham. 2022. Why are nlp models fumbling at elementary math? a survey of deep learning based word problem solvers. *ArXiv*, abs/2205.15683.

Dhanasekar Sundararaman, Shijing Si, Vivek Subramanian, Guoyin Wang, Devamanyu Hazarika, and Lawrence Carin. 2020. Methods for numeracy-preserving word embeddings. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4742–4753, Online. Association for Computational Linguistics.

Avijit Thawani, Jay Pujara, and Filip Ilievski. 2021a. Numeracy enhances the literacy of language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6960–6967, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Avijit Thawani, Jay Pujara, Filip Ilievski, and Pedro Szekely. 2021b. 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, Online. Association for Computational Linguistics.

Shyam Upadhyay and Ming-Wei Chang. 2017. Annotating derivations: A new evaluation strategy and dataset for algebra word problems. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 494–504, Valencia, Spain. Association for Computational Linguistics.

Eric Wallace, Yizhong Wang, Sujian Li, Sameer Singh, and Matt Gardner. 2019. Do NLP models know numbers? probing numeracy in embeddings. 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 5307–5315, Hong Kong, China. Association for Computational Linguistics.

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

Yonghui Wu, Mike Schuster, Z. Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason R. Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Gregory S. Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *ArXiv*, abs/1609.08144.

Zhipeng Xie and Shichao Sun. 2019. A goal-driven tree-structured neural model for math word problems. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 5299–5305. International Joint Conferences on Artificial Intelligence Organization.

Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. TaBERT: Pretraining for joint understanding of textual and tabular data. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8413–8426, Online. Association for Computational Linguistics.

Dongxiang Zhang, Lei Wang, Luming Zhang, Bing Tian Dai, and Heng Tao Shen. 2020a. The gap of semantic parsing: A survey on automatic math word problem solvers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(9):2287–2305.

Jipeng Zhang, Lei Wang, Roy Ka-Wei Lee, Yi Bin, Yan Wang, Jie Shao, and Ee-Peng Lim. 2020b. Graph-to-tree learning for solving math word problems. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3928–3937, Online. Association for Computational Linguistics.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103.

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

A Formal Definition of Perturbations

We provide the formalized definition of the perturbations as follows. In all definitions, "$\star$" denotes perturbed version.

Noise Perturbation. To apply noise perturbation to an number n, an variable X is uniformly sampled on the interval (1, 10). Then a fractional part corresponding to X is added the concerned number n, *i.e.*,

$$X \sim \mathcal{U}(1, 10)$$
$$n^\star = n + 0.1 \times \lfloor X \rfloor$$

Distribution Perturbation. The Distribution Perturbation changes the number distribution in the dataset by adding an normally distributed random variable X to the concerned number n. *I.e.*,

$$X \sim \mathcal{N}(\mu_d, \delta_d^2)$$
$$n^\star = n + \lfloor X \rfloor$$

In this paper we adopt $\mu_d = 1000$ and $\delta_d = 300$.

Language Perturbation. The concerned number string n_s is replaced by the English word describing the same quantity, *i.e.*,

$$n_s^\star = Num2Words(n_s)$$

Type Perturbation. To apply the Type Perturbation, the concerned number is expected to be an integral number. The number string n_s is concatenated with an extra ".0" string to change the type of the concerned number from integer to float-point, *i.e.*,

$$n_s^\star = Concat(n_s, Stringfy(.0))$$

Verbosity Perturbation. The Verbosity Perturbation aims to introduce irrelevant numbers without changing the semantics of the problem. To perturb a number string n_s, we concatenate it with an irrelevant number in parentheses, the irrelevant number is preceded by "not", *i.e.*,

$$X \sim \mathcal{N}(\mu_v, \delta_v^2)$$
$$n_s^\star = Concat(ns, (not, Stringfy(X)))$$

In this paper we adopt $\mu_v = 100$ and $\delta_v = 30$.

Extra Perturbation To apply the Extra Perturbation to a problem $(\mathcal{B}, \mathcal{P})$, an irrelevant sentence containing numbers from the corpus is added to the body $\mathcal{B}$, *i.e.*,

$$\mathcal{P}_\Delta = SampleOtherQs()$$
$$\mathcal{P}^\star = \mathcal{P} \oplus \mathcal{P}_\Delta$$
$$\mathcal{B}^\star = \mathcal{B}$$

Logic Perturbation To apply the Logic Perturbation to a problem $(\mathcal{B}, \mathcal{P})$, the prompt is altered to convert the problem logic used in the problem, *i.e.*,

$$\mathcal{P}^\star = ConvertLogic(\mathcal{P})$$
$$\mathcal{B}^\star = \mathcal{B}$$

Order Perturbation For the Order Perturbation, the sentence order in the problem body is manually altered in a manner that changes the order of number occurrence but not the problem logic, *i.e.*,

$$\mathcal{P}^\star = ChangeOrder(\mathcal{P})$$
$$\mathcal{B}^\star = \mathcal{B}$$

B Details of the Perturbing Process

B.1 The Filtering Conditions

The filtering conditions for Perturbing Algorithms is different across perturbations. The perturbations can be divided into two major categories: 1) perturbations that do not change the solving equation or final results (Language, Type, Verbosity, Extra, Order), and 2) perturbations that changes the solving equation or final results (Noise, Distri, Operation).

For perturbations in category 1), there is no limitation on the perturbing process, thus all questions naturally pass the the filtering condition.

For perturbations in category 2), the filtering conditions follow the principles of Unambiguity, Suitability and Visibility.

Unambiguity. The filtered question should have an unambiguous mapping between the number to be perturbed and the their location in the context. One example that violates this principle is when there are duplicated numbers in the problem body, then it cannot be determined which occurrence of the number affects the final result.

Suitability. The number to be perturbed should be suitable for the perturbation to be conducted. *E.g.* A float-point number should not be used as the target of the Noise perturbation which adds fractional part to integral numbers. In DNC, the Noise and Type perturbations requires the concerned number to be integral, and the Operation perturbation requires the question to match a manually created template.

Visibility. The concerned number should be occur in the the problem since the perturbations can only be applied to known input numbers.

B.2 The Formalized Perturbing Process

Algorithm 1: The Perturbing Process

Data: $D = \{D_{train}, D_{val}, D_{test}\}$
$AllPert = \{Noise, ...\}$ as in §4.2
$AllSet \in \{Attack, ...\}$ as in §4.3
$Filter = \{Integral, ...\}$ as in §B
Result: $D^\star = \{D^\star_{train}, D^\star_{val}, D^\star_{test}\}$

```
/* Decide perturbations to use */
```
$Perturbs \leftarrow SelectBy(AllPert, D)$;
```
/* Create perturbed dataset for
   each perturbation and setting */
```
for $setting \in AllSet$ **do**
  `/* Decide split to perturb */`
  $D_\Delta, D_{remain} \leftarrow SelectBy(D, setting)$;
  for $perturb \in Perturbs$ **do**
    $D^\star_\Delta \leftarrow \{\}$;
    for $d \in D_\Delta$ **do**
      if *Filter(d)* **then**
        $D^\star_\Delta \leftarrow D^\star_\Delta + perturb(d)$;
      end
      else
        $D^\star_\Delta \leftarrow D^\star_\Delta + d$;
      end
    end
    $D^\star_{perturb} = \{D^\star_\Delta, D_{remain}\}$
  end
end
$D^\star = \{D^\star_{perturb}\}|_{\, perturb \in Perturbs}$

C Hyperparameters

In our exploratory experiment, it it observed that while the hyperparameters such as learning rate and batchsize do affect the absolute performance of the models, they have a modest effect on the general trend of the models' strengths and weaknesses against the numerical perturbations. We hypothesize that this is due to the numerical capabilities of a model being contributed mostly by the model architecture instead of hyperparameters.

For example, when the hyperparameters are varying from the default setting (1e-5 for learning rate, 32 for batch size), the following results are observed:

On Graph2Tree, the results of changing the learning rate and batch size are shown in Table 7, the trend of results with the varied hyperparameters align with the default result as shown in Table 3.

Perturbation	Acc_{eq} lr=1e-2	Acc_{eq} batchsize=128	Default
Language	-4.10%	-9.84%	-7.65%
Type	3.28%	3.28%	0.27%
Noise	4.10%	1.64%	0.27%
Distribution	-4.92%	-10.66%	-6.56%
Verbosity	-33.61%	-38.52%	-33.33%
Extra	-44.26%	-57.38%	-53.83%
Logic	-18.03%	-27.87%	-28.42%
Order	-33.61%	-13.11%	-33.33%
Original	62.30%	71.31%	66.94%

Table 7: The results of the attack Acc_{eq} for Graph2Tree on ASDiv-a

Similar behavior can also be observed on large transformer-based model such as T5, as shown in Table 8:

Perturbation	Acc_{eq} lr=1e-4	Acc_{eq} batchsize=16	Default
Language	-17.21%	-17.21%	-18.85%
Type	-20.49%	-32.79%	-37.70%
Noise	-23.77%	-32.79%	-36.89%
Distribution	-18.03%	-13.11%	-16.39%
Verbosity	-39.34%	-38.52%	-41.80%
Extra	-40.16%	-21.31%	-25.41%
Logic	-27.05%	-28.69%	-29.51%
Order	-31.15%	-33.61%	-34.43%
Original	61.48%	63.11%	68.03%

Table 8: The results of the attack Acc_{eq} for T5 on ASDiv-a

Considering this observation, and the fact that the number of our experiment is large due to the combination of different models, datasets, DNC settings, and DNC perturbations, we chose one general setting to reduce search space. We chose the setting as close as possible to the reported setting in the original papers of Graph2Tree and T5. We verified that this setting provides sufficiently good performance to demonstrate the performance gap corresponding to the perturbations, since our experiment focused more on the performance of a same model checkpoint against the datasets before and after the perturbations.

D DNC Results

D.1 Raw Performance And Relative Performance Drop

We provide the original result in Table 9 and the relative performance drop in Table 10.

D.2 Observation Calculation Details

We denote the experiment results table in Table 11. The values, observation explanation, and the formula used are provided in Table 12.

E DNC Experiments that Are Not Applicable

The following types of experiments are not applicable in current DNC framework:

E.1 The Defense of the Logic perturbation on ASDiv-a

The Logic perturbation requires the problem to be perturbed in a way that the logic is changed while the semantics of the problem is still cohesive. This requirement proposes challenge on the scalability of the perturbation. For the Attack setting, we utilized manually annotated labels. However, under the Defense setting the perturbations are expected to automatically augment the dataset. Thus, the Defense setting results of Logic perturbation on ASDiv-a is not applicable.

E.2 Noise and Type perturbations on DROP-num and TATQA-a

DROP-num and TATQA-a do not provide supervision of the operand origins, therefore a mapping from the operands in equation to the context quantities cannot be built, which results in the Noise and Distribution perturbation not applicable on the DROP-num and TATQA-a datasets.

E.3 Logic Perturbation on DROP-num

DROP-num does not provide ground truth reasoning steps or logical forms, thus Logic perturbations that has dependency on the provided supervision is not applicable on DROP-num.

E.4 Order Perturbation on DROP-num

DROP-num is a reasonding dataset based on real-world paragraphs that usually have logical or temporal order information. Order perturbation breaks the semantic of the paragraph and will also confuse humans. Thus Order perturbation is not valid on DROP-num and the results are not applicable.

Configuration		ASDiv-a								DROP-num		TATQA-a
		T5		BART		GPT2		Graph2Tree		T5	BART	TagOps
Setting	Perturbation	Acc_{eq}	Acc_{ans}	Acc_{eq}	Acc_{ans}	Acc_{eq}	Acc_{ans}	Acc_{eq}	Acc_{ans}	Acc	Acc	Acc
Attack	Language	49.18%	54.10%	43.44%	45.90%	31.97%	32.79%	59.29%	61.20%	38.80%	35.62%	23.79%
	Type	30.33%	61.48%	34.43%	57.38%	27.05%	34.43%	67.21%	69.67%	41.72%	39.30%	37.07%
	Noise	31.15%	36.07%	48.36%	51.64%	34.43%	36.07%	67.21%	69.13%	-	-	-
	Distribution	51.64%	58.20%	37.70%	54.92%	31.15%	31.97%	60.38%	62.02%	-	-	-
	Verbosity	26.23%	28.69%	41.80%	43.44%	33.61%	33.61%	33.61%	34.70%	39.84%	37.04%	40.52%
	Extra	42.62%	45.08%	25.41%	27.05%	15.57%	16.39%	13.11%	13.93%	37.63%	38.69%	41.21%
	Logic	38.52%	45.08%	30.33%	37.70%	18.85%	21.31%	38.52%	46.72%	-	-	28.12%
	Order	33.61%	67.21%	33.61%	68.85%	16.39%	37.70%	33.61%	61.48%	-	-	43.53%
Defense	Language	55.74%	59.02%	47.54%	48.36%	46.72%	47.54%	59.29%	61.20%	49.49%	48.52%	34.83%
	Type	56.56%	60.66%	62.30%	66.39%	47.54%	49.18%	68.58%	70.49%	49.88%	49.41%	45.34%
	Noise	53.28%	58.20%	63.93%	68.03%	47.54%	49.18%	67.49%	68.85%	-	-	-
	Distribution	47.54%	52.46%	59.02%	63.11%	36.07%	36.07%	60.11%	62.57%	-	-	-
	Verbosity	52.46%	56.56%	61.48%	65.57%	43.44%	45.08%	66.67%	69.67%	44.29%	48.52%	44.67%
	Extra	68.03%	74.59%	64.75%	68.85%	27.05%	27.05%	46.72%	50.82%	38.10%	39.92%	33.28%
	Logic	-	-	-	-	-	-	-	-	-	-	56.05%
	Order	42.62%	68.85%	39.34%	65.57%	42.62%	68.85%	37.70%	60.66%	-	-	61.89%
Original	None	68.03%	72.95%	67.21%	72.95%	44.26%	45.08%	66.94%	68.58%	49.42%	50.36%	42.41%

Table 9: The raw results of the DNC Framework. Notations here follow the ones in the main experiment results

Configuration		ASDiv-a								DROP-num		TATQA-a
		T5		BART		GPT2		Graph2Tree		T5	BART	TagOps
Setting	Perturbation	Acc_{eq}	Acc_{ans}	Acc_{eq}	Acc_{ans}	Acc_{eq}	Acc_{ans}	Acc_{eq}	Acc_{ans}	Acc	Acc	Acc
Attack (Δ%)	Language	-27.71%	-25.84%	-35.37%	-37.08%	-27.78%	-27.27%	-11.43%	-10.76%	-21.49%	-29.26%	-43.90%
	Type	-55.42%	-15.73%	-48.78%	-21.35%	-38.89%	-23.64%	0.41%	1.59%	-15.59%	-21.96%	-12.60%
	Noise	-54.22%	-50.56%	-28.05%	-29.21%	-22.22%	-20.00%	0.41%	0.80%	-	-	-
	Distribution	-24.10%	-20.22%	-43.90%	-24.72%	-29.63%	-29.09%	-9.80%	-9.56%	-	-	-
	Verbosity	-61.45%	-60.67%	-37.80%	-40.45%	-24.07%	-25.45%	-49.80%	-49.40%	-19.38%	-26.44%	-4.47%
	Extra	-37.35%	-38.20%	-62.20%	-62.92%	-64.81%	-63.64%	-80.41%	-79.68%	-23.86%	-23.17%	-2.85%
	Logic	-43.37%	-38.20%	-54.88%	-48.31%	-57.41%	-52.73%	-42.45%	-31.87%	-	-	-33.69%
	Order	-50.60%	-7.87%	-50.00%	-5.62%	-62.96%	-16.36%	-49.80%	-10.36%	-	-	2.64%
Defense (Δ%)	Language	-18.07%	-19.10%	-29.27%	-33.71%	5.56%	5.45%	-11.43%	-10.76%	0.14%	-3.65%	-17.89%
	Type	-16.87%	-16.85%	-7.32%	-8.99%	7.41%	9.09%	2.45%	2.79%	0.93%	-1.88%	6.91%
	Noise	-21.69%	-20.22%	-4.88%	-6.74%	7.41%	9.09%	0.82%	0.40%	-	-	-
	Distribution	-30.12%	-28.09%	-12.20%	-13.48%	-18.52%	-20.00%	-10.20%	-8.76%	-	-	-
	Verbosity	-22.89%	-22.47%	-8.54%	-10.11%	-1.85%	0.00%	-0.41%	1.59%	-10.38%	-3.65%	5.31%
	Extra	0.00%	2.25%	-3.66%	-5.62%	-38.89%	-40.00%	-30.20%	-25.90%	-22.90%	-20.72%	-21.54%
	Logic	-	-	-	-	-	-	-	-	-	-	32.16%
	Order	-37.35%	-5.62%	-41.46%	-10.11%	-3.70%	52.73%	-43.67%	-11.55%	-	-	45.92%
Original	None	68.03%	72.95%	67.21%	72.95%	44.26%	45.08%	66.94%	68.58%	49.42%	50.36%	42.41%

Table 10: The relative drop results of the DNC Framework. Notations here follow the ones in the main experiment results

	Configuration		A	B	C	D	E	F	G	H	I	J	K
			ASDiv-a								DROP-num		TATQA-a
			T5		BART		GPT2		Graph2Tree		T5	BART	TagOps
	Setting	Perturbation	Acc_{eq}	Acc_{ans}	Acc_{eq}	Acc_{ans}	Acc_{eq}	Acc_{ans}	Acc_{eq}	Acc_{ans}	Acc	Acc	Acc
1	Attack (Δ)	Language	-18.85%	-18.85%	-23.77%	-27.05%	-12.30%	-12.30%	-7.65%	-7.38%	-10.62%	-14.73%	-18.62%
2		Type	-37.70%	-11.48%	-32.79%	-15.57%	-17.21%	-10.66%	0.27%	1.09%	-7.70%	-11.06%	-5.34%
3		Noise	-36.89%	-36.89%	-18.85%	-21.31%	-9.84%	-9.02%	0.27%	0.55%	-	-	-
4		Distribution	-16.39%	-14.75%	-29.51%	-18.03%	-13.11%	-13.11%	-6.56%	-6.56%	-	-	-
5		Verbosity	-41.80%	-44.26%	-25.41%	-29.51%	-10.66%	-11.48%	-33.33%	-33.88%	-9.58%	-13.31%	-1.90%
6		Extra	-25.41%	-27.87%	-41.80%	-45.90%	-28.69%	-28.69%	-53.83%	-54.64%	-11.79%	-11.67%	-1.21%
7		Logic	-29.51%	-27.87%	-36.89%	-35.25%	-25.41%	-23.77%	-28.42%	-21.86%	-	-	-14.29%
8		Order	-34.43%	-5.74%	-33.61%	-4.10%	-27.87%	-7.38%	-33.33%	-7.10%	-	-	1.12%
9	Defense (Δ)	Language	-12.30%	-13.93%	-19.67%	-24.59%	2.46%	2.46%	-7.65%	-7.38%	0.07%	-1.84%	-7.59%
10		Type	-11.48%	-12.30%	-4.92%	-6.56%	3.28%	4.10%	1.64%	1.91%	0.46%	-0.95%	2.93%
11		Noise	-14.75%	-14.75%	-3.28%	-4.92%	3.28%	4.10%	0.55%	0.27%	-	-	-
12		Distribution	-20.49%	-20.49%	-8.20%	-9.84%	-8.20%	-9.02%	-6.83%	-6.01%	-	-	-
13		Verbosity	-15.57%	-16.39%	-5.74%	-7.38%	-0.82%	0.00%	-0.27%	1.09%	-5.13%	-1.84%	2.25%
14		Extra	0.00%	1.64%	-2.46%	-4.10%	-17.21%	-18.03%	-20.22%	-17.76%	-11.32%	-10.44%	-9.14%
15		Logic	-	-	-	-	-	-	-	-	-	-	13.64%
16		Order	-25.41%	-4.10%	-27.87%	-7.38%	-1.64%	23.77%	-29.23%	-7.92%	-	-	19.47%
	Original	None	68.03%	72.95%	67.21%	72.95%	44.26%	45.08%	66.94%	68.58%	49.42%	50.36%	42.41%

Table 11: The main experiment result table with cell coordinates

Value	Explanation	Formula
19.66%	Average performance drop of all models caused by perturbations according to the Semantic Parsing stage	AVERAGE(B5:B8,D5:D8,F5:F8,H5:H8,I5:I8,J5:J8,K5:K8)
13.15%	Average performance drop of all models caused by perturbations according to the Numerical Parsing stage	AVERAGE(B1:B4,D1:D4,F1:F4,H1:H4,I1:I4,J1:J4,K1:K4)
17.42%	Average performance drop of all Transformer-based models caused by perturbations according to the Semantic Parsing stage	AVERAGE(B5:B8,D5:D8,F5:F8,I5:I8,J5:J8,K5:K8)
3.07%	Average performance drop of the Graph2Tree system caused by perturbations according to the Semantic Parsing stage	AVERAGE(B1:B4,D1:D4,F1:F4,I1:I4,J1:J4,K1:K4)
17.96%	Average performance recovery of all models caused by perturbations according to the Semantic Parsing stage	AVERAGE(B13:B16,D13:D16,F13:F16,H13:H16,I13:I16,J13:J16,K13:K16)
6.52%	Average performance recovery of all models caused by perturbations according to the Numerical Parsing stage	AVERAGE(B9:B12,D9:D12,F9:F12,H9:H12,I9:I12,J9:J12,K9:K12)
12.53%	Average performance recovery of all Transformer-based models caused by perturbations according to the Semantic Parsing stage	AVERAGE(B13:B16,D13:D16,F13:F16,I13:I16,J13:J16,K13:K16)
11.58%	Average performance recovery of the Graph2Tree system caused by perturbations according to the Semantic Parsing stage	AVERAGE(B9:B12,D9:D12,F9:F12,I9:I12,J9:J12,K9:K12)

Table 12: The Value, Explanation, And Formula Used of The Experiment Observations.