Chaining Simultaneous Thoughts for Numerical Reasoning

Zhihong Shao, Fei Huang, Minlie Huang*
The CoAI group, DCST, Tsinghua University, Institute for Artificial Intelligence;
State Key Lab of Intelligent Technology and Systems;
Beijing National Research Center for Information Science and Technology;
Tsinghua University, Beijing 100084, China
{szh19, f-huang18}@mails.tsinghua.edu.cn
aihuang@tsinghua.edu.cn

Abstract

Given that rich information is hidden behind ubiquitous numbers in text, numerical reasoning over text should be an essential skill of AI systems. To derive precise equations to solve numerical reasoning problems, previous work focused on modeling the structures of equations, and has proposed various structured decoders. Though structure modeling proves to be effective, these structured decoders construct a single equation in a pre-defined autoregressive order, potentially placing an unnecessary restriction on how a model should grasp the reasoning process. Intuitively, humans may have numerous pieces of thoughts popping up in no pre-defined order; thoughts are not limited to the problem at hand, and can even be concerned with other related problems. By comparing diverse thoughts and chaining relevant pieces, humans are less prone to errors. In this paper, we take this inspiration and propose CANTOR, a numerical reasoner that models reasoning steps using a directed acyclic graph where we produce diverse reasoning steps simultaneously without pre-defined decoding dependencies, and compare and chain relevant ones to reach a solution. Extensive experiments demonstrated the effectiveness of CANTOR under both fully-supervised and weakly-supervised settings.

1 Introduction

Numerical reasoning over text is an essential skill for a neural model to help analyze rich numerical information from large-scale textual data (Chen et al., 2021). Many question answering benchmarks (Dua et al., 2019; Patel et al., 2021) have been created to promote the numerical reasoning ability of neural models, where, typically, models are required to answer questions about given contexts with numerical answers. This is challenging, as it requires comprehensive structural analyses of text as well as precise and possibly complex deduction.

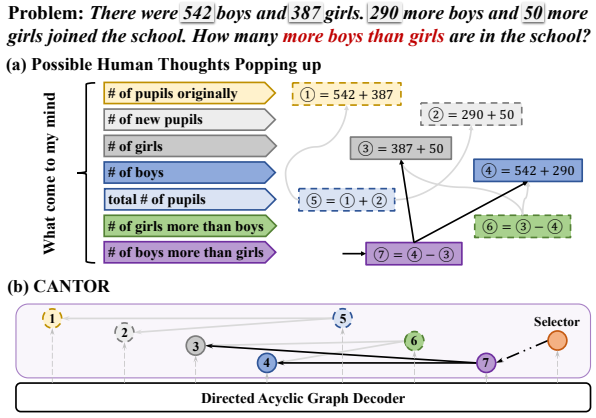

Figure 1: (a) Possible pieces of human thoughts that pops up in no pre-defined order; (b) How our model captures the reasoning process similarly. Reasoning steps inside solid frames and dashed frames are necessary and loosely-relevant ones, respectively.

Existing models mostly decode the equations and return the execution results. To better exploit structures of equations, many complex structured decoders (Xie and Sun, 2019; Cao et al., 2021) have been proposed and significantly outperform sequential decoding (Tan et al., 2021). However, all these methods construct a single equation in a pre-defined order (e.g., top-down or bottom-up order), which may place an unnecessary restriction on how a model should grasp the reasoning process.

Intuitively, after reading a reasoning problem, humans may have several pieces of thoughts which pop up in no pre-defined order, and finalize a solution by comparing and chaining relevant pieces. Take Fig 1(a) for example. Possible thoughts include necessary reasoning steps (e.g., how to get the number of boys and girls separately) and loosely-relevant ones (e.g., those learned from previous similar questions like "how many more girls than boys are in the school?"). There is arguably no pre-defined strict order where a thought should conditionally emerge after some other thoughts. By comparing these diverse thoughts, we finally select

*Corresponding author: Minlie Huang.

Findings of the Association for Computational Linguistics: EMNLP 2022, pages 2533–2547
December 7-11, 2022

and chain proper ones to reach a solid solution, which will be less prone to mistakes.

In this paper, we propose **CANTOR**, which compares and **C**hains simult**AN**eous **Th**O**ughts for numerical **R**easoning. As in Fig 1(b), CANTOR constructs a Directed Acyclic Graph (DAG) of diverse reasoning steps in a non-autoregressive way: all vertices are produced simultaneously, which correspond to operations like addition, and edges in the graph are constructed by chaining operations with their best-matched operands; the final equation is a selected sub-graph in the whole DAG. With no pre-defined decoding order, logical dependencies among reasoning steps are freely captured by the model internally. With our training methods, CANTOR captures diverse reasoning steps at different vertices, and learns to prune away possibly-distracting candidates during both training and inference, resulting in chaining reasoning steps that are more consistent with given problems.

To summarize, compared with previous models with structured decoding, CANTOR has no pre-defined restrictions on the decoding dependencies while also benefiting from modeling the structures of equations. Besides, by comparing diverse reasoning steps and chaining logically consistent ones, our model is less prone to errors. Our model establishes a new state-of-the-art record on two math word problem datasets under the fully-supervised setting, and is also applicable to weakly-supervised scenarios (where problems are only annotated with final answers, and the equations are unavailable) with significant improvements over baselines. Though not directly comparable, on two numerical reasoning datasets, fully-supervised CANTOR achieves even higher accuracies than hundreds of times larger language models (e.g., PaLM-62B (Chowdhery et al., 2022)) that use the effective chain-of-thought prompting technique (Wei et al., 2022), demonstrating CANTOR's great potential.

2 Related Work

Numerical Reasoning Numerical reasoning tasks can be formulated in many ways (Mishra et al., 2022), such as (1) question answering with numerical answers directly derived from arithmetic operations (Koncel-Kedziorski et al., 2016; Wang et al., 2017; Dua et al., 2019; Amini et al., 2019; Miao et al., 2020; Patel et al., 2021), (2) or other tasks like quantitative natural language inference (Ravichander et al., 2019) whose expected outputs are non-numerical but require implicit arithmetic reasoning. In this work, we focus on the former type of task which is widely studied. To generate equations precisely, previous work proposed to enhance number-related representations in problem encoding (Zhang et al., 2020; Shen and Jin, 2020; Liang et al., 2021), re-rank equation samples with a verifier (Shen et al., 2021; Cobbe et al., 2021), or exploit the structures of equations with complex top-down tree-structured decoding (Xie and Sun, 2019; Li et al., 2022) or bottom-up DAG-structured decoding (Cao et al., 2021; Jie et al., 2022). Our numerical reasoner also models equations with DAGs but with three major differences: (1) there is no pre-defined decoding order which may place unnecessary burden on how a model should learn the dependencies among operations; (2) the decoding process is largely simplified, which is reduced to simultaneous predictions of an operator and operands at each vertex of a graph; (3) our model explores diverse operations in a DAG and is trained to compare and chain relevant ones, so that logical consistency between given problems and equations are better captured during both training and inference.

Non-Autoregressive Decoding Our model is also relevant to non-autoregressive decoding. For machine translation, non-autoregressive translation (Gu et al., 2018; Ghazvininejad et al., 2020; Du et al., 2021) aims at fast inference; the recently proposed DA-Transformer (Huang et al., 2022), which utilizes a DAG to capture diverse translations, has made great progress in bridging the performance gap with autoregressive models. Recent work has also proposed non-autoregressive models for efficient task-oriented semantic parsing (Babu et al., 2021; Shrivastava et al., 2021), which achieved comparable performance with autoregressive parsers. All these methods model a target as a sequence and adopt token-wise decoding (one token at a position). By contrast, we model a target as a DAG and adopt step-wise decoding (one complete reasoning step at each vertex), which facilitates structure modeling and learning meaningful vertex representations. Experimental results show that our model significantly outperforms both autoregressive and non-autoregressive baselines. Notably, for open text generation, autoregressive methods are probably still the better choice for strong probabilistic modeling of diverse targets. However, for the numerical reasoning task we focus on, it

is the logical relationships among quantities (both known and unknown in a given problem) that matter, and non-autoregressive methods, with proper designs, suffice to decode equations precisely and can provide new perspectives on how numerical reasoning can be better grasped by neural models.

3 Task Definition

Given a problem description X which mentions a list of numbers $\mathcal{N} = \{n_1, n_2, ..., n_{|\mathcal{N}|}\}$, our task is to return the numerical answer A which is derived from an equation Y that takes arithmetic operations (e.g., addition, subtraction, multiplication, division, and exponentiation) on $\mathcal{N}$ as well as a set of pre-defined constants $\mathcal{C} = \{c_1, c_2, ..., c_{|\mathcal{C}|}\}$.

For scenarios that consider only binary operators[1], a ground-truth equation Y can be formally defined as follows:

$$\begin{aligned} Y =& \{y_1, y_2, ..., y_{|Y|}\},\ y_i = \langle y_i^f, y_i^a, y_i^b \rangle \\ s.t.\ & y_i^f \in \mathcal{F} \wedge y_i^a, y_i^b \in \mathcal{C} \cup \mathcal{N} \cup \{y_k | k < i\} \end{aligned}$$

where $\mathcal{F}$ is the set of pre-defined operators. y_i is an operation that applies the operator y_i^f to the two operands y_i^a and y_i^b. Y can be directly transformed into a DAG with c_i, n_i, and y_i being vertices, and $y_i \rightarrow y_i^a$ and $y_i \rightarrow y_i^b$ being edges. The final operation (the root vertex) $y_{|Y|}$ returns the answer.

4 CANTOR

4.1 Overview

We propose to model diverse reasoning steps with a DAG. Vertices of the graph correspond to reasoning steps which are decoded in parallel. This is analogous to humans' burst of thoughts after reading a reasoning problem. No pre-defined restriction is placed on how a reasoning step should conditionally depend on others; logical dependencies among reasoning steps are captured by the model internally. Our DAG also allows the model to explore diverse reasoning steps at different vertices, including necessary or wrong ones; the model is trained to compare the semantics of diverse reasoning steps and chain the most proper ones to be the final equation, which benefits model performance.

4.2 Architecture

Our model (Fig 2) comprises a pre-trained Transformer encoder (e.g., RoBERTa) and a shallow Transformer-based DAG decoder. The encoder encodes a problem X; from the encoder outputs, we can obtain the representations of mentioned numbers $\mathbf{N} = [\mathbf{n_1}, \mathbf{n_2}, ..., \mathbf{n}_{|\mathcal{N}|}] \in \mathbb{R}^{d \times |\mathcal{N}|}$ (d is the hidden size). The DAG decoder, with positional embeddings as inputs and cross attention over encoder outputs, produces representations for L vertices $\mathcal{V} = \{v_1, v_2, ..., v_L\}$ in a non-autoregressive way, which are denoted as $\mathbf{V} = [\mathbf{v_1}, \mathbf{v_2}, ..., \mathbf{v_L}] \in \mathbb{R}^{d \times L}$. Each vertex representation encodes the semantics of a reasoning step, including its operator, the expected operands, and the meaning of the resulting quantity. We then verbalize the operator for each vertex and chain it with its best-matched operands in parallel, and finally, select one root vertex and return its execution result. The selected root vertex along with its vertex descendants constitutes a decoded sub-graph, which is also the DAG representation of an equation. Let Z be the decoded sub-graph, which can be formulated as:

$$\begin{aligned} Z =& \{z_1, z_2, ..., z_{|Z|}\},\ z_j = \langle p_j, z_j^f, z_j^a, z_j^b \rangle \\ s.t.\ & 1 \le p_1 < p_2 < ... < p_{|Z|} \le L \\ & z_j^f \in \mathcal{F} \wedge z_j^a, z_j^b \in \mathcal{C} \cup \mathcal{N} \cup \{z_k | k < j\} \end{aligned}$$

where z_j is the operation for the vertex at position p_j, with z_j^f being the operator, and z_j^a and z_j^b being its operands. $p_{|Z|}$ is the index of the root vertex, and $\{p_j | j < |Z|\}$ are indices of its vertex descendants.

The probability of a target equation Y can be formulated as follows:

$$P_\theta(Y|X) = \sum_Z P_\theta(Y|Z, X) P_\theta(Z|X) \tag{1}$$

Definition of $P_\theta(Y|Z, X)$: Given Y and Z, $P_\theta(Y|Z, X)$ is defined to be 1 if and only if mapping y_i to the vertex at position p_i ($\forall 1 \le i \le |Y|$) produces Z exactly; otherwise, $P_\theta(Y|Z, X)$ is 0.

Therefore, $P_\theta(Y|X)$ can be re-written as:

$$\begin{aligned} P_\theta(Y|X) &= \sum_{Z \in \Gamma} P_\theta(Z|X) \\ \Gamma &= \{Z | P_\theta(Y|Z, X) = 1\} \end{aligned} \tag{2}$$

Any $Z \in \Gamma$ is a DAG representation of Y (see the example in Fig 2). For a given Y, Γ can be created by enumerating $\{p_1, ..., p_{|Y|}\}$ that satisfies $1 \le p_1 < ... < p_{|Y|} \le L$, and then mapping y_i to v_{p_i} ($\forall 1 \le i \le |Y|$). Therefore, $|\Gamma| = \binom{L}{|Y|}$.

Definition of $P_\theta(Z|X)$: $P_\theta(Z|X)$ can be further decomposed based on operations in Z:

$$P_\theta(Z|X) = P_r(p_{|Z|}|X) \prod_{j=1}^{|Z|} P_z(z_j | p_j, X) \tag{3}$$

$$P_z(z_j | p_j, X) = P_f(z_j^f | p_j, X) P_a(z_j^a | p_j, X) P_b(z_j^b | p_j, X)$$

[1] In this paper, we only consider binary operators while it is feasible to extend our model to utilize other n-ary operators.

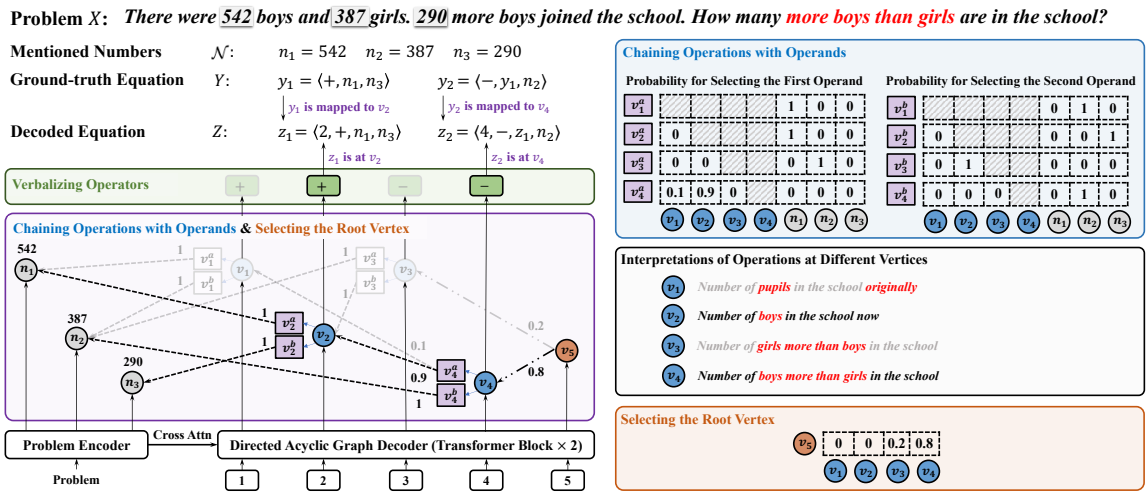

Figure 2: Overview of CANTOR. CANTOR models diverse operations using a DAG. Each vertex corresponds to an operation, which is chained with its operands via edges in the graph. We decode an equation by simultaneously verbalizing operators at each vertex, chaining operations with operands, and selecting the root vertex; the selected root vertex along with all its descendants is the resulting equation in a DAG format. In this example, the ground-truth equation Y can be represented by the decoded sub-graph Z, as mapping y_1 to v_2 and y_2 to v_4 produces Z exactly.

where $P_r(\cdot)$ and $P_z(\cdot)$ are the probability functions of the root vertex and an operation, respectively; $P_f(\cdot)$ and $P_a(\cdot)$ ($P_b(\cdot)$) are for operator verbalization and operand matching, respectively.

4.2.1 Verbalizing Operators

We verbalize an operator for each vertex based on its representation:

$$P_f(z_j^f|p_j, X) = \text{softmax}(\mathbf{W_f v_{p_j}})$$

where $\mathbf{W_f} \in \mathbb{R}^{|\mathcal{F}|\times d}$ is trainable parameters.

4.2.2 Chaining Operations with Operands

Each operation is connected with its best-matched operands chosen from all available quantities (including the other operations, constants, and mentioned numbers). Let $\mathbf{C} = [\mathbf{c_1}, \mathbf{c_2}, ..., \mathbf{c_{|C|}}]^\top$ be the embedding matrix for pre-defined constants. Then the representation matrix for all quantities is denoted as $\mathbf{Q} = [\mathbf{V}, \mathbf{C}, \mathbf{N}] \in \mathbb{R}^{d\times(L+|\mathcal{C}|+|\mathcal{N}|)}$. The probability distribution over candidates when predicting the first operand for the vertex at position p_j can be computed as follows:

$$P_a(z_j^a|p_j, X) = \text{softmax}(\frac{(\mathbf{W_q Q})^\top \mathbf{v^a_{p_j}}}{\sqrt{d}}),\ \mathbf{v^a_{p_j}} = \mathbf{W_a v_{p_j}}$$

The probability for predicting the second operand $P_b(\cdot)$ can be computed likewise. $\mathbf{W_q}, \mathbf{W_a}, \mathbf{W_b} \in \mathbb{R}^{d\times d}$ are trainable parameters. To avoid cycles in the graph, we apply probability masking so that a vertex can not use itself or vertices with larger indices as its operands.

4.2.3 Selecting the Root Vertex and Finalizing the Equation

The final equation is represented by a sub-graph of the whole DAG, which comprises a selected root vertex and all its descendants. We introduce a special vertex v_{L+1} at position $L+1$ of the decoder, and use its representation to select the best-matched root vertex:

$$P_r(p_{|Z|}|X) = \text{softmax}(\frac{(\mathbf{W_q V})^\top \mathbf{v_{L+1}}}{\sqrt{d}})$$

where $\mathbf{v_{L+1}}$ is the representation of v_{L+1}, computed the same way as other vertex representations.

4.3 Training

To capture diverse reasoning steps at different vertices, we explore four training methods[2], namely, naïve mapping, hard EM, MML, and hard EM with annealing. Notably, as will be discussed by Section 5.7.3, in practice, one has no need to consider all four training methods; hard EM with annealing should be the default choice.

4.3.1 Naïve Mapping

A naïve way of mapping from Y to $\mathcal{V}$ is to map y_i to v_i ($\forall 1 \le i \le |Y|$). Let Z' be the resulting sub-graph, then the training objective is:

$$\mathcal{L} = -\log P_\theta(Z'|X) \quad (4)$$

which leaves $\{v_j||Y| < j \le L\}$ unused.

[2]See Appendix B for implementation details of hard EM and MML.

4.3.2 Hard EM

Hard EM is to optimize the probability of Z^* that best aligns with Y:

$$\mathcal{L} = -\log P_\theta(Z^*|X),\ Z^* = arg\max_{Z\in\Gamma} P_\theta(Z|X) \quad (5)$$

As $|\Gamma|$ can be quite large[3], we use beam search to find Z^* approximately, which is feasible as $P_\theta(Z|X)$ can be factorized into probabilities of constituent operations of Z (Eq 3). Notably, the probability of an operation depends on which vertices its operands (if being operations) are mapped to. We search Z by iteratively determining where to map y_i, until $y_{|Y|}$ is settled.

4.3.3 MML

MML optimizes the marginal likelihood of Z:

$$\mathcal{L} = -\log \sum_{Z\in\Gamma} P_\theta(Z|X) \quad (6)$$

Marginalization is expensive due to the large size of Γ. We therefore adopt a strong (but risky) assumption so that we can use dynamic programming to marginalize $P_\theta(Z|X)$ in polynomial time. Specifically, for any operation y_i, we assume that the two sub-graphs rooted at y_i^a and y_i^b respectively (in the DAG counterpart of Y) are independently mapped to $\{v_1, v_2, ..., v_L\}$. Notably, with this assumption, we in fact marginalize $P_\theta(Z|X)$ over a superset of Γ and even allow mapping multiple operations to a single vertex. However, we empirically found that MML (with this assumption) works well on short equations[4], and can be used to warm up hard EM.

4.3.4 Hard EM with Annealing

To avoid optimizing the model on its early decisions, we follow Min et al. (2019) to apply annealing to hard EM: we optimize the model using MML for τ training steps and use hard EM afterwards.

4.4 Inference

During inference, we adopt greedy decoding which conducts the argmax operation for operator prediction, operand matching, and root vertex selection in parallel. The execution result at the root vertex is returned as the numerical answer.

5 Experiments

5.1 Datasets

We applied CANTOR to Math Word Problem (MWP) solving under the fully-supervised setting and discrete reasoning under a weakly-supervised setting. Data statistics are shown in Table 1.

Dataset	Train	Dev	Test	$\lvert\mathcal{C}\rvert$	$\mathcal{F}$	$\max\lvert Y\rvert$
Fully-Supervised MWP Solving						
MathQA	16,191	2,411	1,605	24	$\{+,-,\times,/,**\}$	15
SVAMP	3,138	-	1,000	17	$\{+,-,\times,/\}$	7
Weakly-Supervised Discrete Reasoning						
DROP_{num}	46,973	5,850	-	2	$\{+,-\}$	1
DROP	77,409	9,536	9,615	2	$\{+,-\}$	1

Table 1: Data statistics. Note that DROP_{num} and DROP are only annotated with answer texts but not equations Y; we followed previous work to enumerate binary operations that evaluate to the answers ($\max|Y| = 1$).

Fully-Supervised MWP Solving

(a) MathQA (Amini et al., 2019) consists of GRE level math problems from multiple domains. We used the dataset from Jie et al. (2022) which has wrongly-annotated instances removed.

(b) SVAMP (Patel et al., 2021) was created for robustness evaluation, which consists of problems from the ASDiv-A dataset (Miao et al., 2020) with manual perturbations. We strictly followed Patel et al. (2021) to use both MAWPS (Koncel-Kedziorski et al., 2016) and ASDiv-A for training and SVAMP for testing.

Weakly-Supervised Discrete Reasoning

(a) DROP_{num} (Dua et al., 2019) consists of all problems with numerical answers from the reading comprehension dataset called DROP. Problems are only annotated with final answers but not the corresponding equations.

(b) DROP is a reading comprehension dataset consisting of problems with different types of answers, e.g., number, date, and span(s).

5.2 Metrics

For MWP solving, we evaluated models with **value accuracy** and **equation accuracy**. Previous work evaluated equation accuracy with string matching, failing to recognize positive equations that are structurally different from the ground-truth. In our evaluation, an equation is considered correct if it has consistent results with the annotated equation for 100 random replacements of numbers mentioned in the problem. For discrete reasoning on DROP, we followed previous work to use **F1**.

5.3 Baselines

We considered the following three categories:

Sequential Models generate an equation sequentially based on a given problem. mBERT2Seq (Tan

[3]Suppose $L = 60$ and $|Y| = 15$, then $|\Gamma| \approx 5 \times 10^{13}$.

[4]We provide detailed discussion on the properties of our MML in Appendix C.

Model	Dev	Test
Sequential Model		
mBERT2Seq (Tan et al., 2021)	-	77.1
Structured Model		
Graph2Tree (Zhang et al., 2020)	-	69.5
BERT2Tree (Li et al., 2022)	-	73.8
DEDUCTREASONER (Jie et al., 2022)	-	78.6
CANTOR	**81.7**	**82.9**

Table 2: Value accuracy on MathQA.

Model	Test
Sequential Model	
GPT-2 (Lan et al., 2021)	25.7
RoBERTaGen (Lan et al., 2021)	30.3
Structured Model	
RoBERTa-Graph2Tree (Patel et al., 2021)	43.8
BERT2Tree (Li et al., 2022)	32.4
DEDUCTREASONER (Jie et al., 2022)	45.1
CANTOR	**49.6**$_{\pm 0.63}$

Table 3: Value accuracy on SVAMP.

et al., 2021) comprises a multilingual BERT (Devlin et al., 2019) encoder and an LSTM (Hochreiter and Schmidhuber, 1997) decoder. We also compared our model with two sequential models from (Lan et al., 2021), namely, GPT-2 (Radford et al., 2019) and RoBERTaGen (Liu et al., 2019).

Structured Models utilize structured autoregressive decoders to generate an equation. Graph2Tree (Zhang et al., 2020) and DEDUCTREASONER (Jie et al., 2022) are the representative tree-structured model and DAG-structured model, respectively.

Tagging-based Models refer to the arithmetic modules of those modular networks dominant on DROP, which assign a plus, minus, or zero to each constant and number mentioned in a problem, and return the sum of the signed numbers. TASE (Segal et al., 2020) is a representative modular network which consists of modules specialized for different types of answers, e.g., a tagging-based arithmetic module, a count module, and modules for span-typed answers. We referred to a TASE model with only an arithmetic module as TASE$_{\text{arith}}$.

5.4 Implementation Details

For all experiments, we used two Transformer blocks (Vaswani et al., 2017) as the DAG decoder, which was trained with random initialization.

For MWP solving, we used RoBERTa$_{\text{base}}$ as the problem encoder. We experimented with different training methods whose effect on model performance will be discussed in Section 5.7.3 with the graph size L set to 60 and the beam size B for hard EM set to 20. We further investigated the effect of graph size L (Table 9 in Section 5.7.3) and beam size B (Table 13 in Appendix D.1). The best model on MathQA used hard EM with annealing ($\tau = 2,000, B = 20$), with $L = 80$, and the best model on SVAMP used MML, with $L = 60$. Following previous work, all experiments on SVAMP were run with 5 random seeds, with both the average performance and standard deviation reported.

For discrete reasoning on DROP, we followed TASE to use RoBERTa$_{\text{large}}$ for encoding and MML for training. L was chosen from $\{5, 10\}$ based on F1. The best models on DROP$_{\text{num}}$ and DROP used $L = 5$ and $L = 10$, respectively.

Breakdown	MathQA				SVAMP			
	Baseline		CANTOR		Baseline		CANTOR	
	Equ.	Val.	Equ.	Val.	Equ.	Val.	Equ.	Val.
Breakdown w.r.t. # Operation								
1	76.4	79.1	**78.2**	**80.0**	48.8	49.1	**54.9**	**55.2**
2	81.0	83.5	**83.1**	**84.8**	**31.2**	**32.1**	30.7	31.6
3	80.8	83.6	**82.6**	**86.7**	-	-	-	-
4	78.5	82.0	**81.3**	**84.4**	-	-	-	-
≥ 5	65.7	71.3	**74.4**	**79.4**	-	-	-	-
Breakdown w.r.t. Equation Novelty								
Seen	90.5	91.4	**95.2**	**96.1**	48.8	49.2	**53.5**	**53.8**
Unseen	34.5	45.8	**38.3**	**49.3**	12.2	13.9	**15.8**	**16.9**
Overall Performance								
Full	74.7	78.6	**79.2**	**82.9**	44.6	45.1	**49.2**	**49.6**

Table 4: Breakdowns of performance on the MWP solving task. *Baseline* refers to the previous best model DEDUCTREASONER. *Equ.* and *Val.* are equation accuracy and value accuracy, respectively.

Variations	Baseline		CANTOR	
	Equ.	Val.	Equ.	Val.
Question Sensitivity	21.6	22.3	**29.6**	**30.3**
Reasoning Ability	49.5	49.8	**53.2**	**53.4**
Structural Invariance	37.3	38.1	**42.4**	**43.2**

Table 5: A breakdown of robustness evaluation w.r.t. different variations in SVAMP. *Baseline* refers to the previous best model DEDUCTREASONER. *Equ.* and *Val.* are equation accuracy and value accuracy, respectively.

5.5 Results for MWP Solving

As shown by Table 2 and Table 3, CANTOR established a new state-of-the-art record on MathQA and SVAMP with large improvements. The fine-grained analyses in Table 4 and Table 5 show that CANTOR (1) outperforms the best baseline on nearly all problems of different levels of complexity measured by the number of operations needed,

(2) is better at exploiting equation templates[5] seen in training or creating novel ones to solve problems, (3) and is more robust to different types of variations, including those that evaluate question sensitivity (whether questions asked in problems are ignored in prediction), reasoning ability (how predictions are adjusted to subtle changes in given problems), and structural invariance (whether predictions are invariant to structural changes of given problems that preserve the reasoning logic).

5.6 Results for Discrete Reasoning

Model	Dev
$\text{TASE}_{\text{arith}}$	76.4
CANTOR	**78.1**

(a) DROP_{num}

Model	Dev	Number (Dev)	Test
TASE	83.58	81.38	83.62
w/ CANTOR	**83.93**	**81.95**	**84.25**

(b) DROP

Table 6: F1 scores on DROP_{num} and DROP. *w/ CANTOR* is a TASE model that replaces the original tagging-based arithmetic module with CANTOR; all modules share one problem encoder.

CANTOR is also applicable to weakly-supervised scenarios where only final answers are annotated. Given problem-answer pairs $\{\langle X, A\rangle\}$, if it is feasible to find Y that evaluates to A, we can adapt hard EM, MML, and hard EM with annealing for weakly-supervised training by simply re-defining Γ for the objective functions as follows:

$$\Gamma = \{Z|\exists Y\ P(A|Y)P_\theta(Y|Z,X) = 1\}$$

where $P(A|Y)$ is 1 if and only if Y evaluates to A.

For weakly-supervised training on DROP_{num}, we followed TASE to enumerate Y by searching addition or subtraction of two numbers, and used MML for training[6]. As each Y has only one operation, MML conducts exact marginalization over Γ.

As shown by Table 6a, CANTOR significantly outperforms $\text{TASE}_{\text{arith}}$ on DROP_{num}. If using CANTOR as a drop-in replacement for the arithmetic module of TASE, we can obtain further improvements on DROP (Table 6b).

5.7 Ablation Study

5.7.1 No Pre-defined Order Restrictions

To investigate the effect of removing restrictions on decoding dependencies, we considered a variant of CANTOR called **vanilla CANTOR**, which also produces all operations in parallel, but is not designed to have diverse and possibly redundant operations for comparisons in both operand matching and root vertex selection. Specifically, instead of using a pre-specified value of L, vanilla CANTOR predicts the number of operations needed to solve a given problem as L (using the `[CLS]` representation from the encoder), and was trained with naïve mapping; the last vertex v_L is the root vertex. As shown by Table 7, vanilla CANTOR already outperforms the best baseline which adopts a pre-defined decoding order, indicating that our model does well in capturing the structures of equations internally, and that using a pre-defined decoding order may be an unnecessary burden on model learning.

Model	MathQA (Dev)		MathQA (Test)		SVAMP	
	Equ.	Val.	Equ.	Val.	Equ.	Val.
Autoregressive Model						
Pre-defined Decoding Order (✓); Structure Modeling (✓)						
DEDUCTREASONER	74.0	77.5	74.7	78.6	44.6	45.1
Non-autoregressive Models						
Pre-defined Decoding Order (✗); Structure Modeling (✗)						
Vanilla NAR	76.9	79.1	77.4	79.6	$36.4_{\pm 1.56}$	$37.0_{\pm 1.48}$
Pre-defined Decoding Order (✗); Structure Modeling (✓)						
Vanilla CANTOR	**77.4**	**80.4**	**78.3**	**81.4**	$\mathbf{46.8}_{\pm 0.55}$	$\mathbf{47.3}_{\pm 0.47}$

Table 7: Comparisons between (1) models without and with a pre-defined decoding order (Vanilla CANTOR vs. DEDUCTREASONER) (2) and models with and without modeling the structures of equations (Vanilla CANTOR vs. Vanilla NAR).

5.7.2 Structure Modeling

Previous work has proposed non-autoregressive models for semantic parsing, but without explicit structure modeling. To investigate the effect of structure modeling, we compared vanilla CANTOR with the non-autoregressive parser proposed by Shrivastava et al. (2021) which we name as **vanilla NAR**. Vanilla NAR predicts a length of the decoder L', and produces an L'-sized equation text with token-wise generation (one token at a position)[7]. By contrast, vanilla CANTOR structures an equation as a DAG with vertices corresponding to reasoning steps. As shown by Table 7, vanilla CANTOR outperforms vanilla NAR, which verifies the value of structure modeling.

[5] Equation templates are equations with numbers replaced with placeholders, e.g., `const_10 + num@7` adds 10 to the 7-th number in a problem.

[6] There are more advanced weakly-supervised training methods (Chen et al., 2020; Shao et al., 2021) for discrete reasoning on DROP. Investigation of how CANTOR is compatible with them is left for future work.

[7] For numbers in an equation, following Shrivastava et al. (2021), vanilla NAR decodes their positions in the problem instead of their constituent tokens. An example of an equation text is `( const_1 + pos@7 ) × const_2` where `pos@7` denotes the number mentioned at position 7.

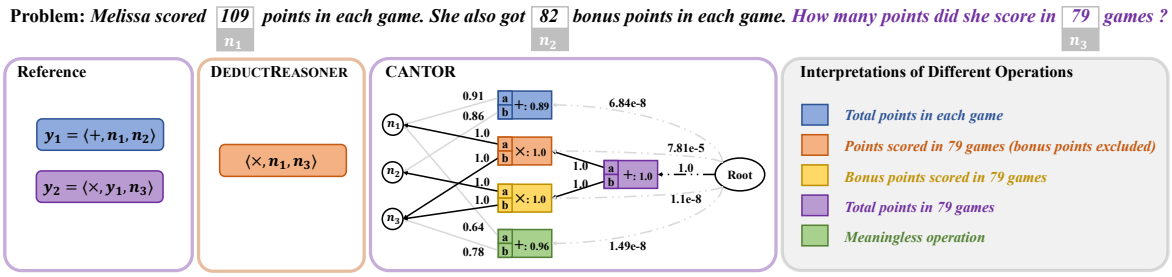

Figure 3: A test case from SVAMP. Operations leading to the same quantity are marked with the same color. Purple ones are operations evaluating to the correct answer. For a clear presentation of our DAG, we only retain top-5 root vertices along with their descendants. We also present probabilities of predicted operators, operands, and root vertices. The best baseline DEDUCTREASONER overlooks *bonus points* in its prediction; while the same prediction appears as a sub-graph in our DAG, CANTOR succeeds in filtering it out and recognizes the correct one.

Training Method	MathQA (Dev)		MathQA (Test)		SVAMP	
	Val.@1	Val.@5	Val.@1	Val.@5	Val.@1	Val.@5
Naïve Mapping	80.51	81.63	81.00	81.87	$48.22_{\pm 0.94}$	$55.98_{\pm 1.30}$
Hard EM	81.29	82.46	82.06	83.68	$47.08_{\pm 0.63}$	$66.56_{\pm 1.45}$
MML	68.39	71.09	69.91	72.65	$\mathbf{49.58}_{\pm 0.63}$	$63.44_{\pm 1.38}$
Hard EM with Annealing						
$\tau = 500$	81.46	**83.66**	**82.93**	**84.86**	$47.84_{\pm 0.64}$	$\mathbf{67.52}_{\pm 1.78}$
$\tau = 1,000$	81.54	83.20	82.55	83.99	$48.06_{\pm 0.91}$	$66.46_{\pm 2.95}$
$\tau = 1,500$	81.50	83.37	82.68	84.24	$48.82_{\pm 0.58}$	$67.12_{\pm 1.63}$
$\tau = 2,000$	**81.54**	83.45	82.80	84.30	$48.14_{\pm 0.48}$	$65.58_{\pm 1.79}$

Table 8: Comparisons among different training methods. *Val.@k* is the recall of answers over execution results at top-k root vertices (top-k $P_r(p_{|Z|}|X)$).

L	MathQA (Dev)		MathQA (Test)		SVAMP	
	Val.@1	Val.@5	Val.@1	Val.@5	Val.@1	Val.@5
20	81.00	82.66	82.74	83.86	$48.56_{\pm 0.43}$	$59.00_{\pm 2.28}$
40	81.21	82.70	82.74	83.86	$48.96_{\pm 0.74}$	$62.94_{\pm 2.26}$
60	81.54	**83.45**	82.80	84.30	$\mathbf{49.58}_{\pm 0.63}$	$\mathbf{63.44}_{\pm 1.38}$
80	**81.67**	83.16	**82.93**	**84.42**	$48.58_{\pm 0.48}$	$62.32_{\pm 0.72}$
100	81.58	83.20	82.87	84.42	$48.60_{\pm 0.75}$	$63.24_{\pm 1.87}$

Table 9: *Val.@k* with varying graph sizes L. Models were trained using hard EM with annealing ($\tau = 2000$) on MathQA and MML on SVAMP. *Val.@k* is the answer recall over execution results at top-k root vertices.

5.7.3 Capturing Diverse Reasoning Steps

CANTOR decodes an L-sized DAG that encompasses diverse reasoning steps which are necessary or possibly redundant. Comparing diverse choices is beneficial to pick out the proper one. In this section, we investigate how well CANTOR captures diverse reasoning steps and its effect on model performance. As it is pointless to merely have different operations at different vertices, we focused on the quality of top-k root vertices (top-k $P_r(p_{|Z|}|X)$)[8] and evaluated the recall of answers (*Val.@k*).

Training Methods Compared with vanilla CANTOR, CANTOR trained with methods that leverage more vertices than necessary (for ground-truth equations) achieved higher *Val.@k* most of the time (Table 8). One exception was applying MML on MathQA, which led to much worse performance. We conjecture that this is because our assumption in MML is incompatible with the complex equations in MathQA (please refer to Appendix C for detailed discussion on the limitations of our MML). However, it is still helpful to warm up hard EM with MML, which is demonstrated by the improvements of hard EM with annealing over hard EM. Notably, CANTOR trained with naïve mapping outperforms vanilla CANTOR on SVAMP; this is because the former was trained to leverage more vertices than necessary in testing (due to $\max |Y|$ on the train set being larger than $\max |Y|$ on SVAMP) and compares different vertices for root vertex selection, while the latter has no access to extra vertices and uses the last vertex as the root vertex without comparisons.

In practice, hard EM with annealing should be the default training method; as in Table 8, it always outperforms naïve mapping and hard EM, and is at least competitive with MML. As shown by Table 8 and Table 13, the two hyperparameters to tune, i.e., the number of warm-up steps τ and the beam size B, are robust to a wide range of values.

Graph Size L A larger DAG can encompass more reasoning steps, but also increases the difficulty of operand matching and root vertex selection. Training methods like hard EM may even suffer from suppressing false negative operations. Table 9 shows the effect of varying graph sizes L. Model performance improves until L reaches 80 and 60 on MathQA and SVAMP, respectively.

[8] When selecting the top-k root vertices, we skipped repeated logically-equivalent equations; logical equivalence is evaluated the same way as equation accuracy.

Model	Params	SVAMP	GSM8K
Few-Shot Setting			
8-shot CoT (Wei et al., 2022)			
LaMDA	137B	37.5	14.3
GPT-3	175B	68.9	46.9
PaLM	62B	46.7	29.9
	540B	79.0	56.9
Fully-Supervised Setting			
GPT-3 (Cobbe et al., 2021)	175B	-	∼35
DEDUCTREASONER			
RoBERTa$_{base}$	125M	45.1	-
RoBERTa$_{large}$	355M	50.4	-
CANTOR			
RoBERTa$_{base}$	125M	49.6	-
RoBERTa$_{large}$	355M	55.4	30.2

Table 10: Value accuracy on SVAMP and GSM8K. CoT is short for Chain-of-Thought prompting.

5.8 Case Study

Fig 3 presents a test case from SVAMP. For a clear presentation of our DAG, we only show top-5 root vertices[8] along with their descendants. By comparing diverse operations and chaining relevant ones, CANTOR succeeds in discriminating logically correct operations from distracting ones (e.g., the one predicted by DEDUCTREASONER which overlooks *bonus points*), even though the final equation is structurally different from the annotated reference.

5.9 CANTOR vs. LLMs with Chain-of-Thought Prompting

Recently, Wei et al. (2022) proposed chain-of-thought prompting which endows large language models with the ability to generate a series of intermediate reasoning steps to reach the final answer of a given problem, achieving state-of-the-art performance on a wide range of reasoning tasks. Table 10 compares CANTOR and chain-of-thought prompting. Though being 392× smaller, CANTOR with RoBERTa$_{\text{base}}$ already outperforms PaLM-62B on SVAMP; using RoBERTa$_{\text{large}}$ gives an aggressive improvement, demonstrating CANTOR's great potential.

CANTOR is also applicable to the challenging GSM8K dataset (Cobbe et al., 2021) which was created to probe the reasoning ability of large language models and has high diversity among problems. As GSM8K was annotated with natural language solutions, the extracted equations are noisy and incomplete; we ended up with 6,312 (out of 7,473) noisy training examples. As shown in Table 10, CANTOR is close to the 175B GPT-3 model fine-tuned on the whole train set, and is on a par with PaLM-62B with chain-of-thought prompting.

6 Conclusion

We propose a numerical reasoner called CANTOR. Unlike previous structured decoders that model a single equation with pre-defined restrictions on the decoding dependencies, CANTOR models diverse reasoning steps using a directed acyclic graph without a pre-defined decoding order, and derives equations by comparing and chaining relevant reasoning steps. With our training methods, CANTOR is capable of capturing the logical dependencies among reasoning steps internally, and produces equations that are more consistent with the reasoning problems by comparing diverse reasoning steps. CANTOR achieves state-of-the-art results on two math word problem datasets under the fully-supervised setting, and is applicable to weakly-supervised scenarios with significant improvements.

In future work, we plan to extend CANTOR for general structured prediction tasks, e.g., sequence labeling and parsing.

7 Limitations

Though CANTOR significantly outperforms baselines, there is still a large room for improvement in solving numerical reasoning problems with novel equation templates and being robust to variations in the problems. For example, our value accuracy on SVAMP problems with unseen equation templates is lower than 20% (Table 4), and the value accuracy on problems that evaluate question sensitivity barely reaches 30% (Table 5). We also argue for more benchmarks that expose weaknesses of existing models, as we observe that more than half of test problems in MWP datasets can be solved with equation templates seen in training, which may overestimate the numerical reasoning ability of neural models.

Acknowledgements

This work was supported by the National Science Foundation for Distinguished Young Scholars (with No. 62125604) and the NSFC projects (Key project with No. 61936010 and regular project with No. 61876096). This work was also supported by the Guoqiang Institute of Tsinghua University, with Grant No. 2019GQG1 and 2020GQG0005, and sponsored by Tsinghua-Toyota Joint Research Fund.

References

Aida Amini, Saadia Gabriel, Shanchuan Lin, Rik Koncel-Kedziorski, Yejin Choi, and Hannaneh Hajishirzi. 2019. Mathqa: Towards interpretable math word problem solving with operation-based formalisms. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 2357–2367. Association for Computational Linguistics.

Arun Babu, Akshat Shrivastava, Armen Aghajanyan, Ahmed Aly, Angela Fan, and Marjan Ghazvininejad. 2021. Non-autoregressive semantic parsing for compositional task-oriented dialog. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, pages 2969–2978. Association for Computational Linguistics.

Yixuan Cao, Feng Hong, Hongwei Li, and Ping Luo. 2021. A bottom-up DAG structure extraction model for math word problems. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 39–46. AAAI Press.

Xinyun Chen, Chen Liang, Adams Wei Yu, Denny Zhou, Dawn Song, and Quoc V. Le. 2020. Neural symbolic reader: Scalable integration of distributed and symbolic representations for reading comprehension. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Zhiyu Chen, Wenhu Chen, Charese Smiley, Sameena Shah, Iana Borova, Dylan Langdon, Reema Moussa, Matt Beane, Ting-Hao Huang, Bryan R. 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, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 3697–3711. 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 Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben 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 Garcia, 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 Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2022. Palm: Scaling language modeling with pathways. *CoRR*, abs/2204.02311.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *CoRR*, abs/2110.14168.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. 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, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics.

Cunxiao Du, Zhaopeng Tu, and Jing Jiang. 2021. Order-agnostic cross entropy for non-autoregressive machine translation. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 2849–2859. PMLR.

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, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 2368–2378. Association for Computational Linguistics.

Marjan Ghazvininejad, Vladimir Karpukhin, Luke Zettlemoyer, and Omer Levy. 2020. Aligned cross entropy for non-autoregressive machine translation. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 3515–3523. PMLR.

Jiatao Gu, James Bradbury, Caiming Xiong, Victor O. K. Li, and Richard Socher. 2018. Non-autoregressive neural machine translation. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

Fei Huang, Hao Zhou, Yang Liu, Hang Li, and Minlie Huang. 2022. Directed acyclic transformer for non-autoregressive machine translation. *CoRR*, abs/2205.07459.

Zhanming Jie, Jierui Li, and Wei Lu. 2022. Learning to reason deductively: Math word problem solving as complex relation extraction. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 5944–5955. Association for Computational Linguistics.

Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. 2016. MAWPS: A math word problem repository. In *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016*, pages 1152–1157. The Association for Computational Linguistics.

Yihuai Lan, Lei Wang, Qiyuan Zhang, Yunshi Lan, Bing Tian Dai, Yan Wang, Dongxiang Zhang, and Ee-Peng Lim. 2021. Mwptoolkit: An open-source framework for deep learning-based math word problem solvers. *CoRR*, abs/2109.00799.

Zhongli Li, Wenxuan Zhang, Chao Yan, Qingyu Zhou, Chao Li, Hongzhi Liu, and Yunbo Cao. 2022. Seeking patterns, not just memorizing procedures: Contrastive learning for solving math word problems. In *Findings of the Association for Computational Linguistics: ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 2486–2496. Association for Computational Linguistics.

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

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

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, ACL 2020, Online, July 5-10, 2020*, pages 975–984. Association for Computational Linguistics.

M.L. Miller, H.S. Stone, and I.J. Cox. 1997. Optimizing murty's ranked assignment method. *IEEE Transactions on Aerospace and Electronic Systems*, 33(3):851–862.

Sewon Min, Danqi Chen, Hannaneh Hajishirzi, and Luke Zettlemoyer. 2019. A discrete hard EM approach for weakly supervised question answering. 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 2019, Hong Kong, China, November 3-7, 2019*, pages 2851–2864. Association for Computational Linguistics.

Swaroop Mishra, Arindam Mitra, Neeraj Varshney, Bhavdeep Singh Sachdeva, Peter Clark, Chitta Baral, and Ashwin Kalyan. 2022. Numglue: A suite of fundamental yet challenging mathematical reasoning tasks. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 3505–3523. 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, NAACL-HLT 2021, Online, June 6-11, 2021*, pages 2080–2094. Association for Computational Linguistics.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Abhilasha Ravichander, Aakanksha Naik, Carolyn Penstein Rosé, and Eduard H. Hovy. 2019. EQUATE: A benchmark evaluation framework for quantitative reasoning in natural language inference. In *Proceedings of the 23rd Conference on Computational Natural Language Learning, CoNLL 2019, Hong Kong, China, November 3-4, 2019*, pages 349–361. Association for Computational Linguistics.

Elad Segal, Avia Efrat, Mor Shoham, Amir Globerson, and Jonathan Berant. 2020. A simple and effective model for answering multi-span questions. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pages 3074–3080. Association for Computational Linguistics.

Zhihong Shao, Lifeng Shang, Qun Liu, and Minlie Huang. 2021. A mutual information maximization approach for the spurious solution problem in weakly supervised question answering. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, pages 4111–4124. Association for Computational Linguistics.

Jianhao Shen, Yichun Yin, Lin Li, Lifeng Shang, Xin Jiang, Ming Zhang, and Qun Liu. 2021. Generate & rank: A multi-task framework for math word problems. In *Findings of the Association for Computational Linguistics: EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 16-20 November, 2021*, pages 2269–2279. 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, COLING 2020, Barcelona, Spain (Online), December 8-13, 2020*, pages 2924–2934. International Committee on Computational Linguistics.

Akshat Shrivastava, Pierce Chuang, Arun Babu, Shrey Desai, Abhinav Arora, Alexander Zotov, and Ahmed Aly. 2021. Span pointer networks for non-autoregressive task-oriented semantic parsing. In *Findings of the Association for Computational Linguistics: EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 16-20 November, 2021*, pages 1873–1886. Association for Computational Linguistics.

Minghuan Tan, Lei Wang, Lingxiao Jiang, and Jing Jiang. 2021. Investigating math word problems using pretrained multilingual language models. *CoRR*, abs/2105.08928.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008.

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, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*, pages 845–854. Association for Computational Linguistics.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed H. Chi, Quoc Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. *CoRR*, abs/2201.11903.

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 2019, Macao, China, August 10-16, 2019*, pages 5299–5305. ijcai.org.

Jipeng Zhang, Lei Wang, Roy Ka-Wei Lee, Yi Bin, Yan Wang, Jie Shao, and Ee-Peng Lim. 2020. Graph-to-tree learning for solving math word problems. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 3928–3937. Association for Computational Linguistics.

A Implementation Details

	MWP Solving	Discrete Reasoning
Batch Size	32	12
Learning Rate	2e-5	5e-6
Learning Rate Warum-up Steps	500	0

Table 11: Hyperparameters for training CANTOR.

We trained CANTOR for up to 100k training steps for the MWP task and up to 20 epochs for the discrete reasoning task, using hyperparameters specified in Table 11. All experiments were conducted with V100 GPUs.

B Training Methods

B.1 Hard EM

The training objective of hard EM is formulated as:

$$\mathcal{L} = -\log P_\theta(Z^*|X),\ Z^* = arg\max_{Z\in\Gamma} P_\theta(Z|X)$$

where $\Gamma = \{Z|P_\theta(Y|Z,X)=1\}$. For any $Z \in \Gamma$, we have $|Z| = |Y|$, that is,

$$\begin{aligned} Z =& \{z_1, z_2, ..., z_{|Y|}\},\ z_i = \langle p_i, z_i^f, z_i^a, z_i^b \rangle \\ s.t.\ & 1 \le p_1 < p_2 < ... < p_{|Y|} \le L \\ & z_i^f = y_i^f \end{aligned}$$

where z_i is the operation y_i mapped to the vertex at position p_i.

As $\{p_1, ..., p_{|Y|}\}$ defines a valid mapping from Y to Z, finding Z^* is equivalent to finding the optimal mapping $\{p_1, ..., p_{|Y|}\}$, which we search for via beam search. For convenience of illustration, we define the level of an operation in Y as the length of the longest path from its corresponding vertex in the DAG counterpart of Y to a leaf vertex (which is a constant or a number mentioned in the problem). Let D_l be the set of indices of operations with the same level l. For any $Z \in \Gamma$, $P_\theta(Z|X)$ can be factorized as follows:

$$P_\theta(Z|X) = P_r(p_{|Y|}|X) \prod_l \prod_{i\in D_l} P_z(z_i|p_i, X)$$

Therefore, we can use beam search to approximately find the optimal mapping level-by-level. To guarantee valid mappings, we restrict that

$$\forall i \in D_l,\ \max\{p_j|j \in D_{l-1}\} < p_i \le L - \sum_{s>l} |D_s|$$

To find the B-best mappings from D_l according to $\prod_{i\in D_l} P_z(z_i|p_i, X)$, we utilize an open-source implementation[9] of Murty's algorithm (Miller et al., 1997), whose worse case complexity is $O(B|D_l|^3)$.

[9] https://github.com/motrom/fastmurty

B.2 MML

The training objective of MML is formulated as:

$$\mathcal{L} = -\log \sum_{Z \in \Gamma} P_\theta(Z|X)$$

We adopt a strong (but risky) assumption so that we can use dynamic programming to marginalize $P_\theta(Z|X)$ in polynomial time. Specifically, for any operation y_i, we assume that the two sub-graphs rooted at y_i^a and y_i^b respectively (in the DAG counterpart of Y) are independently mapped to $\{v_1, v_2, ..., v_L\}$. Let $\mathbf{M}_{i,j}$ be the marginal probability of the sub-graph rooted at y_i mapped to $\{v_1, ..., v_j\}$, and $G(y_i)$ is the set of indices of constituent operations in the sub-graph, then $\mathbf{M}_{i,j}$ is computed as (we omit X for a brief presentation):

$$\mathbf{M}_{i,j} = \sum_{\substack{\{p_k | k \in G(y_i)\} \\ p_i = j}} \prod_{k \in G(y_i)} P_z(z_k|p_k)$$

$$P_z(z_k|p_k) = P_f(y_k^f|p_k) P_a(z_k^a|p_k) P_b(z_k^b|p_k)$$

Based on our assumption, if $y_i^a = y_u \in Y$ and $y_i^b = y_v \in Y$, we have:

$$\mathbf{M}_{i,j} = P_f(y_i^f|j) \sum_{p_u=1}^{j-1} \mathbf{M}_{u,p_u} P_a(z_u|j) \sum_{p_v=1}^{j-1} \mathbf{M}_{v,p_v} P_b(z_v|j)$$

otherwise, we have:

$y_i^a \in \mathcal{C} \cup \mathcal{N}, y_i^b = y_v \in Y$:

$$\mathbf{M}_{i,j} = P_f(y_i^f|j) P_a(y_i^a|j) \sum_{p_v=1}^{j-1} \mathbf{M}_{v,p_v} P_b(z_v|j)$$

$y_i^a = y_u \in Y, y_i^b \in \mathcal{C} \cup \mathcal{N}$:

$$\mathbf{M}_{i,j} = P_f(y_i^f|j) P_b(y_i^b|j) \sum_{p_u=1}^{j-1} \mathbf{M}_{u,p_u} P_a(z_u|j)$$

$y_i^a, y_i^b \in \mathcal{C} \cup \mathcal{N}$:

$$\mathbf{M}_{i,j} = P_f(y_i^f|j) P_a(y_i^a|j) P_b(y_i^b|j)$$

Finally, the training objective can be computed as:

$$\mathcal{L} = -\log \sum_{j=1}^{L} P_r(j) \mathbf{M}_{|Y|,j}$$

which takes $O(|Y|)$ parallel operations.

C Limitations of Our MML

For our MML method, we impose an independence assumption for efficient marginalization of $P_\theta(Z|X)$ over all Z that denote valid mappings from operations in Y to decoding positions, but at the cost of failing to compute exact marginalization and giving a noisy training objective when the target equation Y is complex, like those in MathQA.

# Branch	All			# Operation ≤ 3			# Operation ≥ 4		
	MML	Naïve	Hard EM	MML	Naïve	Hard EM	MML	Naïve	Hard EM
0	**83.8**	82.5	82.9	**84.7**	83.4	83.4	79.5	78.0	**80.3**
1	69.2	83.0	**85.2**	82.3	82.3	**87.1**	65.6	83.2	**84.7**
≥ 2	35.0	73.1	**73.5**	-	-	-	35.0	73.1	**73.5**

Table 12: Value accuracy breakdown on the test set of MathQA w.r.t. the number of branches (*# Branch*) and the number of operations (*# Operation*) in annotated gold equations. *Naïve* stands for naïve mapping.

When does Our MML Conduct Exact Marginalization? And What are the Effects on Model Performance? Our MML conducts exact marginalization only if Y has a linear structure, i.e., Y has no branches; we define a branch in Y to be an operation taking another two operations as operands. If Y have branches, our MML will include the probability of invalid Z where different operations share one decoding position, which may mislead a model. As validated by Table 12, (a) our MML works well on test problems whose gold equations have no branches (# Branch=0: value accuracy=83.8%), even when equations are long (# Branch=0 and # Operation>=4: value accuracy=79.5%); (b) However, it becomes poor if equations have more branches (# Branch>=2: value accuracy=35.0%).

Empirically, our MML works well when most equations are linear, and short equations are likely linear in existing datasets (e.g., SVAMP and DROP). When target equations are complex, hard EM should be more suitable, but we can still benefit from using our MML for warming up.

D Ablation Study

D.1 Effect of Beam Size B on Hard EM

Training Method	Dev		Test	
	Equ.	Val.	Equ.	Val.
Random Mapping	18.08	19.20	17.76	18.44
Hard EM				
$B = 1$	77.93	81.13	**78.75**	**82.24**
$B = 10$	77.64	81.17	78.38	81.99
$B = 20$	**78.43**	**81.29**	78.63	82.06

Table 13: Value accuracy of models trained with hard EM using different beam sizes. *Random Mapping* is a baseline which uses random $Z \in \Gamma$ for training.

As shown by Table 13, model performance is insensitive to beam size when using hard EM on MathQA. To investigate whether the choices of Z matter for optimization, we considered a baseline called **random mapping**, which optimizes a model

on random $Z \in \Gamma$. We observed that hard EM outperforms random mapping substantially, indicating that beam search finds effective Z for training.

E Inference Efficiency

Due to non-autoregressive decoding, CANTOR is significantly faster than previous autoregressive baselines in terms of inference efficiency. For example, on a single V100 32G GPU, CANTOR achieves a $7\times$ speedup over DEDUCTREASONER on the dev set of MathQA.

F Case Study on MathQA

Fig 4 presents two test cases from MathQA. In the upper case, the baseline DEDUCTREASONER misunderstands "increase" and "decrease", and conducts wrong operations. In the lower case which mentions numerous quantities in the problem, DEDUCTREASONER, despite arriving at the correct value, operates on wrong quantities at the second and the third reasoning steps. By contrast, our proposed model CANTOR produces precise reasoning processes with proper choices of quantities to operate on.

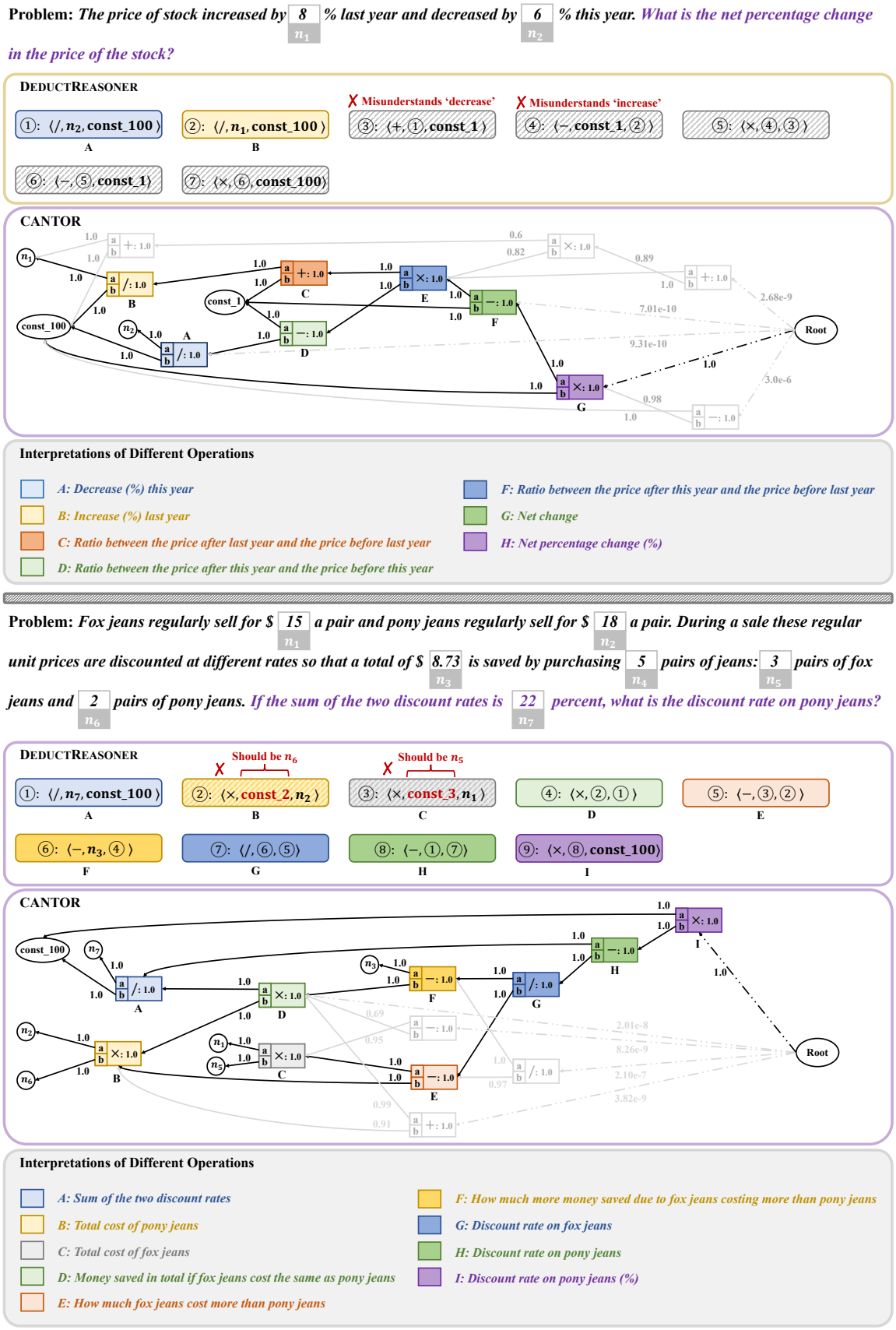

Figure 4: Two test cases from MathQA. Operations leading to the same value are marked with the same color and letter (e.g., A, B, etc.). Purple ones are operations evaluating to the correct answer. For a clear presentation of our DAG, we only retain top-5 root vertices along with their descendants. We also present probabilities of predicted operators, operands, and root vertices. For predictions from DEDUCTREASONER, we mark the decoding order of operations with circled numbers; operations with forward slashes in the background are erroneous ones.