MULTIPROVER: Generating Multiple Proofs for Improved Interpretability in Rule Reasoning

Swarnadeep Saha **Prateek Yadav** **Mohit Bansal**
UNC Chapel Hill
{swarna, prateek, mbansal}@cs.unc.edu

Abstract

We focus on a type of linguistic formal reasoning where the goal is to reason over explicit knowledge in the form of natural language facts and rules (Clark et al., 2020). A recent work, named PROVER (Saha et al., 2020), performs such reasoning by answering a question and also generating a proof graph that explains the answer. However, compositional reasoning is not always unique and there may be multiple ways of reaching the correct answer. Thus, in our work, we address a new and challenging problem of generating multiple proof graphs for reasoning over natural language rule-bases. Each proof provides a different rationale for the answer, thereby improving the interpretability of such reasoning systems. In order to jointly learn from all proof graphs and exploit the correlations between multiple proofs for a question, we pose this task as a set generation problem over structured output spaces where each proof is represented as a directed graph. We propose two variants of a proof-set generation model, MULTIPROVER. Our first model, *Multilabel*-MULTIPROVER, generates a set of proofs via multi-label classification and implicit conditioning between the proofs; while the second model, *Iterative*-MULTIPROVER, generates proofs iteratively by explicitly conditioning on the previously generated proofs. Experiments on multiple synthetic, zero-shot, and human-paraphrased datasets reveal that both MULTIPROVER models significantly outperform PROVER on datasets containing multiple gold proofs. *Iterative*-MULTIPROVER obtains state-of-the-art proof F1 in zero-shot scenarios where all examples have single correct proofs. It also generalizes better to questions requiring higher depths of reasoning where multiple proofs are more frequent.

1 Introduction

Formal reasoning over explicit multi-sentence knowledge (Newell and Simon, 1956) has often proved to be challenging (Musen and Van Der Lei, 1988), owing to the difficulty in creating logical forms from such sentences, thereby restricting the application of semantic parsers (Zettlemoyer and Collins, 2005; Berant et al., 2013; Berant and Liang, 2014). Thus, in a recent work, Clark et al. (2020) bypass the creation of intermediate logical forms and show that transformers (Vaswani et al., 2017) can act as "soft theorem provers" by answering questions over natural language (English) rule-bases, consisting of facts and rules. In order to reliably interpret these predicted answers, Saha et al. (2020) propose PROVER, a transformer-based model that generates the corresponding proof graph, thus emulating formal reasoning closely. Consider the two example rule-bases with two questions and corresponding proofs in Figure 1, where a proof is a directed graph consisting of the relevant facts and rules from the corresponding rule-base.

PROVER shows good single-proof generation accuracy but is designed and trained in a way to generate only a single proof for each question. This is not ideal because formal proofs are not always unique and there may be multiple correct ways of arriving at the answer. For example, Q_1 and Q_2 in Figure 1 have three and four correct proofs respectively. Hence, in order to enhance the human-interpretability of linguistic formal reasoning systems, it is desirable to develop methods that can generate multiple proofs, each providing a different rationale for the predicted answer. Such interpretable methods, while possessing the flexibility of operating over natural language, can also aid in verifying claims when constructing proofs from scratch is tedious or infeasible.

We find that PROVER (Saha et al., 2020), when trained on all proofs as independent training examples (Eq. 2) and extended to generate top-p proofs during inference (Eq. 3), fails drastically, achieving a low proof precision of 34%. The subsequent proofs are often incorrect because it is not

Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 3662–3677
June 6–11, 2021.

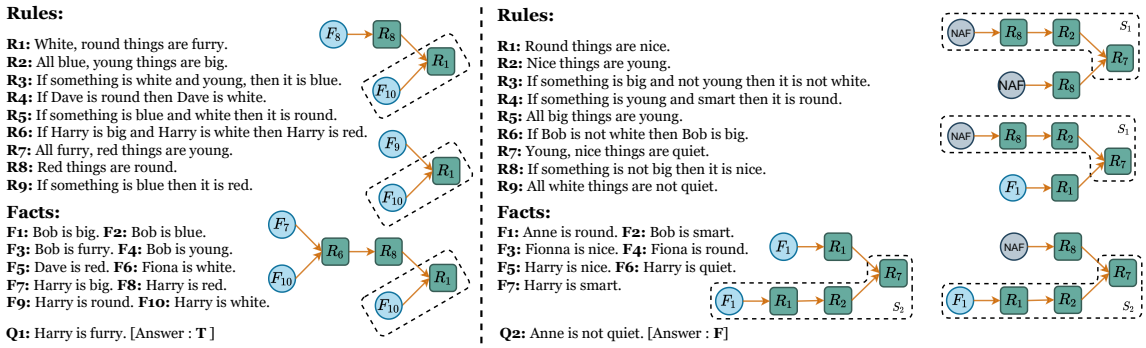

Figure 1: Diagram showing two rule-bases with rules, facts, questions, answers and all possible proofs. The first question has three correct proofs while the second question has four correct proofs. MULTIPROVER answers both questions correctly and also generates all the corresponding proofs accurately for each question.

trained jointly with all proofs and hence, is unable to exploit the inter-proof correlations and also does not learn the correct number of proofs for a question. Thus, we propose MULTIPROVER, a transformer-based model that can generate a set of proof graphs with appropriate cardinality for a given question. Since multiple proofs can be generated in any arbitrary order, we pose this task as a set generation problem over graphs and train MULTIPROVER jointly with a permutation-invariant Hungarian Loss (Zhang et al., 2019a,b) over all proofs.

A proof graph is generated through a node module which selects the relevant facts and rules as part of the proof and an edge module which determines the edges between the chosen nodes. Similar to PRover, we first enforce multiple structural constraints during training and inference to ensure that a generated proof is valid. Next, in order to generate a set of proofs jointly, we propose our first model, *Multilabel*-MULTIPROVER, a multi-label classification framework which performs *implicit conditioning* among the proofs and predicts p binary labels for each node and edge, denoting its presence or absence in each of the p proofs that we want to generate. It is efficient in terms of number of parameters and training time and also achieves a better proof F1 than PROVER. However, the lack of explicit conditioning between the proofs is not ideal because a question with multiple proofs often has certain common sub-graphs across the proofs. E.g., all the 3 proofs for Q_1 in Figure 1 have the sub-graph $\{F_{10} \rightarrow R_1\}$ common. Thus, in order to exploit these correlations which *Multilabel*-MULTIPROVER cannot capture explicitly, we further propose an improved variant of MULTIPROVER, named *Iterative*-MULTIPROVER, which generates appropriate number of proofs by stacking multiple node and edge encoders, each of which generates one proof at each time step by conditioning on the previously generated proofs. This enables the model to better learn the correlations between multiple proofs for a given question. To capture the set-based nature of the task, we train MULTIPROVER using a permutation-invariant Hungarian Loss (Sec. 3.5), which solves an assignment problem between a set of predicted and gold proofs.

Empirical evaluation on synthetic and human paraphrased QA rule-bases (Clark et al., 2020) show that both of our MULTIPROVER models achieve a significantly higher proof F1 compared to PROVER while retaining the QA accuracy. Further, on a challenging hand-authored zero-shot dataset, where all examples have single gold proofs, *Iterative*-MULTIPROVER achieves state-of-the-art proof F1. It also generalizes better to questions requiring higher depths of reasoning with more multiple proofs. Overall, our contributions are:

- We address a new and challenging problem of generating a set of multiple logical proof graphs for reasoning over natural language rule-bases by proposing two set-based joint models, *Multilabel*-MULTIPROVER and *Iterative*-MULTIPROVER.[1]
- *Iterative*-MULTIPROVER's joint training and explicit conditioning helps it to better learn the relative importance of rules and facts for a particular question and uncover common subgraphs across multiple proofs. Thus, compared to *Multilabel*-MULTIPROVER and PROVER, it is able to transfer well in zero-shot settings because it learns to assign a soft prior over the rule-base.
- *Iterative*-MULTIPROVER's conditional generation also enables it to generalize better to questions requiring higher depths of reasoning where the presence of multiple proofs is frequent.

[1]Our code and models are publicly available at https://github.com/swarnaHub/multiPRover.

2 Related Work

The task of rule reasoning (Clark et al., 2020) is related to other recently proposed tasks on QA (Weston et al., 2015; Yang et al., 2018; Lin et al., 2019; Tafjord et al., 2019; Richardson et al., 2020) and NLI (MacCartney and Manning, 2014). However, most of these tasks require implicit reasoning rules as opposed to explicit ones and the focus is either on broad language understanding or on single rule application. Below we discuss MULTIPROVER's relation to multiple areas of NLP and ML.

Structured Explanations: There is useful previous work on developing interpretable and explainable models (Doshi-Velez and Kim, 2017; Rudin, 2019; Hase and Bansal, 2020; Jacovi and Goldberg, 2020) for NLP. Explanations in NLP take three major forms – (1) extractive rationales or highlights (Zaidan et al., 2007; Lei et al., 2016; Yu et al., 2019; DeYoung et al., 2020) where a subset of the input text explain a prediction, (2) free-form or natural language explanations (Camburu et al., 2018; Rajani et al., 2019; Zhang et al., 2020; Kumar and Talukdar, 2020) that are not constrained to the input, and (3) structured explanations that range from semi-structured text (Ye et al., 2020) to chain of facts (Khot et al., 2020; Jhamtani and Clark, 2020; Gontier et al., 2020) to explanation graphs (based on edges between chains of facts) (Jansen et al., 2018; Jansen and Ustalov, 2019; Xie et al., 2020).

Generating Multiple Outputs: Generating a set of proofs can be viewed as a task of generating multiple structured outputs (Prasad et al., 2014). Multiple prior studies focus on generating diverse unstructured texts (Gimpel et al., 2013; Dai et al., 2017; Xu et al., 2018; Raffel et al., 2020). which broadly span two categories – (1) using improved decoding techniques like beam search with inter-sibling ranking penalty (Li et al., 2016), iterative beam search (Kulikov et al., 2018), diverse beam search (Vijayakumar et al., 2018), and sentence codes (Shu et al., 2019), (2) varying the hidden representations or using multiple decoders (Dai et al., 2017; Jain et al., 2017; Shen et al., 2019). Our baseline, PROVER-top-p, which extends PROVER to generate top-p proofs during inference falls in the first category while MULTIPROVER falls in the second category, where the multiple node and edge encoders vary the node and edge representations for generating multiple proofs.

Machine Learning over Sets: Set-based ML models (Zaheer et al., 2017; Lee et al., 2018; Zhang et al., 2019a; Kosiorek et al., 2020) have a wide range of applications including generating multiple image captions (Vinyals et al., 2015), generating diverse translations (Cho et al., 2014; Bahdanau et al., 2015), enumerating rules in a logical inference system (Gao et al., 2019). Set problems are challenging because the number of valid solutions for a set of size n are $n!$, which increases faster than exponential in n and ignoring the set structure produces sub-optimal solutions (Zhang et al., 2019a). Thus, we use a set-based Hungarian Loss (Zhang et al., 2019a,b) for capturing the permutation-invariant nature of generating a set of proofs.

3 Method

3.1 Task Description and Notations

The input to our task is a tuple of the form $(\mathcal{C}, \mathcal{Q})$, where $\mathcal{C}$ is a rule-base context and $\mathcal{Q}$ is the question. We want to predict a binary answer $\mathcal{A} \in \{True, False\}$ for the question and generate a set of proof graphs $P = \{\mathcal{P}_1, \ldots, \mathcal{P}_p\}$, each of which provides a diverse rationale for the answer (see Figure 1). The context $\mathcal{C}$ consists of a set of facts and rules, denoted by $\mathcal{F}$ and $\mathcal{R}$ respectively. Facts $\mathcal{F} = \{F_1, \ldots F_f\}$ are unambiguous statements, while rules $\mathcal{R} = \{R_1, \ldots R_r\}$ are logical statements, which can be used in conjunction with the facts to arrive at a logical conclusion. Each proof $\mathcal{P}_i = (\mathcal{V}_i, \mathcal{E}_i)$ is a directed graph, with a set of nodes $\mathcal{V}_i \subseteq \mathcal{N}$ and a set of edges $\mathcal{E}_i \subseteq \mathcal{V}_i \times \mathcal{V}_i$, where $\mathcal{N} = \mathcal{F} \cup \mathcal{R} \cup \{\textbf{NAF}\}$ and $k = |\mathcal{N}|$. If a statement (E.g. "Anne is big") cannot be deduced from the context, then Negation as Failure (**NAF**) contains the negation of that statement (E.g. "Anne is not big"), which is considered true in a closed-world assumption. See appendix for more details of the syntax of proof graphs.

3.2 Baseline PROVER Model

PROVER (Saha et al., 2020) builds on top of RoBERTa (Liu et al., 2019) and consists of a question answering (QA) module, a node module and an edge module where the node and edge modules are used to predict a single proof graph. The input to RoBERTa is the concatenation of the facts, rules and the question. The QA module takes in the representation of the $[CLS]$ token and predicts a binary label for the question. The node module computes the node embeddings $\mathbf{N} \in \mathbb{R}^{k \times d}$

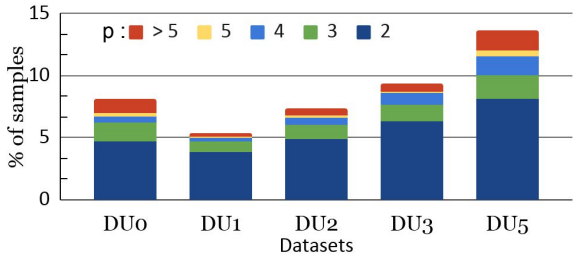

Figure 2: Plot showing the percentage of samples with $p > 1$ proofs for different training datasets, DU0-DU5.

consisting of the representations of each fact, rule and **NAF** where d is the embedding dimension. The i^{th} row n_i of $\mathbf{N}$ denotes the embedding of node i. A node classifier takes in these embeddings to output the node probabilities $np_i \in \mathbb{R}^k$ for each fact, rule and **NAF** being present in the proof. The edge module computes the edge embeddings $\mathbf{E} \in \mathbb{R}^{k^2 \times 3d}$ for every edge (i, j) through the function $\phi(i, j) = [n_i; n_j; (n_i - n_j)]$ where ; is the concatenation operation and outputs probabilities $ep_{i,j} \in \mathbb{R}^{k^2}$ of each edge being present in the proof. PROVER is trained using the joint cross-entropy loss over the QA, node and edge modules. The authors pose inference as a Integer Linear Program (ILP). Given a set of nodes and the edge probabilities from the trained model, the following global score over the edge probabilities is maximized, subject to multiple structural constraints $\mathcal{S}$ that ensure the validity of a proof graph (like checking for graph connectivity).

$$\underset{e_{i,j} \in \{0,1\}, s \in \mathcal{S}}{\arg\max} \sum_{i,j,i \neq j} ep_{i,j} * e_{i,j} + (1 - ep_{i,j}) * (1 - e_{i,j}) \quad (1)$$

Extending PROVER to Generate Proof-Sets: Since Saha et al. (2020) focus on generating one proof per question, they also train their model with one gold proof per question. For multiple proof generation, an obvious extension is to treat each proof for a question as a separate training example. Formally, for each sample l, given a context $\mathcal{C}^l$, a question $\mathcal{Q}^l$, an answer $\mathcal{A}^l$ and a set of gold proofs $\mathcal{P}^l_i$, where $i \in \{1, \ldots, p_l\}$, the extended training dataset can be defined as:

$$\mathcal{D} = \bigcup_{l=1}^{L} \left\{ \left(\mathcal{Q}^l, \mathcal{C}^l, \mathcal{A}^l, \mathcal{P}^l_i \right)_{i=1}^{p_l} \right\}_l \quad (2)$$

Once PROVER is trained with this dataset, during inference, we generate top-p proofs by first selecting the top-p node sets according to Eqn. 3 and then choosing the corresponding edge sets using the optimization function in Eqn. 1.

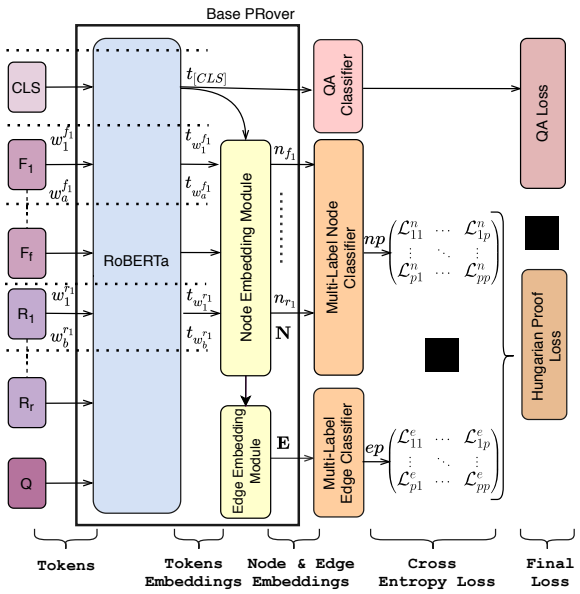

Figure 3: *Multilabel*-MULTIPROVER.

$$\underset{\mathbf{v} \in \{0,1\}^k}{\arg\max} \sum_{i=1}^{k} np_i * \mathbf{v}_i + (1 - np_i) * (1 - \mathbf{v}_i) \quad (3)$$

The top-p solutions of Eqn. 3 are $\mathbf{v}^1, \ldots, \mathbf{v}^p$ which indicate a node's presence or absence in the proofs. Although simple, this approach has two major issues. First, the lack of coupling between the proofs can potentially confuse the model as there are multiple possible proofs for the same (question, context) pair. Second, inference is inflexible and always generates a fixed number of proofs for every example, thus leading to the generation of many incorrect proofs (Section 5.1). As shown in Figure 1, certain questions can have multiple possible proofs. Figure 2 demonstrates this phenomenon statistically – the datasets we experiment with (Clark et al., 2020) contain up to 13% of the samples with > 1 correct proof. Thus, in the light of PROVER's limitations, we propose two novel architectures of a proof-set generation model, MULTIPROVER.

3.3 *Multilabel*-MULTIPROVER

As described in the previous section, a desired property for generating a set of proofs is to have the proofs conditioned on each other as opposed to treating them independently. Thus, we propose *Multilabel*-MULTIPROVER (see Figure 3), which poses the problem of generating a set of proofs as a multi-label classification task over all the nodes and edges corresponding to the set of p proofs. Each training example is a tuple $\left(\mathcal{Q}^l, \mathcal{C}^l, \mathcal{A}^l, \{\mathcal{P}^l_i\}_{i=1}^{p_l}\right)$,

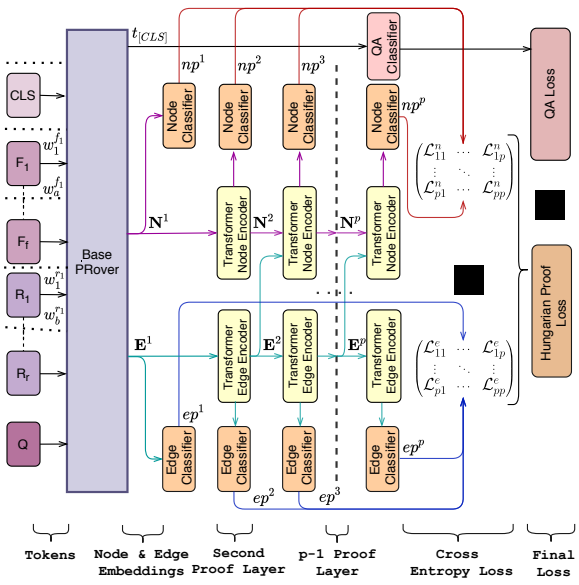

Figure 4: *Iterative*-MULTIPROVER.

consisting of a set of gold proofs $\{\mathcal{P}_i^l\}_{i=1}^{p_l}$ per example. It consists of a QA module, a node module and an edge module. Following PROVER (Section 3.2), we obtain the node representations $\mathbf{N} \in \mathbb{R}^{k \times d}$ by mean-pooling over the constituent RoBERTa representations. These are then passed through a *multi-label node classifier*, which consists of two linear layers and produces the probabilities $np_i \in \mathbb{R}^p$ of a node being present in the p proofs. The node embeddings n_i and n_j for a pair of nodes are transformed by the function $\phi(i, j)$, described in Section 3.2, to output the edge embeddings $\mathbf{E} \in \mathbb{R}^{k^2 \times 3d}$. We also have a *multi-label edge classifier*, which takes in the edge embeddings to generate the probabilities $ep_{i,j} \in \mathbb{R}^p$ of an edge (i, j) being present in the p proofs. Lastly, a *question answering* module predicts a binary answer for the question. Following PROVER, during training, we mask certain impossible edges like fact to fact, rule to fact and non-nodes. Given the outputs from the three modules, we train our model jointly over all proofs using a set-based Hungarian Loss.

This model is advantageous because there is *implicit conditioning* between the proofs as all the proofs are generated in parallel from the same node embeddings and edge embeddings. Thus, it has no additional time or memory overhead while also generating proof-sets better than PROVER (Section 5.1). However, it suffers from two major drawbacks. First, since the proofs are generated in parallel, the model is trained by padding empty proof graphs. Hence for higher values of p, the model has to learn more empty proofs, which makes the learning problem harder. Second, the proofs are not explicitly conditioned on each other. This motivates us to propose *Iterative*-MULTIPROVER.

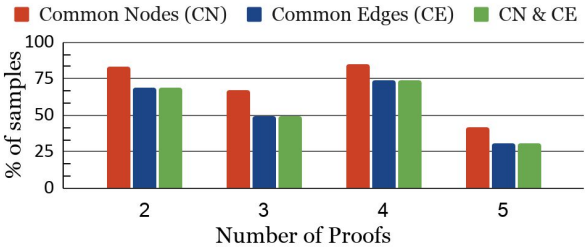

Figure 5: Plot showing the percentage of samples in DU5 with at least one common node, common edge or both between the proofs for varying number of proofs.

3.4 *Iterative*-MULTIPROVER

As a motivating example for why explicit conditioning among proofs is necessary, consider the proofs for Q_1 in Figure 1 where the sub-graph $\{F_{10} \rightarrow R_1\}$ is common across all the proofs. F_{10} and R_1 are essential for answering the question and hence conditioning on the previously generated proofs will help the model adjust the relevance of nodes and edges in the subsequent proofs. Quantitatively, we find that about 75% of the samples with 4 proofs have at least one node and one edge common across all the proofs (see Figure 5). Thus, we propose *Iterative*-MULTIPROVER (see Figure 4), which broadly consists of a base PROVER architecture, as in Figure 3 and an additional p node and edge encoders for generating a maximum of p proofs. The proofs are generated iteratively until an empty graph is generated to denote the end.

Base PROVER architecture computes the first level of node embeddings $\mathbf{N}^1 \in \mathbb{R}^{k \times d}$ and edge embeddings $\mathbf{E}^1 \in \mathbb{R}^{k^2 \times d}$. These are passed respectively through a node and edge classifier to generate the node probabilities $np^1 \in \mathbb{R}^k$ and edge probabilities $ep^1 \in \mathbb{R}^{k^2}$, corresponding to the first proof. In the next iteration, two transformer encoders generate the node and edge embeddings corresponding to the second proof. Specifically, we condition the generation of the next node embeddings $\mathbf{N}^2$ on the previous node ($\mathbf{N}^1$) and edge ($\mathbf{E}^1$) embeddings simultaneously. Conditioning on both is crucial because $\mathbf{N}^1$ captures the relevance of nodes for the first proof, while $\mathbf{E}^1$ contains information about the strength of the connections between these nodes. We condition $\mathbf{E}^2$ only on $\mathbf{E}^1$, because the edge embeddings corresponding to the nodes predicted by

$\mathbf{N}^1$ are already updated in $\mathbf{E}^1$. Formally,

$$\mathbf{T}^1 = W^{(1)}\mathbf{E}^1 W^{(2)}, W^{(1)} \in \mathbb{R}^{k \times k^2}, W^{(2)} \in \mathbb{R}^{3d \times d}$$
$$\mathbf{N}' = [\mathbf{N}^1; \mathbf{T}^1] W^{(3)}, W^{(3)} \in \mathbb{R}^{2d \times d}$$
$$\mathbf{N}^2 = \mathit{Transformer}(\mathbf{N}');\ \ \mathbf{E}^2 = \mathit{Transformer}(\mathbf{E}^1)$$

These next set of embeddings, when passed through the respective node and edge classifiers, predict the node probabilities $np^2 \in \mathbb{R}^k$ and edge probabilities $ep^2 \in \mathbb{R}^{k^2}$, denoting the likelihood of their presence in the second proof. We repeat this process of stacking up node and edge encoders for generating a maximum of p proofs. Given the node and edge probabilities corresponding to each proof and a QA probability from the QA module, we train *Iterative*-MULTIPROVER jointly with all proofs using the Hungarian Loss, described below.

3.5 Permutation-Invariant Hungarian Loss

Unlike words in text generation, proofs can be generated in any arbitrary order. Consequently, computing cross-entropy loss between the i^{th} predicted proof and the i^{th} gold proof, $i \in \{1, ..., p\}$ will be sub-optimal. Thus, we use a permutation-invariant Hungarian Loss (Zhang et al., 2019a,b) which finds the most optimal assignment between the predicted proofs and the gold proofs such that the overall loss is minimized. Formally, the Hungarian loss $\mathcal{L}_H$ and total loss $\mathcal{L}$ are denoted as follows:

$$\mathcal{L}_H = \min_{\pi \in \Pi} \sum_{i=1}^{p} CE(np^i, y_n^{\pi(i)}) + CE(ep^i, y_e^{\pi(i)})$$
$$\mathcal{L} = \mathcal{L}_{QA} + \mathcal{L}_H$$

where $CE(.,.)$ is the cross entropy loss, np^i and ep^i are the respective node and edge probabilities for the i^{th} predicted proof while $y_n^{\pi(i)} \in \{0,1\}^k$ and $y_e^{\pi(i)} \in \{0,1\}^{k^2}$ are the respective true node and edge labels for the gold proof $\pi(i)$, where π is the most optimal permutation. The Hungarian Loss is implemented by first summing the node and edge cross-entropy loss matrices $\mathcal{L}^n \in \mathbb{R}^{p \times p}$ and $\mathcal{L}^e \in \mathbb{R}^{p \times p}$ respectively, each entry (i, j) of which corresponds to the proof loss between the i^{th} predicted proof and j^{th} gold proof (see Figures 3 and 4). Then we find the best assignment between the gold and predicted proofs through the Hungarian algorithm (Kuhn and Yaw, 1955). Our final loss sums the Hungarian proof loss and the QA loss.

3.6 Integer Linear Program (ILP) Inference

Following PROVER, we generate valid proofs during inference using an ILP, subject to multiple global constraints (see Saha et al. (2020)). For each predicted proof, the predicted nodes and edge probabilities from MULTIPROVER, we obtain the corresponding predicted edges using Eqn. 1.

4 Experimental Setup

We experiment on synthetic, hand-authored zero-shot, and human paraphrased datasets, following Clark et al. (2020); Saha et al. (2020).

Datasets: The five synthetic datasets **DU0-DU5** consist of 100k questions with their own train, validation and test splits (70/10/20) and reasoning depths up to $D = 0, 1, 2, 3, 5$. Each example in these datasets is annotated with all possible proofs. The second dataset is a **Birds-Electricity** dataset, consisting of 5k hand-authored samples aimed at evaluating the zero-shot performance of the models. Unlike the previous datasets, all examples in this dataset have a unique gold proof. Third, **ParaRules** is a human-paraphrased dataset, consisting of 40k examples with all possible proofs, where the facts and rules are paraphrased by crowdworkers. Further details of the datasets and model's hyperparameters can be found in the appendix.

Evaluation Metrics: Following PROVER, QA evaluation is done through accuracy. For proofs, we compute the following metrics: (1) **Node Precision, Recall, F1** (2) **Edge Precision, Recall, F1**, (3) **Proof Precision, Recall, F1**, and (4) **Full Accuracy (FA)**. For each sample, given a set of gold proofs and predicted proofs, node precision is computed as the fraction of predicted proofs where the predicted node set matches exactly with a gold proof's node set. Similarly, node recall for each sample is computed as the fraction of gold proofs where the corresponding node sets match exactly. The overall node precision, recall and F1 are the respective sample-wise precision, recall and F1 scores averaged over all the samples. Edge metrics are computed similarly but with respect to the edges only and the proof metrics consider both nodes and edges in conjunction. Our final metric, full accuracy evaluates a sample as a whole and is given by the fraction of samples where the answer and all corresponding proofs are exactly correct.

5 Results and Analysis

5.1 Comparison of PROVER variants with MULTIPROVER

In Table 1, we compare ML-MULTIPROVER and IT-MULTIPROVER with five variants of PROVER

		Node			Edge			Proof			
	QA	P	R	F1	P	R	F1	P	R	F1	FA
PROVER (Saha et al., 2020)	99.3	89.2	84.9	86.0	87.5	84.2	85.3	87.1	84.0	84.7	81.2
PROVER-all	99.3	87.9	83.8	84.9	87.1	83.6	84.6	85.9	82.8	83.7	80.3
PROVER-top-p	99.3	34.4	88.4	48.4	33.8	87.4	47.7	33.3	86.7	47.2	00.0
PROVER-top-p-classifier	99.3	85.7	84.4	83.8	84.8	84.1	83.5	83.9	83.4	82.6	77.3
PROVER-top-p-threshold	99.3	84.4	88.0	85.0	83.6	87.1	84.4	83.0	86.5	83.8	77.2
ML-MULTIPROVER	**99.5**	89.4	89.2	89.0	87.7	87.8	87.4	87.2	87.3	87.0	83.8
IT-MULTIPROVER	**99.5**	**90.6**	**90.2**	**90.0**	**89.6**	**89.4**	**89.2**	**89.1**	**89.0**	**88.7**	**85.5**

Table 1: Comparison of our MULTIPROVER models with PROVER variations on DU5 test set. *Iterative*-MULTIPROVER's improvement in Full Accuracy over *Multilabel*-MULTIPROVER is statistically significant with $p < 0.001$.

– (1) PROVER, as introduced in Saha et al. (2020), trained with one proof per example and also generates a single proof, (2) PROVER-all, trained with all proofs as separate examples and generates a single proof per example, (3) PROVER-top-p, an extension of PROVER-all, generating top-p proofs for all examples, (4) PROVER-top-p-classifier, an improvement over the vanilla top-p model, where we first predict the number of proofs by training a RoBERTa classifier with concatenated question and context and then generate those many top proof graphs, and (5) PROVER-top-p-threshold, another improved model over vanilla top-p, where we use the optimization score from Equation 3 to predict the number of proofs to generate, i.e., we stop generating proofs when the score difference between two consecutive proofs exceeds a certain threshold (tuned on the validation set). All models are trained on the DU5 train set and tested on the corresponding test set. Based on Figure 2 which shows that 98% of the dataset contains samples with $\leq$ 3 proofs, we set max-proofs, $p = 3$. 87% of the examples in the dataset have a single gold proof, thereby making PROVER a strong baseline.

We observe that PROVER-all has a slightly lower proof F1 than PROVER, because the model likely gets confused with multiple possible proofs for the same context and question. PROVER-top-p's huge drop in precision is unsurprising because the subsequent non-empty proofs are always incorrect, causing full accuracy to drop to 0%. When we perform careful inference over PROVER either by predicting the number of proofs or by thresholding and do not generate a fixed p number of proofs for all examples, we observe a boost in precision over the vanilla top-p model, with very little drop in recall. However, PROVER continues to be a stronger baseline than all the top-p variants because of a lot of single-proof examples in the dataset.

Both MULTIPROVER models improve significantly on the state-of-the-art proof F1, while retaining a near perfect QA accuracy. IT-MULTIPROVER is a significantly stronger model because of its explicit conditioning mechanism and obtains up to a statistically significant[2] ($p < 0.001$) 4% improvement on proof F1 and full accuracy. While our model is expected to improve the proof recall compared to PROVER and PROVER-all because of the generation of multiple proofs, the improvement in precision is particularly important as it shows that the subsequently generated proofs by IT-MULTIPROVER are mostly correct. Similarly, its improvement in proof recall compared to PROVER-top-p also shows the strength of the model considering that PROVER-top-p generates the maximum number of proofs for every sample. Overall, IT-MULTIPROVER outperforms all other models in all metrics. In summary, careful inference strategies over a single-proof generation model like PROVER are largely ineffective for generating multiple proofs and an effective proof-set generation model needs to exploit and learn the inter-proof correlations during the training phase itself. Our experiments on the ParaRules dataset demonstrate similar findings, details of which and the effect of varying p for MULTIPROVER is in the appendix.

Iterative-MULTIPROVER performs equally well on the subset of questions where the context has *negations*, achieving a high proof F1 of 90.8. As part of error analysis, we find that 58% of *Iterative*-MULTIPROVER's wrongly predicted proofs have more nodes and edges than those in the gold proof, suggesting that our model tends to overestimate the essential rules and facts and their inter-connections. In the following subsections, we analyze MULTIPROVER's generalization capabilities in three dif-

[2]We use bootstrap test (Efron and Tibshirani, 1994) for calculating the statistical significance score.

		Node			Edge			Proof			
	QA	P	R	F1	P	R	F1	P	R	F1	FA
PROVER (Saha et al., 2020)	**86.5**	81.3	81.3	81.3	81.4	81.4	81.4	80.7	80.7	80.7	80.7
PROVER-all	85.9	80.9	80.9	80.9	80.4	80.4	80.4	80.2	80.2	80.2	80.0
ML-MULTIPROVER	85.1	79.2	79.9	79.4	79.4	79.9	79.5	78.7	79.1	78.8	78.1
IT-MULTIPROVER	86.3	**82.7**	**83.3**	**82.9**	**82.4**	**83.0**	**82.6**	**82.2**	**82.7**	**82.3**	**81.8**

Table 2: Comparison of all models on the zero-shot Birds-Electricity dataset containing one gold proof per sample. *Iterative*-MULTIPROVER's improvement in Full Accuracy over PROVER is statistically significant with $p < 0.001$.

		Proof F1			Full Acc		
d	MP	PR	ML	IT	PR	ML	IT
0	7.2	93.8	97.8	**98.2**	92.6	96.7	**97.0**
1	10.3	88.0	92.8	**93.5**	85.7	91.0	**91.7**
2	15.7	80.8	86.1	**87.1**	76.5	81.8	**83.7**
3	17.7	78.0	80.7	**83.0**	72.2	75.9	**78.0**
4	19.9	71.1	72.3	**77.2**	65.9	66.4	**70.1**
5	23.1	67.7	64.9	**70.6**	61.0	58.7	**63.7**

Table 3: Comparison of PROVER-all and MULTIPROVER models on the subset of samples in DU5 test set requiring d depth of reasoning.

ferent contexts – zero-shot settings, higher depth questions and training with less training data.

5.2 Generalization to Zero-Shot Dataset with Single Gold Proofs

The Birds-Electricity test-only dataset evaluates the zero-shot performance. It contains examples with single gold proofs; hence, if a multiple-proof generation model like MULTIPROVER transfers well to it, this indicates strong generalization capabilities because along with generating correct proofs, it also needs to infer the correct number of proofs. With that motivation, in Table 2, we compare PROVER and PROVER-all, both trained on DU5 to generate a single proof, with our MULTIPROVER models, also trained on DU5 and find that IT-MULTIPROVER obtains state-of-the-art result on all proof-related metrics, while retaining the QA performance. Note that IT-MULTIPROVER has two important design choices which explains its good performance on out-of-domain transfer – (1) it trains on all proofs jointly, (2) explicit proof conditioning. Both of these, when combined, enable it to learn the correlations between the proofs to identify the degree of relevance of facts and rules, ranging from essential to sometimes useful to irrelevant, for a given question. Thus, on out-of-domain test data, it assigns soft prior relevance scores to the context which helps it to better learn the significantly smaller space of correct proofs and be more accurate even for a single-proof dataset.

	QA		Proof F1		Full Acc	
Count	ML	IT	ML	IT	ML	IT
10k	**87.2**	86.1	**41.5**	41.4	39.0	**39.5**
30k	97.7	**98.2**	74.3	**74.9**	71.2	**72.0**
50k	**99.4**	**99.4**	83.7	**84.5**	80.0	**81.0**
70k (All)	**99.5**	**99.5**	87.0	**88.7**	83.8	**85.5**

Table 4: Comparative study between the two MULTIPROVER models with varying amount of training data on DU5. Count = number of training examples.

5.3 Generalization to Higher Depths

The DU5 dataset consists of questions requiring reasoning up to a maximum depth of 5. Thus, we test the generalization capabilities of the MULTIPROVER models on higher depth questions. Specifically, in Table 3, we compare the DU5-trained models of PROVER-all, ML-MULTIPROVER and IT-MULTIPROVER on the subset of DU5 test examples with varying depths of reasoning (d). Each row also shows the percentage of examples with multiple gold proofs (MP) which, unsurprisingly, increases as the depth increases. We observe that much of IT-MULTIPROVER's improvement compared to ML-MULTIPROVER comes at higher depths where the presence of multiple proofs is a more frequent phenomenon. At depth-5, where 23% of the examples have > 1 correct proof, IT-MULTIPROVER obtains a 6% improvement over ML-MULTIPROVER. This shows that joint training with all proofs and explicit conditioning between them leads to better generalization at higher depths.

5.4 Generalization with Less Training Data

Collecting proofs for supervised training is expensive in most real-world scenarios. Hence, on top of the zero-shot and depth generalization results presented so far, we ask if our MULTIPROVER models can learn from less training data. Table 4 shows that these models obtain near perfect QA accuracy with only 40% of the training data (30k examples). However, proof generation proves to be challenging and only improves with sufficient training

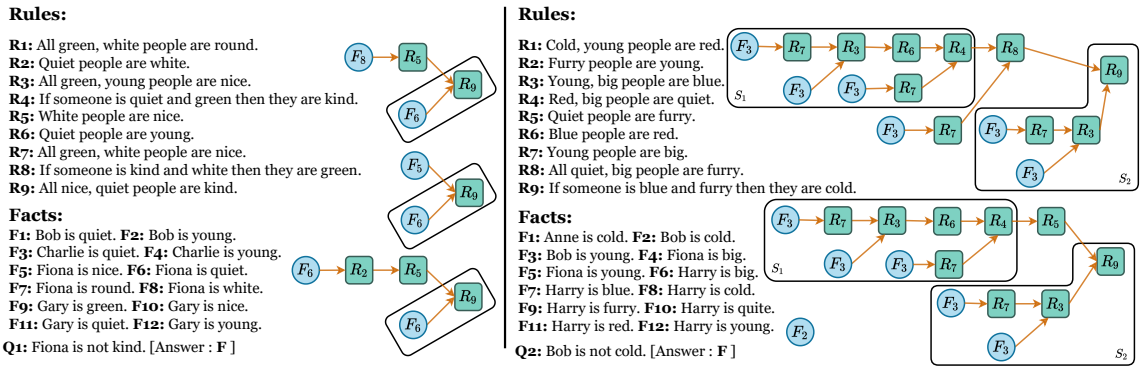

Figure 6: Figure showing all the proofs correctly generated by our *Iterative*-MULTIPROVER model for two randomly chosen questions corresponding to two different rule-bases.

data. Another interesting observation is that while both MULTIPROVER models perform comparably with less training data, IT-MULTIPROVER starts to outperform ML-MULTIPROVER upon training with more examples. IT-MULTIPROVER consists of more trainable parameters because of its multiple node and edge encoders, which get learned better with more data. See appendix for runtime and parameter space of these models.

5.5 Comparison of MULTIPROVER with the Skyline Single-Proof Generation Model

We find that an ideal (skyline) single-proof generation model's proof recall for the DU5 dataset is upper-bounded by 92% as it contains about 87% of single-proof examples. This is computed by considering exactly 1 correct proof per question. Hence, we ask how well our MULTIPROVER models compare with this ideal performance (Figure 7). Our results are encouraging, not only because IT-MULTIPROVER generates more correct proofs than all other models but also because it almost matches the performance of the skyline single-proof generation model. The PROVER model is 9.2% worse as compared to the skyline single-proof generation model while IT-MULTIPROVER reduces this gap to 3%. Given the dataset mostly contains single-proof examples, the skyline is a strong upper-bound on proof generation performance and IT-MULTIPROVER significantly reduces the gap. See appendix for ablations of IT-MULTIPROVER, including the effect of Hungarian Loss.

6 Qualitative Analysis of MULTIPROVER

Fig. 6 shows the sets of proofs correctly generated by *Iterative*-MULTIPROVER for two randomly chosen questions. For Q_1, it generates all the possible proofs by identifying the common subgraph

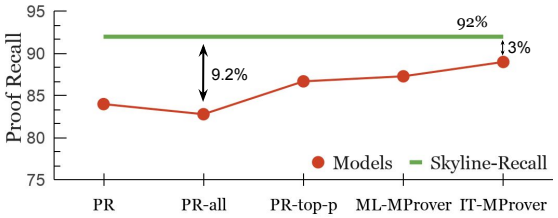

Figure 7: Comparison of proof recall for all models with that of the skyline single-proof generation model.

$F_6 \rightarrow R_9$. Q_2 is interesting, because (i) the single-node proof F_2 is significantly different from the other proofs in both structure and size, and (ii) the two larger proofs have two distinct common subgraphs. Here, PROVER performs simple lookup in the rule-base to generate the proof F_2, thereby limiting our understanding of its reasoning capabilities. However, MULTIPROVER, through its ability to also generate the larger and more complex proofs enhances the transparency and verification of its reasoning abilities, and hence is a crucial step towards bridging the gap between neural and symbolic approaches.

7 Conclusion

We proposed *Multilabel*-MULTIPROVER and *Iterative*-MULTIPROVER, two variants of a proof-set generation model where the former performs implicit conditioning between the proofs to generate them in parallel while the latter generates a proof-set through explicit conditioning on the previously generated proofs. Both models obtain strong proof F1 improvements on synthetic and human-paraphrased datasets and *Iterative*-MULTIPROVER also obtains state-of-the-art proof F1 on a zero-shot dataset with single proofs. MULTIPROVER's modeling is fairly generic and similar methods can be used in generating a set of structured explanations for other NLP tasks like multi-hop QA.

Ethical Considerations

Despite the overwhelming success of pre-trained language models for various NLP tasks, a common criticism is their lack of interpretability. Generating structured proofs from such models allows us to explain their reasoning capabilities and also bridges the gap between neural and symbolic systems. In this work, we take a step closer towards improving the interpretability of rule-based reasoning by generating a set of multiple proofs, each providing a diverse rationale for the reasoning process. We experiment with a wide variety of rule-bases ranging from synthetic to hand-authored to human-paraphrased rule-bases. Our results show good generalization performance of our models across three different aspects – (1) zero-shot settings, (2) questions requiring higher depths of reasoning, and (3) availability of less training data. We hope our models and findings will inspire future work on generating multiple structured explanations for different compositional reasoning tasks in NLP.

Acknowledgements

We thank the reviewers and Peter Hase for their helpful feedback. This work was supported by DARPA MCS Grant N66001-19-2-4031, NSF-CAREER Award 1846185, DARPA YFA17-D17AP00022, ONR Grant N00014-18-1-2871, Microsoft Investigator Fellowship, and Munroe & Rebecca Cobey Fellowship. The views in this article are those of the authors and not the funding agency.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *ICLR*.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on freebase from question-answer pairs. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1533–1544.

Jonathan Berant and Percy Liang. 2014. Semantic parsing via paraphrasing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1415–1425.

Oana-Maria Camburu, Tim Rocktäschel, Thomas Lukasiewicz, and Phil Blunsom. 2018. e-SNLI: Natural language inference with natural language explanations. In *Advances in Neural Information Processing Systems*, pages 9539–9549.

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *EMNLP*.

Peter Clark, Oyvind Tafjord, and Kyle Richardson. 2020. Transformers as soft reasoners over language. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 3882–3890. International Joint Conferences on Artificial Intelligence Organization. Main track.

Bo Dai, Sanja Fidler, Raquel Urtasun, and Dahua Lin. 2017. Towards diverse and natural image descriptions via a conditional gan. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2970–2979.

Jay DeYoung, Sarthak Jain, Nazneen Fatema Rajani, Eric Lehman, Caiming Xiong, Richard Socher, and Byron C Wallace. 2020. Eraser: A benchmark to evaluate rationalized nlp models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4443–4458.

Finale Doshi-Velez and Been Kim. 2017. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*.

Bradley Efron and Robert J Tibshirani. 1994. *An introduction to the bootstrap*. CRC press.

Tian Gao, Jie Chen, Vijil Chenthamarakshan, and Michael Witbrock. 2019. A sequential set generation method for predicting set-valued outputs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 2835–2842.

Kevin Gimpel, Dhruv Batra, Chris Dyer, and Gregory Shakhnarovich. 2013. A systematic exploration of diversity in machine translation. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1100–1111, Seattle, Washington, USA. Association for Computational Linguistics.

Nicolas Gontier, Koustuv Sinha, Siva Reddy, and Chris Pal. 2020. Measuring systematic generalization in neural proof generation with transformers. *Advances in Neural Information Processing Systems*, 33.

Peter Hase and Mohit Bansal. 2020. Evaluating explainable AI: Which algorithmic explanations help users predict model behavior? In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*.

Alon Jacovi and Yoav Goldberg. 2020. Towards faithfully interpretable nlp systems: How should we define and evaluate faithfulness? In *ACL*.

Unnat Jain, Ziyu Zhang, and Alexander G Schwing. 2017. Creativity: Generating diverse questions using variational autoencoders. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6485–6494.

Peter Jansen and Dmitry Ustalov. 2019. Textgraphs 2019 shared task on multi-hop inference for explanation regeneration. In *Proceedings of the Thirteenth Workshop on Graph-Based Methods for Natural Language Processing (TextGraphs-13)*, pages 63–77.

Peter A Jansen, Elizabeth Wainwright, Steven Marmorstein, and Clayton T Morrison. 2018. Worldtree: A corpus of explanation graphs for elementary science questions supporting multi-hop inference. *arXiv preprint arXiv:1802.03052*.

Harsh Jhamtani and Peter Clark. 2020. Learning to explain: Datasets and models for identifying valid reasoning chains in multihop question-answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 137–150.

Tushar Khot, Peter Clark, Michal Guerquin, Peter Jansen, and Ashish Sabharwal. 2020. Qasc: A dataset for question answering via sentence composition. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 05, pages 8082–8090.

Adam R Kosiorek, Hyunjik Kim, and Danilo J Rezende. 2020. Conditional set generation with transformers. *arXiv preprint arXiv:2006.16841*.

H. W. Kuhn and Bryn Yaw. 1955. The hungarian method for the assignment problem. *Naval Res. Logist. Quart*, pages 83–97.

Ilya Kulikov, Alexander H Miller, Kyunghyun Cho, and Jason Weston. 2018. Importance of a search strategy in neural dialogue modelling. In *iNLG*.

Sawan Kumar and Partha Talukdar. 2020. NILE: Natural language inference with faithful natural language explanations. In *ACL*.

Juho Lee, Yoonho Lee, Jungtaek Kim, Adam R. Kosiorek, Seungjin Choi, and Yee Whye Teh. 2018. Set transformer: A framework for attention-based permutation-invariant neural networks. In *ICML*.

Tao Lei, Regina Barzilay, and Tommi Jaakkola. 2016. Rationalizing neural predictions. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 107–117.

Jiwei Li, Will Monroe, and Dan Jurafsky. 2016. A simple, fast diverse decoding algorithm for neural generation. *arXiv preprint arXiv:1611.08562*.

Kevin Lin, Oyvind Tafjord, Peter Clark, and Matt Gardner. 2019. Reasoning over paragraph effects in situations. In *Proceedings of the 2nd Workshop on Machine Reading for Question Answering*, pages 58–62, Hong Kong, China. Association for Computational Linguistics.

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

Bill MacCartney and Christopher D Manning. 2014. Natural logic and natural language inference. In *Computing meaning*, pages 129–147. Springer.

Mark A Musen and Johan Van Der Lei. 1988. Of brittleness and bottlenecks: Challenges in the creation of pattern-recognition and expert-system models. In *Machine Intelligence and Pattern Recognition*, volume 7, pages 335–352. Elsevier.

Allen Newell and Herbert Simon. 1956. The logic theory machine–a complex information processing system. *IRE Transactions on information theory*, 2(3):61–79.

Adarsh Prasad, Stefanie Jegelka, and Dhruv Batra. 2014. Submodular meets structured: Finding diverse subsets in exponentially-large structured item sets. In *Advances in Neural Information Processing Systems*, volume 27, pages 2645–2653. Curran Associates, Inc.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67.

Nazneen Fatema Rajani, Bryan McCann, Caiming Xiong, and Richard Socher. 2019. Explain yourself! leveraging language models for commonsense reasoning. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4932–4942.

Kyle Richardson, Hai Hu, Lawrence Moss, and Ashish Sabharwal. 2020. Probing natural language inference models through semantic fragments. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8713–8721.

Cynthia Rudin. 2019. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215.

Swarnadeep Saha, Sayan Ghosh, Shashank Srivastava, and Mohit Bansal. 2020. PRover: Proof generation for interpretable reasoning over rules. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 122–136.

Tianxiao Shen, Myle Ott, Michael Auli, and Marc'Aurelio Ranzato. 2019. Mixture models for diverse machine translation: Tricks of the trade. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 5719–5728, Long Beach, California, USA. PMLR.

Raphael Shu, Hideki Nakayama, and Kyunghyun Cho. 2019. Generating diverse translations with sentence codes. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1823–1827.

Oyvind Tafjord, Matt Gardner, Kevin Lin, and Peter Clark. 2019. QuaRTz: An open-domain dataset of qualitative relationship questions. 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 5941–5946, Hong Kong, China. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Ashwin K. Vijayakumar, Michael Cogswell, Ramprasaath R. Selvaraju, Qing Sun, Stefan Lee, David J. Crandall, and Dhruv Batra. 2018. Diverse beam search: Decoding diverse solutions from neural sequence models. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

O. Vinyals, A. Toshev, S. Bengio, and D. Erhan. 2015. Show and tell: A neural image caption generator. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3156–3164.

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

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45.

Zhengnan Xie, Sebastian Thiem, Jaycie Martin, Elizabeth Wainwright, Steven Marmorstein, and Peter Jansen. 2020. Worldtree v2: A corpus of science-domain structured explanations and inference patterns supporting multi-hop inference. In *Proceedings of The 12th Language Resources and Evaluation Conference*, pages 5456–5473.

Qiongkai Xu, Juyan Zhang, Lizhen Qu, Lexing Xie, and Richard Nock. 2018. D-page: Diverse paraphrase generation. *arXiv preprint arXiv:1808.04364*.

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

Qinyuan Ye, Xiao Huang, Elizabeth Boschee, and Xiang Ren. 2020. Teaching machine comprehension with compositional explanations. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings*, pages 1599–1615.

Mo Yu, Shiyu Chang, Yang Zhang, and Tommi Jaakkola. 2019. Rethinking cooperative rationalization: Introspective extraction and complement control. 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 4085–4094.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. 2017. Deep sets. In *Advances in Neural Information Processing Systems*, volume 30, pages 3391–3401. Curran Associates, Inc.

Omar Zaidan, Jason Eisner, and Christine Piatko. 2007. Using "annotator rationales" to improve machine learning for text categorization. In *Human language technologies 2007: The conference of the North American chapter of the association for computational linguistics; proceedings of the main conference*, pages 260–267.

Luke S. Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, UAI'05, page 658–666. AUAI Press.

Hongming Zhang, Xinran Zhao, and Yangqiu Song. 2020. WinoWhy: A deep diagnosis of essential commonsense knowledge for answering Winograd schema challenge. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5736–5745, Online. Association for Computational Linguistics.

Yan Zhang, Jonathon Hare, and Adam Prugel-Bennett. 2019a. Deep set prediction networks. In *Advances in Neural Information Processing Systems*, volume 32, pages 3212–3222. Curran Associates, Inc.

Yan Zhang, Jonathon Hare, and Adam Prügel-Bennett. 2019b. Fspool: Learning set representations with featurewise sort pooling. In *International Conference on Learning Representations*.

		Node			Edge			Proof			
	QA	P	R	F1	P	R	F1	P	R	F1	FA
PROVER (Saha et al., 2020)	99.3	90.0	85.3	86.4	88.6	85.6	86.2	88.0	84.7	85.5	82.1
PROVER-all	99.4	88.4	84.2	85.3	87.9	84.4	85.4	86.5	83.5	84.3	81.1
PROVER-top-p	99.4	34.4	88.6	48.4	34.0	88.0	48.0	33.4	87.3	47.4	00.0
PROVER-top-p-classifier	99.4	86.2	85.1	84.4	85.6	85.1	84.4	84.6	84.2	83.4	78.2
PROVER-top-p-threshold	99.4	85.0	88.4	85.6	84.5	87.8	85.2	83.9	87.1	84.5	78.0
ML-MULTIPROVER	99.3	89.9	89.6	89.3	88.3	88.3	88.0	87.7	87.6	87.3	83.9
IT-MULTIPROVER-seq	99.4	88.5	86.8	87.2	87.4	86.0	86.3	86.7	85.3	85.6	81.4
IT-MULTIPROVER-nec	99.3	90.2	89.7	89.5	89.4	89.1	89.0	89.0	88.6	88.5	85.2
IT-MULTIPROVER	**99.5**	**90.6**	**90.5**	**90.1**	**89.9**	**90.0**	**89.5**	**89.4**	**89.4**	**89.0**	**85.3**

Table 5: Comparison of our final MULTIPROVER models with variants of PROVER and other ablations of IT-MULTIPROVER on the validation set of DU5. IT-MULTIPROVER-seq = *Iterative*-MULTIPROVER with sequential loss. IT-MULTIPROVER-nec = *Iterative*-MULTIPROVER with no edge conditioning. The final *Iterative*-MULTIPROVER model outperforms all other models across all metrics.

A Appendix

A.1 Experimental Setup

MULTIPROVER is developed on top of the Hugging Face transformers library (Wolf et al., 2020).[3] Experiments with PROVER (Saha et al., 2020) are performed using their publicly released code and hyperparameters.[4] All MULTIPROVER hyperparameters are chosen based on the best Full Accuracy on the corresponding validation sets. We use RoBERTa-large (Liu et al., 2019) as the pre-trained language model. The batch size and maximum sequence length are set to 8 and 300 respectively. We train all our models for a maximum of 7 epochs using an initial learning rate of 10^{-5}, a weight decay of 0.1 and a dropout probability of 0.1. We use a random seed of 42 across all our experiments. All experiments are performed on one V100 Volta GPU. Batch size and learning rate are manually tuned in the range {8,16} and $\{10^{-5}, 2 * 10^{-5}\}$ respectively. For inference, we use PROVER's ILP optimization code, which is modeled using PuLP.[5] In all the datasets, the maximum number of facts and rules corresponding to a context is 25.

A.2 Datasets

Our experiments are conducted on the datasets introduced in Clark et al. (2020).[6] These consist of 5 datasets with synthetic rule-bases, DU0-DU5, a zero-shot test-only dataset called Birds-Electricity and a dataset with human-paraphrased rules called ParaRules. All datasets, except Birds-Electricity, have their corresponding train, validation and test splits.

DU0-DU5: Each of these consists of 100k questions with sythetic rule-bases and requires reasoning chains up to a maximum depth of D ($D = 0, 1, 2, 3, 5$). The number of train, validation and test examples in each of the datasets are 70k, 10k and 20k respectively. Further, each question in the datasets is annotated with all possible proofs. The total number of proofs in the DU5 train set range from 1 to 1350, with a mean and median of 1.45 and 1 respectively.

Birds-Electricity: The Birds-Electricity dataset comprises of two test-only datasets where the contexts are about birds and electric circuits. The vocabulary of the entities, attributes and predicates, apart from `is()` are all new at test time, thus providing a benchmark for testing the generalization capability of the models on out-of-distribution data. Another interesting aspect of this dataset is that all examples are annotated with a unique gold proof.

ParaRules: The ParaRules dataset is one where the facts and rules are paraphrased by humans into more natural language. It consists of a total of 40k questions, with 28k, 4k, and 8k questions in the train, validation and test splits respectively. This dataset tests the model's ability to reason over more complex human-like language. Similar to the synthetic datasets, each example is annotated with all possible proofs.

A.3 Syntax of Proof Graph

Each proof $\mathcal{P}_i = (\mathcal{V}_i, \mathcal{E}_i)$ is a directed graph, with a set of nodes $\mathcal{V}_i \subseteq \mathcal{N}$ and a set of edges $\mathcal{E}_i \subseteq \mathcal{V}_i \times \mathcal{V}_i$. Each node $n_i \in \mathcal{N}$ is either a fact $F \in \mathcal{F}$ or a rule

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

[4] https://github.com/swarnaHub/PRover

[5] https://pypi.org/project/PuLP/

[6] https://rule-reasoning.apps.allenai.org/

		Node			Edge			Proof			
p	QA	P	R	F1	P	R	F1	P	R	F1	PA
2	99.4	90.5	89.1	89.2	89.3	88.3	88.4	88.8	87.8	87.9	84.4
3	99.3	89.9	89.6	89.3	88.3	88.3	88.0	87.7	87.6	87.3	83.9
4	99.3	89.1	89.2	88.8	87.8	82.0	87.8	87.2	87.5	87.1	83.6
5	99.2	88.6	89.1	88.5	87.2	87.8	87.2	86.6	87.2	86.6	83.1

Table 6: Effect of varying maximum number of proofs (p) on *Multilabel*-MULTIPROVER. All models are trained on the DU5 training set and evaluated on the corresponding validation set. The proof metrics start to decrease marginally with increase in p.

		Node			Edge			Proof			
p	QA	P	R	F1	P	R	F1	P	R	F1	PA
2	99.5	90.0	89.0	89.0	89.2	88.4	88.3	88.6	87.8	87.7	84.1
3	99.5	90.6	90.5	90.1	89.9	90.0	89.5	89.4	89.4	89.0	85.3
4	99.5	90.2	89.7	89.5	89.5	89.2	89.1	89.1	88.7	88.6	85.2
5	99.5	90.1	89.6	89.4	89.5	89.2	89.1	89.0	88.6	88.5	85.2

Table 7: Effect of varying maximum number of proofs (p) on *Iterative*-MULTIPROVER. All models are trained on the DU5 training set and evaluated on the corresponding validation set. Unlike *Multilabel*-MULTIPROVER, it is significantly robust to variation in p.

$R \in \mathcal{R}$ from the context or a special **NAF** node, denoting "Negation as Failure". A **NAF** node in a proof indicates the truthfulness of the negation of statement(s) that cannot be proved using the set of rules (under closed-world assumption). Edges in the graph can be directed either from a fact (or **NAF**) to a rule or between two rules. An edge from a fact to a rule means that the rule applies on the fact to generate a new fact. Similarly, an edge from a rule $R_1 \in \mathcal{R}$ to another rule $R_2 \in \mathcal{R}$ implies the application of R_2 on the fact generated by R_1. Proofs are either successful or failed. A successful proof is one where the question statement can be logically reached (to be either proved or disproved) using the given rule-base while for failed proofs, no conclusion can be reached, in which case the shallowest branch of the proof tree that fails is generated. For more details and examples of proofs, we refer the readers to prior work (Saha et al., 2020; Clark et al., 2020).

A.4 Ablation Analysis

In Table 5, we compare our baselines PROVER, PROVER-all and PROVER-top-p variants with our MULTIPROVER models on the validation set of DU5 dataset. Additionally, we also show two ablations of IT-MULTIPROVER – in the first, we replace the Hungarian loss with a sequential loss, which computes the cross-entropy loss of the i^{th} predicted proof with the i^{th} gold proof and in the second, we condition the node embeddings on the previous node embeddings only instead of both node and edge embeddings. All models, except PROVER and PROVER-all, generate a maximum of 3 proofs. PROVER-top-p suffers from a huge drop in proof precision due to the generation of many incorrect proofs. Although carefully choosing the value of p either by thresholding or through a classifier helps boost the proof precision, PROVER continues to be a superior baseline on this dataset due to a high skew towards single-proof examples. ML-MULTIPROVER improves upon PROVER's proof F1 and full accuracy (FA) which are further bettered by IT-MULTIPROVER, owing to its explicit conditioning mechanism between the proofs. Replacing the Hungarian loss with a sequential loss leads to a significant drop in proof F1, thereby showing the effectiveness of modeling multiple proof generation as a set generation problem. Finally, conditioning the node embeddings on both node and edge embeddings leads to marginal improvement in proof F1. Overall, IT-MULTIPROVER outperforms all other models across all metrics.

A.5 MULTIPROVER with Varying Maximum Number of Proofs

We analyze the effect of varying the maximum number of proofs p on ML-MULTIPROVER and IT-MULTIPROVER in Table 6 and 7 respectively. All models are trained on the DU5 training set and evaluated on the corresponding validation set. Although all models maintain the QA accuracy, we

		Node			Edge			Proof			
	QA	P	R	F1	P	R	F1	P	R	F1	FA
PROVER-all	98.6	95.9	94.1	94.5	95.4	93.8	94.3	95.3	93.7	94.2	92.3
PROVER-top-p	98.6	39.3	96.6	55.0	38.9	96.0	54.6	38.9	95.9	54.5	00.1
ML-MULTIPROVER	**98.9**	96.7	96.4	96.4	96.4	96.2	96.2	96.2	96.0	96.0	95.2
IT-MULTIPROVER	**98.9**	**97.3**	**97.2**	**97.2**	**97.2**	**97.0**	**97.0**	**96.8**	**96.7**	**96.7**	**96.1**

Table 8: Comparison of models trained on DU3 and ParaRules training sets and evaluated on ParaRules validation set. IT-MULTIPROVER outperforms all other models across all metrics.

		Node			Edge			Proof			
	QA	P	R	F1	P	R	F1	P	R	F1	FA
PROVER-all	98.2	95.3	92.8	93.5	94.7	92.7	93.3	94.4	92.4	93.0	90.5
PROVER-top-p	98.2	38.7	95.9	54.3	38.3	95.5	53.9	38.2	95.3	53.8	00.1
ML-MULTIPROVER	**98.3**	96.0	95.6	95.7	95.9	95.5	95.6	95.5	95.2	95.2	93.8
IT-MULTIPROVER	**98.3**	**96.8**	**96.2**	**96.3**	**96.5**	**96.3**	**96.3**	**96.2**	**96.0**	**96.0**	**94.5**

Table 9: Comparison of models trained on DU3 and ParaRules training sets and evaluated on ParaRules test set. IT-MULTIPROVER outperforms all other models across all metrics.

find that the proof F1 for ML-MULTIPROVER starts to decrease marginally with the increase in p. Note that this model is trained with padding of empty proof graphs since it generates all p proofs in parallel. Thus, the amount of padding increases with the increase in p, thereby leading to a harder learning problem as the model needs to predict more number of empty graphs. IT-MULTIPROVER, on the other hand, is significantly robust to such variations in p, because it generates proofs iteratively with one empty graph at the end, indicating end of set.

A.6 Evaluation on Human-Paraphrased Rule-Bases

Following PROVER, we also test MULTIPROVER's effectiveness in generating proofs for more human-like complex rule-bases. The ParaRules dataset is constructed by first creating a set of fact groups where each fact group consists of all facts in the theory concerning a particular person and then paraphrasing these fact groups into more complex language. E.g., a fact group "Alan is blue. Alan is rough. Alan is young.", may be re-worded into "Alan is on the young side, but rough. He often feels rather blue." Thus, unlike the DU datasets or the Birds-Electricity dataset where the proof graphs are composed of facts and rules, ParaRules proofs are composed of fact groups and rules. Following past work (Clark et al., 2020; Saha et al., 2020) we train our models combining the DU3 and ParaRules train sets, and evaluate on the ParaRules validation and test set in Tables 8 and 9 respectively. We find that similar conclusions to the DU5 dataset hold for this dataset as well – ML-MULTIPROVER achieves a better proof F1 and full accuracy than PROVER, which are further improved by IT-MULTIPROVER due to its explicit conditioning mechanism between the proofs.

	# Parameters			Time/epoch (in hours)		
p	PR	ML	IT	PR	ML	IT
1	361M	361M	488M	5.0	3.4	3.6
2	361M	361M	615M	5.0	3.5	4.0
3	361M	361M	742M	5.0	3.6	4.6
4	361M	361M	869M	5.0	3.7	5.1
5	361M	361M	996M	5.0	3.8	5.7

Table 10: Comparative study of the number of parameters and training time per epoch (in hours) for PROVER-all (PR), ML-MULTIPROVER and IT-MULTIPROVER with varying number of maximum proofs (p).

A.7 Training Time and Size Comparison

Table 10 shows the number of trainable parameters and training times per epoch for the baseline model PROVER and our proposed models, ML-MULTIPROVER and IT-MULTIPROVER across varying number of maximum proofs (p) per sample. Since ML-MULTIPROVER adopts the same PROVER architecture but with multi-label classification, it has the same number of parameters as PROVER, which also remains unchanged irrespective of the maximum number of proofs. The number of parameters for IT-MULTIPROVER, however, increases with the increase in p because of the presence of multiple node and edge encoders.

While IT-MULTIPROVER has more parameters than PROVER, our empirical findings reveal that just having a similarly-sized, larger PROVER model will not be sufficient and exploiting the correlations between multiple proofs with a permutation-invariant loss is necessary for the task of generating a set of multiple proofs.

The training time of PROVER is more than that of ML-MULTIPROVER because the former treats each proof as a separate example, causing an increase in the training data size from 70k to 110k. ML-MULTIPROVER is the most time-efficient model and its running time only increases marginally with the increase in p. This is due to the additional node and edge classifications that the model has to perform corresponding to each extra proof. Unsurprisingly, IT-MULTIPROVER takes longer to train but encouragingly for $p \leq 4$, still has a comparable running time to PROVER.