ProofInfer: Generating Proof via Iterative Hierarchical Inference

Zichu Fei[1,2], **Qi Zhang**[1,2]**Xin Zhou**[1,2], **Tao Gui**[3], **Xuanjing Huang**[1,2,3]
School of Computer Science, Fudan Unviersity[1]
Shanghai Key Laboratory of Intelligent Information Processing, Shanghai, China[2]
Institute of Modern Languages and Linguistics, Fudan University, Shanghai, China[3]
{zcfei19, qz, xzhou20, tgui,xjhuang}@fudan.edu.cn

Abstract

Proof generation focuses on deductive reasoning: given a hypothesis and a set of theories, including some supporting facts and logical rules expressed in natural language, the model generates a proof tree indicating how to deduce the hypothesis from given theories. Current models with state-of-the-art performance employ the stepwise method, linking an individual node to the proof step-by-step. However, these methods actually focus on generating several proof paths rather than a whole tree. To address this problem, we propose ProofInfer, which generates the proof tree via iterative hierarchical inference. At each step, ProofInfer generates the entire layer for proof tree, where all nodes in this layer are generated simultaneously. Since the conventional autoregressive generation architecture cannot simultaneously predict multiple nodes, ProofInfer employs text-to-text paradigm to avoid it. To this end, we propose a divide-and-conquer algorithm to encode the proof tree as the plain text recursively without structure information loss. Experimental results show that ProofInfer significantly outperforms the state-of-the-art (SOTA) models on several widely-used datasets. In addition, ProofInfer still performs well with data-limited, achieving comparable performance to the SOTA models with only **40%** of the training data. [1]

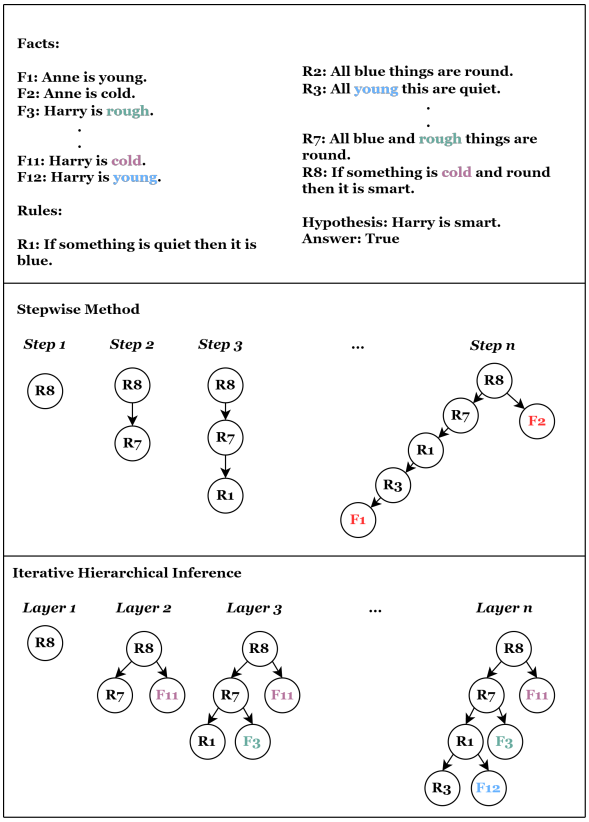

Figure 1: An example of the proof generation task. The existing stepwise methods add the nodes and edges step by step. They focus on generating several proof paths instead of the whole proof tree, so they often link the wrong branches (the nodes with red text) and miss the critical branches (the nodes with other color text).

1 Introduction

Automated reasoning is a fundamental goal of AI (Newell and Simon, 1956; McCarthy et al., 1960): the ability to draw valid conclusions from explicitly provided knowledge. Traditional research in automated reasoning focus on structured domains such as formal logic (Robinson and Voronkov, 2001; Musen and Van der Lei, 1988). Recently, (Clark et al., 2021) proposed a new version of the task by replacing the formal representations of rule-bases with natural language. Given a hypothesis and a set of theories, including some supporting facts and logical rules expressed in natural language, the model generates a proof tree indicating how the hypothesis is deduced from a subset of the theories.

Existing works generates the proof either in a single shot (Saha et al., 2020; Sun et al., 2021) or step-by-step (Liang et al., 2021; Tafjord et al., 2021; Yang et al., 2022; Qu et al., 2022). Stepwise

[1]Our code and models are publicly available at https://github.com/sion-zcfei/ProofInfer

Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, pages 10883–10892
December 7-11, 2022

methods link an individual node to the proof step-by-step, which leverage the compositionality of proofs, making it easier for the model to learn and generalize to long proofs (Tafjord et al., 2021). However, the existing stepwise methods actually focus on generating several proof paths rather than a whole tree (Yang et al., 2022; Qu et al., 2022). During generation, these models only focus on the most relevant areas of the currently generated path while neglecting the rest of the proof tree. This problem makes the existing stepwise methods struggle to generate proof steps that are both valid and relevant (Sanyal et al., 2022; Yang et al., 2022) as shown in Figure 1 .

To address this challenge, we propose ProofInfer, which generates the proof tree via iterative hierarchical inference. Unlike the stepwise methods to link nodes *one-by-one*, the iterative hierarchical inference takes the hypothesis as the root node and infers the proof tree *layer-by-layer* until each intermediate node in the tree finds its supporting facts. At each step, ProofInfer adds the entire layer to the proof, where all nodes in this layer are generated simultaneously. Since convention autoregressive generation architecture with teacher-forcing (See et al., 2017; Lewis et al., 2020; Radford et al., 2019) cannot simultaneously predict the multiple nodes at each step (Qi et al., 2020; Goodman et al., 2020), ProofInfer employs the text-to-text paradigm to generate the proof instead of generating the tree structure directly. In detail, we map the generated proof tree to a plain text and take it as input for a generation model to infer the proof text with a new layer, which can be decoded into the corresponding tree structure. Text-to-text paradigm can ensure that all the nodes with the same depth in the proof tree are generated simultaneously. Furthermore, the standard text-to-text paradigm loses the structure information hidden in the proof tree (Qu et al., 2022). To this end, we propose a divide-and-conquer algorithm to encode the plain text for the proof tree, ensuring that the plain text is unique for each tree and ultimately retains the structure information.

We conduct experiments to evaluate our approach on several datasets widely used in previous studies with different settings. Experimental results show that our ProofInfer outperforms significantly more than the state-of-the-art (SOTA) models. In addition, ProofInfer shows strong generalization ability and performs well with data-limited, achieving comparable performance to the SOTA models with about **40%** of the training data. Our main contributions are summarized as follows:

- We present ProofInfer, a novel proof generation model via iterative hierarchical inference. ProofInfer employs the text-to-text paradigm to avoid the shortcoming of conventional autoregressive generation architecture on proof generation tasks. Unlike the previous stepwise methods, Proofinfer focuses on generating the whole proof tree rather than several proof paths.
- We propose a divide-and-conquer algorithm to encode the proof tree as a linear structure without losing structure information.
- Experimental results show that ProofInfer significantly outperforms the state-of-the-art models. ProofInfer performs well with data-limited, achieving comparable performance to the SOTA models with about **40%** of the training data.

2 Related Work

Existing methods for generating natural language proofs include single-shot and stepwise methods. Single-shot methods generate the entire proof tree at once by predicting the nodes and edges in the proof. PROVER (Saha et al., 2020) trains two binary classifiers to predict whether each node and each edge is a part of the proof, and it employs linear integer programming to ensure the proof is connected. Similarly, PROBR (Sun et al., 2021) employs a probabilistic graph to generate the proof via one-shot.

On the other hand, stepwise methods link the node to the proof one by one, which leverages the compositionality of proofs, making it easier for the model to learn the long proof. EVR (Liang et al., 2021) splits the question into sub-questions, using generated intermediate texts to guide proof generation step-by-step. ProofWriter (Tafjord et al., 2021) shares a similar idea but annotates the extra intermediate conclusions and a much more powerful T5-11B model (Raffel et al., 2019) for generation, which is hard to reproduce. In addition, ProofWriter employs the text-to-text paradigm to generate the proof step-by-step, which ignores the structure information in the proof tree. To retain the structure information, recent works

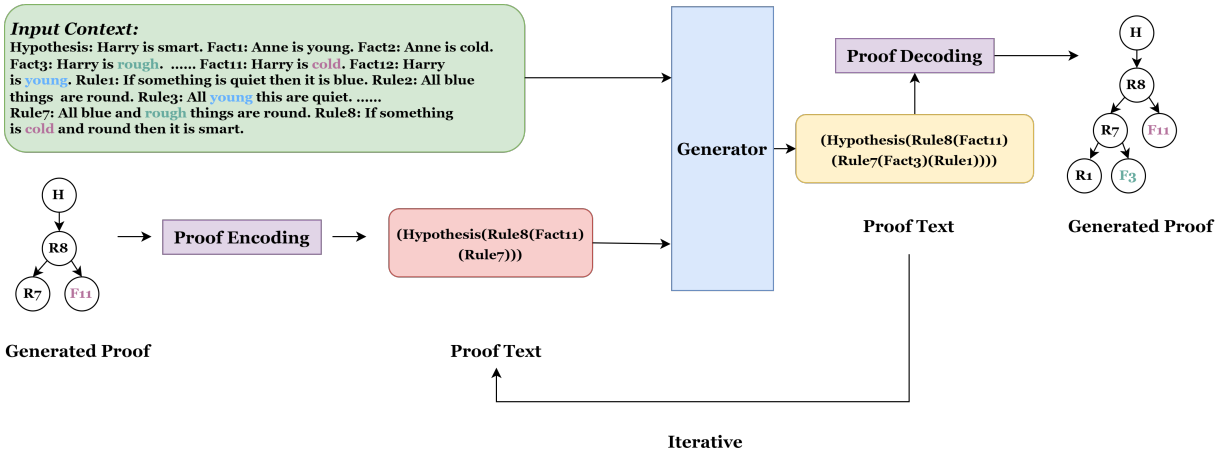

Figure 2: The model architecture of ProofInfer. At each step, ProofInfer encodes the generated proof tree into a proof text and inputs it into a generator to generate the proof text with the new layer. The iterative inference will end when all the nodes find their supporting facts.

(Yang et al., 2022; Qu et al., 2022; Sanyal et al., 2022) focus on training the module to guide the model to generate the proof path. However, prior works (Tafjord et al., 2021; Yang et al., 2022) have observed that the stepwise methods struggle to generate valid and relevant proof steps. One reason is that the stepwise methods focus on generating the proof path instead of the proof tree (Qu et al., 2022). But the conventional generation model with teacher-forcing cannot generate multiple nodes simultaneously, making it inevitable that the model can only focus on the proof path.

ProofInfer employs the text-to-text paradigm to achieve the hierarchical inference, which generates the proof layer-by-layer and simultaneously generates the nodes with the same depth. This way, ProofInfer focuses on the whole proof tree rather than the paths. In addition, to avoid structure information loss caused by the text-to-text paradigm (Tafjord et al., 2021), ProofInfer uses a novel divide-and-conquer algorithm to encode the proof tree as a plain text without information loss.

3 Methodology

In this section, we formalize the proof generation task and introduce our ProofInfer. In particular, we first describe the proof generation framework via iterative hierarchical inference. Following this, we describe the divide-and-conquer algorithm for proof encoding and proof decoding. The overall architecture of ProofInfer is shown in Figure 2.

3.1 Task Definition

We first formulate the proof generation task as follows. As shown in Figure 1, the input consists of a hypothesis H and a set of theories $T = \{F, R\}$ containing several textural rules $R = \{R_i\}$ and facts $F = \{F_i\}$. Both H and T are natural language sentences. H can be deduced from a subset of T through reasoning, which may require multiple steps.

The output is a proof tree P specifying how H is deduced from T. The node in P can be a fact or a rule. The directed edges in the proof indicate that the end nodes can support the start nodes during reasoning. As shown in Figure 1, $R1$ can be proved by $R3$ and $F12$. To successfully perform the task, the model must select relevant facts and rules from T and use them as the nodes to compose a valid proof tree to deduce the H.

3.2 The Proof Generation Framework via Iterative Hierarchical Inference

We present ProofInfer, our method for generating natural language proof via iterative hierarchical inference. At first, to ensure the fluency of reasoning, we add the hypothesis H to the node-set and take it as the initial node to produce the proof tree P. Then we concatenate the text representations of all facts, rules, and the hypothesis as the context X_{theory} as shown in Figure 2.

ProofInfer employs a top-down architecture and links all the supporting nodes that can be used to prove the current leaf nodes at each iterative step

as follow:

$$p(y_d) = p(y_d|y_{d-1}) \quad (1)$$

$$p(y_d|y_{d-1}) = p(x_1^d, ..., x_{num_d}^d|y_{d-1}) \quad (2)$$

where y_d is the proof tree has d layers, x^d is the node in the dth layer of the proof tree, and num_d is the number of nodes in the dth layer.

In a word, ProofInfer generates the proof tree layer-by-layer until each intermediate node in the tree finds the supporting facts.

Compared to existing stepwise methods, the iterative hierarchical inference is a novel paradigm for proof generation, which focuses on constructing the entire proof tree instead of proof paths. As for iterative hierarchical inference, all the nodes in the same layer are generated simultaneously. However, the autoregressive model with teacher-forcing that is used widely in the generation tasks can predict only one new node as follow:

$$p(x_i) = p(x_i|x_1, ..., x_{i-1}) \quad (3)$$

where x is the nodes that are linked to the proof.

To achieve iterative hierarchical inference, we take text-to-text paradigm to generate multiple nodes simultaneously. As shown in Figure 2, at each iterative step, ProofInfer infers the new layer of proof tree by three steps as follow:

$$X_{proof}^t = \text{ProofEncoding}(Y^t) \quad (4)$$

$$X^t = X_{theory}\,[\text{SEP}]\,X_{proof}^t \quad (5)$$

$$X_{proof}^{t+1} = \text{Generator}(X^t) \quad (6)$$

$$Y^{t+1} = \text{ProofDecoding}(X_{proof}^{t+1}) \quad (7)$$

1) ProofInfer encodes the generated proof tree Y^t as the text X_{proof}^t. 2) ProofInfer concatenates theories X_{theory} and X_{proof}^t as X^t, separated by special [SEP] token. ProofInfer input it to the generator and generates X_{proof}^{t+1} the text representation of the proof tree with the nodes in the next layer. 3) ProofInfer decodes X_{proof}^{t+1} as the tree structure Y^{t+1}.

If Y^t is the same as Y^{t+1}, which shows all the nodes find their supporting nodes, the generation processing will end.

3.3 Divide-and-Conquer Algorithm for Proof Encoding

Although ProofInfer employs the text-to-text paradigm to avoid the shortcoming of the autoregressive model with teach-forcing, the text-to-text ignores the structure information hidden in the proof tree (Qu et al., 2022), which is vital for proof generation, especially for iterative hierarchical inference.

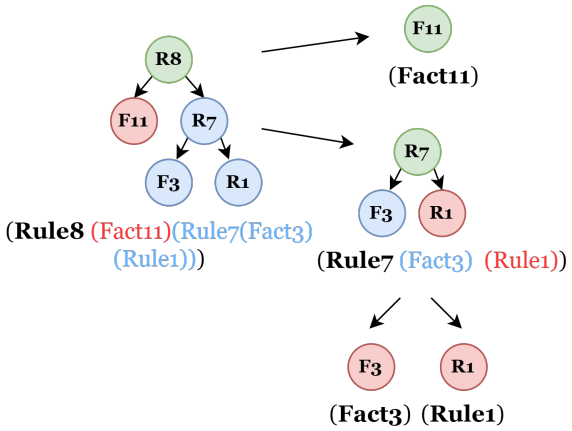

Figure 3: An example for our divide-and-conquer algorithm. The example shows how to encode the proof into plain text recursively. We make each subtree, and corresponding plain text has the same color as red or blue, and the green text indicates the root node for each tree.

Algorithm 1 Proof Encoding

input : The root node of tree $root$,
the child nodes of each node g
output : The proof text T

if *root is a leaf node* **then**
  $text \leftarrow$ the name of $root$
  $T \leftarrow$ "(" + $text$ + ")"
  return T
end
$T \leftarrow empty$
$g[root] \leftarrow \text{SORT}(g[root])$
for *child in* $g[root]$ **do**
  $childT \leftarrow ProofEncoding(child)$
  $T \leftarrow T$ + $childT$
end
$text \leftarrow$ the name of $root$
$T \leftarrow$ "(" + $text$ + ")"
return T

To solve this issue, we propose a divide-and-conquer algorithm, which can encode the proof tree to a unique plain text and decode it into the original tree structure. More importantly, all structure information in the proof tree is preserved in this text. Next, we introduce the encoding and decoding

algorithms, respectively. Figure 3 shows how to convert the proof to the plain text.

Algorithm1: Proof Encoding. Algorithm 1 is implemented by recursion. We use the divide-and-conquer method to encode the tree structure. For each subtree of the root node, we recursively encode it separately and then merge it with the root node. Each piece of text surrounded by paired parentheses is the textual representation of a subtree. In addition, for the nodes that have the same father, we sort them by lexicographical order. For example, the node representing *fact1* will be sorted before *fact2*, and the *fact2* will be sorted before *rule1*. The purpose is to ensure each proof tree has a unique text representation.

Algorithm2: Proof Decoding. Algorithm 2 is also a divide-and-conquer algorithm implemented by recursion. For each textual representation of a tree, we first exclude the root node's name. Then we decode each subtree recursively and link it to the root node.

Algorithm 2 Proof Decoding

input : The proof text T
output : The proof tree $Tree$ corresponding to T

if *there is no* "(" *or* ")" *in* T **then**
  Create a new node $Tree$ whose name is T
  return $Tree$
end
$text \leftarrow$ the first name in T
Create a new node $Tree$ whose name is $text$
$T \leftarrow T$ remove the $text$
$N \leftarrow$ the length of T
$i \leftarrow 0$
while $i < N$ **do**
  // $T[i]$ must be "("
  $j \leftarrow$ the index of the matched ")"
  $childT \leftarrow$ Proof Decoding($T[i+1:j-1]$)
  Addedge($Tree$, $childT$)
  $i \leftarrow j+1$
end
return $Tree$

3.4 Training and Inference

Training: To train the iterative model, we create an augmented set of training examples for each sample in the training data with one sequence of iteratively inferred proof trees in turn. In detail, we split the sample whose target proof tree's maximum depth is d into d samples whose target proof tree's depth is $1, .., d$. It ensures that all example's inferences are depth-1 and the model generates all nodes in the next layer simultaneously. We employ T5 model (Raffel et al., 2019) as the generator and train it by the negative log-likelihood for the target sequence $Y = \{y_t\}$:

$$L = \frac{1}{T}\sum_{t=1}^{T} logP(\tilde{y}_t = y_t) \quad (8)$$

where $\tilde{y}_t$ is the prediction token at the tth decoding step.

Furthermore, we train a linear classifier to predict the answer of the hypothesis, where we input the theories, hypothesis, and the proof tree text into it.

Inference: We input the theories and the initial proof tree in the inference stage with only a $hypothesis$ node. ProofInfer generates the proof tree by iteratively applying the model until no new layer is generated (the generated proof is the same as the input proof). The answer predictor employs the final proof tree to predict the answer of the hypothesis.

4 Experiments

To evaluate the model's ability to generate the proof, following (Qu et al., 2022) we conduct experiments on three datasets and four settings, including fully-supervised training, training using fewer samples, testing on out-of-distribution samples, and generalization to more complex proofs or language.

4.1 Datasets

We conduct the experiments on three datasets raised by [2] (Clark et al., 2021), and we use the same setting as previous works for fair comparison:

DU0-DU5: The dataset comprises five synthesized datasets created by translating hand-crafted rules and formal language to natural language. The five datasets are named DU0, DU1, DU2, DU3, and DU5, each containing 100k questions. It is divided by the highest depth of the proof tree, where DU stands for "Depth Upto" ($DU = 0, 1, 2, 3, 5$). Data in higher DU values also contain samples with lower depth. Each dataset is split 70/10/20 into train/dev/test. Specifically, the proofs in DU0 only have one supporting fact. **All related results are reported on DU5.**

Bird-Electricity: It consists of two test-only datasets of test-only datasets of 5k samples used to

[2] https://allenai.org/data/ruletaker

Model	Answer Accuracy							Proof Accuracy						
	0	1	2	3	4	5	all	0	1	2	3	4	5	all
PROVER (Saha et al., 2020)	100	99.0	98.8	99.1	98.8	99.3	98.4	99.3	93.2	84.8	80.5	72.5	65.1	87.1
PROBR (Sun et al., 2021)	100	99.9	99.9	**100**	100	100	99.9	98.4	94.3	86.1	82.0	76.1	72.2	88.8
EVR (Liang et al., 2021)	99.4	99.3	96.9	93.3	88.9	88.3	94.4	95.8	92.5	87.7	79.3	77.3	68.8	83.6
IBR (Qu et al., 2022)	100	99.2	99.2	98.9	99.3	99.6	99.4	99.5	95.6	93.0	90.7	86.5	81.7	93.5
ProofWriter(T5-11B + extra information) (Tafjord et al., 2021)	100	99.1	98.6	98.5	98.7	99.3	99.2	99.6	98.7	97.3	94.4	91.0	86.4	96.2
ProofWriter(T5-large + extra informaiton) (Tafjord et al., 2021)	99.0	98.8	98.3	98.6	98.0	97.7	98.7	99.0	95.0	91.0	89.0	86.3	85.4	94.4
ProofInfer(T5-large)(ours)	**100**	**100**	**100**	99.9	**100**	**100**	**99.9**	**99.6**	**99.3**	**98.7**	**96.8**	**92.8**	**90.7**	**97.2**

Table 1: Test results on **DU0-DU5** for different depths of the proof tree. Models are trained and tested on the **D5** subset. Our model employs T5-large and is trained **without any extra information**.

	Model	10k	30k	70k
Answer	PROVER (Saha et al., 2020)	87.1	97.8	99.3
	PROBR (Sun et al., 2021)	**99.9**	99.9	99.9
	EVR (Liang et al., 2021)	94.8	97.8	99.2
	IBR (Qu et al., 2022)	97.9	99.4	99.5
	ProofINfer	99.2	**99.9**	**99.9**
Proof	PROVER (Saha et al., 2020)	72.4	86.8	88.8
	PROBR (Sun et al., 2021)	72.4	86.8	88.8
	EVR (Liang et al., 2021)	80.2	80.9	83.6
	IBR (Qu et al., 2022)	75.7	89.8	93.5
	ProofInfer	**83.5**	**92.9**	**97.2**

Table 2: Performance comparison using fewer training samples. The models are tested on the full test split of DU5 after training on the subset of train-set on DU5.

evaluate the out-of-distribution performance of the models.

ParaRules: ParaRules consists of 40k questions against theories expressed in paraphrased natural language, obtained through crowdsourcing.

4.2 Metrics

Following previous works, we evaluate the performance of models via answer prediction of hypothesis answer accuracy and proof generation accuracy. As for proof generation accuracy, we evaluate the fraction of samples where the generated proof tree matches exactly with the gold proof. Since some samples may have multiple gold proofs, a generated proof will be considered correct if it matches exactly with any of the gold proofs.

4.3 Baselines

We compare our proposed model against several strong baselines on proof generation.

PROVER (Saha et al., 2020): a single-shot method that treats the proof as a graph and predicts all its nodes and edges at once.

PROBR (Sun et al., 2021): a single-shot method improves the PROBER via the probabilistic, which jointly considers the answer, nodes, and edges.

ProofWriter (Tafjord et al., 2021): a text-to-text method, which introduce the extra intermediate conclusions to reduce the difficulty of reasoning.

EVR (Liang et al., 2021): an iterative stepwise method that predicts the next proof item by generating textual sub-questions based on a logical operator.

IBR (Qu et al., 2022) an iterative stepwise method that trains several modules to predict the proof path at each step.

4.4 Implementation Details

We use the T5-large model (Raffel et al., 2019) loaded from transformers in huggingface library [3]. We use the AdamW (Loshchilov and Hutter, 2018) as the optimizer, and the learning rate for local fine-tuning is set to 2e-5. We stop the training if the validation BLEU-4 score stops improving for 5 epochs. We clip the gradient at length 10. The batch size is 32 and the beam search width is 5. All hyperparameters are tuned on the development set.

4.5 Results for Varying Depths

We first train and evaluate ProofInfer on the train and test splits for the DU5 dataset and compare the performances of varying depths. We report the results on Table 1.

ProofInfer achieves the best proof accuracy and answer accuracy among all baseline models on samples in every depth. ProofInfer is much better on samples with the deep proof tree, which is 6.3 and 9.0 higher than IBR (Qu et al., 2022) when the depth are 4 and 5. ProofInfer obtains significantly stronger performance on all metrics, benefiting from the hierarchical inference.

Compared to ProofWriter(T5-large + extra information) (Tafjord et al., 2021) that employs text-to-text paradigm and additional intermediate conclusions information, our ProofInfer does not

[3] huggingface.co/transformers

	Model	DU0	DU1	DU2	DU3	DU5
Answer	PROBER (Saha et al., 2020)	68.7	73.7	89.6	98.6	99.3
	PROBR (Sun et al., 2021)	56.9	97.7	**99.9**	99.9	99.9
	IBR (Qu et al., 2022)	53.5	73.1	89.6	98.6	99.4
	ProofInfer	**72.6**	89.4	**99.9**	**99.9**	**99.9**
Proof	PROBER (Saha et al., 2020)	44.4	63.8	72.6	79.1	87.1
	PROBR (Sun et al., 2021)	50.7	63.9	74.5	83.2	88.8
	IBR (Qu et al., 2022)	47.0	64.6	76.3	87.4	93.5
	ProofInfer	**51.1**	**66.1**	**78.9**	**91.7**	**97.2**

Table 3: Performance of generalization ability on proof generation between models when testing one the DU5, after trained on DU0, DU1, DU2, DU, and DU5.

	Models	0	1	2	3	4	all
Answer	PROBER (Saha et al., 2020)	99.7	98.6	98.2	96.5	88.0	98.4
	PROBR (Sun et al., 2021)	99.8	99.7	99.9	99.8	100	99.8
	IBR (Qu et al., 2022)	99.9	98.8	97.5	96.3	88.7	98.4
	ProofInfer	**99.8**	99.6	**100**	**99.9**	**100**	**99.9**
Proof	PROBER (Saha et al., 2020)	99.5	98.0	88.9	90.0	76.1	95.4
	PROBR (Sun et al., 2021)	99.5	98.0	88.9	90.1	82.4	95.6
	IBR (Qu et al., 2022)	99.8	98.8	91.1	89.0	75.3	95.9
	ProofInfer	**100**	**100**	**100**	**99.5**	**91.9**	**99.6**

Table 4: Performance of proof generation on ParaRules test set whose samples are expressed in more human-like natural language.

use any extra information and still achieves much better performance than ProofWriter. In addition, our ProofInfer with T5-large model (770 million parameters) achieves better performance than ProofWriter with T5-11B (11 billion parameters), which outperforms ProofWriter with **14x fewer** parameters. Using the same T5-large model, our ProofInfer performs significantly better than ProofWriter with extra information. The results show that our proof encoding algorithm improves the text-to-text paradigm's performance and hierarchical inference's effectiveness.

4.6 Using Fewer Training Samples

We report the performance of ProofInfer on proof accuracy when using fewer training data, ranging from 10k to 30k to all the samples (70k) in DU5. The comparison between ProofInfer, PROVER, PROBR, and IBR is shown in Table 2 Our model significantly has the best proof generation than the other baselines in all settings. Iterative hierarchical inference focus on the whole proof tree instead of a path, which does not need to switch the generated path and thus fewer training samples.

Our ProofInfer with almost 40% training samples (30k) is only a bit lower (0.6%) than state-of-the-art method IBR with all training samples. Compared to PROVER and PROBR, ProofInfer with 30k samples has much higher accuracy on proof accuracy. In addition, the performance of ProofInfer with 10k samples is still competitive with other methods. The results show our ProofInfer can perform well with data-limited.

4.7 Generalize to Higher Depths

To test the generalization ability of ProofInfer, we train the model on the training splits of DU0, DU1, DU2, and DU3, then we test them on the DU5 with deeper proof paths respectively and report the results in Table 3. We notice that the performance of all models drops significantly, especially when the proof depth of the training set is lower because it is hard for the model to learn complex reasoning-based simple training data. However, ProofInfer also performs well, especially on DU3. ProofInfer obtains 91.7% for proof accuracy training on the DU3 dataset, which is 4.3 % higher than SOTA model IBR and is even higher than PBOBER and PROBR training on DU5. The results prove that our ProofInfer, which focuses on generating the whole proof, can obtain a better generalization capability than stepwise methods.

4.8 Generalize to Complex Language

In this section, we evaluate the ability of the samples expressed in more human-like natural language. Following (Clark et al., 2021), we train models on the train-set of DU3 and ParaRules and test on the ParaRules test-set. ParaRules is a dataset that is close to real-world applications. Table 4 shows that our ProofInfer improves the performance significantly. The results indicate that ProofInfer has good applicability when reasoning on more complicated and natural texts. Our ProofInfer shows strong competitiveness for real-world applications.

Moreover, our ProofInfer performs well in the sample with the deep proof tree compared to other methods. The performance of ProofInfer is 10.5% and 16.6% higher than IBR in depth-3 and depth-4, respectively. Our ProofInfer obtains **99.5%** at depth-3 samples and **99.6%** for all samples on ParaRules, which is a near-perfect performance.

4.9 Out-of-Domain Evaluation

Following previous work (Saha et al., 2020) we test the out-of-domain performance of ProofInfer on the Birds-Electricity dataset, and we show the results in Table 5. We train the model on DU5 and test it on six datasets, two from the birds domain(B1,B2) and the other four from the

	Model	B1	B2	E1	E2	E3	E4	All
Answer	PROBER	95.0	95.0	100.0	100.0	89.7	84.8	86.5
	PROBR	100.0	100.0	100.0	100.0	98.2	95.6	96.3
	EVR	75.0	72.5	100.0	75.0	98.1	93.8	93.6
	IBR	100.0	97.5	100.0	100.0	89.2	84.1	86.0
	ProofInfer	**100.0**	**100.0**	**100.0**	**100.0**	**98.1**	**95.8**	**96.5**
Proof	PROBER	92.5	95.0	95.1	91.7	72.3	80.6	80.7
	PROBR	100.0	100.0	**97.5**	93.3	79.3	77.7	79.3
	EVR	28.6	35.7	100.0	50.0	70.8	81.6	77.2
	IBR	100.0	100.0	95.6	94.4	80.2	82.4	83.2
	ProofInfer	**100.0**	**100.0**	96.4	**94.8**	**81.6**	**83.1**	**84.0**

Table 5: Out-of-domain performance comparison on Birds-Electricity dataset after training on DU5.

Model	Proof Accuracy
ProofInfer	97.2
w/o hypothesis node	95.9
w/o proof encoding	93.6
w/o iterative hierachical inference	92.4
w/o iterative inference	91.7
w/o proof encoding + iterative hierachical inference	89.4
w/o proof encoding + iterative inference	87.2

Table 6: Results of ablation studies on DU5 dataset.

electricity domain (E1, E2, E3, E4). We show the reults in Table 5. Overall, our ProofInfer achieves a 10.5% promotion in answer accuracy while a 0.8% upgrade in proof accuracy compared to IBR. Our method does not significantly improve performance for proof generation on out-of-domain datasets. We think the reason is that our ProofInfer focuses on solving the shortcomings of existing stepwise methods for constructing tree structures. The main challenge for out-of-domain datasets is the transfer learning ability of the model.

4.10 Ablation Study

To explore the effects of different components in our model, we consider the following ablations:

ProofInfer *w/o* proof encoding: removing our divide-and-conquer algorithm but use Polish notation following (Tafjord et al., 2021).

ProofInfer *w/o* iterative hierarchical inference: removing iterative hierarchical inference and predicting the node one-by-one, the model predicts the node by the level traversal order of the proof tree.

ProofInfer *w/o* hypothesis node: removing the hypothesis node in the proof tree and inputting the empty tree to the model initially.

ProofInfer *w/o* proof encoding + iterative hierarchical inference: removing divide-and-conquer proof encoding algorithm and iterative hierarchical inference.

ProofInfer *w/o* iterative inference: removing the iterative inference and predicting the whole proof tree at once.

ProofInfer *w/o* proof encoding + iterative inference: removing the divide-and-conquer proof encoding algorithm and predicting the whole proof tree simultaneously.

At first, there is a huge gap between ProofInfer and ProofInfer *w/o* iterative hierarchical inference. The result shows that our iterative hierarchical inference that generates the proof layer-by-layer is much better than stepwise methods. On the other hand, ProofInfer *w/o* proof encoding drops significantly compared to ProofInfer. It demonstrates that our proof encoding algorithm retains the structure information in the proof tree, which can considerably improve performance.

Secondly, ProofInfer *w/o* iterative inference is much lower than ProofInfer, demonstrating that iterative inference reduces the search space and makes it easier to learn the long proofs. However, we can find that there is not much difference (only 0.7%) between ProofInfer *w/o* iterative inference and ProofInfer *w/o* iterative hierarchical inference. The result shows the existing stepwise method that links the nodes one by one degrades the performance of the iterative inference. Furthermore, ProofInfer *w/o* proof encoding + iterative inference is 2.2% lower than ProofInfer *w/o* proof encoding + iterative hierachical inference. We can see that the methods that generate the proof at once rely more on structural information than iterative methods.

ProofInfer *w/o* hypothesis node is 1.3% lower than ProofInfer. The result shows the effective for the hypothesis node, which improves the integrity of the reasoning process.

5 Conclusion

This paper presents ProofInfer, a proof generation model via iterative hierarchical inference. We argue that existing stepwise methods that add the node to the proof one-by-one focus on generating several proof paths instead of a whole proof tree. ProofInfer employs iterative hierarchical inference, which simultaneously generates all nodes at the same layer by the text-to-text paradigm. We propose a divide-and-conquer proof encoding algorithm to retain the structure information in the proof tree. Our work improves the proof generation task significantly and provides a new method to handle the structure samples.

Limitations

Although ProofInfer achieves the new SOTA performance on all datasets with different settings, the performance for out-of-domain evaluation is not good enough, which is only 0.8% better than IBR model. How to improve the out-of-domain ability is a future work for us.

Acknowledgments

The authors wish to thank the anonymous reviewers for their helpful comments. This work was partially funded by National Natural Science Foundation of China (No. 61906176, 62206057, 62076069, 61976056), Program of Shanghai Academic Research Leader (No. 22XD1401100) and Beijing Academy of Artificial Intelligence (BAAI).

References

Peter Clark, Oyvind Tafjord, and Kyle Richardson. 2021. Transformers as soft reasoners over language. In *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, pages 3882–3890.

Sebastian Goodman, Nan Ding, and Radu Soricut. 2020. Teaforn: Teacher-forcing with n-grams. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 8704–8717.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880.

Zhengzhong Liang, Steven Bethard, and Mihai Surdeanu. 2021. Explainable multi-hop verbal reasoning through internal monologue. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1225–1250.

Ilya Loshchilov and Frank Hutter. 2018. Decoupled weight decay regularization. In *International Conference on Learning Representations*.

John McCarthy et al. 1960. *Programs with common sense*. RLE and MIT computation center Cambridge, MA, USA.

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.

Weizhen Qi, Yu Yan, Yeyun Gong, Dayiheng Liu, Nan Duan, Jiusheng Chen, Ruofei Zhang, and Ming Zhou. 2020. Prophetnet: Predicting future n-gram for sequence-to-sequencepre-training. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 2401–2410.

Hanhao Qu, Yu Cao, Jun Gao, Liang Ding, and Ruifeng Xu. 2022. Interpretable proof generation via iterative backward reasoning. *arXiv preprint arXiv:2205.10714*.

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

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*.

Alan JA Robinson and Andrei Voronkov. 2001. *Handbook of automated reasoning*, volume 1. Elsevier.

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.

Soumya Sanyal, Harman Singh, and Xiang Ren. 2022. Fairr: Faithful and robust deductive reasoning over natural language. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1075–1093.

Abigail See, Peter J Liu, and Christopher D Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1073–1083.

Changzhi Sun, Xinbo Zhang, Jiangjie Chen, Chun Gan, Yuanbin Wu, Jiaze Chen, Hao Zhou, and Lei Li. 2021. Probabilistic graph reasoning for natural proof generation. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 3140–3151, Online. Association for Computational Linguistics.

Oyvind Tafjord, Bhavana Dalvi, and Peter Clark. 2021. Proofwriter: Generating implications, proofs, and abductive statements over natural language. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 3621–3634.

Kaiyu Yang, Jia Deng, and Danqi Chen. 2022. Generating natural language proofs with verifier-guided search. *arXiv preprint arXiv:2205.12443*.