Are LLMs Rigorous Logical Reasoners? Empowering Natural Language Proof Generation by Stepwise Decoding with Contrastive Learning

Ying Su[1], **Mingwen Liu**[2], **Zhijiang Guo**[3,4]
South China University of Technology[1], Likelihood Lab[2], HKUST(GZ)[3], HKUST[4]
yingsu@scut.edu.cn, maxwell@alphafuture.cn, zhijiangguo@hkust-gz.edu.cn

Abstract

Logical reasoning is a pivotal component in the field of artificial intelligence. Proof planning, particularly in contexts requiring the validation of explanation accuracy, continues to present challenges. The recent advancement of large language models (LLMs) has led to significant progress in natural language proof planning, evolving from one-stage generators to more complex three-stage systems that include additional searchers or verifiers. While these assisted methods improve the quality of generated results, they also introduce increased search efforts and computational costs. Furthermore, the generative process itself remains underexplored. In this study, we propose a stepwise decoding approach augmented by contrastive learning to address two common errors encountered during the LLM generator's decoding process. We fine-tune the language model using both vanilla and enhanced hard negatives to mitigate these decoding errors. Empirical results demonstrate the effectiveness of our strategy. Additionally, our further analysis reveals that even larger LLMs still struggle to generate rigorous logical chains.

1 Introduction

Logical reasoning underpins the comprehension of human cognition and intelligence in machines (Goel et al., 2017). Large Language Models (LLMs) like GPT (Brown et al., 2020; Ouyang et al., 2022) and PaLM (Chowdhery et al., 2022; Anil et al., 2023) have pioneered using natural language as a platform for logical reasoning, complementing the traditional use of formal languages (Kazemi et al., 2022; Creswell et al., 2022). The incorporation of natural language broadens the scope of logical reasoning by allowing flexible querying and tapping into the extensive implicit knowledge encapsulated within LLMs.

In examining the capacity of LLMs for logical reasoning, it is crucial to consider not only the accuracy of their answers but also the correctness of their explanations (Xu et al., 2023). Utilizing prompting methods such as in-context learning (Brown et al., 2020) and chain-of-thought (Wei et al., 2022), LLMs have shown promising results across various deductive reasoning tasks in question-answering formats (Weston et al., 2015; Tafjord et al., 2021; Saparov and He, 2022; Han et al., 2022). These approaches decompose the final task goal by guiding the LLMs through intermediate reasoning steps in a carefully constructed context. However, providing correct explanations, which covers *completeness*, *redundancy*, *correctness* (Xu et al., 2023), emerges as a more daunting challenge. This is particularly evident in tasks that involve generating reasoning chains from premises leading to a conclusion, known as proof generation (Clark et al., 2020; Dalvi et al., 2021). Unfortunately, LLMs often fall short in creating concise and exact proof trees, commonly producing superfluous or imprecise intermediate steps.

Previous studies have utilized LLMs to generate proof trees, employing a range of techniques from holistic approaches (Qu et al., 2022) to incremental steps (Tafjord et al., 2021; Sanyal et al., 2022). Recent methods increasingly rely on post-processing to enhance the quality of generated results, introducing verification- and search-based systems (Hong et al., 2022; Yang et al., 2022). However, as the methods become more complex, there is a corresponding increase in search efforts and computational costs. Conversely, there has been insufficient focus on refining the generative process itself. Current models exhibit proficiency in selecting relevant premises but struggle to deduce intermediary conclusions, highlighting a deficiency in their understanding of semantic nuances during stepwise deductive reasoning.

Addressing this issue, we introduce a novel strategy dubbed ConDec (**Con**trastive learning based stepwise **Dec**oding), designed to enhance the generative aspect of LLMs for deductive reasoning

Proceedings of the 14th International Joint Conference on Natural Language Processing and the 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics, pages 1694–1708

Method	Stepwise Generation	Stepwise Correction	Direction	Search	Verifier	Human-authored Benchmark	Stage
EntailmentWriter (Dalvi et al., 2021)	✗	✗	→	✗	✗	✓	1
IRGR (Ribeiro et al., 2022)	✓	✗	→	✓	✗	✓	2
SCSearch (Bostrom et al., 2022)	✓	✗	→	✓	✗	✗	2
MetGen (Hong et al., 2022)	✓	✗	both	✓	✗	✓	2
ADGV (Sprague et al., 2022)	✓	✗	both	✓	✓	✓	3
NLProofs (Yang et al., 2022)	✓	✗	→	✓	✓	✓	3
ConDec	✓	✓	→	✗	✗	✓	1

Table 1: Comparison of methods over natural language proof generation. *Stepwise Correction* means that if stepwise generation is enhanced in training. *Stage* calculates if the method contains generation, verification, and search.

tasks. ConDec leverages carefully constructed hard negatives – outputs that are deceivingly similar in form yet differ semantically – to refine generation precision. These hard negatives can be simple sequence alterations or products of an intricate sampling and reasoning process, aided by an external reasoner and checker. Intuitively, the hard negatives are designed to solve decoding errors analyzed in (Dalvi et al., 2021): *repetition* and *invalid entailment*. Finetuning with these hard negatives notably advances the LLMs' proficiency in intermediate step and conclusion generation, culminating in overall improved proof accuracy. The main contributions of this study are threefold:

- We introduce ConDec, a stepwise decoding with contrastive learning strategy that enhances stepwise generative quality in proof generation tasks, and devise an automatic method for hard negative generation involving a reasoner and a checker;
- We conduct an extensive empirical analysis on the Entailment benchmark, demonstrating the effectiveness of the proposed method;
- We reveal that LLMs even equipped with chain-of-thought strategies still struggle to perform rigorous logical reasoning in natural language proof generation tasks.

2 Related Work

2.1 Logical Reasoning with Natural Language

Logical reasoning is an important ability to realize human-level cognition and intelligence in AI (Nunes, 2012). Early research of logical reasoning uses formal language to represent knowledge and conducts symbolic reasoning (Muggleton and De Raedt, 1994). Recent research uses pretrained language models for logical reasoning in the form of natural language to alleviate the representation challenge with formal language (Musen and Van der Lei, 1988).

Among the logical reasoning over natural language (Yang et al., 2023), deductive reasoning covers aspects including hypothesis classification, proof generation, proof generation with incomplete information, and implication enumeration. Several tasks have been proposed to evaluate these reasoning abilities. Specifically, hypothesis classification is conducted over RuleTaker with transformers (Clark et al., 2020). Proof generation providing rationals along with the predicted answer for emulating formal reasoning is further proposed to increase the explanability (Saha et al., 2020). ProofWriter (Tafjord et al., 2021) produces a deductive chain of reasoning over proof generation and implication enumeration with an iterative generating style. To enhance the chain of reasoning for multi-step premises, EntailmentWriter tests transformers' explainability in the form of entailment trees over EntailmentBank (Dalvi et al., 2021).

2.2 Proof Generation

Methods for finetuning language models for proof generation vary in the proof direction, inference with or without hypothesis, and whether search or verification is involved. One line of research is inference without a hypothesis available. FaiRR (Sanyal et al., 2022) breaks proof generation into three steps: rule selection, fact selection, and knowledge composition. MetGen (Hong et al., 2022) iteratively generates the entailment tree by conducting a single-step entailment with separate modules and a reasoning controller. SCSearch (Bostrom et al., 2022) decomposes the deductive reasoning task into separate steps coordinated by a search procedure, producing a tree of intermediate conclusions that faithfully reflects the system's reasoning process. ADGV (Sprague

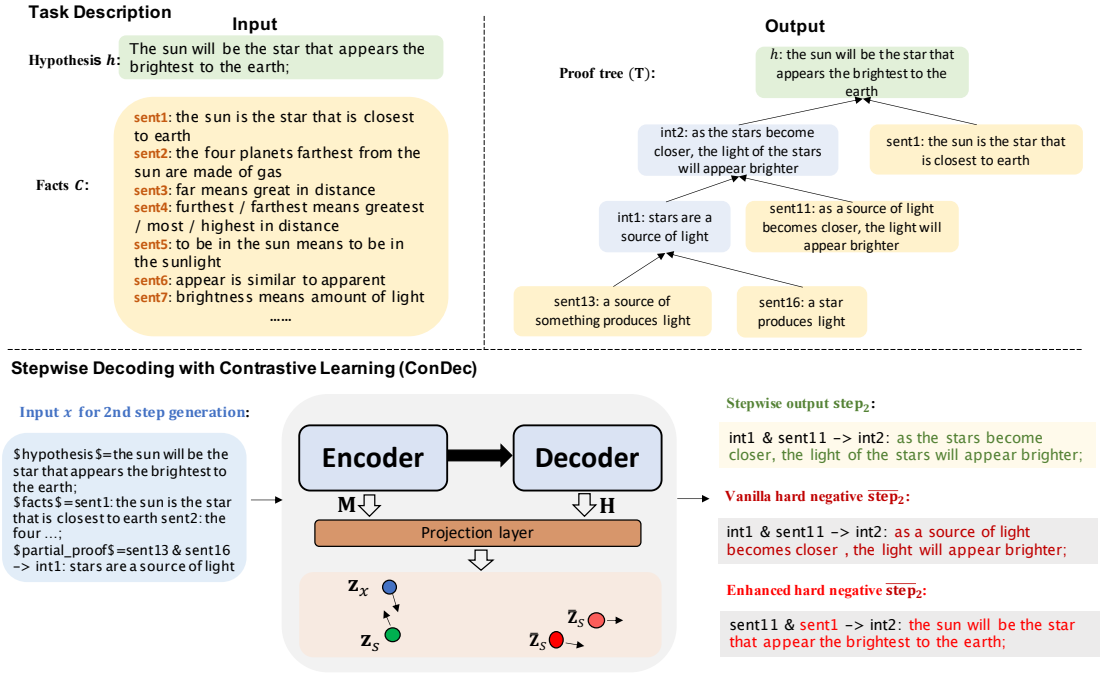

Figure 1: Architecture of the stepwise decoding with contrastive learning over hard negatives. The hard negatives are constructed by vanilla and enhanced strategies. Vanilla strategy means simple conclusion substitution. The enhanced strategy uses a reasoner and a checker to generate hard negatives.

et al., 2022) proposes to abductively infer a premise given another premise and a conclusion, as well as to search over two fingers interleaving deductive (forward-chaining) and abductive (backward-chaining) inferences.

Another line of work is with the hypothesis available. IBR (Qu et al., 2022) enhances the interpretability of reasoning procedures by predicting nodes and edges in the proof tree iteratively backward from the question, as well as increasing efficiency and accuracy by simplifying the intermediate process of reasoning. IRGR (Ribeiro et al., 2022) explains a given hypothesis by iteratively searching for suitable premises, constructing a single entailment step at a time. NLProofs (Yang et al., 2022) trains an independent verifier to check the validity of the proof steps to improve decoding accuracy. Our work follows this line and we focus on stepwise decoding correction on the generator itself. A comparison of our method with other approaches with LLMs is presented in Table 1.

3 Problem Formulation

Following the task definition in Yang et al. (2022), the proof generation task is to derive a proof tree T given a hypothesis h and a set of supporting facts $C = \{\text{sent}_1, \text{sent}_2, ..., \text{sent}_n\}$. The proof tree T is represented as a tuple $T = (h, \hat{C}, U, S)$. $\hat{C}$ is a subset of the facts $\{\text{sent}_i\} \in C$, denoting leaf nodes on the tree T. $U = \{\text{int}_1, \text{int}_2, ..., \text{int}_m\}$ denotes the intermediate nodes on the tree. The intermediate nodes are deduced during the reasoning process. Each intermediate node represents an intermediate conclusion. The intermediate nodes are internal tree nodes. On top of the tree, h is the root node as well as the final conclusion needs to be proved.

The structure of the tree is denoted by reasoning steps $S = \{\text{step}_1, \text{step}_2, ..., \text{step}_t\}$. Each internal tree node corresponds to a reasoning step $\text{step}_i \in S$ with $\text{int}_j \in U$ as the conclusion and its children as premises, i.e., sent_1 & $\text{int}_1 \rightarrow \text{int}_2$, representing that intermediate node int_2 is the conclusion of leaf node sent_1 and intermediate node int_1. The premise of a reasoning step can be from leaf nodes or intermediate nodes. Under the stepwise generation setting, intermediate nodes and proof steps up to the current step are added to given facts as input to generate new intermediate nodes for the next step.

4 Approach

4.1 Stepwise Decoding with Contrastive Learning

With stepwise training, subtrees are sampled on the original entire proof graph. The decoding goal can

be the intermediate node or the final hypothesis of the subtree, depending on the sampling strategy. As analyzed in (Dalvi et al., 2021), there are decoding errors leading to inaccurate proof step generation, finally leading to entire proof generation failure. An overview is presented in Figure 1.

To address the problem, we adopt a contrastive learning technique to improve the stepwise decoding quality. Learning with contrasting positive and negative pairs can improve the generalization ability of conditional text generations (Lee et al., 2020; An et al., 2022). Inspired by this, we construct negative decoding samples to improve the reasoning ability of the generator.

The goal of the generator is to output a stepwise reasoning step $\text{step}_j^{(i)}$ with tokens $(\tilde{s}_1^{(i)}, \tilde{s}_2^{(i)}, ..., \tilde{s}_L^{(i)})$ with length L conditioned on the input text sequence $x^{(i)} = (\tilde{x}_1^{(i)}, \tilde{x}_2^{(i)}, ...)$. The input text sequence is a concatenation of contexts from hypothesis h, facts C, and previous steps $\{\text{step}_1^{(i)}, ..., \text{step}_{j-1}^{(i)}\}$. i is the index of instances in a batch. The finetuning loss is to maximize the conditional log-likelihood $\log p_\theta(\text{step}_j|x)$ for a given N observations $\{(x^{(i)}, \text{step}_j^{(i)})_{i=1}^N\}$ as follows:

$$\mathcal{L}_{MLE}(\theta) = \sum_{i=1}^{N} \log p_\theta(\text{step}_j^{(i)}|x^{(i)}), \quad (1)$$

$$p_\theta(\tilde{s}_1^{(i)}, ..., \tilde{s}_L^{(i)}|x^{(i)}) = \prod_{l=1}^{L} p_\theta(\tilde{s}_l^{(i)}|\tilde{s}_{<l}^{(i)}, x^{(i)}), \quad (2)$$

$$\mathbf{h}_l^{(i)} = g(\tilde{s}_{l-1}^{(i)}, \mathbf{M}^{(i)}; \theta), \mathbf{M}^{(i)} = f(x^{(i)}; \theta), \quad (3)$$

where f, g denote the encoder and the decoder respectively. $\mathbf{M}^{(i)}$ is the concatenation of the hidden representations of the source tokens $x^{(i)}$. $\mathbf{H}^{(i)}$ is the concatenation of the hidden states $[\mathbf{h}_1^{(i)}, ..., \mathbf{h}_L^{(i)}]$ at the decoder output.

With a linear projection layer, the hidden states $\mathbf{M}^{(i)}$ and $\mathbf{H}^{(i)}$ of the encoder and decoder are mapped onto the latent embedding space:

$$\mathbf{z}_x^{(i)} = \text{AvgPool}(\text{ReLU}(\mathbf{W}_{proj}\mathbf{M}^{(i)} + \mathbf{b}_{proj})), \quad (4)$$

$$\mathbf{z}_s^{(i)} = \text{AvgPool}(\text{ReLU}(\mathbf{W}_{proj}\mathbf{H}^{(i)} + \mathbf{b}_{proj})). \quad (5)$$

The semantic similarity sim between them can be calculated by distance with a dot or cosine function. A contrastive loss maximizes the similarity between the pair of source sequence and target sequence while minimizing the similarity between the negative pairs as follows:

$$\mathcal{L}_{cont}(\theta) = \sum_{i=1}^{N} \log \frac{\exp(sim(\mathbf{z}_x^{(i)}, \mathbf{z}_s^{(i)})/\tau)}{\sum_{\mathbf{z}_s^{(k)} \in \mathcal{S}} \exp(sim(\mathbf{z}_x^{(i)}, \mathbf{z}_s^{(k)})/\tau)}, \quad (6)$$

where $\mathcal{S}$ is the set of negative samples in the same batch and τ is the temperature parameter.

4.2 Training with Hard Negative

Though stepwise generation improves over a single-shot generation (training over the entire proof tree and the decoding goal is a hypothesis), it still has typical errors (Dalvi et al., 2021): 1) **repetition**(the entailed conclusion simply repeats one of the input sentences); 2) **invalid entailment**(the entailed conclusion does not follow from input sentences). To improve the decoding quality, we design two types of hard negatives for these errors and finetune the model with these hard negatives.

The hard negative sequence $\overline{\text{step}}_j^{(i)}$ is constructed based on the gold proof step sequence $\text{step}_j^{(i)}$. With the hard negative sequences, the decoding loss becomes:

$$\mathcal{L}_{cont-hard}(\theta) = \sum_{i=1}^{N} \log \frac{\exp(sim(\mathbf{z}_x^{(i)}, \mathbf{z}_s^{(i)})/\tau)}{\sum_{\mathbf{z}_s^{(k)} \in \mathcal{S} \bigcup \{\bar{\mathbf{z}}_s^{(i)}\}} \exp(sim(\mathbf{z}_x^{(i)}, \mathbf{z}_s^{(i)})/\tau)}, \quad (7)$$

where $\bar{\mathbf{z}}_s^{(i)}$ is the projected hidden state of hard negatives. The final loss for finetuning is:

$$\mathcal{L} = \mathcal{L}_{MLE} + \alpha\mathcal{L}_{cont-hard}, \quad (8)$$

α is a weighted parameter.

4.2.1 Vanilla Hard Negative

Vanilla hard negatives are constructed based on substitution, which mimics the form of gold stepwise proofs, to address the repetition error. we randomly select one of the premises in a proof step step_j and replace the conclusion with its context. For example in Figure 1, the vanilla hard negative is constructed by replacing the gold standard conclusion with the context of input node *sent11*.

4.2.2 Enhanced Hard Negative

To increase the entailment quality over stepwise proofs, we also propose to construct enhanced hard

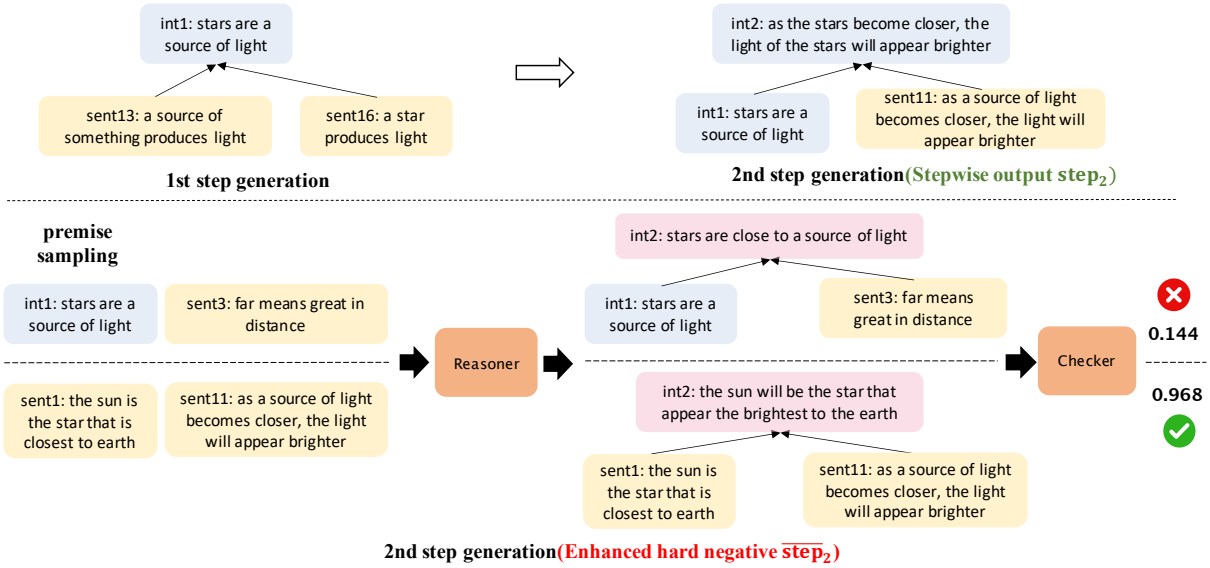

Figure 2: Enhanced hard negative construction is implemented by exploring the unseen combination of premises, inferencing with the reasoner, and filtering with a $score$ from the checker.

negatives by exploring unseen proof steps with a reasoner and a checker. The reasoner is first trained with proof steps $\text{step}_\text{j} \in S$ in natural language and then utilized to generate a conclusion given an unseen combination of premises in the supporting facts C.

Given proof step step_j, the premises and conclusion in natural language are denoted as set $\{p_1, p_2, ...\}$ and c. The premises are concatenated as input and the conclusion is output for training the reasoner.

$$\mathcal{L}_{MLE}(\phi) = \log p_\phi(c|p_1, p_2, ...), \quad (9)$$

After training, the reasoner can generate a conclusion given an unseen combination of premises. The premise combinations are first sampled from supporting fact set. The sampled premises $\{p_1^s, p_2^s, ...\}$ and generated conclusion c^s constructs an enhanced hard negative step. Details are presented in Figure 2. In the example, for premises *int1* and *sent11* in gold proof, randomly sample one premise to substitute one of them and use the reasoner to generate a new conclusion given the recombined premises.

To improve the quality of the hard negatives generated from the reasoner, we further adopt the checker Vera (Liu et al., 2023) to score the hard negatives and filter those with low scores. Vera is finetuned over T5-11B with commonsense datasets. The $score$ from the checker indicates the extent of the reasonableness of the deductive commonsense knowledge generated by the reasoner:

$$score = \text{sigmoid}([p_1^s, p_2^s, ..., c^s]; \gamma), \quad (10)$$

where the $score$ is calculated with a sigmoid function after hidden layers γ.

5 Experiment

5.1 Dataset

EntailmentBank (Dalvi et al., 2021): Entailment trees are made up of individual and multi-premise textual entailment steps. EntailmentBank contains 1,840 multi-step entailment trees for accompanying QA pairs. For the proof generation task, only the hypothesis H and context set C are used as inputs. Each proof tree T contains an average of 6.6 nodes and 2.7 entailment steps. Train/Validation/Test splits are 1,313/187/340 respectively. EntailmentBank consists of three tasks as follows:
(1) **Task 1 (no-distractor)**: C consists of exactly the leaf nodes of the ground truth proof tree;
(2) **Task 2 (distractor)**: C consists of 15-20 distractor sentences besides the leaf nodes on the ground truth proof tree;
(3) **Task 3 (full-corpus)**: C is a large corpus of 12K sentences derived from WorldTree V2 (Xie et al., 2020), requiring the model to retrieve relevant supporting facts from the corpus. For each hypothesis, 25 supporting facts are retrieved. Following Dalvi et al. (2021), we evaluate the zero-shot performance of the model from Task 2.

5.2 Evaluation

Following Dalvi et al. (2021), we use the official tools[1] to evaluate the generated entailment tree $T = (h, \mathcal{L}, \mathcal{E}, \mathcal{S})$ with the golden entailment tree

[1]https://github.com/allenai/entailment_bank

Reasoner	Enhanced Negative		Filtered Negative	
8,819	Task 1	Task 2	Task 1	Task 2
	47,386	104,665	16,371	21,948

Table 2: Distribution of samples from training data, constructed enhanced hard negatives and filtered hard negatives on Task 1 and Task 2. In filtering, threshold score is 0.9.

$T^* = (h, \mathcal{L}^*, \mathcal{E}^*, \mathcal{S}^*)$. These metrics evaluate the correctness along 4 dimensions:
(1) **Leaves** (F1, AllCorrect): F1 measures the precision of leaf nodes of T comparing to gold tree T^*. ALLCorrect=1 if F1=1, and ALLCorrect=1 if F1<1.
(2) **Steps** (F1, AllCorrect): F1 measures the precision of proof steps structurally correctness. Each step contains an internal node $u \in T^*$ (aligned to $v \in T$). The predicted step is correct if u and v are perfectly aligned. For each tree, ALLCorrect=1 if F1=1 otherwise 0.
(3) **Intermediates** (F1, AllCorrect): For the internal node $u \in T^*$ (aligned to $v \in T$), the intermediate conclusion is correct if the BLEURT (Sellam et al., 2020) score between u and v is greater than 0.28 (Dalvi et al., 2021). F1 and AllCorrect from all intermediate conclusions in T^* and T are calculated. ALLCorrect=1 if F1=1 otherwise 0.
(4) **Overall** (AllCorrect): The metric evaluates whether leaves, steps, and intermediates are all correct, AllCorret = 1 if and only if all the leaves, steps, and intermediates are all correct.

5.3 Implementation Details

The optimizer is set as Adam (Kingma and Ba, 2014) for all the training. The average running time is 18 hours for Task 1 and 24 hours for Task 2. Experiments are conducted on A800.

Generator. We use Flan-T5-Large as the generator. Flan-T5-Large is a finetuned version of T5-Large (Raffel et al., 2020) over a collection of FLAN instructions (Chung et al., 2022; Longpre et al., 2023). The generator is trained for 500 epochs on Task1 and 600 epochs on Task2. The learning rate for the first-stage generator is 1e-4 and 5e-5 for Task 1 and Task 2 respectively.

ConDec. The vanilla hard negatives are constructed by substituting the conclusion in the gold proof step with a randomly selected premise node context. For example, for *sent11* & *sent24* -> *int: neptune orbits the sun in the solar system*, the hard negatives substitute conclusion *neptune orbits the sun in the solar system* with the context of *sent11* or *sent24*. The temperature τ is set as 0.05 and α is 0.1. The learning rate of the stepwise decoding stage is the same as finetuning with original proof trees. The MLE loss and contrastive loss are alternatively trained for 10 epochs based on the finetuned generator with MLE loss only.

Enhanced Hard Negatives. We use Flan-T5-Large as an additional reasoner to generate unseen proof steps based on labeled proof trees as hard negatives. To train the reasoner, details of data collection are presented in Appendix A.1. The reasoner is trained for 30 epochs with a learning rate of 1e-4. We further apply Vera (Liu et al., 2023) to filter unreasonable generated proof steps. Details of stepwise proofs sampled for reasoner and enhanced new negatives filtered by the checker are presented in Table 2. To increase diversity, enhanced hard negatives and vanilla hard negatives are jointly sampled for training.

6 Result Analysis

6.1 Main Results

The main results are presented in Table 3. By analyzing the results, we can find that:
Stepwise correction matters. Stepwise generation methods (MetGen, NLProofs, ConDec) outperform single-shot training (EntailmentWriter), even for EntailmentWriter with a much larger parameter size (11B). When comparing ConDec with three-stage generation methods MetGen and NLProofs, ConDec achieves comparable or even better performance on Task 2 and Task 3. This shows that contrastive decoding with hard negatives can improve language models' reasoning ability, demonstrating our methods' effectiveness. While our research focuses on the generator, combining our method with theirs may still improve the final accuracy and it is worth exploring.
Enhanced hard negatives facilitate reasoning. With enhanced hard negatives, we can find that the ability of proof planning over three tasks is all improved. Unlike vanilla negative construction, the enhanced hard negatives contain harder or more accurate conclusions given premises. Detailed evaluation of enhanced hard negatives is in Appendix A.1. It further improves the training coverage over reasoning steps, thus leading to better performance

Task	Method	Leaves		Steps		Intermediates		Overall
		F1	AllCorrect	F1	AllCorrect	F1	AllCorrect	AllCorrect
Task 1 (no-distractor)	EntailmentWriter	98.7	86.2	50.5	37.7	67.6	36.2	33.5
	EntailmentWriter (11B)	99.0	89.4	51.5	38.2	71.2	38.5	35.3
	MetGen*	**100.0**	**100.0**	**57.7**	41.9	70.8	39.2	36.5
	NLProofs*	97.8	90.1	55.6	42.3	72.4	40.6	**38.9**
	ConDec	99.9	98.2	55.7	42.1	72.3	38.9	36.2
	ConDec★	99.9	98.2	57.3	**43.2**	**72.9**	**41.5**	37.9
Task 2 (distractor)	EntailmentWriter	84.3	35.6	35.5	22.9	61.8	28.5	20.9
	EntailmentWriter (11B)	89.1	48.8	41.4	27.7	66.2	31.5	25.6
	MetGen*	82.7	46.1	41.3	29.6	61.4	32.4	27.7
	NLProofs*	90.3	58.8	47.2	34.4	70.2	37.8	33.3
	ConDec	91.0	59.1	50.2	36.5	70.3	**38.2**	34.1
	ConDec★	**91.1**	**60.6**	**50.7**	**37.4**	**70.7**	**38.2**	**34.7**
Task 3 (full-corpus)	EntailmentWriter	35.7	2.9	6.1	2.4	33.4	7.7	2.4
	EntailmentWriter (11B)	39.9	3.8	7.4	2.9	35.9	7.1	2.9
	MetGen*	34.8	8.7	9.8	**8.6**	36.6	**20.4**	**8.6**
	NLProofs*	43.2	8.2	11.2	6.9	42.9	17.3	6.9
	ConDec	43.3	8.2	11.1	6.5	**43.4**	18.0	6.5
	ConDec★	**44.7**	**9.4**	**11.7**	7.1	42.3	17.7	7.1

Table 3: Main results on EntailmentBank with finetuning method. Methods with * are **three-stage**. ConDec denotes finetuning with vanilla hard negatives and ConDec★ denotes finetuning with combination of vanilla and enhanced hard negatives. Best results are **boldface** and second-best results are underlined.

over proof planning.

ConDec is more efficient with distractors. Specifically, contrastive learning with hard negatives achieves obvious performance gain over Task 2 and Task 3 while deteriorating the performance on Task 1. For Task 2 and Task 3, there are distractor premises or full-corpus, which increases the challenges for rigorous reasoning. For Task 1, there is no distractor. Adding hard negatives deviates the generator from generating correct steps instead of intermediate nodes.

Predicting intermediate node is still challenging. ConDec achieves the best performance mostly over leave or step accuracy. The contrastive loss helps the generator discriminate between premises and finds semantically correlated premises to deduce a conclusion. However, deductive reasoning is still challenging as the improvement over intermediates is not as obvious as that on leaves or steps.

6.2 Computational Cost Analysis

ConDec's design can reduce the inference cost with a one-stage process during inference as shown in Table 4. We provide a concrete comparison using the validation set for Task 2, with the same model size (Flan-T5-Large), on an A800 GPU. It clearly shows that ConDec can significantly improve time efficiency compared to two and three-stage methods (NLProofs).

Method	Stage	Time(s)	Improvement
ConDec	generator	294	-
NLProofs	generator+search	430	31.6%
NLProofs	generator+search+verify	544	46.0%

Table 4: Computation time comparison(percentage reduction). Time denotes the inference time. NLProofs requires additional search and verification for prediction.

6.3 Ablation Study

Results of the ablation study on Task 2 test split are presented in Table 5.

Backbone model. The results indicate that LLaMA-3.2-1B fails to adequately differentiate between input premises and output reasoning proof steps. As a decoder model focused on next-word prediction, LLaMA simply predicts proof steps as the next sentence following the premises. With stepwise training, the presence of intermediate or hypothesis nodes can confuse the model regarding when to stop generating output, as a subtree with an intermediate node is a subsequence of the entire tree that includes the hypothesis node. This is why LLaMA tends to produce shorter outputs when trained in a stepwise manner.

Moreover, the results for LLaMA are significantly lower than those for Flan-T5-Large, despite the latter's smaller size. Natural proof generation is fundamentally a sequence-to-sequence task. Distinguishing between input and output is crucial, as the

Method	Leaves		Steps		Intermediates		Overall
	F1	AllCorrect	F1	AllCorrect	F1	AllCorrect	AllCorrect
LLaMA-3.2-1B (1.2B, w/o stepwise)	19.5	5.3	6.4	2.9	13.9	5.3	2.7
LLaMA-3.2-1B (1.2B, stepwise)	15.6	3.5	4.1	3.2	9.9	5.0	3.2
Flan-T5-Large (0.8B, stepwise)	90.7	58.8	49.2	36.2	69.6	36.8	33.5
+ contrastive loss	90.9	60.3	49.5	35.9	69.4	37.1	32.4
+ contrastive loss, vanilla hard	91.0	59.1	50.2	36.5	70.3	**38.2**	34.1
+ contrastive loss, vanilla and enhanced hard	**91.1**	**60.6**	**50.7**	**37.4**	**70.7**	**38.2**	**34.7**
Flan-T5-XL (3B, stepwise)	90.9	57.1	50.2	36.5	68.8	35.9	33.8

Table 5: Ablation study results on EntailmentBank test set in Task 2.

Method	Leaves		Steps		Intermediates		Overall
	F1	AllCorrect	F1	AllCorrect	F1	AllCorrect	AllCorrect
ConDec$^\bigstar$	**92.4**	**63.1**	**55.3**	**45.5**	**72.8**	**43.9**	**41.2**
GPT3 (5-shot)	64.2	15.3	17.6	12.3	53.6	22.3	12.3
GPT3.5-turbo (5-shot)	61.9	9.0	16.9	4.3	51.9	15.1	3.74
GPT4 (5-shot)	78.1	32.6	30.2	22.5	63.9	30.8	21.9
GPT4 (SI)	79.1	30.0	24.3	14.0	65.4	32.1	13.9
GPT4 (CoT)	79.0	33.7	30.9	22.5	64.7	31.6	22.5
GPT4o-mini (5-shot)	63.1	18.2	20.0	13.9	54.5	24.1	13.9
o1-mini (5-shot)	72.5	27.3	28.6	13.9	50.9	18.7	11.8
o1-preview (5-shot)	84.0	43.3	38.7	28.0	67.8	33.7	21.9

Table 6: Results on EntailmentBank validation set in Task 2 with finetuning and prompting methods. GPT3.5 is *GPT3.5-turbo-0613*. GPT4 is *GPT4-0613*. GPT4o-mini is *gpt-4o-mini-2024-07-18*. o1-mini is *o1-mini-2024-09-12*. o1-preview is *o1-preview-2024-09-12*.

encoder-decoder architecture of Flan-T5 processes different texts. A case study comparing LLaMA and Flan-T5 is presented in the Appendix A.6.

Model size. For the model size, we also try Flan-T5-XL (3B). Stepwise training over Flan-T5-Large provides a strong baseline. Though much larger in model size, Flan-T5-XL achieves similar performance over the metrics. It shows that proof planning requires a higher level of understanding of deductive reasoning over multi-hops.

Negative sample. ConDec without hard negatives improves the leave accuracy while decreasing the step accuracy, leading to a slight drop in the overall performance. Adding vanilla or enhanced hard negatives can generally improve over all the metrics.

7 Analysis of Closed-Source LLMs

In-context learning (ICL; Brown et al. 2020) and CoT (Wei et al., 2022) have been widely used in various reasoning tasks (Patel et al., 2021; Cobbe et al., 2021; Fu et al., 2022; Xiong et al., 2023). We apply 5-shot prompting and CoT to evaluate how LLMs perform on the proof generation task. The vanilla prompt method simply adopts a k-shot in-context learning strategy. The CoT decomposes the reasoning process into step-by-step generation. Based on it, Select-inference(SI)(Creswell et al., 2022) is a two-stage COT that decomposes each reasoning step into a premise selection stage and a conclusion inference state. Details of the CoT is illustrated in Appendix A.2.2.

As shown in Table 6. Comparing different LLMs, GPT4 generally outperforms GPT3 and GPT3.5-turbo on all the metrics. One detailed case study of GPT3.5-turbo and GPT4 with k-shot prompting results is in Appendix A.2.1. It shows that the GPT3.5-turbo tends to generate more irrelevant and inaccurate steps with simple imitation of premises during inferencing new conclusions. In contrast, GPT4 conducts deductive reasoning and generates correct conclusions based on premises. When accompanied by CoT, GPT4 achieves better performance than the vanilla prompt. While with SI, GPT4 makes more mistakes in the premise selection stage.

Our ConDec$^\bigstar$ outperforms all closed-source LLMs in all metrics. Although recent LLMs such as o1-mini and o1-preview show improvements in several metrics compared to their predecessors (GPT-3.5-turbo and GPT-4), a substantial performance gap remains when compared to finetuning-based methods. Through stepwise training with

curated datasets, these language models can better capture the correlations between premises.

8 Conclusion

Logical reasoning is both challenging and fundamental in artificial intelligence. Proof generation serves as a measure of the explanatory capabilities of large language models in the context of logical reasoning. To enhance stepwise deductive reasoning, we propose a decoding strategy augmented by contrastive learning. The carefully designed hard negatives address the typical errors encountered during the decoding process. The experimental results across standard benchmarks demonstrate the effectiveness of our method. Additionally, our analysis of larger LLMs reveals that they continue to struggle with proof planning tasks.

Limitations

From the analysis of the paper, natural proof planning ability is still a challenging topic in evaluating LLMs' deductive reasoning ability. The current curated human-annotated dataset in our experiments is of limited size to improve LLMs' deductive reasoning ability. Knowledge from related corpora such as cause and effect, logic reasoning can be further applied to improve the proof generation ability with pre-training or transfer learning.

References

Chenxin An, Jiangtao Feng, Kai Lv, Lingpeng Kong, Xipeng Qiu, and Xuanjing Huang. 2022. Cont: Contrastive neural text generation. *Advances in Neural Information Processing Systems*, 35:2197–2210.

Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, et al. 2023. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*.

Kaj Bostrom, Zayne Sprague, Swarat Chaudhuri, and Greg Durrett. 2022. Natural language deduction through search over statement compositions. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 4871–4883.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2022. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. 2022. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*.

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 2020*, pages 3882–3890. ijcai.org.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Antonia Creswell, Murray Shanahan, and Irina Higgins. 2022. Selection-inference: Exploiting large language models for interpretable logical reasoning. *arXiv preprint arXiv:2205.09712*.

Bhavana Dalvi, Peter Jansen, Oyvind Tafjord, Zhengnan Xie, Hannah Smith, Leighanna Pipatanangkura, and Peter Clark. 2021. Explaining answers with entailment trees. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 7358–7370.

Yao Fu, Hao Peng, Ashish Sabharwal, Peter Clark, and Tushar Khot. 2022. Complexity-based prompting for multi-step reasoning. *arXiv preprint arXiv:2210.00720*.

Vinod Goel, Gorka Navarrete, Ira A Noveck, and Jérôme Prado. 2017. The reasoning brain: the interplay between cognitive neuroscience and theories of reasoning.

Simeng Han, Hailey Schoelkopf, Yilun Zhao, Zhenting Qi, Martin Riddell, Luke Benson, Lucy Sun, Ekaterina Zubova, Yujie Qiao, Matthew Burtell, et al. 2022. Folio: Natural language reasoning with first-order logic. *arXiv preprint arXiv:2209.00840*.

Ruixin Hong, Hongming Zhang, Xintong Yu, and Changshui Zhang. 2022. METGEN: A module-based entailment tree generation framework for answer explanation. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pages 1887–1905, Seattle, United States. Association for Computational Linguistics.

Seyed Mehran Kazemi, Najoung Kim, Deepti Bhatia, Xin Xu, and Deepak Ramachandran. 2022. Lambada: Backward chaining for automated reasoning in natural language. *arXiv preprint arXiv:2212.13894*.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Seanie Lee, Dong Bok Lee, and Sung Ju Hwang. 2020. Contrastive learning with adversarial perturbations for conditional text generation. In *International Conference on Learning Representations*.

Jiacheng Liu, Wenya Wang, Dianzhuo Wang, Noah A Smith, Yejin Choi, and Hannaneh Hajishirzi. 2023. Vera: A general-purpose plausibility estimation model for commonsense statements. *arXiv preprint arXiv:2305.03695*.

Tengxiao Liu, Qipeng Guo, Xiangkun Hu, Yue Zhang, Xipeng Qiu, and Zheng Zhang. 2022. Rlet: A reinforcement learning based approach for explainable qa with entailment trees. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 7177–7189.

Shayne Longpre, Le Hou, Tu Vu, Albert Webson, Hyung Won Chung, Yi Tay, Denny Zhou, Quoc V Le, Barret Zoph, Jason Wei, et al. 2023. The flan collection: Designing data and methods for effective instruction tuning. *arXiv preprint arXiv:2301.13688*.

Stephen Muggleton and Luc De Raedt. 1994. Inductive logic programming: Theory and methods. *The Journal of Logic Programming*, 19:629–679.

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.

Terezinha Nunes. 2012. Logical reasoning and learning. pages 2066–2069.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are nlp models really able to solve simple math word problems? *arXiv preprint arXiv:2103.07191*.

Hanhao Qu, Yu Cao, Jun Gao, Liang Ding, and Ruifeng Xu. 2022. Interpretable proof generation via iterative backward reasoning. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2968–2981, Seattle, United States. Association for Computational Linguistics.

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. *The Journal of Machine Learning Research*, 21(1):5485–5551.

Danilo Neves Ribeiro, Shen Wang, Xiaofei Ma, Rui Dong, Xiaokai Wei, Henghui Zhu, Xinchi Chen, Peng Xu, Zhiheng Huang, Andrew Arnold, et al. 2022. Entailment tree explanations via iterative retrieval-generation reasoner. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pages 465–475.

Stephen Robertson, Hugo Zaragoza, et al. 2009. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389.

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, Dublin, Ireland. Association for Computational Linguistics.

Abulhair Saparov and He He. 2022. Language models are greedy reasoners: A systematic formal analysis of chain-of-thought. *arXiv preprint arXiv:2210.01240*.

Thibault Sellam, Dipanjan Das, and Ankur Parikh. 2020. Bleurt: Learning robust metrics for text generation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7881–7892.

Zayne Sprague, Kaj Bostrom, Swarat Chaudhuri, and Greg Durrett. 2022. Natural language deduction with incomplete information. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 8230–8258.

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.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*.

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*.

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 Twelfth Language Resources and Evaluation Conference*, pages 5456–5473.

Jiong Xiong, Zixuan Li, Chuanyang Zheng, Zhijiang Guo, Yichun Yin, Enze Xie, Zhicheng Yang, Qingxing Cao, Haiming Wang, Xiongwei Han, Jing Tang, Chengming Li, and Xiaodan Liang. 2023. Dq-lore: Dual queries with low rank approximation re-ranking for in-context learning. *ArXiv*, abs/2310.02954.

Fangzhi Xu, Qika Lin, Jiawei Han, Tianzhe Zhao, Jun Liu, and Erik Cambria. 2023. Are large language models really good logical reasoners? a comprehensive evaluation from deductive, inductive and abductive views. *arXiv preprint arXiv:2306.09841*.

Kaiyu Yang, Jia Deng, and Danqi Chen. 2022. Generating natural language proofs with verifier-guided search. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 89–105.

Zonglin Yang, Xinya Du, Rui Mao, Jinjie Ni, and Erik Cambria. 2023. Logical reasoning over natural language as knowledge representation: A survey. *arXiv preprint arXiv:2303.12023*.

Threshold score	Accuracy(%)
0.7	36
0.8	50
0.9	60

Table 7: Human check accuracy on enhanced hard negatives from reasoner and checker with different threshold scores.

A Appendix

A.1 Details of Enhanced Hard Negatives

Training data of the reasoner is basically all the proof steps in the training set. Each proof step is converted into the form of natural language: 1) premises are concatenated with conjunction "and"; 2) premises and conclusion are linked with "Because" and "Therefore". The reasoner is trained with 8,819 proof steps for both Task 1 and Task 2.

For each proof step $\mathrm{step_j}$, substitute one premise from the the supporting fact set C except the ones in current proof step. Recombined proof premises $\{p_1^s, p_2^s, ...\}$ are converted into natural language form and used as input to the reasoner. The reasoner generates a new conclusion c^s based on the premises, forming a hard negative $\overline{\mathrm{step}}_j^{(i)}$.

The constructed hard negative $\overline{\mathrm{step}}_j^{(i)}$ is then filtered by the checker with a threshold score. To determine the threshold score, three PhD students from the CSE department are assigned with the task to check the quality of 100 randomly selected constructed hard negatives. Each hard negative is first judged by Vera. The $score$ from Vera indicates the reasonableness of the hard negative. By filtering the $score$ with a threshold, we choose the constructed negatives with high quality for further training. The result of human checking over the filtered negatives is shown in the table 7. We find that with a threshold score of 0.9, 60% of the filtered hard negatives are reasonable.

A.2 Prompting Methods

A.2.1 Case study for prompting

Result case from GPT3.5-turbo and GPT4 on Task2 dev split with vanilla prompts is shown in Table 8.

A.2.2 Template for prompting

Prompting template for GPT4 with CoT is in Table 9.

A.3 Baseline Details

MetGen and NLProofs are three-stage methods. MetGen divides single-step entailment into basic logical operations. It reasons in both forward deductive and backward abductive steps. A controller finally selects promising steps among the reasoning results. NLProofs uses a trained verifier to guide the search process of proof generation. The verifier scores the generation expansion steps and finally, a proof tree is selected according to the scores.

A.4 Discussion with RLET

RLET(Liu et al., 2022) is trained using cumulative signals across the entire entailment tree, marking the first introduction of reinforcement learning into the entailment tree generation task. It employs a one-stage generation method that does not involve search or verification. RLET flexibly assigns rewards to each generated step and utilizes the overall cumulative reward to optimize training based on the full trajectory. In contrast, our approach and previous methods (as listed in Table 1) rely on exact matches of gold steps as training signals.

In Task 3, the retrieved supporting facts may contain noise and are longer than those in Task 1 and Task 2. RLET's focus on long-path proof training provides it with an advantage over our method (and previous methods listed in Table 1) in this task. However, since the supporting facts are sourced from a relevant corpus, they may deviate from the training setting, particularly under zero-shot performance conditions. As a result, RLET's performance is significantly lower than that of previous methods in Task 1 and Task 2.

Hypothesis: new york state has the least sunlight during december
Context:
sent1: precipitation is a kind of weather
sent2: daylight hours means time during which there is daylight
sent3: state means form
sent4: water is in the liquid state , called liquid water , for temperatures between 273 / 32 / 0 and 373 / 212 / 100 k / f / c
sent5: united states is located in the northern hemisphere
...
sent9: when the seasons change from summer to fall / from summer to winter , the amount of daylight will decrease
...
sent14: new york / new york state is a state located in the united states of america
sent15: winter has the least sunlight
...
sent22: the earth being tilted on its rotating axis causes seasons
sent23: december is during the winter in the northern hemisphere
...

Gold Proof: sent14 & sent5 -> int1: new york state is located in the northern hemisphere; int1 & sent23 -> int2: december is during the winter for new york state; int2 & sent15 -> hypothesis;

GPT3.5-turbo: sent23 & sent22 -> int1: December is during the winter in the northern hemisphere; int1 & sent9 -> int2: the amount of daylight will decrease during December; int2 & sent15 -> int3: December has the least sunlight; int3 & sent14 -> hypothesis;

GPT4: sent14 & sent5 -> int1: New York state is located in the northern hemisphere; int1 & sent23 -> int2: December is during the winter in New York state; int2 & sent15 -> hypothesis;

Table 8: Case study of vanilla prompt with GPT3.5-turbo and GPT4.

Hypothesis: if a fossil of a bird cannot be identified then that kind of bird is probably extinct.
Context:
sent1: identifying is similar to determining.
sent2: if a fossil is of an organism that cannot be identified then that organism is probably extinct.
sent3: discovering something usually requires seeing that something.
sent4: a dinosaur is a kind of extinct animal.
sent5: fossils can be used as evidence for the ancient environment.
sent6: dead means not alive.
...
sent25: fossils can be used to study the history of organisms and environments on earth

Reasoning step by step and finally output the proof:
From sent13 & sent24, we can infer that a bird is a kind of organism (int1);
From int1 & sent2, we can infer that if a fossil of a bird cannot be identified then that kind of bird is probably extinct (hypothesis);
Proof: sent13 & sent24 -> int1: a bird is a kind of organism; int1 & sent2 -> hypothesis;

Table 9: Prompting template of stepwise proof generation for GPT4 with COT.

A.5 Selection of Enhanced Hard Negatives

Besides randomly selecting premises to construct enhanced hard negatives, we also select according to the top similarity score calculated with BM25 (Robertson et al., 2009). Results are presented in Table 10. The results show that similar distracted premises do not contribute to intermediate node accuracy as much as random premises do. This is because randomly selected premises can introduce more diversity for the constructed hard negatives.

A.6 Case Study

We present a case study comparing the generation capabilities of LLaMA 3.2-1B and Flan-T5-Large to illustrate the differences between encoder-decoder and decoder-only LLMs, as shown in Table 11. The table indicates that LLaMA (without stepwise training) generates longer proofs because it is trained with complete proof chains. In contrast,LLaMA (with stepwise training) typically produces one-step proofs that conclude with either a "hypothesis" or an "intermediate node," reflecting the fact that both are endpoints in the sequences used for stepwise training. Flan-T5-Large (with stepwise training) strikes a balance between generating long and short proof chains due to its encoder-decoder architecture. However, it still faces challenges in accurately selecting premises.

Method	Leaves		Steps		Intermediates		Overall
	F1	AllCorrect	F1	AllCorrect	F1	AllCorrect	AllCorrect
Random	91.0	60.6	50.7	37.4	70.7	38.2	34.7
BM25	90.8	60.3	50.3	36.8	68.8	37.9	34.7

Table 10: Ablation study of enhanced hard negative construction on EntailmentBank test set on Task 2.

Hypothesis: as the distance of the star to earth decreases, the star will appear brighter.
Context:
sent1: a star produces light and heat
sent2: far is the opposite of close
sent3: as the distance from an object increases , the force of gravity on that object will decrease
sent4: brightness means amount of light
sent5: if two or more things are in a relationship then those things impact each other
sent6: feature is synonymous with characteristic
...
sent25: moving away from the source increases the distance

Gold Proof: sent1 & sent13 -> int1: a star is a source of light; int1 & sent21 & sent23 -> hypothesis;

LLaMA-3.1-1B(w/o stepwise): sent13 & sent1 -> int1: stars are a source of light; int1 & sent21 -> int2: as the distance of a star to earth decreases, the star will appear brighter; int2 & sent7 -> hypothesis;

LLaMA-3.1-1B(stepwise): sent21 & sent3 -> hypothesis;

Flan-T5-Large(stepwise): sent1 & sent13 -> int1: a star is a source of light and produces light; sent17 & int1 & sent21 -> hypothesis;

Table 11: Case study of LLaMA and Flan-T5.