

METGEN: A Module-Based Entailment Tree Generation Framework for Answer Explanation

Ruixin Hong¹, Hongming Zhang², Xintong Yu¹, Changshui Zhang¹

¹Institute for Artificial Intelligence, Tsinghua University (THUAI);

¹Beijing National Research Center for Information Science and Technology (BNRist);

¹Department of Automation, Tsinghua University, Beijing, P.R.China

²Tencent AI Lab, Seattle

{hrx20, yuxt16}@mails.tsinghua.edu.cn,

hongmzhang@tencent.com, zcs@mail.tsinghua.edu.cn,

Abstract

Knowing the reasoning chains from knowledge to the predicted answers can help construct an explainable question answering (QA) system. Advances on QA explanation propose to explain the answers with entailment trees composed of multiple entailment steps. While current work proposes to generate entailment trees with end-to-end generative models, the steps in the generated trees are not constrained and could be unreliable. In this paper, we propose METGEN, a Module-based Entailment Tree GENERation framework that has multiple modules and a reasoning controller. Given a question and several supporting knowledge, METGEN can iteratively generate the entailment tree by conducting single-step entailment with separate modules and selecting the reasoning flow with the controller. As each module is guided to perform a specific type of entailment reasoning, the steps generated by METGEN are more reliable and valid. Experiment results on the standard benchmark show that METGEN can outperform previous state-of-the-art models with only 9% of the parameters.

1 Introduction

Explanation is recognized as a key factor toward responsible AI systems (Arrieta et al., 2020). In the context of question answering (QA), providing an explanation of the predicted answers can help improve the understandability, debuggability, and trustworthiness of QA models. Great efforts have been devoted to revealing how the models predict the answers and give explanations in various forms, including showing an attention map over passages (Seo et al., 2017), giving a snippet of textual evidence (DeYoung et al., 2020), and selecting answer-supporting sentences (Xie et al., 2020; Jansen and Ustalov, 2019). Among all explanation forms, the *entailment trees* (Dalvi et al., 2021) provide the most detailed and informative explanation by exposing the *chains of reasoning*

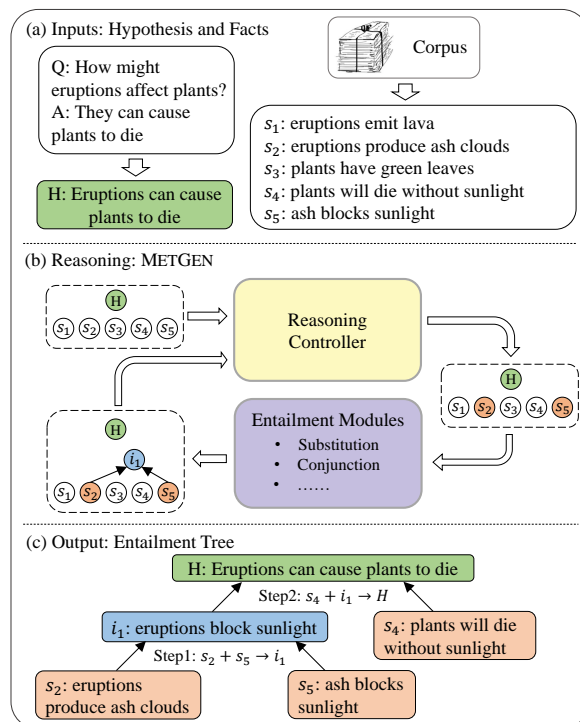


Figure 1: Given facts related to the question+answer, METGEN iteratively generates an entailment tree that contains the hypothesis (green), used facts (orange), and intermediate conclusions (blue) with several separate entailment modules and a reasoning controller.

from the knowledge to the predictions. As shown in Figure 1(a) and (c), given a hypothesis (summarizing a question+answer pair) and supporting facts (retrieved from a corpus), the goal is to generate an entailment tree where each non-leaf node is an entailment of its children. Providing a valid entailment tree would help users to understand how the hypothesis is proved, obtain novel intermediate conclusions from the basic knowledge, and gain detailed information to support decision making.

To generate the entailment trees, Dalvi et al. (2021) propose EntailmentWriter, an end-to-end sequence-to-sequence generative model, trained by maximizing the generation likelihood of the

linearized gold trees. However, they do not have an explicit strategy to constrain the validity of every single step and the tree structure. Thus, the steps are not guaranteed to satisfy the reasoning rules and could be incorrect and unreliable. For example, the step conclusion may not be entailed by the input premises or simply repeat one of the input premises (Dalvi et al., 2021). Furthermore, although their outputs are trees that can indicate the reasoning chains, the mapping mechanisms from the inputs to the trees remain implicit and invisible.

To tackle the above problems, we propose METGEN, a module-based framework to generate entailment trees in a more explicit approach and constrain the entailment steps with reasoning rules. As shown in Figure 1(b), given the target hypothesis and known facts, METGEN first uses the reasoning controller to select some steps that can help get closer to the hypothesis. Subsequently, METGEN executes the selected steps with single-step entailment modules and adds the generated intermediate facts into the known facts for the next round of reasoning. Through this iterative approach, METGEN proves the hypothesis step by step and generates the overall entailment tree.

Each module in METGEN is a generative model that can perform a specific type of entailment reasoning (e.g., making a substitution inference). To guide the modules to generate correct and sound conclusions, we train the modules with well-formed synthetic data containing the corresponding logical regularities of the reasoning types (Bostrom et al., 2021). Inspired by the forward chaining and backward chaining algorithms in logic programming (Chein and Mugnier, 2008), we adopt both deductive and abductive modules to execute forward and backward reasoning steps, respectively.

Experiments on the standard benchmark EntailmentBank (Dalvi et al., 2021) show that METGEN can outperform the previous best model with 9.0% of the model parameters. Manual evaluation results demonstrate that METGEN can generate more reliable steps. Further experiments under the data-scarce setting and cross-dataset setting (on eQASC and eOBQA (Jhamtani and Clark, 2020)) show that METGEN is more data-efficient and has better generalization capability compared with the baselines.

2 Related Works

Explainability in Question Answering. Recent works have explored the explainability of QA in

various forms (Seo et al., 2017; Ye et al., 2020; Dalvi et al., 2021; Lamm et al., 2021; Wiegrefe and Marasovic, 2021; Thayaparan et al., 2020; Rosenthal et al., 2021). One way is to retrieve multiple supporting facts related to the question or answer (Xie et al., 2020; Jansen and Ustalov, 2019; Jhamtani and Clark, 2020; Inoue et al., 2020; Yadav et al., 2019, 2020; Valentino et al., 2021; Cartuyvels et al., 2020; Zhang et al., 2020). These “rationales” (DeYoung et al., 2020) provide insights about *what* are used by the model to inform its predictions, but do not show *how* the facts are combined to generate novel intermediate conclusions. Some other works explain QA systems in a generative way, including generating explanation sentences that directly link a question to an answer (Camburu et al., 2018; Rajani et al., 2019) and thus expose the relevant knowledge used by models (Laticinnik and Berant, 2020; Shwartz et al., 2020). However, as these models generate explanations in a free form, the generated facts may not be necessarily sound (Bostrom et al., 2021). Recently, Bostrom et al. (2021) propose ParaPattern, an automated pipeline for building two kinds of single-step deductions. Different from the above work, our method generates the explanations in a multi-step tree structure (Dalvi et al., 2021), showing *what* and *how* facts are combined to draw novel intermediate conclusions and reach the final answer. The intermediate conclusions are generated by deductive and abductive entailment modules that are constrained to perform specific types of reasoning.

Multi-Hop Proof Generation. Recently, several works propose to use the transformers for multi-hop logical reasoning and generate reliable formal proofs (Clark et al., 2020; Talmor et al., 2020; Saha et al., 2020, 2021; Tafjord et al., 2021). However, they mainly focus on synthetic sentences, which have low linguistic variation and struggle to represent the flexible sentences in real QA scenarios.

Neural Module Networks. Decomposing the reasoning process into several pre-defined operations overlaps with the idea of neural module networks (Andreas et al., 2016; Hu et al., 2017; Gupta and Lewis, 2018; Gupta et al., 2020; Jiang et al., 2019). They typically assume that the question could be parsed into an executable program, i.e., the question *explicitly* describes the process to arrive at the answer. In our work, we tackle the questions/hypotheses that do not trivially describe the reasoning process and could be more challenging.

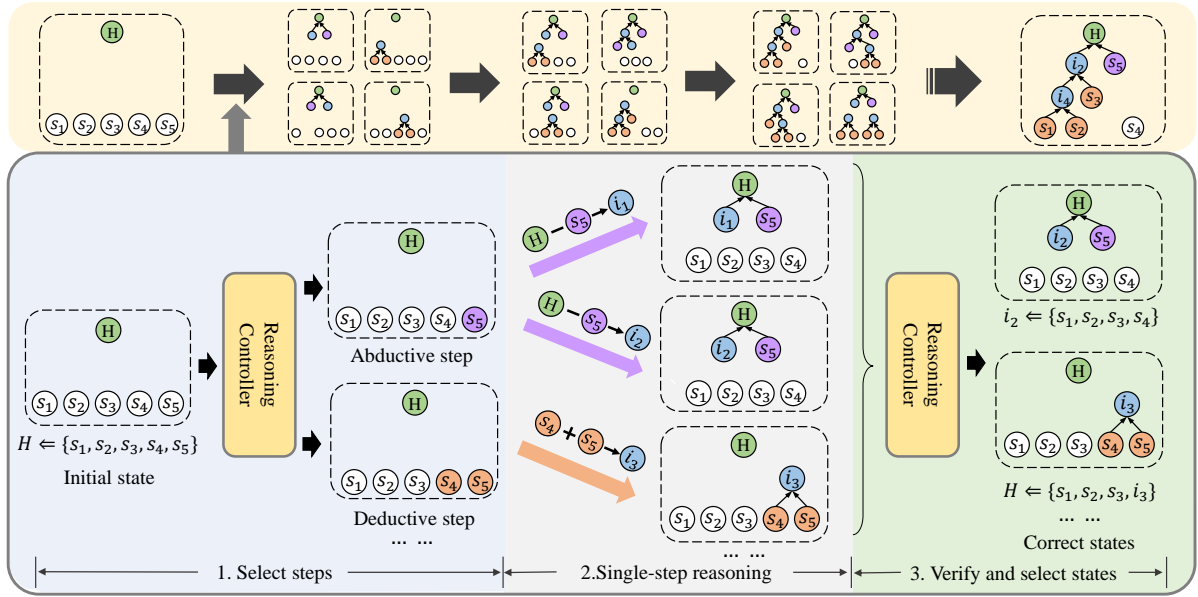


Figure 2: Reasoning process of METGEN framework. The goal is to prove the hypothesis with the given facts through reasoning iterations (the upper part). In the first reasoning iteration (the lower part), the initial state is denoted as $H \leftarrow \{s_1, s_2, s_3, s_4, s_5\}$. First, the controller selects promising steps, such as the backward abductive step $H - s_5$ and the forward deductive one $s_4 + s_5$. Then, single-step entailment modules perform the reasoning steps and generate novel intermediate facts including i_1, i_2, i_3 . After that, the controller verifies that the states $i_2 \leftarrow \{s_1, s_2, s_3, s_4\}$ and $H \leftarrow \{s_1, s_2, s_3, i_1\}$ are closer to the completion of reasoning and thus selects them for the next reasoning iteration.

3 Task Definition

As shown in Figure 1, the inputs are a hypothesis H and some fact sentences $S = \{s_1, s_2, \dots, s_n\}$ (including both relevant and irrelevant ones) expressing knowledge. H is a declarative sentence derived from a question+answer pair and can be proved by the knowledge in S . The desired output is a valid entailment tree T with the root node being H , the leaves being facts selected from S , and the intermediate nodes being novel intermediate facts (e.g., i_1, i_2). T is considered valid if each non-leaf node is a valid entailment (a conclusion that “a person would typically infer” (Dagan et al., 2013)) of its immediate children. We denote the annotated gold tree as T_{gold} and its leaf facts as S_{gold} . Following Dalvi et al. (2021), we consider three increasingly difficult tasks with different S :

Task1 (no-distractor): $S = S_{gold}$,

Task2 (distractor): $S = S_{gold} + 15\text{-}20$ distractors,

Task3 (full-corpus): $S =$ a corpus C .

4 METGEN

Figure 2 illustrates the reasoning process of METGEN. We reason one step at a time and iteratively generate the entailment trees. In each iteration, given a reasoning state (e.g., the initial state $R_0 : H \leftarrow S$, where we aim to prove H using S),

the reasoning controller selects promising steps, including forward deductive steps and backward abductive ones. We then use the corresponding modules to perform single-step entailment on the selected steps and generate novel intermediate facts. Finally, we use the controller to verify the generated facts and select the correct states to perform further reasoning. We introduce details about the module design, reasoning controller, and reasoning algorithm in Sec 4.1, 4.2, and 4.3, respectively.

4.1 Single-step Entailment Modules

4.1.1 Module Definition

We propose to divide the single-step entailment reasoning ability into a set of well-defined basic logical operations. Such a design could help improve the generalization capability (Bostrom et al., 2021; Rudin, 2019). As shown in Table 1, we adopt three common reasoning types, covering over 90% of the steps in EntailmentBank according to the analysis by Dalvi et al. (2021). Note that the entailment module types could be adjusted according to the specific tasks or domains, which allows our method to be flexibly applied to other problems.

We adopt both the deductive and abductive versions of the reasoning types. Take a gold step $s_1 + s_2 \rightarrow i_1$ as an example. Deduction is the process of reasoning from the premises to reach

Type	Definition	Logical Regularity	Example
Substitution	performing taxonomic, merynomic or other kinds of reasoning that substitute one entity for another	$s_1: \forall x \in X P(x)$ $s_2: a \in X$ $i_1: P(a)$	s_1 : the mass of a planet causes the pull of gravity on that planet . s_2 : earth is a kind of planet . i_1 : the mass of earth causes the pull of gravity on earth .
Conjunction	combining the details of both input sentences into a single output sentence	$s_1: P(x)$ $s_2: Q(x)$ $i_1: P \wedge Q(x)$	s_1 : chemical splashing can cause harm to humans / to the eyes . s_2 : chemical splashing sometimes occurs during experiments . i_1 : chemical splashing during experiments can cause harm to the eyes .
If-then	applying a conditional claim or a specific rule (one of the input sentences) to the other input sentences	$s_1: P(x) \rightarrow Q(x)$ $s_2: P(x)$ $i_1: Q(x)$	s_1 : if something requires something else then that something else is important to that something . s_2 : nuclear fusion is required for star formation . i_1 : nuclear fusion is important to star formation .

Table 1: The used reasoning types. Here, s_1 and s_2 denote input premises for deductive modules, while i_1 denotes the entailed conclusion. For logical regularity, $P(x)$ means that the predicate P is true for the entity x .

a logical conclusion. A deductive module takes the two premises s_1 and s_2 as inputs and outputs a conclusion \hat{i}_1 according to its reasoning types (denoted as $s_1 + s_2 \rightarrow \hat{i}_1$). Abduction is to find the best explanation given complete/incomplete observations (Harman, 1965). In the context of the entailment steps, given a conclusion i_1 and a premise fact s_2 as observations, the abductive module yields a plausible premise \hat{s}_1 (denoted as $i_1 - s_2 \rightarrow \hat{s}_1$), where the generated premise \hat{s}_1 and the observed premise s_2 would most likely infer the conclusion i_1 . Although the steps in the EntailmentBank may have more than two premises, we only consider the case of two premises. The reason is that the n -premise step ($n > 2$) could be further decomposed into several valid 2-premise steps (Dalvi et al., 2021) (See Appendix Figure 8 for a specific example).

4.1.2 Module Training

Training the entailment modules with data that contains the corresponding logical regularities would guide them to perform correct inferences and ensure soundness (Bostrom et al., 2021). We first train the modules with synthetic sentences to learn the logical transformations and then further fine-tune them with the end task.

We follow ParePattern (Bostrom et al., 2021), a pipeline based on syntactic retrieval, rule-based example construction, and automatic paraphrasing, to collect synthetic sentences from Wikipedia. Since Bostrom et al. (2021) only consider the substitution and contraposition deductions, we extend the method to conjunction and if-then deductions by designing the specific syntactic templates and construction rules (See Appendix A.1). In addition, we also considered the abductive form of these modules. We then fine-tune the modules with corresponding steps in EntailmentBank to adapt the modules to the science domain. Since the original

steps in EntailmentBank are not annotated with reasoning types, we manually label 400 steps of the training split and train a classifier with these steps. The remaining steps are labeled with the pseudo labels predicted by the classifier. We freeze the parameters of modules once the training is complete.

4.2 Reasoning Controller

In addition to single-step reasoning modules, we need to search for the correct path to reach the target hypothesis. The entire reasoning search space would grow rapidly as the number of input facts increases and there would also be complex branching in the trees. We introduce a reasoning controller to filter out incorrect facts, steps, and states to reduce the search space and complete the reasoning accurately and efficiently.

Figure 2 shows how the controller is used in each reasoning iteration. At the beginning of the iteration, the controller scores all possible *steps* and selects the most promising ones for single-step entailment. After the entailment modules generate intermediate facts, the controller estimates which *state* with a generated fact gets closer to the completion of reasoning and selects the best states for the next iteration. Besides the usage within each iteration, the controller also rates all *facts* at the start of the whole reasoning process and keeps only the relevant facts for the initial state when fact distractors exist.

4.2.1 Controller Model

The controller model scores steps, facts, and states based on a transformer, and its structure is shown in Figure 3.

Encoding. We first encode the target hypothesis and facts of state with a pre-trained transformer: $[\text{CLS}]H[\text{SEP}]s_1[\text{SEP}]\dots[\text{SEP}]s_n[\text{SEP}]$. We obtain the contextualized representation \mathbf{h} for H and \mathbf{f}_i for s_i using the average contextualized representa-

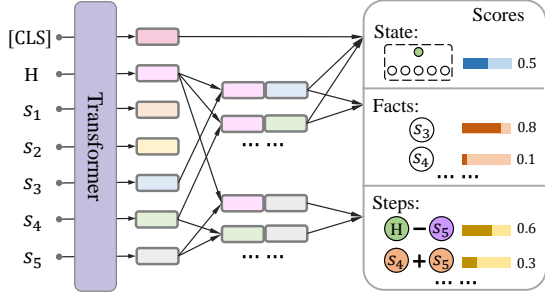


Figure 3: Reasoning controller illustration. Given a state, the controller predicts a score for the whole state, scores for facts, and scores for all possible steps.

tion of all tokens within the sentence.

Steps. We introduce feed forward networks FFN_{ded} and FFN_{abd} for deductive steps and abductive steps, respectively. Each combination of two facts is a possible deductive step (s_i, s_j) . Each combination of the target hypothesis and a fact is a possible abductive step (H, s_k) . We score them by a score function G_{step} ,

$$\begin{aligned} G_{\text{step}}(s_i, s_j) &= \text{FFN}_{\text{ded}}([\mathbf{f}_i, \mathbf{f}_j]), \\ G_{\text{step}}(H, s_k) &= \text{FFN}_{\text{abd}}([\mathbf{h}, \mathbf{f}_k]), \end{aligned} \quad (1)$$

where $[\cdot]$ is the concatenate operation. We normalize the step scores by applying Softmax over all possible deductive and abductive steps.

Facts. The fact score indicates whether the fact is useful by how similar the fact is to the state’s target hypothesis. We assume that if a fact has a smaller depth in the gold entailment tree (i.e., closer to the root), it would be more similar to the target hypothesis than those facts with a larger depth. We introduce FFN_{fact} as a learnable similarity function and determine the fact score by comparing it with the target,

$$G_{\text{fact}}(s_i) = \sigma(\text{FFN}_{\text{fact}}([\mathbf{h}, \mathbf{f}_i])), \quad (2)$$

where σ is the Sigmoid function.

State. The state score reflects the quality of the current state and indicates whether this state should be used for further reasoning. We assign the state score using the following two parts:

$$G_{\text{state}}(R) = \frac{\lambda}{n} \sum_{s_i \in S} G_{\text{fact}}(s_i) + (1-\lambda)\sigma(\text{FFN}_{\text{cls}}(\mathbf{f}_{[\text{CLS}]})), \quad (3)$$

where λ is a learnable weight, $\mathbf{f}_{[\text{CLS}]}$ is the representation of $[\text{CLS}]$, FFN_{cls} is a feed forward network. The first part helps choose states that contain more relevant facts and fewer distractors. The second part comprehensively considers the whole state and gives the promising one a higher score.

4.2.2 Controller Training

Training State Construction. We decompose the gold entailment trees into several intermediate states for training. We add disturbances to the trees to make positive and negative states. For each gold deductive step (e.g., $s_1 + s_2 \rightarrow i_1$), we use the deductive module to predict a conclusion \hat{i}_1 . If the predicted \hat{i}_1 is correct, we replace i_1 in the state with \hat{i}_1 to make new positive states. Otherwise, we replace i_1 with \hat{i}_1 to make negative states. The abductive modules are also used in a similar way.

Loss Function. We train the controller with corresponding margin ranking losses $\mathcal{L}_{\text{step}}$, $\mathcal{L}_{\text{fact}}$, and $\mathcal{L}_{\text{state}}$ to learn to rank the correct steps, facts, and states ahead of incorrect ones, respectively. Specifically, the loss for scoring steps is

$$\mathcal{L}_{\text{step}} = \frac{1}{N_1} \sum_{(p^+, p^-)} \varphi(G_{\text{step}}(p^+), G_{\text{step}}(p^-), m_{\text{step}}), \quad (4)$$

where p^+ and p^- are the positive and negative step, N_1 is the number of (p^+, p^-) pairs, $\varphi(x_1, x_2, m) = \max(0, x_2 - x_1 + m)$ is the margin loss, and m_{step} is the margin for steps.

For facts, we have

$$\begin{aligned} \mathcal{L}_{\text{fact}} &= \frac{1}{N_2} \sum_{s_1^+, s_2^+ \in S_{\text{gold}}} \varphi(G_{\text{fact}}(s_1^+), G_{\text{fact}}(s_2^+), m_{\text{fact}}) \\ &\quad + \frac{1}{N_3} \sum_{s^- \notin S_{\text{gold}}} -\log(1 - G_{\text{fact}}(s^-)), \end{aligned} \quad (5)$$

where s_1^+ is a fact which has smaller depth in the gold tree than s_2^+ , s^- is the distractor, N_2 is the number of (s_1^+, s_2^+) pairs, N_3 is the number of distractors, and m_{fact} is the margin for facts.

For states, we sample a positive state R^+ and a negative state R^- from a tree and train the controller with

$$\mathcal{L}_{\text{state}} = \varphi(G_{\text{state}}(R^+), G_{\text{state}}(R^-), m_{\text{state}}), \quad (6)$$

where m_{state} is the margin for states.

Finally, we average the above losses over all trees in the training split and train the controller with

$$\mathcal{L} = \mathcal{L}_{\text{step}} + \mathcal{L}_{\text{fact}} + \mathcal{L}_{\text{state}}. \quad (7)$$

Appendix B gives more controller training details.

4.3 Reasoning Algorithm

Since the entailment trees are generated iteratively and the search space for reasoning could be large for each iteration, we adopt beam search for efficient reasoning. Given the initial state $H \Leftarrow S$, we first remove s_i with a low *fact score* to filter distractors. Subsequently, we perform several reasoning

	Train	Dev	Test	All
Questions / Trees	1,131	187	340	1,840
Entailment steps	4,175	597	1,109	5,881

Table 2: EntailmentBank statistics.

iterations until the target hypothesis is proved or the maximum reasoning depth is reached. In each iteration, we select the steps with the highest *step scores*, execute the steps with all types of deductive or abductive modules, and construct novel states with the generated intermediate facts. We remain the top- K states ranked with *state scores* for the next iteration, where K is the beam size. More algorithm details are in Appendix Algorithm 1.

5 Experiments

We conduct experiments on EntailmentBank (Dalvi et al., 2021), the first dataset supporting QA explanations in the form of the entailment tree. EntailmentBank contains 1,840 entailment trees, each of which corresponds to a question from the ARC dataset (Clark et al., 2018). On average, each tree contains 7.6 nodes across 3.2 steps. Summary statistics are shown in Table 2.

5.1 Evaluation Metrics

Following Dalvi et al. (2021), we first align nodes in the predicted tree T_{pred} with nodes in the gold tree T_{gold} and then evaluate with three dimensions:

- **Leaves:** To evaluate whether T_{pred} uses the correct leaf facts, we compute **F1** score by comparing the predicted leaf facts S_{pred} to S_{gold} .
- **Steps:** To evaluate whether the individual steps are *structurally* correct, we compare all steps in two trees and compute **F1**. A predicted step is considered *structurally* correct if its children’s identifiers (e.g., s_1, i_2) perfectly match the gold ones.
- **Intermediates:** To evaluate whether the intermediate conclusions are correct, we report the **F1** of comparing the aligned intermediate conclusions. A predicted intermediate sentence \hat{i} is considered correct if the BLEURT-Large-512 score of the aligned intermediate pair (\hat{i}, i) is larger than 0.28¹.

The **AllCorrect** score is 1 if F1 is 1, 0 otherwise². Given the above scores, we comprehensively evaluate T_{pred} with **Overall AllCorrect** whose value

¹The threshold was picked using 300 manually labeled pairs (Dalvi et al., 2021).

²We repair a bug in the official evaluation code, which makes the Intermediate AllCorrect = 1 if the precision = 1 (rather than if F1 = 1), which leads to an overestimation on the Intermediate AllCorrect.

is 1 if and only if all the leaves, steps and intermediates are all correct. This is a strict metric since any error in T_{pred} will lead to a score of 0.

5.2 Baselines

We compare with the SOTA entailment tree generation method **EntailmentWriter** (Dalvi et al., 2021), which directly generates the linearized trees (e.g., $s_2 + s_5 \rightarrow i_1$: eruptions block sunlight; $s_4 + i_1 \rightarrow H$) given $H + QA + S$ with an end-to-end encoder-decoder framework. We also follow the “Iterative” ProofWriter (Tafjord et al., 2021), which is one of the SOTA proof generation methods for logical reasoning, to extend EntailmentWriter to **EntailmentWriter-Iter**. EntailmentWriter-Iter iteratively generates a part of the linearized tree in one forward process (e.g., $s_2 + s_5 \rightarrow i_1$: eruptions block sunlight;) and concatenates all parts to make the final tree. It completes the step selection and entailment reasoning in a seq2seq model and does not provide the reasoning types of steps.

5.3 Implementation Details

Modules. We implement the entailment modules on top of T5-large (Raffel et al., 2020) with the following two implementations. (1) **Separated.** We implement each module separately. We have six models in total, corresponding to the three reasoning types of deductive and abductive versions. (2) **Prefixed.** We implement all modules with a single model. To specify which reasoning type the model should perform, we follow Raffel et al. (2020) to add a type-specific prefix (e.g., “deductive substitution:”) to the input before feeding it to the model. To evaluate the modules, we annotate the types of 275 steps in the dev split. We train the modules with a batch size of 20 for 100 epochs.

Controller. The controller is implemented with albert-xxlarge-v2 (Lan et al., 2019). We train two individual controllers for Task1 and Task2. For Task3, we reuse the Task2 model without additional training. The controllers are trained with a batch size of 10 for 1,000 epochs. The margins m_{step} , m_{fact} , and m_{state} are tuned on the development split and all set to 0.1.

Algorithm. For Task1, we iterate until all facts in S are used. For Task2, we use a fact score threshold of 0.001 to filter distractors and a maximum reasoning depth of 5. We select the top 10% steps for each state and set the beam size to 10. All hyper-parameters are selected using the dev split (Appendix C). For Task3, we follow Dalvi et al.

Task	Method	n_{para}	Leaves		Steps		Intermediates		Overall AllCorrect
			F1	AllCorrect	F1	AllCorrect	F1	AllCorrect	
Task1 (no-distractor)	EntailmentWriter (T5-11B)†	11.00	99.0	89.4	51.5	38.2	71.2	52.9 †	35.6
	EntailmentWriter (T5-large)	0.77	98.4	84.1	50.0	38.5	67.0	35.9	34.4
	EntailmentWriter-Iter (T5-large)	0.77	99.8	97.6	51.6	38.5	68.3	36.5	35.0
	METGEN-separated (Ours)	0.22+6×0.77	100.0	100.0	57.9	42.1	71.3	39.2	37.0
	METGEN-prefixed (Ours)	0.22+0.77	100.0	100.0	57.7	41.9	70.8	39.2	36.5
Task2 (distractor)	EntailmentWriter (T5-11B)†	11.00	89.1	48.8	41.4	27.7	66.2	53.2 †	25.6
	EntailmentWriter (T5-large)	0.77	83.2	35.0	39.5	24.7	62.2	28.2	23.2
	EntailmentWriter-Iter (T5-large)	0.77	85.2	40.9	38.9	26.8	63.5	29.1	25.0
	METGEN-separated (Ours)	0.22+6×0.77	83.7	48.6	41.7	30.4	62.7	32.7	28.0
	METGEN-prefixed (Ours)	0.22+0.77	82.7	46.1	41.3	29.6	61.4	32.4	27.7
Task3 (full-corpus)	EntailmentWriter (T5-11B)†	11.00	39.7	3.8	7.8	2.9	36.4	13.2	2.9
	EntailmentWriter (T5-large)	0.77	30.9	1.2	4.4	1.2	28.8	5.6	1.2
	EntailmentWriter-Iter (T5-large)	0.77	32.4	1.8	4.4	1.5	29.7	6.5	1.5
	METGEN-separated (Ours)	0.22+6×0.77	34.8	8.7	9.8	8.6	36.7	20.4	8.6
	METGEN-prefixed (Ours)	0.22+0.77	34.8	8.7	9.8	8.6	36.6	20.4	8.6

Table 3: Automatic evaluation results on the EntailmentBank test split. † indicates results from the published paper². n_{para} denotes the number of model parameters (B).

Method	Task1		Task2	
	Automatic	Manual	Automatic	Manual
EntailmentWriter (T5-large)	35	46	21	26
EntailmentWriter-Iter (T5-large)	35	47	25	35
METGEN-prefixed (Ours)	36	53	27	39

Table 4: Entailment tree evaluation results on 100 uniformly sampled questions from the test split. We report the proportion (%) of the predicted trees that are rated as valid, following automatic and manual evaluation.

(2021) to retrieve 25 sentences from the corpus C using the H as the query. We use the same retrieval results as EntailmentWriter for a fair comparison. Model checkpoints are selected using the dev split. More implementation details can be found in the Appendix.

6 Result Analysis

6.1 Automatic Evaluation

As shown in Table 3, our methods outperform all baseline methods on the strictest metric Overall AllCorrect for all three tasks. Notice that the trees generated by our methods only contain 2-premise steps, which would lead to a 0 Overall AllCorrect score on 26% of test samples whose annotations contain n -premise ($n > 2$) steps. Even so, our METGEN-separated still obtains an absolute improvement of 1.4%/2.4%/5.7% on Task1/2/3 in comparison to the strongest baseline. With only 9.0% of the model parameters, METGEN-prefixed can outperform the EntailmentWriter (T5-11B) by absolute 0.9%/2.1%/5.7% on Task1/2/3. In the case of using a comparable amount of model parameters, METGEN-prefixed also outperforms the EntailmentWriter-Iter (T5-large) by a large margin. For Task3, we note that all methods perform

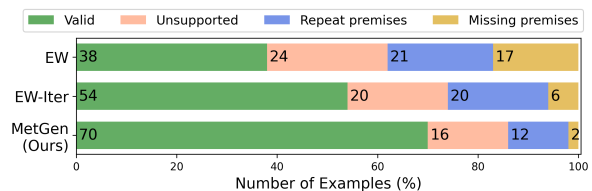


Figure 4: Manual evaluation results of 100 single-step entailments uniformly sampled from the predicted trees of Task2 test split. EW denotes EntailmentWriter.

poorly. The main reason is that the retrieved facts may not contain all the required facts S_{gold} (68% of the cases). We note that METGEN underperforms the baselines on some metrics, probably due to the inaccuracy of the tree alignment algorithm in the automatic evaluation (Appendix G).

6.2 Manual Evaluation

As analysed by Dalvi et al. (2021), the automated metrics might misjudge some valid trees and thus underestimate the performance. To make a more accurate comparison, we perform the manual evaluation. We compare three methods with a comparable amount of model parameters, EntailmentWriter (T5-large), EntailmentWriter-Iter (T5-large), and METGEN-prefixed. For each step and tree, we invite three students as experts to evaluate the validity. The inter-annotator agreement (Cohen’s kappa statistic) is 0.85/0.76 for the step/tree, indicating the substantial agreement between annotators.

Validity of Full Entailment Trees. As shown in Table 4, under the manual evaluation, METGEN outperforms the baselines with large margins.

Validity of Individual Entailment Steps. We review the validity of the single-step entailment and annotate each step with one of the four categories:

- **Valid:** The step conclusion can be inferred from

Implementation	Models	Reasoning Type	Training Data	Overall AllCorrect	Single-step Accuracy
(a)	Sep 6×T5-large	✓	S+E	28.0	81.0
(b)	Sep 6×BART-large	✓	S+E	26.2	77.0
(c)	Sep 6×T5-base	✓	S+E	27.3	78.0
(d)	Sep 6×T5-large	✓	E	27.8	79.5
(e)	Sep 6×T5-large	✓	S	23.5	43.6
(f)	Pre 1×T5-large	✓	S+E	27.7	78.4
(g)	Pre 1×T5-large	✓	E	27.4	78.1
(h)	Pre 1×T5-large	×	E	25.9	76.0

Table 5: Ablation results on entailment modules. Sep/Pre indicates separated/prefixed. S/E denotes the synthesis/EntailmentBank step training data.

Task	Method	Leaves	Steps	Intermediate	Overall
Task1	controller	100.0	42.1	39.2	37.0
	w/o abduction	100.0	41.4	38.4	36.2
	heuristic	100.0	31.2	31.2	28.8
Task2	controller	48.6	30.4	32.7	28.0
	w/o abduction	44.5	28.3	31.6	27.0
	heuristic	3.2	3.2	12.1	3.2

Table 6: Ablation results on the reasoning controller. We report the AllCorrect scores on the test split.

the premises and does not trivially repeat them.

- **Unsupported:** The conclusion is in conflict with, irrelevant with, or not followed from the premises.
- **Repeat premises:** The conclusion trivially repeats one or more of the premises.
- **Missing premises:** The conclusion uses knowledge unstated in the premises. The step would be correct if one additional premise from S is added.

As shown in Figure 4, METGEN achieves considerable improvement in the validity of steps compared to the baseline methods. We note that 17% of the steps of EntailmentWriter belong to missing premises. METGEN constrains the reasoning types of steps and uses the premise-related and context-independent entailment modules to perform every single step. This can reduce the cases of missing premises (from 17% to 2%) and improve the validity of the conclusions (from 38% to 70%).

6.3 Ablation Study

Entailment Modules Analysis. Table 5 reports the ablation results on modules. We report the Overall AllCorrect on test split and the single-step entailment accuracy on the labeled dev steps, and can make the following observations. (1) **Separated vs. Prefixed.** We can see that METGEN-prefixed achieves slightly worse performance than METGEN-separated ((a) vs. (f) and (d) vs. (g)). This is mainly because separate modules could better learn different types of reasoning. However, in our final system, we still choose to use METGEN-prefixed

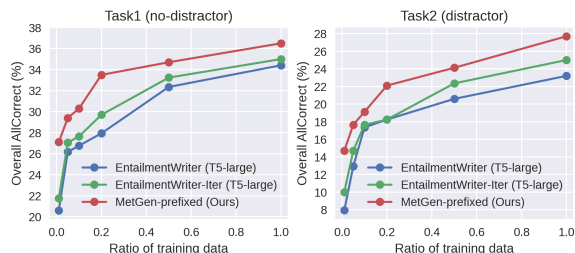


Figure 5: Results on different ratios (0.01, 0.05, 0.10, 0.20, 0.50, 1.00) of EntailmentBank training data.

due to the consideration of model size. (2) **Clarifying Reasoning Types.** We train a module to infer without distinguishing or assigning specific reasoning types. We find that the performance drops from 27.4% to 25.9% ((g) vs. (h)), suggesting that clarifying the reasoning types of the entailment steps is crucial for generating entailment trees. (3) **Training Data.** Comparing (a) and (d), we find that training with the synthesis data could improve the accuracy. Without tuning on EntailmentBank (setting (e)), the modules might not adapt to the science domain and obtain low step accuracy. However, the well-trained controller would verify and filter the error conclusions, thus our method can still achieve 23.5% on Overall AllCorrect. (4) **Generative Model.** A stronger generative model, which achieves higher single-step accuracy, could achieve higher tree generation performance (comparing (a), (b) and (c)), indicating that our method can be further improved with stronger entailment modules.

Controller and Algorithm Analysis. (1) **Is the reasoning controller necessary?** To answer this question, we design a **heuristic** generation algorithm without the controller (Appendix D). It uses the BLEURT scores as heuristic information to guide the reasoning. As shown in Table 6, the heuristic method achieves observable lower performance. The controller could aid in eliminating the error steps and states, so as to find the valid trees efficiently and accurately. Without the controller, we find it difficult to find effective heuristic information. (2) **Effect of Abductive Steps.** The generation performance drops when abductive steps are not used. This suggests that abductive steps, as a way of backward searching, could help improve the quality of generated trees.

6.4 Data-scarce Setting

Figure 5 reports the results in the data-scarce setting. Our method is more data-efficient. With only 1% of the EntailmentBank training data, our

Method	eQASC		eOBQA	
	P@1	NDCG	P@1	NDCG
EntailmentWriter (T5-large)	52.48	73.14	69.07	89.05
EntailmentWriter-Iter (T5-large)	52.56	73.28	72.15	90.19
METGEN-prefixed (Ours)	55.81	74.19	74.89	90.50

Table 7: Cross-dataset results on the eQASC and eOBQA test split.

method obtains 14.7% on Task2 Overall AllCorrect, in comparison to 10.0% of the strongest baseline. When the data is scarce, the advantage of training our modules with synthetic data becomes more significant. It can help alleviate the overfitting on few EntailmentBank sentences.

6.5 Cross-dataset Setting

To test the generalization capability of our method, we conduct cross-dataset experiments on datasets eQASC and eOBQA (Jhamtani and Clark, 2020), which collect *one-step* entailment trees for questions from QASC (Khot et al., 2020) and OpenBookQA (Mihaylov et al., 2018), respectively. Given H and S , their task requires selecting the valid one-step trees (e.g., $s_1 + s_2 \rightarrow H$) from a candidate set. We apply the Task2 models (without fine-tuning on eQASC or eOBQA) to select from the candidate trees (Appendix E). Following Jhamtani and Clark (2020), we evaluate models with the **P@1** and **NDCG** metrics. Questions with no valid tree are filtered. As shown in Table 7, our method achieves better generalization performance. Instead of training a seq2seq model with a single generation loss, our method explicitly models the step and state selection ability (equation (1) and (3)) and guides the controller with specific losses to rank the correct ones ahead of incorrect ones. Such a manner could aid in alleviating the overfitting on training data and improve the generality.

7 Conclusion

We propose METGEN, a module-based framework to generate the entailment trees for explaining answers. METGEN reasons with single-step entailment modules and the reasoning controller. Experiments on EntailmentBank benchmark show METGEN can generate valid trees with reliable steps and achieve SOTA performance.

8 Acknowledgements

We appreciate the anonymous reviewers for their insightful comments. We would like to

thank Zhihong Shao for the helpful discussions. This work is supported by the National Key Research and Development Program of China (No. 2018AAA0100701), and the Guoqiang Institute of Tsinghua University with Grant No. 2020GQG0005.

References

- Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. 2016. [Neural module networks](#). In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 39–48. IEEE Computer Society.
- Alejandro Barredo Arrieta, Natalia Díaz Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador García, Sergio Gil-Lopez, Daniel Molina, Richard Benjamins, Raja Chatila, and Francisco Herrera. 2020. [Explainable artificial intelligence \(XAI\): concepts, taxonomies, opportunities and challenges toward responsible AI](#). *Inf. Fusion*, 58:82–115.
- Kaj Bostrom, Xinyu Zhao, Swarat Chaudhuri, and Greg Durrett. 2021. [Flexible generation of natural language deductions](#). In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 6266–6278. Association for Computational Linguistics.
- 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 31, NeurIPS 2018*, pages 9560–9572.
- Ruben Cartuyvels, Graham Spinks, and Marie-Francine Moens. 2020. [Autoregressive reasoning over chains of facts with transformers](#). In *Proceedings of the 28th International Conference on Computational Linguistics, COLING 2020*, pages 6916–6930. International Committee on Computational Linguistics.
- Michel Chein and Marie-Laure Mugnier. 2008. *Graph-based knowledge representation: computational foundations of conceptual graphs*. Springer Science & Business Media.
- Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. [Think you have solved question answering? try arc, the AI2 reasoning challenge](#). *CoRR*, abs/1803.05457.
- 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.

- Ido Dagan, Dan Roth, Mark Sammons, and Fabio Massimo Zanzotto. 2013. *Recognizing Textual Entailment: Models and Applications*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers.
- 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, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 7358–7370. Association for Computational Linguistics.
- 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, ACL 2020, Online, July 5-10, 2020*, pages 4443–4458. Association for Computational Linguistics.
- Nitish Gupta and Mike Lewis. 2018. *Neural compositional denotational semantics for question answering*. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 2152–2161. Association for Computational Linguistics.
- Nitish Gupta, Kevin Lin, Dan Roth, Sameer Singh, and Matt Gardner. 2020. *Neural module networks for reasoning over text*. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.
- Gilbert H Harman. 1965. The inference to the best explanation. *The philosophical review*, 74(1):88–95.
- Ronghang Hu, Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Kate Saenko. 2017. *Learning to reason: End-to-end module networks for visual question answering*. In *IEEE International Conference on Computer Vision, ICCV 2017*, pages 804–813. IEEE Computer Society.
- Naoya Inoue, Pontus Stenetorp, and Kentaro Inui. 2020. *R4C: A benchmark for evaluating RC systems to get the right answer for the right reason*. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 6740–6750. Association for Computational Linguistics.
- 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@EMNLP 2019, Hong Kong, November 4, 2019*, pages 63–77. Association for Computational Linguistics.
- 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 2020, Online, November 16-20, 2020*, pages 137–150. Association for Computational Linguistics.
- Yichen Jiang, Nitish Joshi, Yen-Chun Chen, and Mohit Bansal. 2019. *Explore, propose, and assemble: An interpretable model for multi-hop reading comprehension*. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 2714–2725. Association for Computational Linguistics.
- Tushar Khot, Peter Clark, Michal Guerquin, Peter Jansen, and Ashish Sabharwal. 2020. *QASC: A dataset for question answering via sentence composition*. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020*, pages 8082–8090. AAAI Press.
- Matthew Lamm, Jennimaria Palomaki, Chris Alberti, Daniel Andor, Eunsol Choi, Livio Baldini Soares, and Michael Collins. 2021. *QED: A framework and dataset for explanations in question answering*. *Trans. Assoc. Comput. Linguistics*, 9:790–806.
- Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. *ALBERT: A lite BERT for self-supervised learning of language representations*. *CoRR*, abs/1909.11942.
- Veronica Latcinnik and Jonathan Berant. 2020. *Explaining question answering models through text generation*. *CoRR*, abs/2004.05569.
- Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. *Roberta: A robustly optimized BERT pretraining approach*. *CoRR*, abs/1907.11692.
- Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. *Can a suit of armor conduct electricity? A new dataset for open book question answering*. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 2381–2391. 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*. *J. Mach. Learn. Res.*, 21:140:1–140:67.
- Nazneen Fatema Rajani, Bryan McCann, Caiming Xiong, and Richard Socher. 2019. *Explain yourself! leveraging language models for commonsense reasoning*. In *Proceedings of the 57th Conference of*

- the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 4932–4942. Association for Computational Linguistics.
- Sara Rosenthal, Mihaela A. Bornea, Avirup Sil, Radu Florian, and J. Scott McCarley. 2021. [Do answers to boolean questions need explanations? yes](#). *CoRR*, abs/2112.07772.
- 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 2020, Online, November 16-20, 2020*, pages 122–136. Association for Computational Linguistics.
- Swarnadeep Saha, Prateek Yadav, and Mohit Bansal. 2021. [multiprover: Generating multiple proofs for improved interpretability in rule reasoning](#). In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, pages 3662–3677. Association for Computational Linguistics.
- Min Joon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. 2017. [Bidirectional attention flow for machine comprehension](#). In *5th International Conference on Learning Representations, ICLR 2017*. OpenReview.net.
- Noam Shazeer and Mitchell Stern. 2018. [Adafactor: Adaptive learning rates with sublinear memory cost](#). In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 4603–4611. PMLR.
- Vered Shwartz, Peter West, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2020. [Unsupervised commonsense question answering with self-talk](#). In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pages 4615–4629. 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, Online, August 1-6, 2021*, pages 3621–3634. Association for Computational Linguistics.
- Alon Talmor, Oyvind Tafjord, Peter Clark, Yoav Goldberg, and Jonathan Berant. 2020. [Leap-of-thought: Teaching pre-trained models to systematically reason over implicit knowledge](#). In *Advances in Neural Information Processing Systems 33, NeurIPS 2020*.
- Mokanarangan Thayaparan, Marco Valentino, and André Freitas. 2020. [A survey on explainability in machine reading comprehension](#). *CoRR*, abs/2010.00389.
- Marco Valentino, Mokanarangan Thayaparan, and André Freitas. 2021. [Unification-based reconstruction of multi-hop explanations for science questions](#). In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume, EACL 2021*, pages 200–211. Association for Computational Linguistics.
- Sarah Wiegrefe and Ana Marasovic. 2021. Teach me to explain: A review of datasets for explainable natural language processing. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*.
- Zhengnan Xie, Sebastian Thiem, Jaycie Martin, Elizabeth Wainwright, Steven Marmorstein, and Peter A. 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, LREC 2020*, pages 5456–5473. European Language Resources Association.
- Vikas Yadav, Steven Bethard, and Mihai Surdeanu. 2019. [Quick and \(not so\) dirty: Unsupervised selection of justification sentences for multi-hop question answering](#). In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 2578–2589. Association for Computational Linguistics.
- Vikas Yadav, Steven Bethard, and Mihai Surdeanu. 2020. [Unsupervised alignment-based iterative evidence retrieval for multi-hop question answering](#). In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 4514–4525. Association for Computational Linguistics.
- Qinyuan Ye, Xiao Huang, Elizabeth Boschee, and Xiang Ren. 2020. [Teaching machine comprehension with compositional explanations](#). In *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020*, volume EMNLP 2020 of *Findings of ACL*, pages 1599–1615. Association for Computational Linguistics.
- 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, ACL 2020, Online, July 5-10, 2020*, pages 5736–5745. Association for Computational Linguistics.

A Entailment Modules Training Details

A.1 Synthetic Data

We follow the ParaPattern (Bostrom et al., 2021) to collect synthetic training data for the entailment modules. Since they only consider the substitution and contraposition deductions, we extend the method to conjunction and if-then deductions by designing the specific syntactic templates and construction rules. Table 9 shows the used syntactic patterns. We use Spacy³ to match sentences from Wikipedia (version “20200501.en”). In total, we collect about 24k, 443k, and 97k sentences for substitution, conjunction, and if-then modules, respectively. We follow Bostrom et al. (2021) to train the modules on the synthetic data with a learning rate of 3e-5 for 1 epoch.

A.2 Reasoning Type Annotations of EntailmentBank

The original steps in the EntailmentBank are not annotated with reasoning types. We manually annotated the reasoning types of 400 steps in the training split (Train-manual) and 275 steps in the development split (Dev-manual). To label the remaining steps in the training split, we train a classifier with the Train-manual steps. We use the Roberta-large (Liu et al., 2019) as our classifier. It achieves an accuracy rate of 88% on the Dev-manual steps. We use the classifier to predict the reasoning types of the remaining 2-premise steps and take the predicted types as the pseudo labels (Train-pseudo). Table 8 shows the statistics of the reasoning type annotations.

Split	Sub.	Conj.	If-then	All
Train-manual	211	105	84	400
Train-pseudo	2,441	812	535	3,788
Dev-manual	153	71	51	275

Table 8: Statistics of the step reasoning type annotations.

B Controller Training Details

Training Data. We decompose the gold entailment trees into several intermediate states for training. For example, the tree in Figure 1(c) can be decomposed into the following positive states: $R_0 : H \Leftarrow \{s_1, s_2, s_3, s_4, s_5\}$, $R_1 : H \Leftarrow \{s_1, s_3, s_4, i_1\}$, and $R_2 : i_1 \Leftarrow \{s_1, s_2, s_3, s_5\}$. The state R_0 has two distractors s_1 and s_3 , one positive deductive

³<https://spacy.io/>

step $s_2 + s_5 \rightarrow i_1$, and one positive abductive step $H - s_4 \rightarrow i_1$. We add disturbances to the trees to make positive and negative states. For the state R_1 , the fact i_1 is the conclusion of gold step $s_2 + s_5 \rightarrow i_1$. We use a deductive module to predict a conclusion \hat{i}_1 given s_2 and s_5 . If the predicted \hat{i}_1 is correct, we replace i_1 with \hat{i}_1 to make new positive states $R_1^+ : H \Leftarrow \{s_1, s_3, s_4, \hat{i}_1\}$. The R_1^+ can be used to perform further reasoning. Otherwise, we replace i_1 with \hat{i}_1 to make negative states R_1^- . The R_1^- contains an incorrect conclusion \hat{i}_1 and thus should not be used for further reasoning. The reasoning controller should be trained to learn to distinguish between R_1^+ and R_1^- and give the R_1^+ a higher state score than R_1^- . To judge whether the generated \hat{i}_1 is correct, we follow the evaluation metrics (Dalvi et al., 2021) to use BLEURT. The predicted \hat{i}_1 is considered correct if the BLEURT score between \hat{i}_1 and the gold i_1 is larger than 0.28.

C Reasoning Algorithm and Hyperparameter Analysis

Algorithm 1 shows the whole reasoning process. The hyperparameters are selected with the development split, as shown in Figure 6. We select a beam size of 10, a max reasoning depth of 5, a distractor threshold of 0.001, and a step sampling rate of 10%. We only consider the steps whose sentences have word overlap. When constructing the entailment tree, we use the BLEURT scores to align the target of a state to the most similar fact. Note that when making a new reasoning state with the step p and the novel intermediate fact i , if the step p is a backward abductive step, we replace the original target hypothesis with i and treat the i as the target hypothesis which the new state aims to prove (as shown in Figure 2). We run our method three times and report the average performance.

D Heuristic Reasoning Algorithm without the Controller

To investigate the effect of the reasoning controller for entailment tree generation, we design a heuristic generation algorithm that does not use the reasoning controller. Since the cost of traversing the entire search space is unaffordable, we adopt the beam search. In each reasoning state, we try all possible steps with entailment modules and make new candidate reasoning states. To select the correct states, we use the BLEURT scores as the heuristic information to guide the search process. Specifically,

Algorithm 1 Reasoning Algorithm

Input: Hypothesis H , fact sentences S , controller, deductive modules M_{ded} , abductive modules M_{abd}

Parameter: Beam size K , max reasoning depth D , distractor threshold θ , step sampling rate τ

```
1: // Construct initial reasoning state
2: Remove  $s_i$  with fact score less than  $\theta$  in  $S$ 
3:  $R_{init} \leftarrow (H \Leftarrow \text{the filtered sentences } S')$ 
4:  $\mathcal{R}_{beam} \leftarrow \{R_{init}\}, \mathcal{R} \leftarrow \mathcal{R}_{beam}$ 
5: // Reasoning with beam search
6: while the depth does not reach  $D$  do
7:    $\mathcal{R}'_{beam} \leftarrow \{\}$ 
8:   for  $R \in \mathcal{R}_{beam}$  do
9:     // Select promising steps
10:    for  $p \in \text{steps of } R$  with top  $\tau\%$  step score do
11:      // Single-step entailment reasoning
12:      for  $m \in M_{ded}$  or  $m \in M_{abd}$  do
13:        execute step  $p$  with module  $m$  and
14:        obtain a novel intermediate fact  $i$ 
15:        construct a new state  $R_{new}$  with the
16:        step  $p$  and the fact  $i$ 
17:         $R'_{beam} \leftarrow R'_{beam} \cup \{R_{new}\}$ 
18:      end for
19:    end for
20:    // Verify and select states
21:     $\mathcal{R}_{beam} \leftarrow K$  states with the highest state
22:    scores from  $R'_{beam}$ 
23:     $\mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{R}_{beam}$ 
24:  end while
25: // Construct the entailment tree
26: for  $R \in \mathcal{R}$  do
27:   Align the target of  $R$  to the most similar fact
28:   sentence of  $R$  to make a tree  $T$ 
29: end for
30: Select the tree  $\hat{T}$  with highest score
31: Return The entailment tree  $\hat{T}$ 
```

given a candidate state $R : H \Leftarrow S$, we estimate the similarity between a fact $s_i \in S$ and the target H by

$$G'_{fact}(s_i) = \text{BLEURT}(H, s_i), \quad (8)$$

and then score a candidate state by

$$G'_{state}(R) = \frac{1}{n} \sum_{s_i \in S} G'_{fact}(s_i). \quad (9)$$

The top- K candidate states with the highest state scores are selected to perform further reasoning,

where K is the beam size. We use the same beam size as the algorithm with the controller uses.

E Experiment Details on eQASC and eOBQA

For each question+answer pair, the eQASC/eOBQA provides the corresponding hypothesis H , about 10/4 facts as S , and a candidate set of steps. Each candidate step is a 2-premise single step from two facts to H (e.g., $s_1 + s_2 \rightarrow H$) and can be viewed as a *one-step entailment tree* with three nodes. The target is to select the correct trees/steps from the candidate set. There might be more than one correct tree in the candidate set. We conduct experiments on the questions with at least one correct entailment tree (677 eQASC questions and 79 eOBQA questions). Since the given S contains distractors, we adopt the Task2 models trained on EntailmentBank (without further fine-tuning on eQASC and eOBQA) to perform cross-dataset experiments.

For our method, we follow our Task2 reasoning algorithm to select from the candidate trees/steps. Specifically, we first filter out the facts in S with low fact scores using a threshold (selected using the development split). Then we predict the step scores for the candidate steps and select the step with the highest score. For the EntailmentWriter, we feed the S and H to the EntailmentWriter and score each candidate step with the $\frac{1}{\text{PPL}}$, where PPL is the perplexity of the sequence segment representing the step (e.g., *sent1* & *sent2* for $s_1 + s_2$ in the official EntailmentWriter implementation).

We follow the official evaluation metrics of eQASC and eOBQA. The P@1 (Precision@1) measures the fraction of cases where the selected tree (topmost ranked) is correct. It is equivalent to the Overall AllCorrect score between the top-1 predicted one-step tree and the best-matching gold tree. The NDCG (Normalized Discounted Cumulative Gain) metric measures how well ranked the candidate trees are when ordered by the predicted scores. It reflects the model's ability to distinguish the validity of trees and rank the correct trees ahead of the incorrect ones.

F Main Experimental Environments

We deploy all models on a server with 500GB of memory and one 40G A100 GPU. Specifically, the configuration environment of the server is ubuntu 21.04 and our code mainly depends on

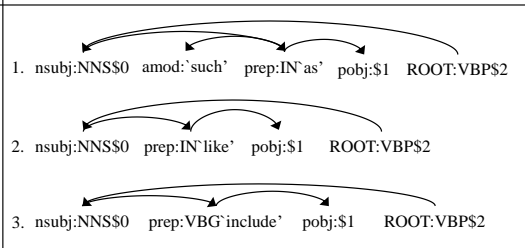
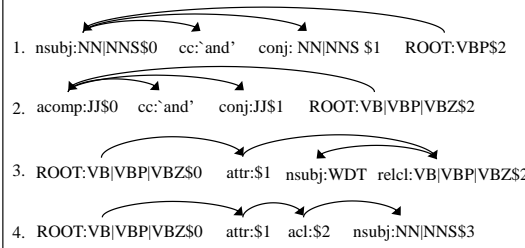
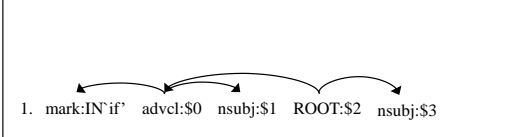
Type	Dependency Patterns	Example
Substitution	 <p>1. nsubj:NNSS\$0 amod:'such' prep:IN'as' obj:\$1 ROOT:VBP\$2</p> <p>2. nsubj:NNSS\$0 prep:IN'like' obj:\$1 ROOT:VBP\$2</p> <p>3. nsubj:NNSS\$0 prep:VBG'include' obj:\$1 ROOT:VBP\$2</p>	<p>Original Sentence: Slime molds like <i>Physarum polycephalum</i> are useful for studying cytoplasmic streaming.</p> <p>Premises: <i>Physarum polycephalum</i> are a slime mold. Slime molds are useful for studying cytoplasmic streaming.</p> <p>Paraphrased: The polycephalum is a slime mold. The slimy molds are useful for studying streaming.</p> <p>Conclusion: <i>Physarum polycephalum</i> are useful for studying cytoplasmic streaming.</p>
Conjunction	 <p>1. nsubj:NN NNS\$0 cc:'and' conj:NN NNS \$1 ROOT:VBP\$2</p> <p>2. acomp:JJ\$0 cc:'and' conj:JJ\$1 ROOT:VB VBP VBZ\$2</p> <p>3. ROOT:VB VBP VBZ\$0 attr:\$1 nsubj:WDT relcl:VB VBP VBZ\$2</p> <p>4. ROOT:VB VBP VBZ\$0 attr:\$1 acl:\$2 nsubj:NN NNS\$3</p>	<p>Original Sentence: <i>Aman</i> is an Indian anti-war movie directed by Mohan Kumar.</p> <p>Premises: <i>Aman</i> is an Indian anti-war movie. <i>Aman</i> is directed by Mohan Kumar.</p> <p>Paraphrased: <i>Aman</i> is a Indian film that is anti-war. Mohan Kumar was the director of <i>Aman</i>.</p> <p>Conclusion: <i>Aman</i> is an Indian anti-war movie directed by Mohan Kumar.</p>
If-then	 <p>1. mark:IN'if' advcl:\$0 nsubj:\$1 ROOT:\$2 nsubj:\$3</p>	<p>Original Sentence: If the rebels occupy territory, they make a gain.</p> <p>Premises: If the rebels occupy territory they make a gain. The rebels occupy territory</p> <p>Paraphrased: The rebels are able to make a gain if they hold on to territory. The territory was occupied by the rebels.</p> <p>Conclusion: The rebels make a gain.</p>

Table 9: The syntactic patterns used on data scraping and the training examples for deductive entailment modules. Pattern nodes are denoted as dep: POS‘lemma’ \$i, where dep contains the dependency relations of the matching token, POS contains the part-of-speech tags of the matching token, ‘lemma’ contains the lemmatized form of the matching token, and \$i indicates that a matching token and its subtree will be used as a match variable for rule-based rewriting. | means “or”.

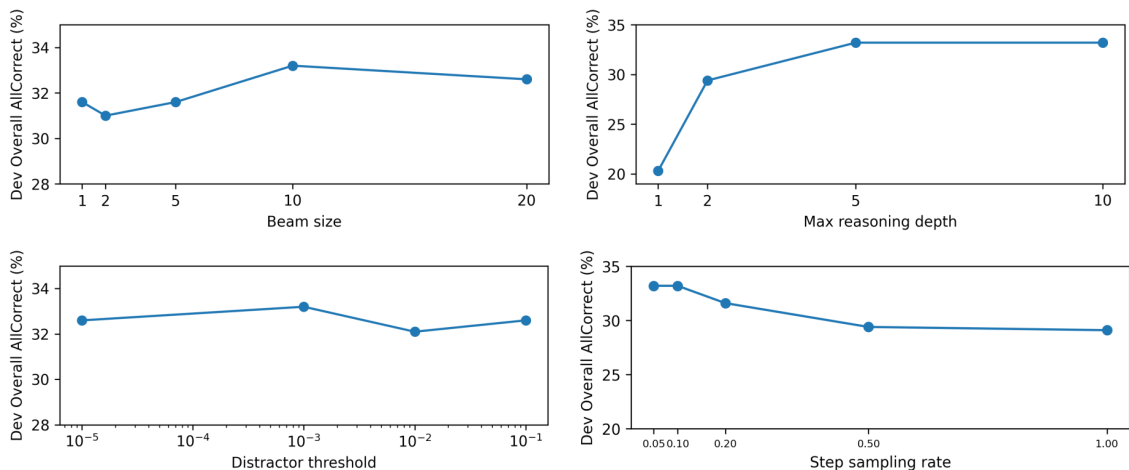


Figure 6: Hyperparameter analysis on the Task2 development split.

python 3.8.10 and PyTorch 1.7.1. We use the pre-trained language models from HuggingFace Transformers⁴. We use the Adafactor optimizer (Shazeer and Stern, 2018) implemented by HuggingFace Transformers.

⁴<https://github.com/huggingface/transformers>

G Discussion on the Automatic Evaluation

As discussed by Dalvi et al. (2021), the automatic entailment tree evaluation metrics might misjudge in some cases (e.g., tree structure variation) and still need to be improved. In fact, how to quantitatively evaluate a predicted tree remains a challenging problem. In the existing metric, the first step is the tree alignment algorithm (Dalvi et al.,

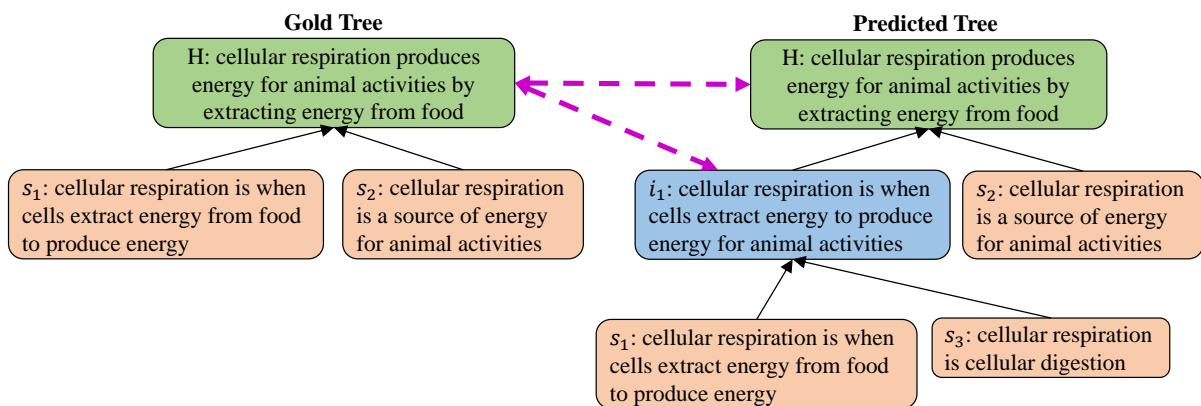


Figure 7: An example case illustrating the potential inaccuracy of the automatic evaluation metrics. In the predicted tree, the fact s_3 is a distractor and the step $s_1 + s_3 \rightarrow i_1$ is not a valid entailment. Following the official evaluation code, the nodes i_1, H in the predicted tree are aligned to the H, H in the gold tree, respectively (the dotted line). By comparing the aligned intermediate nodes (i_1 vs. H, H vs. H), the predicted tree achieves a Step F1 score of 0.0 and an Intermediate F1 score of 1.0. The Intermediate F1 score being 1.0 should have indicated that the predicted tree has perfect intermediate conclusions. However, the i_1 is not entailed by the s_1 and s_3 .

2021). The nodes in the predicted tree T_{pred} are aligned to the nodes in the gold tree T_{gold} for further comparison. Each non-leaf node i_{pred} of T_{pred} is aligned to the first non-leaf node i_{gold} where the Jaccard similarity of their respective leaf sentences is maximum. For any i_{pred} with zero Jaccard similarity to all gold nodes, it is aligned to a dummy gold node with a blank conclusion. In the official implementation, (1) each i_{gold} may correspond to more than one i_{pred} , while there is no penalty for duplication when calculating Intermediate F1; (2) the root node (the *given* hypothesis sentence which is *identical* in T_{pred} and T_{gold}) is trivially viewed as a normal intermediate node (the *novel generated* intermediate sentence). Because of these two reasons, the Intermediate F1 might achieve a high score (indicating the T_{pred} can draw correct intermediate conclusions from the premises), even when the Step F1/AllCorrect is relatively low (indicating the T_{pred} does not select the correct premises for the intermediate nodes). For example, the EntailmentWriter (T5-11B) for Task3 achieves an Intermediate F1 of 36.4% while the Step F1/AllCorrect is only 7.8%/2.9% (Dalvi et al., 2021). Figure 7 shows a specific case.

To alleviate the inaccuracy caused by the above reasons, we mainly use the more strict metrics (i.e., Leaves/Steps/Intermediates/Overall AllCorrect) for comparison. Furthermore, we adopt manual evaluation on the full trees and individual steps to make a more accurate comparison (Sec. 6.2).

H Case Study

We show some entailment trees generated by our METGEN-separated on the Task2 questions in Figure 8, 9, 10, 11. METGEN can generate a valid entailment tree which may have a different structure with the gold one (Figure 8). METGEN can handle medium-complexity questions, generate valid entailment trees and provide the reasoning types of steps (Figure 9 and 10). The questions which require more complex reasoning (e.g., the gold tree in Figure 11 requires 11 leaf facts and 8 entailment steps) remain challenging. Although the full tree generated by our method for such complex question can be not entirely correct, the intermediate conclusions (e.g., i_1, i_2 in Figure 11) are still reliable.

Question Q: Which phase of the Moon occurs after a waxing gibbous?

Answer A: full moon

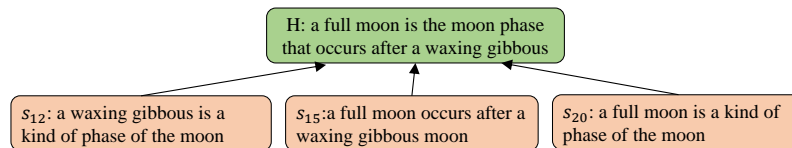
Hypothesis H: a full moon is the moon phase that occurs after a waxing gibbous

Facts S:

s1: state of matter is a property of matter and includes ordered values of solid / liquid / gas
 s2: the moon is earth 's moon
 s3: usually means most of the time
 s4: phase means state
 s5: occur is similar to appear
 s6: the moon orbits the earth
 s7: to be found in means to be contained in
 s8: revolving around something means orbiting that something
 s9: the moon orbiting the earth occurs once per month
 s10: a complete revolution / orbit of the moon around the earth takes 1 / one month
 s11: amount is a property of something and includes ordered values of none / least / little / some / half / much / many / most / all
 s12: a waxing gibbous is a kind of phase of the moon

s13: warm / becoming warm means heat is added
 s14: a phase change is when matter / a substance changes from one state of matter into another state of matter
 s15: a full moon occurs after a waxing gibbous moon
 s16: the moon reflects sunlight towards the earth
 s17: generally means usually
 s18: to happen means to occur
 s19: type of moon / kind of moon means moon phase
 s20: a full moon is a kind of phase of the moon
 s21: visible means able to be seen
 s22: motion / movement means moving / to move
 s23: the moon completes a lunar cycle over a period of 29 days
 s24: the moon rising occurs once per day
 s25: lunar phase is synonymous with moon phase

Gold Tree



Predicted Tree

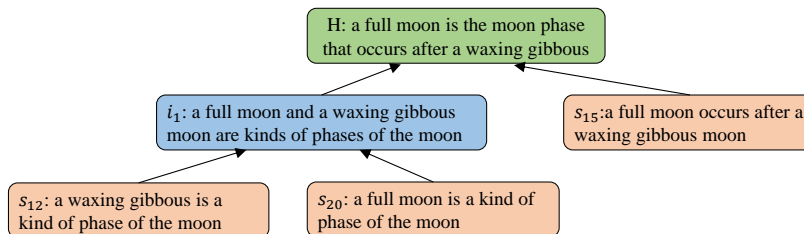


Figure 8: Case 1. The predicted entailment tree consists of two 2-premise steps, while the gold tree consists of one 3-premise step. Under the automatic evaluation metric, the predicted tree would be rated as invalid (Overall AllCorrect = 0), since the predicted steps do not match the gold step. However, the predicted tree should be valid because each step in the tree is a valid entailment (i.e., the 3-premise step can be decomposed into two valid 2-premise steps). It would be rated as valid under manual evaluation.

Question Q: According to the Periodic Table of the Elements, which set of elements has similar properties?

Answer A: He, Ne, Ar

Hypothesis H: he, ne, ar have similar properties

Facts S:

s1: cannot is the opposite of can

s2: helium / neon / argon belong to noble gases family , group 18 on the periodic table

s3: a periodic table is a kind of scientific model

s4: charge is a property of an object / a material / a substance and includes ordered values of negatively-charged / neutral / positively-charged

s5: including means containing

s6: similar means in common

s7: magnetism is a property of materials / objects and includes ordered values of nonmagnetic / magnetic

s8: a proton has a positive 1 electric charge

s9: the chemical symbol for helium is he

s10: similarity means the same property

s11: identical means copy

s12: chemical reactivity is a property of elements and includes ordered values of reactive / unreactive

s13: the chemical symbol for argon is ar

s14: amount is a property of something and includes ordered values of none / least / little / some / half / much / many / most / all

s15: an element is identified by its number of protons

s16: according to is similar to be determined by

s17: a group / family in the periodic table means a column in the periodic table

s18: characteristic means property

s19: the chemical symbol for neon is ne

s20: same means identical / equal in value / amount / number / quantity

s21: made of is similar to contains

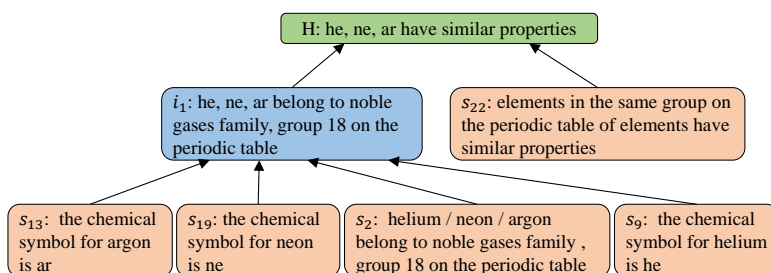
s22: elements in the same group on the periodic table of elements have similar properties

s23: in common is similar to the same

s24: positive charge is the opposite of negative charge

s25: identical is the opposite of different

Gold Tree



Predicted Tree

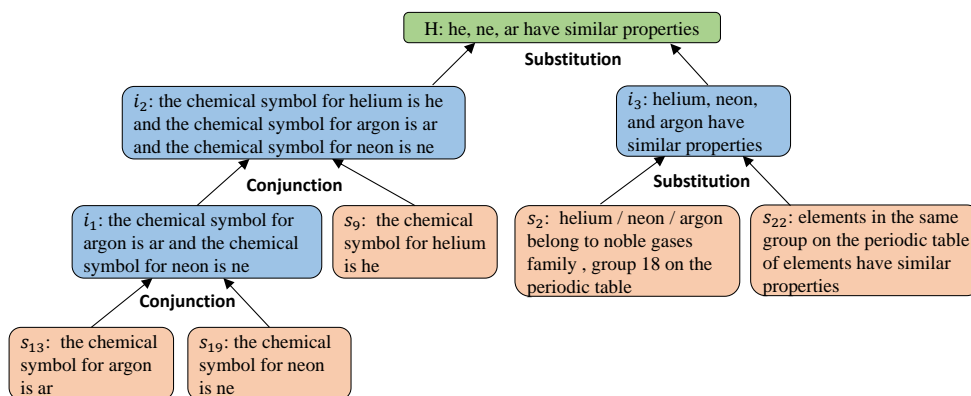


Figure 9: Case 2. Explaining the question and answer in this case requires 5 leaf facts from the given 25 facts. METGEN can select the correct facts, generate valid entailment trees, and provide the reasoning types of steps.

Question Q: Which trait would a cat most likely inherit from its parents?

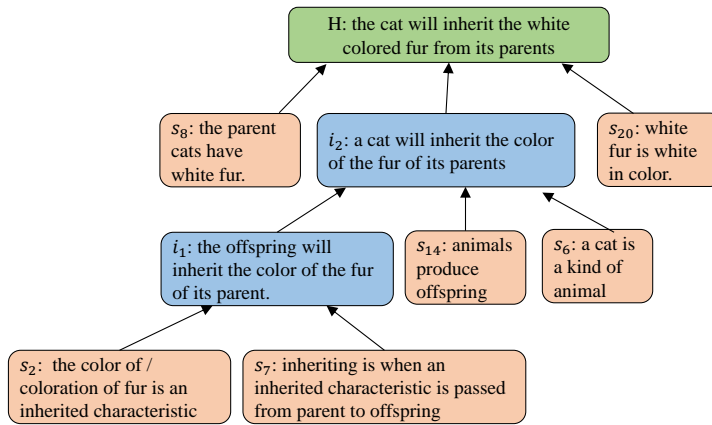
Answer A: having white fur

Hypothesis H: the cat will inherit the white colored fur from its parents

Facts S:

- s1: heredity is similar to inheritance
- s2: the color of / coloration of fur is an inherited characteristic
- s3: the mature / adult form of a kitten is called a cat
- s4: if an organism passes on its traits then future generations will have those traits
- s5: freckles are an inherited characteristic
- s6: a cat is a kind of animal
- s7: inheriting is when an inherited characteristic is passed from parent to offspring
- s8: the parent cats have white fur
- s9: coloration means a thing 's color
- s10: an animal knows how to do instinctive behaviors when it is born
- s11: color is a kind of physical / visual property
- s12: offspring receives half of the genes from each parent
- s13: a homozygous recessive organism contains only recessive genes
- s14: animals produce offspring
- s15: trait means property
- s16: the size of an organism is an inherited characteristic
- s17: genetic / hereditary means of genes / heredity
- s18: coat means fur coat
- s19: genes contains genetic information
- s20: white fur is white in color
- s21: coloration means a pattern of colors
- s22: hair / fur is a part of skin for protection / keeping warm
- s23: the shape of body parts is an inherited characteristic
- s24: hair is similar to fur
- s25: dna are a vehicle for passing genes from parent to offspring

Gold Tree



Predicted Tree

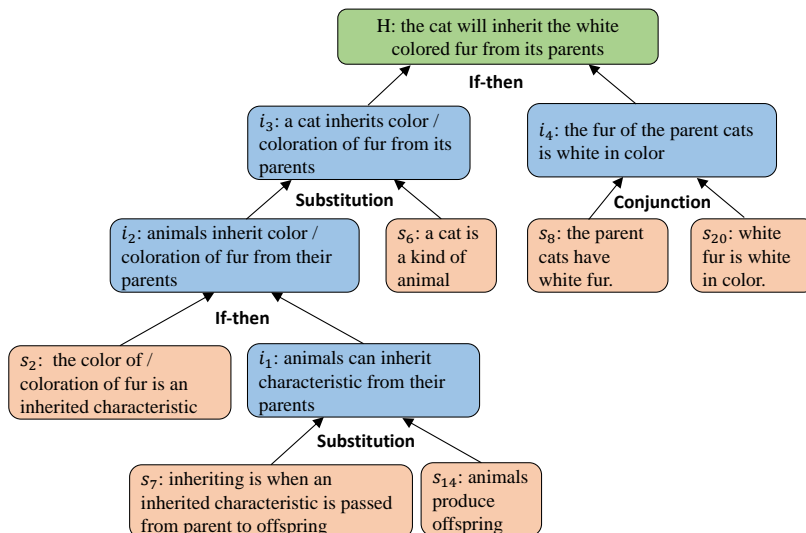


Figure 10: Case 3. METGEN can handle medium-complexity questions and provide the reasoning types of steps.

Question Q: A student is mixing a cup of hot chocolate with a spoon. How is the heat transferred between the hot chocolate and the part of the spoon that is in the hot chocolate?

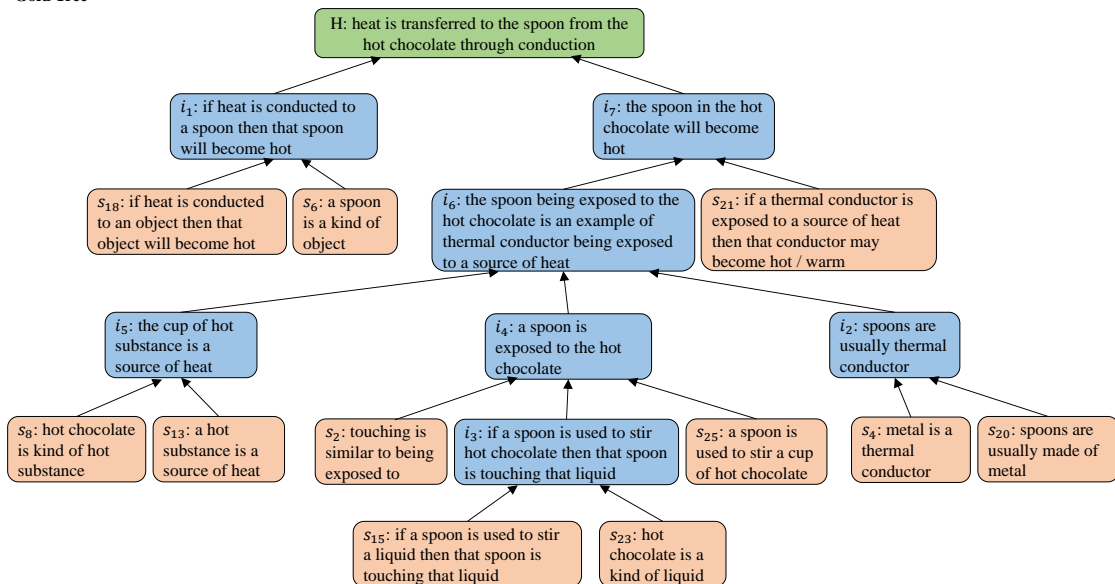
Answer A: Conduction transfers energy from the hot chocolate to the spoon.

Hypothesis H: heat is transferred to the spoon from the hot chocolate through conduction

Facts S:

- s1: static electricity is when electrons are exchanged between objects through friction
- s2: touching is similar to being exposed to
- s3: if something transfers energy to something else then that something else absorbs that energy
- s4: metal is a thermal conductor
- s5: friction occurs when two object 's surfaces move against each other
- s6: a spoon is a kind of object
- s7: if something is in something else , then that something is exposed to that something else
- s8: hot chocolate is kind of hot substance
- s9: conductivity is a measure of how easily electricity travels through a material
- s10: friction causes the temperature of an object to increase
- s11: the heat energy in the cooler object increases in thermal conduction
- s12: if one object absorbs more energy than another object , then the object will be warmer
- s13: a hot substance is a source of heat
- s14: conductivity is a kind of physical property
- s15: if a spoon is used to stir a liquid then that spoon is touching that liquid
- s16: thermal energy is a kind of energy
- s17: absorbing energy causes objects / materials / substances to heat
- s18: if heat is conducted to an object then that object will become hot
- s19: sending electricity through a conductor causes electricity / electric current to flow through that conductor
- s20: spoons are usually made of metal
- s21: if a thermal conductor is exposed to a source of heat then that conductor may become hot / warm
- s22: heat means heat energy
- s23: hot chocolate is a kind of liquid
- s24: heat energy is synonymous with thermal energy
- s25: a spoon is used to stir a cup of hot chocolate

Gold Tree



Predicted Tree

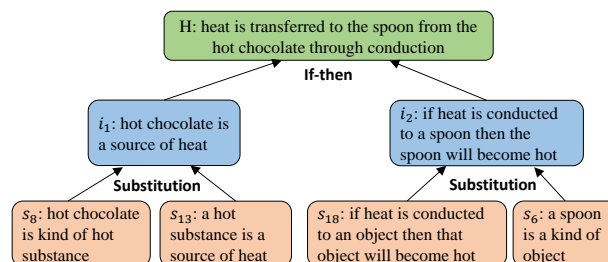


Figure 11: Case 4. The question requires more complex reasoning, where the gold tree contains 11 leaf facts and 8 entailment steps. Although the full tree generated by METGEN is not entirely correct, the intermediate conclusions i_1, i_2 are still reliable.