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);
²Beijing 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 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

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.
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 Mungnier, 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; Wiegreffe 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 (Latcinnik 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.
3 Task Definition

As shown in Figure 1, the inputs are a hypothesis \( H \) and some fact sentences \( S = \{s_1, s_2, \ldots, s_n\} \) (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):** \( S = S_{gold} + 15-20 \) 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...
<table>
<thead>
<tr>
<th>Type</th>
<th>Definition</th>
<th>Logical Regularity</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Substitution</td>
<td>performing taxonomic, merynomic or other kinds of reasoning</td>
<td></td>
<td>$s_1 \forall x \in X \ P(x)$; $s_2: \text{the mass of a planet causes the pull of gravity on that planet.}$</td>
</tr>
<tr>
<td></td>
<td>substitute one entity for another</td>
<td></td>
<td>$s_2: \text{earth is a kind of planet.}$</td>
</tr>
<tr>
<td>Conjunction</td>
<td>combining the details of both input sentences into a single output sentence</td>
<td></td>
<td>$s_1: \ P(x)$; $s_2: \ Q(x)$; $i_1: P(a)$; $i_1: \text{chemical splashing can cause harm to humans/to the eyes.}$</td>
</tr>
<tr>
<td></td>
<td>applying a conditional claim or a specific rule (one of the input sentences)</td>
<td></td>
<td>$s_1: \ P(x) \rightarrow Q(x)$; $s_2: \ P(x)$; $i_1: Q(x)$; $i_1: \text{chemical splashing sometimes occurs during experiments.}$</td>
</tr>
</tbody>
</table>

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 $i_1$ according to its reasoning types (denoted as $s_1 + s_2 \rightarrow i_1$). 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 $s_1$ (denoted as $i_1 - s_2 \rightarrow s_1$), where the generated premise $s_1$ 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 PurePattern (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: [CLS] $H$ [SEP] $s_1$ [SEP] ... [SEP] $s_n$ [SEP]. We obtain the contextualized representation $h$ for $H$ and $f_i$ for $s_i$ using the average contextualized representa-
Steps. We introduce feed forward networks $\text{FFN}_{\text{ded}}$ and $\text{FFN}_{\text{abd}}$ 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 $G_{\text{step}}$.

$$G_{\text{step}}(s_i, s_j) = \text{FFN}_{\text{ded}}([f_i, f_j]),$$

$$G_{\text{step}}(H, s_k) = \text{FFN}_{\text{abd}}([h, f_k]),$$

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 $\text{FFN}_{\text{fact}}$ as a learnable similarity function and determine the fact score by comparing it with the target,

$$G_{\text{fact}}(s_i) = \sigma(\text{FFN}_{\text{fact}}([h, f_i])),\tag{2}$$

where $\sigma$ 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:

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

where $\lambda$ is a learnable weight, $f_{\text{[CLS]}}$ is the representation of [CLS], $\text{FFN}_{\text{cls}}$ 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 $i_1$. If the predicted $i_1$ is correct, we replace $i_1$ in the state with $i_1$ to make new positive states. Otherwise, we replace $i_1$ with $i_2$ 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 $L_{\text{step}}, L_{\text{fact}},$ and $L_{\text{state}}$ to learn to rank the correct steps, facts, and states ahead of incorrect ones, respectively. Specifically, the loss for scoring steps is

$$L_{\text{step}} = \frac{1}{N_1} \sum_{(p^+, p^-)} \varphi(G_{\text{step}}(p^+), G_{\text{step}}(p^-), m_{\text{step}}),\tag{4}$$

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 $m_{\text{step}}$ is the margin for steps.

For facts, we have

$$L_{\text{fact}} = \frac{1}{N_2} \sum_{s_i^+, s_i^- \in S_{\text{gold}}} \varphi(G_{\text{fact}}(s_i^+), G_{\text{fact}}(s_i^-), m_{\text{fact}})$$

$$+ \frac{1}{N_3} \sum_{p^- \not\in S_{\text{gold}}} - \log(1 - G_{\text{fact}}(s^-)),\tag{5}$$

where $s_i^+$ is a fact which has smaller depth in the gold tree than $s_i^-$, $s^-$ is the distractor, $N_2$ is the number of $(s_i^+, s_i^-)$ pairs, $N_3$ is the number of distractors, and $m_{\text{fact}}$ 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

$$L_{\text{state}} = \varphi(G_{\text{state}}(R^+), G_{\text{state}}(R^-), m_{\text{state}}),\tag{6}$$

where $m_{\text{state}}$ is the margin for states.

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

$$L = L_{\text{step}} + L_{\text{fact}} + L_{\text{state}}.\tag{7}$$

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
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.

<table>
<thead>
<tr>
<th></th>
<th>Train</th>
<th>Dev</th>
<th>Test</th>
<th>All</th>
</tr>
</thead>
<tbody>
<tr>
<td>Questions/Trees</td>
<td>1,131</td>
<td>187</td>
<td>340</td>
<td>1,840</td>
</tr>
<tr>
<td>Entailment steps</td>
<td>4,175</td>
<td>597</td>
<td>1,109</td>
<td>5,881</td>
</tr>
</tbody>
</table>

Table 2: EntailmentBank statistics.

5.2 Baselines

We compare with the SOTA entailment tree generation method EntialmentWriter (Dalvi et al., 2021), which directly generates the linearized trees (e.g., $s_2 + s_5 \rightarrow i_1 : \text{eruptions block sunlight}; s_4 + i_1 \rightarrow H$) 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 EntialmentWriter to EntialmentWriter-Iter. EntialmentWriter-Iter iteratively generates a part of the linearized tree in one forward process (e.g., $s_2 + s_5 \rightarrow i_1 : \text{eruptions block sunlight;}$) and concatenates all parts to make the final tree. It completes the step selection and entails 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 al bert-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.
Validity of Individual Entailment Steps. We re-vite three students as experts to evaluate 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

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-separatted 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-prefix can outperform the EntialmentWriter (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-prefix also outperforms the EntialmentWriter-Iter (T5-large) by a large margin. For Task3, we note that all methods perform 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, EntialmentWriter (T5-large), EntialmentWriter-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

---

<table>
<thead>
<tr>
<th>Task</th>
<th>Method</th>
<th>( n_{para} )</th>
<th>Leaves</th>
<th>Steps</th>
<th>Intermediates</th>
<th>Overall</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>F1 AllCorrect</td>
<td>F1</td>
<td>F1 AllCorrect</td>
<td>F1 AllCorrect</td>
<td>F1 AllCorrect</td>
</tr>
<tr>
<td>Task1 (no-distractor)</td>
<td>EntialmentWriter (T5-11B)†</td>
<td>11.00</td>
<td>99.0</td>
<td>89.4</td>
<td>51.5</td>
<td>38.2</td>
</tr>
<tr>
<td></td>
<td>EntialmentWriter (T5-large)</td>
<td>0.77</td>
<td>98.4</td>
<td>84.1</td>
<td>50.0</td>
<td>38.5</td>
</tr>
<tr>
<td></td>
<td>EntialmentWriter-Iter (T5-large)</td>
<td>0.77</td>
<td>99.8</td>
<td>97.6</td>
<td>51.6</td>
<td>38.5</td>
</tr>
<tr>
<td></td>
<td>METGEN-separated (Ours)</td>
<td>0.22±0.77</td>
<td>100.0</td>
<td>100.0</td>
<td>57.9</td>
<td>42.1</td>
</tr>
<tr>
<td></td>
<td>METGEN-prefixixed (Ours)</td>
<td>0.22±0.77</td>
<td>100.0</td>
<td>100.0</td>
<td>57.4</td>
<td>41.9</td>
</tr>
<tr>
<td>Task2 (distractor)</td>
<td>EntialmentWriter (T5-11B)†</td>
<td>11.00</td>
<td>89.1</td>
<td>48.8</td>
<td>41.4</td>
<td>27.7</td>
</tr>
<tr>
<td></td>
<td>EntialmentWriter (T5-large)</td>
<td>0.77</td>
<td>83.2</td>
<td>35.0</td>
<td>39.5</td>
<td>24.7</td>
</tr>
<tr>
<td></td>
<td>EntialmentWriter-Iter (T5-large)</td>
<td>0.77</td>
<td>85.2</td>
<td>40.9</td>
<td>38.9</td>
<td>26.8</td>
</tr>
<tr>
<td></td>
<td>METGEN-separated (Ours)</td>
<td>0.22±0.77</td>
<td>83.7</td>
<td>48.6</td>
<td>41.7</td>
<td>30.4</td>
</tr>
<tr>
<td></td>
<td>METGEN-prefixixed (Ours)</td>
<td>0.22±0.77</td>
<td>82.7</td>
<td>46.1</td>
<td>41.3</td>
<td>29.6</td>
</tr>
<tr>
<td>Task3 (full-corpus)</td>
<td>EntialmentWriter (T5-11B)†</td>
<td>11.00</td>
<td>39.7</td>
<td>3.8</td>
<td>7.8</td>
<td>2.9</td>
</tr>
<tr>
<td></td>
<td>EntialmentWriter (T5-large)</td>
<td>0.77</td>
<td>30.9</td>
<td>3.8</td>
<td>4.4</td>
<td>1.2</td>
</tr>
<tr>
<td></td>
<td>EntialmentWriter-Iter (T5-large)</td>
<td>0.77</td>
<td>32.4</td>
<td>1.8</td>
<td>4.4</td>
<td>1.5</td>
</tr>
<tr>
<td></td>
<td>METGEN-separated (Ours)</td>
<td>0.22±0.77</td>
<td>34.8</td>
<td>8.7</td>
<td>9.8</td>
<td>8.6</td>
</tr>
<tr>
<td></td>
<td>METGEN-prefixixed (Ours)</td>
<td>0.22±0.77</td>
<td>34.8</td>
<td>8.7</td>
<td>9.8</td>
<td>8.6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>Task1</th>
<th>Task2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>35</td>
<td>35</td>
</tr>
<tr>
<td>EntialmentWriter (T5-large)</td>
<td>46</td>
<td>47</td>
</tr>
<tr>
<td>EntialmentWriter-Iter (T5-large)</td>
<td>21</td>
<td>25</td>
</tr>
<tr>
<td>METGEN-prefixixed (Ours)</td>
<td>26</td>
<td>35</td>
</tr>
<tr>
<td>METGEN (Ours)</td>
<td>53</td>
<td>27</td>
</tr>
</tbody>
</table>

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.
We report the AllCorrect scores on the 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 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 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.

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

<table>
<thead>
<tr>
<th>Method</th>
<th>eQASC P@1</th>
<th>eQASC NDCG</th>
<th>eOBQA P@1</th>
<th>eOBQA NDCG</th>
</tr>
</thead>
<tbody>
<tr>
<td>EntailmentWriter (T5-large)</td>
<td>52.48</td>
<td>73.14</td>
<td>69.07</td>
<td>89.05</td>
</tr>
<tr>
<td>EntailmentWriter-Iter (T5-large)</td>
<td>52.56</td>
<td>73.28</td>
<td>72.15</td>
<td>90.19</td>
</tr>
<tr>
<td>METGEN-prefixed (Ours)</td>
<td>55.81</td>
<td>74.19</td>
<td>74.89</td>
<td>90.50</td>
</tr>
</tbody>
</table>

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


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 Spacy\(^3\) 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.

<table>
<thead>
<tr>
<th>Split</th>
<th>Sub.</th>
<th>Conj.</th>
<th>If-then</th>
<th>All</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train-manual</td>
<td>211</td>
<td>105</td>
<td>84</td>
<td>400</td>
</tr>
<tr>
<td>Train-pseudo</td>
<td>2,441</td>
<td>812</td>
<td>535</td>
<td>3,788</td>
</tr>
<tr>
<td>Dev-manual</td>
<td>153</td>
<td>71</td>
<td>51</td>
<td>275</td>
</tr>
</tbody>
</table>

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 step \( s_2 + s_5 \rightarrow i_1 \), and one positive abductive step \( H \rightarrow s_4 \rightarrow i_1 \). 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 $\theta$, step sampling rate $\tau$

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: $R_{beam} \leftarrow \{R_{init}\}$, $R \leftarrow R_{beam}$  
5: // Reasoning with beam search  
6: while the depth does not reach $D$ do  
7: // Select promising steps  
8: for $R \in R_{beam}$ do  
9: // Construct initial reasoning state  
10: for $p \in \text{steps of } R \text{ with top } \tau\% \text{ step score}$ 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 obtain a novel intermediate fact $i$  
14: construct a new state $R_{new}$ with the step $p$ and the fact $i$  
15: $R_{beam} \leftarrow R_{beam} \cup \{R_{new}\}$  
16: end for  
17: end for  
18: // Verify and select states  
19: $R_{beam} \leftarrow K$ states with the highest state scores from $R_{beam}'$  
20: $R \leftarrow R \cup R_{beam}$  
21: end while  
22: // Construct the entailment tree  
23: for $R \in R$ do  
24: Align the target of $R$ to the most similar fact sentence of $R$ to make a tree $T$  
25: end for  
26: Select the tree $T$ with highest score  
27: Return The entailment tree $T$

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., $\text{sent1} \& \text{sent2}$ for $s_1 + s_2$ in the official EntailmentWriter implementation).

We follow the official evaluation metrics of eQASC and eOBQA. The $\text{P@1}$ (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
Table 9: The syntactic patterns used on data scraping and the training examples for deductive entailment modules. Pattern nodes are donated 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 Transformers\(^4\). We use the Adafactor optimizer (Shazeer and Stern, 2018) implemented by 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., 2021).
S1 cellular respiration is when cells extract energy from food to produce energy.

S2 cellular respiration is a source of energy for animal activities.

S3 cellular respiration is cellular digestion.

I1: cellular respiration is when cells extract energy to produce energy for cell activities.

H cellular respiration produces energy for animal activities by extracting energy from food.

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.

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.

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).
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

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

s12: a waxing gibbous is a kind of phase of the moon

s15: a full moon occurs after a waxing gibbous moon

s20: a full moon is a kind of phase of the moon

Predicted Tree

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

s1: a full moon and a waxing gibbous moon are kinds of phases of the moon

s12: a waxing gibbous is a kind of phase of the moon

s20: a full moon is a kind of phase of the moon

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**

![Gold Tree Diagram](image)

**Predicted Tree**

![Predicted Tree Diagram](image)

---

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:
1. heredity is similar to inheritance
2. the color of fur is an inherited characteristic
3. the mature / adult form of a kitten is called a cat
4. if an organism passes on its traits then future generations will have those traits
5. freckles are an inherited characteristic
6. a cat is a kind of animal
7. inheriting is when an inherited characteristic is passed from parent to offspring
8. the parent cats have white fur
9. coloration means a thing’s color
10. an animal knows how to do instinctive behaviors when it is born
11. color is a kind of physical / visual property
12. offspring receives half of the genes from each parent

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 objects’ 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 a 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 an object is exposed to a source of heat then that object will absorb energy.
- 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.
- 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.