Entailment Tree Explanations via Iterative Retrieval-Generation Reasoner

Danilo Ribeiro[1,2,*] **Shen Wang**[2], **Xiaofei Ma**[2], **Rui Dong**[2], **Xiaokai Wei**[2], **Henry Zhu**[2], **Xinchi Chen**[2], **Zhiheng Huang**[2], **Peng Xu**[2], **Andrew Arnold**[2], **Dan Roth**[2],
[1]Northwestern University, [2]AWS AI Labs
{dnribeiro}@u.northwestern.edu,
{shenwa, xiaofeim, ruidong, xiaokaiw, henghui}@amazon.com,
{xcc, zhiheng, pengx, anarnld, drot}@amazon.com

Abstract

Large language models have achieved high performance on various question answering (QA) benchmarks, but the explainability of their output remains elusive. Structured explanations, called *entailment trees*, were recently suggested as a way to explain and inspect a QA system's answer. In order to better generate such entailment trees, we propose an architecture called *Iterative Retrieval-Generation Reasoner* (IRGR). Our model is able to explain a given hypothesis by systematically generating a step-by-step explanation from textual premises. The IRGR model iteratively searches for suitable premises, constructing a single entailment step at a time. Contrary to previous approaches, our method combines generation steps and retrieval of premises, allowing the model to leverage intermediate conclusions, and mitigating the input size limit of baseline encoder-decoder models. We conduct experiments using the EntailmentBank dataset, where we outperform existing benchmarks on premise retrieval and entailment tree generation, with around 300% gain in overall correctness.

1 Introduction

Large neural network models have successfully been applied to different natural language tasks, achieving state-of-the-art results in many natural language benchmarks. Despite this success, these results came with the expense of AI systems becoming less interpretable (Jain and Wallace, 2019; Rajani et al., 2019a).

With the desire to make the output of such models less opaque, we propose a question answering (QA) system that is able to explain their decisions not only by retrieving supporting textual evidence (rationales), but by showing how the answer to a question can be systematically proven from simpler textual premises (natural language reasoning).

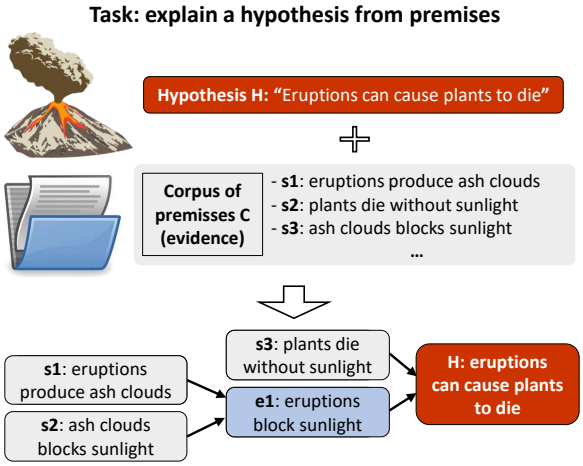

Figure 1: Task has as input a hypothesis H (e.g. an answer to a question) and a corpus of premises C (simple textual evidences), the goal is to generate an *entailment tree* that explains the hypothesis H by using premises from C.

These explanations are represented using *entailment trees*, as depicted in Figure 1. First introduced by Dalvi et al. (2021), entailment trees represents a chain of reasoning that shows *how* a hypothesis (or an answer to a question) can be explained from simpler textual evidence. In comparison, other explanation approaches such as retrieval of passages (rationales) (DeYoung et al., 2020) or multi-hop reasoning (chaining) (Jhamtani and Clark, 2020) are less expressive than entailment trees, which are comprised of multi-premise textual entailment steps.

In order to generate such entailment trees, previous works (Tafjord et al., 2021; Dalvi et al., 2021; Bostrom et al., 2021) have used encoder-decoder models that takes as input a small set of retrieved premises and output a linearized representation of the entailment tree. Such models are limited by (1) the language model's fixed input size, and they may construct incorrect proofs when the retrieval module cannot fetch all relevant premises at once

*Work done during an internship at the AWS AI. Code and model checkpoints are publicly available at https://github.com/amazon-research/irgr.

Findings of the Association for Computational Linguistics: NAACL 2022, pages 465 - 475
July 10-15, 2022

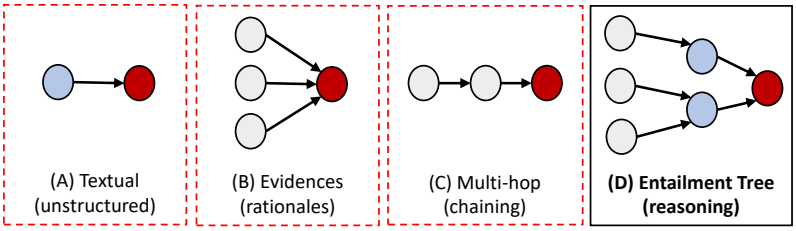

Figure 2: Comparison among different natural language explanation approaches. The images show (A) plain textual explanations (B) retrieval of evidence passages (C) multi-hop explanations (D) entailment trees. The approach in (D) allows for more detailed inspection of the reasoning behind an explanation. Nodes in gray are retrieved from a corpus, nodes in blue are generated, and the red node is the hypothesis or answer that is being explained.

(2) such approaches do not leverage the partially generated entailment trees. In contrast, we propose ***I**terative **R**etrieval-**G**eneration **R**easoner* (IRGR), a novel architecture that iteratively searches for suitable premises, constructing a single entailment step at a time. At every generation step, the model searches for a distinct set of premises that will support the generation of a single step, therefore mitigating the language model's input size limit and improving generation correctness.

Our contributions are two-fold. First, we design a retrieval method that is able to better identify premises needed to generate a chain of reasoning, which explains a given hypothesis. Our retrieval method outperforms previous baselines by 48.3%, while allowing for a dynamic set of premises to be retrieved. Secondly, we propose an iterative retrieval-generation architecture that constructs partial proofs and augments the retrieval probes using intermediate generation results. We show that integrating the retrieval module with iterative generation can significantly improve explanations. Our proposed approach achieves new state-of-the-art result on entailment tree generation with over 306% better results on the All-Correct metric (strict comparison with golden data), while using a model with one order of magnitude fewer parameters.

2 Related Work

Traditionally, natural language processing (NLP) frameworks were based on white-box methods such as rule-based systems (Allen, 1988; Ribeiro et al., 2019; Ribeiro and Forbus, 2021) and decision trees (Boros et al., 2017), which were inherently inspectable (Danilevsky et al., 2020). More recently, large deep learning language models (black-box methods) have gained popularity (Song et al., 2020; Raffel et al., 2020), but their improvements in result quality came with a cost: the system's outputs lack explainability and inspectability.

There have been many attempts to mitigate this issue, including input perturbation (Ribeiro et al., 2018) and premises selection (DeYoung et al., 2020). One promising explanation approach is to combine the model's output with a human-interpretable explanation. For instance, Camburu et al. (2018) introduced the concept of natural language explanation in their e-SNLI dataset while Rajani et al. (2019b) expanded this idea to commonsense explanations. Jhamtani and Clark (2020) further explored the notion of explanation in multi-hop QA, where explanations contain a *chain of reasoning*, instead of simple textual explanations. Different from these explanation approaches, our work generates explanations in the form of *entailment trees*, introduced by Dalvi et al. (2021), which are composed of multi-premise textual entailment steps. Entailment trees are more detailed explanations, making it easier to inspect the reasoning behind the model's answer. Figure 2 shows a diagram compering different natural language explanation methods according to their structure and use of textual evidence.

The first approach used to generate entailment trees was based on the *EntailmentWriter* model by Tafjord et al. (2021). However, their approach is limited by the input size of the encoder-decoder language models, where a fixed set of supporting facts is used to generate an explanation. Instead, our model iteratively fetches a set of premises using dense retrieval conditioned on previous entailment steps, allowing for more precise explanations.

Our work is also related to some recent approaches that combine retrieval and neural networks for QA tasks (Karpukhin et al., 2020; Guu et al., 2020). The work of Lewis et al. (2020) combined dense retrieval with encoder-decoder models, where a different set of passages were retrieved

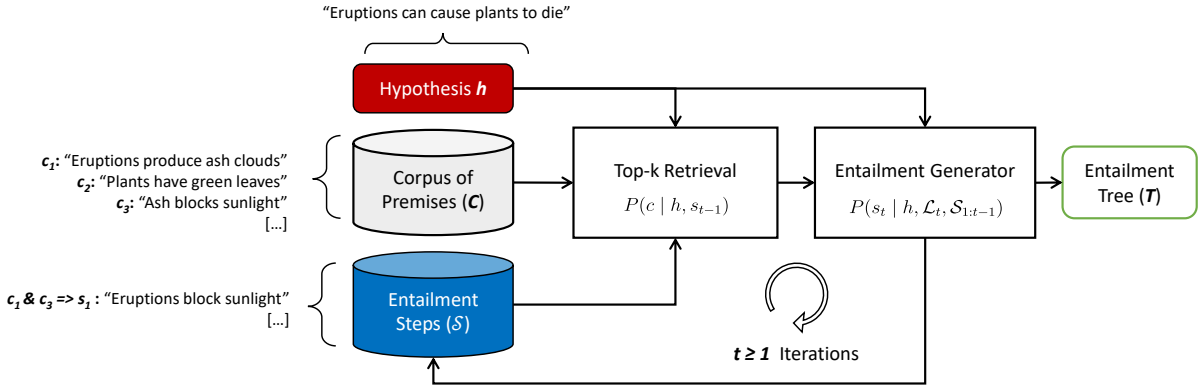

Figure 3: IRGR is composed of two modules, IRGR-retriever and IRGR-generator. The IRGR-retriever iteratively fetches a set of premises from a corpus C in order to generate an entailment tree (structured explanation for a given hypothesis). The IRGR-generator computes a single entailment step at a time, and the intermediate generated steps are stored and used for subsequent retrieval and generation.

for each generated character. Conditioning the retrieval of a passage on previously retrieved passages has been explored in the context of multi-hop QA (Zhao et al., 2021; Xiong et al., 2021), and multi-hop explanations (Valentino et al., 2021; Cartuyvels et al., 2020). However, these approaches either are not used to generate explanations or do not use inferred intermediate reasoning steps to retrieve premises.

3 Approach

3.1 Problem Definition

The problem input consists of a **corpus of premises** C (simple textual statements) and a **hypothesis** h. The objective is to generate an **entailment tree** T that explains the hypothesis h by using a subset of the premises in C as building blocks. Entailment trees are represented as a tuple $T = (h, \mathcal{L}, \mathcal{E}, \mathcal{S})$, where **leaf nodes** $l_i \in \mathcal{L}$ are retrieved from the corpus (i.e. $\mathcal{L} \subseteq C$), **internal tree nodes** $e_i \in \mathcal{E}$ are intermediate conclusions (new sentences not present in corpus C, note that intermediate conclusions are generated by the model), and $s_i \in \mathcal{S}$ is a list of **entailment steps** that can explain the hypothesis h, which is always the tree root and the final conclusion. An illustration of the problem and expected entailment tree can be found in Figure 1.

Each entailment step s_i represents one inference step from a conjunction of premises to a conclusion. For instance, "$l_1 \wedge l_2 \Rightarrow e_1$" or "$l_1 \wedge l_2 \wedge e_1 \Rightarrow h$" could be valid entailment steps in $\mathcal{S}$. Note that the root of T is always the node representing h.

3.2 Architecture

Our approach, which we call ***I**terative **R**etrieval-**G**eneration **R**easoner* (IRGR), consists of two modules, the IRGR-retriever and the IRGR-generator. The initial input to the model is the hypothesis h and the corpus of premises C. The generation process is performed through multiple iterations. At each iteration step $t \geq 1$ the IRGR-retriever selects a subset of premises from the corpus $\mathcal{L}_t \subseteq C$. The IRGR-generator outputs one entailment step s_t per iteration until the entailment tree T is fully generated. Given $\mathcal{S}_{1:t-1} = (s_1, \ldots, s_{t-1})$ as the list of the entailment steps generated up to the previous iterations $t-1$, the generator takes as input $\mathcal{L}_t$ and $\mathcal{S}_{1:t-1}$ and produces the next entailment step s_t. The generation stops when the entailment step's conclusion is the hypothesis h, i.e., the proof is finished. Formally, the t-th iteration of the generation process is defined as:

$$\mathcal{L}_t = \text{IRGR-retriever}(h, s_{t-1}) \quad (1)$$

$$s_t = \text{IRGR-generator}(h, \mathcal{L}_t, \mathcal{S}_{1:t-1}) \quad (2)$$

The IRGR-retriever searches over the premises in corpus C using *dense passage retrieval* (Karpukhin et al., 2020). Meanwhile, the IRGR-generator was implemented using T5, the Text-to-Text Transformer (Raffel et al., 2020), while any other sequence-to-sequence language model could also be used. An overview of the model can be seen in Figure 3.

3.2.1 IRGR-retriever

The IRGR-retriever module proposed in this work aims to retrieve premises from the corpus C. In existing baseline models the retrieval is done in one single step, fetching a fixed set of premises before generation (Tafjord et al., 2021). However, the generation of entailment trees requires a different set of leaves for each entailment step. To address this issue, our IRGR-retriever fetches k_t premises from C to produce $\mathcal{L}_t$ at iteration step t. Note that the size of C can be very large ($k_t << |C|$). The value k_t is chosen such that the size of $\mathcal{L}_t$ is small enough to fit in the context of a language model while still being large enough to fetch as many premises as possible (in our experiments, the value k_t is always below 25). We define the retrieval probability of a premise $c \in C$ at a certain iteration step t as:

$$P(c \mid h, s_{t-1}) = \frac{exp(\langle \mathbf{c}, \mathbf{q_t} \rangle)}{\sum_{c' \in C} exp(\langle \mathbf{c}', \mathbf{q_t} \rangle)} \quad (3)$$

Where ϕ is the sentence encoder function used to encode both premises and hypothesis, transforming the input text into a dense vector representation in $\mathbb{R}^M$. The values $\mathbf{c} = \phi(c)$, $\mathbf{c}' = \phi(c')$ and $\mathbf{q_t} = \phi(h, s_{t-1})$ are dense M-dimensional vectors. The operator $\langle . \rangle$ represents the inner product between two vectors.

The encoder follows the Siamese Network architecture from Reimers and Gurevych (2019). We select a set of N positive and negative examples in the form of query-value pairs $\{(q_j, c_j)\}_{j=1}^{N}$ for training. Queries q_j encode both the hypothesis h and previous entailment step s_{t-1} by concatenating their textual values. The positive examples are taken from the golden entailment trees, where $c_j \in \mathcal{L}$. For negative examples, we pair a query q_j with either random premises from C or premises retrieved using the not fine-tuned version of the encoder (hard negatives).

We define $\hat{y}_j$ as the label given to the training example (q_j, c_j). For positive examples, the label $\hat{y}_j$ depends on how close the leaf node $l_i \in \mathcal{L}$ is to the intermediate step s_{t-1} in the golden tree:

$$\hat{y}_j = \begin{cases} 0, & \text{if negative} \\ \lambda, & \text{if positive and } l_i \notin \texttt{ant}(s_{t-1}) \\ 1, & \text{if positive and } l_i \in \texttt{ant}(s_{t-1}) \end{cases} \quad (4)$$

Where $\texttt{ant}(s_{t-1})$ denotes the set of antecedents in some entailment step s_{t-1}, and $l_i \in \texttt{ant}(s_{t-1})$ means that the leaf node l_i is used in the entailment step s_{t-1}. The value $\lambda \in [0 : 1]$ is used to give lower priority to leaf nodes not relevant to the current entailment step ($\lambda = 0.75$ gave the best results in our experiments). Finally, we fine-tune the encoder ϕ by minimizing the following loss function L_ϕ, where N is the number of training examples:

$$L_\phi = \frac{1}{N} \sum_{j=1}^{N} \left(\hat{y}_j - \frac{\langle \phi(q_j), \phi(c_j) \rangle}{\|\phi(q_j)\| \|\phi(c_j)\|} \right) \quad (5)$$

One significant challenge is that for the first generation step, when $t = 1$, the list of previously generated entailment steps $\mathcal{S}_0$ is empty. The retrieval only depends on h, meaning $\mathcal{L}_1 = \text{IRGR-retriever}(h)$. It is more difficult to retrieve premises for leaf nodes when the entailment tree's

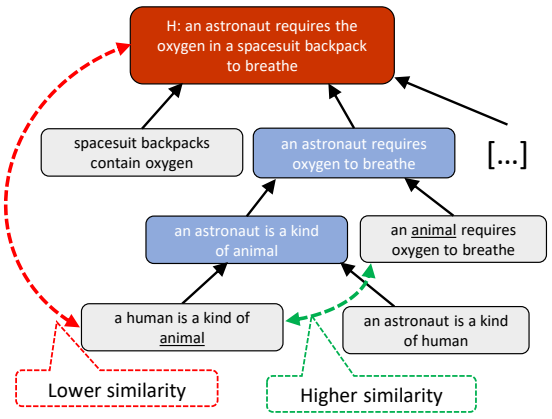

Figure 4: Entailment tree example showing how some retrieval examples are challenging. Leaf sentences are not always directly related to hypothesis.

Algorithm 1: Conditional Retrieval

Data: hypothesis h, corpus C, number of retrieved premises k_0
Result: retrieved premises $\mathcal{L}_0$
Parameter : conditioning factor ω

```
Q ← {h} ;   /* set of queries */
L_0 ← {};
for i ← 0 to k_0 do
    C' ← {c ∈ C : c ∉ L_0};
    if i ≥ ω then
        l_i ← arg max_(c∈C') P(c | Q);
        Q ← Q ∪ {l_i};
    else
        l_i ← arg max_(c∈C') P(c | h);
    end
    L_0 ← L_0 ∪ {l_i};
end
```

depth is large since the leaf nodes have low syntactic and semantic similarity with the hypothesis h. For instance, the example in Figure 4 shows how leaf node "*a human is a kind of animal*" (depth 3) is needed to build the entailment tree, but is syntactically distinct to hypothesis "*an astronaut requires the oxygen in a space suit backpack to breath*".

To mitigate this problem, we perform a conditional retrieval on the first step, where the retrieval module uses partial results as part of the query, as depicted in Algorithm 1. This algorithm assumes that leaf nodes (premises) further from the root node (hypothesis) are more similar to each other than to the root node itself. The parameter ω (value $\omega = 15$ yields the best results on development set) is used to split the search, such that part of the retrieved premises only depend on the hypothesis h. In contrast, the other parts of the retrieved premises depend on the hypothesis and previously retrieved premises stored in the set Q.

3.2.2 IRGR-generator

The IRGR-generator consists of a sequence-to-sequence model that outputs one single entailment step given a context. One key aspect of this module is encoding the input and output as plain text.

Encoding Entailment Trees: Entailment trees are linearized from leaves to root. Each leaf node $l_i \in \mathcal{L}$, intermediate node $e_i \in \mathcal{E}$ and root node h are encoded with the symbols "`sent`", "`int`" and "`hypothesis`", respectively. The entailment steps represent conjunctions with "`&`" and entailment with the symbol "`->`". For instance, the entailment tree depicted in Figure 1 can be represented as:

```
"sent1 & sent2 -> int1:
Eruptions block sunlight;
sent3 & int1 -> hypothesis;"
```

Note that the text of intermediate nodes have to be explicitly represented, since they are not part of the corpus C. Ultimately, they have to be generated by the model. The input to the model encodes the hypothesis h and retrieved premises $l_i^t \in \mathcal{L}_t$, which are straightforwardly encoded as follows:

```
"hypothesis: Eruptions can
cause plants to die;
sent1: eruptions emit lava
sent2: eruptions produce ash
clouds
sent3: [...];"
```

Method	R@25	All-Correct
Okapi BM25	45.01	22.35
EntailmentWriter	59.76	34.70
IRGR-retriever (sing.)	64.41	40.29
IRGR-retriever (cond.)	**68.28**	44.70
IRGR-retriever*	-	**51.47**

Table 1: Retrieval results. The methods with * retrieves more than 25 premises from corpus.

When a leaf sentence l_t^i is used in the entailment step, it is removed from the context for the following step, and the premise `sent` identifier is not used to encode new retrieved premises. A detailed example of input and output for the IRGR-generator module is shown in Appendix A.3.

4 Experiments

4.1 Datasets

We evaluate our architecture on the ENTAILMENTBANK dataset (Dalvi et al., 2021), which is comprised of 1,840 questions (each associated with a hypothesis h_i and entailment tree T_i) with 5,881 total entailment steps. On average, each entailment tree has 7.6 nodes (including leaf, intermediate, and root) and around 3.2 entailment steps. The corpus of premises C has around 11K entries and is derived from the WorldTree V2 (Xie et al., 2020) in addition to a few premises created by the EntailmentBank annotators.

4.2 Evaluation Metrics

4.2.1 Retrieval

We evaluate our IRGR-retriever module using two different sets of metrics. The first one is "Recall at k" (R@k), a standard evaluation metric for information retrieval. The second metric "All-Correct" is more strict, and the results are only considered correct if all the premises from the golden tree are retrieved. Formally, given the retrieved premises $\mathcal{L}$ and the set of gold premises $\mathcal{L}^*$, the metrics R@k is given by $|\mathcal{L} \cap \mathcal{L}^*| \ / \ |\mathcal{L}^*|$, and the metric All-Correct is 1 if $|\{x \in \mathcal{L}^* : x \notin \mathcal{L}\}| = 0$, or 0 otherwise. For our experiments, we consider k = 25 since that's roughly the maximum number of sentences that can fit in the T5 language model's 512 tokens context.

Method	Leaves		Steps		Intermediates		Overall
	F1	All-Cor.	F1	All-Cor.	F1	All-Cor.	All-Cor.
EntailmentWriter	39.7	3.8	7.8	2.9	36.4	13.2	2.9
IRGR	45.6	**12.1**	**16.3**	**11.8**	**38.8**	**36.5**	**11.8**
- w/o iter.	**46.6**	10.0	11.3	8.2	36.3	35.6	8.2
- w/o iter. & cond.	36.1	3.8	6.0	3.2	27.6	14.7	3.2

Table 2: Entailment tree scores for baseline methods and IRGR, along four different dimensions (test set). F1 scores measure predicted/gold overlap, while All-Correct scores are 1 when all the predictions for a tree are correct, 0 otherwise.

4.2.2 Entailment Tree Generation

We adopt the evaluation metrics defined by Dalvi et al. (2021), which compares the generated entailment tree $T = (h, \mathcal{L}, \mathcal{E}, \mathcal{S})$ with the golden entailment tree $T^* = (h, \mathcal{L}^*, \mathcal{E}^*, \mathcal{S}^*)$. The metrics evaluate the correctness along four dimensions: (1) leaf nodes, (2) entailment steps, (3) generated intermediate nodes, (4) and overall correctness. The first step is to align the nodes from T with the nodes from T^* by Jaccard similarity (alignment algorithm and further details of metrics described in Appendix A.2). This method tries to ignore variations between predicted and gold trees that do not change the semantics of the output. The four metric dimensions are described below as follows. For each metric with F1 value, there is also a strict "All-Correct" metric that is equal to 1 when F1 = 1 and 0 otherwise.

Leaf (F1, All-Correct): Tests if the predicted and golden leaf nodes match. This metric compares the sets $\mathcal{L}$ and $\mathcal{L}^*$ using F1 score.

Steps (F1, All-Correct): Tests if the predicted entailment steps follow the correct structure. Given that $s_i \in \mathcal{S}$ matches $s_j \in \mathcal{S}^*$ according to the alignment algorithm, tests if the premises of s_i are equal to those of s_j, and computes the F1 score according to the set of all matched steps.

Intermediates (F1, All-Correct): Tests if the sentences of the generated intermediate nodes are correct. Given that intermediate nodes $e_i \in \mathcal{E}$ and $e_j \in \mathcal{E}^*$ were matched by the alignment algorithm, the F1 score is computed by comparing the textual similarity between the set of the aligned and correct pairs e_i and e_j.

Overall (All-Correct): Tests all previous metrics together. The All-Correct value is only 1 if the All-Correct values for leaves, steps, and intermediates are 1. Note that this is a strict metric, and any semantic difference between T and T^* will cause the score to be zero.

4.3 Implementation Details

All experiments were conducted using a machine with 4 Tesla V100 GPUs with 16GB of memory. Our code is based on HuggingFace's Transformers (Wolf et al., 2020) implementation of the `t5-large` model (Raffel et al., 2020). The retrieval module uses the Sentence Transformers (Reimers and Gurevych, 2019) sentence embeddings by fine-tuning the `all-mpnet-base-v2` encoder. Please refer to Appendix A.1 for further details on hyper-parameters and training settings.

4.4 Results

4.4.1 Retrieval Results

We compare our retrieval module against two baselines: **Okapi BM25** and the retrieval module of **EntailmentWriter**, which constitutes of a classifier that retrieves relevant sentences using RoBERTA (Liu et al., 2020) and performs re-ranking with Tensorflow-Ranking-BERT (Han et al., 2020).

For comparison, we break down the results of our approach (the IRGR-retriever module) into three variations. The **IRGR-retriever (sing.)** method retrieves premises from the corpus using a single query element, namely the hypothesis h. The **IRGR-retriever (cond.)** method performs conditioned retrieval as described by Algorithm 1. This retrieval method is not iterative and fetches a fixed set of premises per example. Finally, **IRGR-retriever** tries to emulate the retrieval when combined with the generation module. It not only performs conditional retrieval, but also fetches a different set of premises for each iteration depending on the generated intermediate nodes. In this retrieval experiment, the IRGR-retriever uses the intermediate nodes from the golden entailment trees.

Task	Method	Leaves		Steps		Intermediates		Overall
		F1	All-Cor.	F1	All-Cor.	F1	All-Cor.	All-Cor.
Gold	EntailmentWriter	98.7	86.2	50.5	37.7	67.6	50.3	34.4
	IRGR	97.9	89.4	50.2	36.8	62.1	45.6	32.3
Gold+Dist.	EntailmentWriter	84.3	38.5	35.7	23.5	62.6	50.9	22.4
	IRGR	69.9	23.8	30.5	22.3	47.7	56.5	22.0

Table 3: Entailment tree scores for baseline methods and IRGR, along four different dimensions (test set). The "Gold" and "Gold+Dist." tasks do not require retrieval and evaluates solely on the model's entailment tree generation capabilities.

Therefore, IRGR-retriever results should be considered an upper bound since the generator might not produce the desirable intermediate steps used for queries.

Table 1 shows the R@25 and All-Correct metrics results for different methods. Our premise retrieval module performs consistently better than baselines. For instance, the "IRGR-retriever (cond.)" outperforms the retriever from EntailmentWriter by 14.2% on R@25 and 28.8% on All-Correct metric. Note that "IRGR-retriever" may retrieve a variable number of premises (greater than 25), so we are not reporting R@25 for this method.

4.4.2 Entailment Tree Generation Results

We compare our method against EntailmentWriter baseline model on entailment tree generation. As shown in Table 2, our method outperforms the EntailmentWriter in all metrics. The overall tree structure better matches the golden tree, where the score for Overall All-Correct metric has an impressive increase of over 300.0%. Note that EntailmentWriter uses the T5-11B model, which has around 10 times more parameters than our model.

We also show the ablation results of combining different retrieval modules with our proposed generation module on Table 2. The "**w/o iter.**" method does not iteratively retrieve premises, relying on one-shot retrieval at the beginning of the generation. As for the "**w/o iter. & cond.**" method, the model does not use the conditioned retrieval, only relying on the trained dense retrieval with the hypothesis h as the query instead.

The work of Dalvi et al. (2021) defines two other simplified entailment tree generation tasks for further ablation studies. We report the results for what they define as "Task-1" and "Task-2", which are generation tasks where the golden premises are given as input, disregarding the retrieval component. Results in Table 2 report what they define as "Task-3". For clarity, we rename "Task-1" and "Task-2" to "Gold" and "Gold+Dist.", respectively, and show the results in Table 3. In the "Gold" task, each context uses the golden leaves as input, while the "Gold+Dist." task uses the golden leaves plus some distractors (up to 25 distractors). When comparing models with the same number of parameters (we use their reported T5-large results), the generation results without retrieval are roughly the same as the EntailmentWriter method. This experiment shows that the iterative generation can create accurate explanations compared to a single pass generation when using golden retrieved premises.

4.5 Results Breakdown

We investigate how well the system performs relative to the number of steps in the gold tree. Figure 5 contains two graphs with results breakdown. The graph on the top shows the all-correct metric values for all three tasks (golden, golden + distractors, and retrieval). The bottom graph shows all F1 metrics (leaves, steps, and intermediates), but only for the "retrieval" task.

The results demonstrate that generating entailment trees becomes increasingly difficult as the size of the tree increases. The IRGR model cannot perfectly predict trees with more than four steps for any of the three different tasks. For the "retrieval" task (without the golden leaf sentences provided as input), the IRGR model cannot successfully generate trees with three or more steps. This could be explained by the fact that the all-correct metric is very strict, and missing or misplacing a single leaf sentence can result in an incorrect tree.

This downwards trend is also present in the "Break Down by Metrics" graph. Most noticeably, the "Intermediates (F1)" metric is especially challenging, having values close to zero for entailment trees with more than five steps. This metric is one

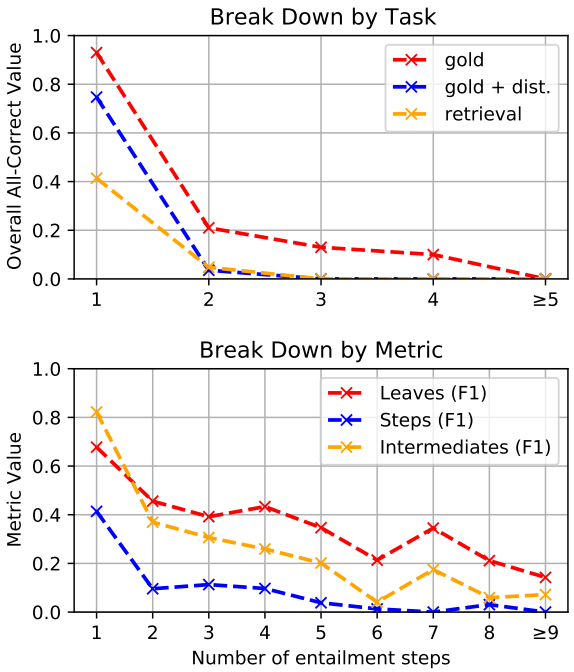

Figure 5: Result breakdown for number of steps in explanation (entailment steps).

of the main bottlenecks that lowers the value of the "Overall All-Correct" metric.

4.6 Analysis

To understand the strengths and weaknesses of our model, we conduct further analysis of the output of the IRGR. When analyzing errors in the generation of entailment trees, we use the results on the development set for the task with distractors. We manually annotate 50 predicted trees that contain some error compared to the golden tree. We categorize the different types of errors, identifying both individual generated steps errors and entailment tree errors.

4.6.1 Retrieval Error Analysis

We use ENTAILMENTBANK's development set to automatically compute metrics that will give us some insights into the type of errors made by the IRGR-retriever module. We use "IRGR-retriever (cond.)" to fetch a set of 25 premises for each data point, where we identify the set of **true positives** (correctly retrieved premises) and the set of **false negatives** (missing premises).

To understand if the false negatives are more challenging to retrieve than the true positives, we compute the number of overlapping uni-grams and bi-grams between premises and hypotheses in these two sets. We notice that true positives contain 28.5% more uni-gram overlap and 68.6% more bi-gram overlap to the hypothesis compared to the false negatives. These results suggest that premises lexically *dissimilar* to the hypothesis are, in theory, more challenging to retrieve.

We also investigate how the depth (number of edges in a path from the tree root) of a leaf node in the gold tree correlates to the errors of the IRGR-retriever module. We compute the average depth of true positive nodes as 2.3, while for false-negative nodes, the average depth is 3.0. These results strengthen the idea that leaf nodes deeper in the tree tend to be harder to retrieve, as depicted in Figure 4.

4.6.2 Entailment Step Error Analysis

The first error case is called **invalid entailment steps** (56% of errors), meaning that the conclusion of a step did not follow from the premises. For instance, in "*kilogram is used to measure heavy objects*" ∧ "*an automobile is usually a heavy object*" ⇒ "*kilogram can be used to measure the mass of an automobile*", the model assumes that "measure" is the same as "measure of mass", even though that is not explicitly stated.

The second error case accounts for **misevaluation and irrelevance** (27% of errors). It happens when the step is correct but does not match the golden tree, or when the step is correct but is not relevant or well placed in the final entailment tree. In the third error case, labeled **repetition** (17% of errors), the conclusion directly copied the premises, not creating a new sentence for the intermediate step.

4.6.3 Entailment Tree Error Analysis

When analyzing errors between the entire generated and golden trees, we noticed that **incorrect or missing leaves** (52% of errors) is the most common type of problem. For instance, when explaining the hypothesis "*light year can be used to measure the distance between the stars in milky way*" the premises "*the milky way is a kind of galaxy*" and "*a galaxy is made of stars*" are missing from the generated tree, making it impossible to explain the second part of the hypothesis.

The remaining errors are categorized as **invalid or skipped steps** (32% of errors), where the model commonly concludes an invalid conclusion from premises. This error often overlaps with **missing leaves** due to the fact that the model uses

fewer premises when it skips important intermediate steps; **Imperfect evaluation** (12% of errors), where the tree produced is valid, but does not match the golden tree; **Disconnected or degenerate trees** (4% of errors), where the generated output does not form a tree, or follows the desired output format.

5 Conclusion

As deep learning models become more ubiquitous in the natural language field, it is desirable that users can understand the model's answer by inspecting the reasoning chain from simple premises to the answer hypothesis. To generate rich, systematic explanations, we proposed a method that can iteratively generate and retrieve premises to produce entailment trees. We show how our approach has advantages over previous baselines, where the retrieved premises and generated explanations are more accurate.

In future work, we plan to improve the generation module by leveraging the structure of the entailment tree instead of relying purely on the encoder-decoder models. This idea could potentially fix the issues with "invalid entailment steps" and "repetition", which account for 73% of entailment step errors. We also plan to understand how explanations can be generated in the case of a false hypothesis, where we would expect the model to build a conclusion explaining why a statement is incorrect. It could help users verify false claims and understand the meaning behind their incorrectness.

Acknowledgements

We are thankful to Felicity M. Lu-Hill for proofreading this paper. The research leading to this paper was supported in part by the Machine Learning, Reasoning, and Intelligence Program of the Office of Naval Research.

References

James Allen. 1988. *Natural language understanding*. Benjamin-Cummings Publishing Co., Inc.

Tiberiu Boros, Stefan Daniel Dumitrescu, and Sonia Pipa. 2017. Fast and accurate decision trees for natural language processing tasks. In *Proceedings of the International Conference Recent Advances in Natural Language Processing, RANLP 2017*, pages 103–110, Varna, Bulgaria. INCOMA Ltd.

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*, pages 6266–6278, Online and Punta Cana, Dominican Republic. 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 S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 9539–9549. Curran Associates, Inc.

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*, pages 6916–6930, Barcelona, Spain (Online). International Committee on Computational Linguistics.

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

Marina Danilevsky, Kun Qian, Ranit Aharonov, Yannis Katsis, Ban Kawas, and Prithviraj Sen. 2020. A survey of the state of explainable AI for natural language processing. In *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing*, pages 447–459, Suzhou, China. 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*, pages 4443–4458, Online. Association for Computational Linguistics.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. 2020. REALM: Retrieval-augmented language model pre-training. *arXiv preprint arXiv:2002.08909*.

Shuguang Han, Xuanhui Wang, Mike Bendersky, and Marc Najork. 2020. Learning-to-rank with BERT in tf-ranking. *CoRR*, abs/2004.08476.

Sarthak Jain and Byron C. Wallace. 2019. Attention is not Explanation. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3543–3556, Stroudsburg, PA, USA. 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)*.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781, Online. Association for Computational Linguistics.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474. Curran Associates, Inc.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2020. Ro{bert}a: A robustly optimized {bert} pretraining approach.

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. 2019a. Explain yourself! leveraging language models for commonsense reasoning. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4932–4942, Florence, Italy. Association for Computational Linguistics.

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

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China. Association for Computational Linguistics.

Danilo Ribeiro, Thomas Hinrichs, Maxwell Crouse, Kenneth Forbus, Maria Chang, and Michael Witbrock. 2019. Predicting state changes in procedural text using analogical question answering. In *7th Annual Conference on Advances in Cognitive Systems*.

Danilo Neves Ribeiro and Kenneth Forbus. 2021. Combining analogy with language models for knowledge extraction. In *3rd Conference on Automated Knowledge Base Construction*.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2018. Anchors: High-precision model-agnostic explanations. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32.

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

Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. 2020. Mpnet: Masked and permuted pre-training for language understanding. In *Advances in Neural Information Processing Systems*, volume 33, pages 16857–16867. Curran Associates, Inc.

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

Marco Valentino, Mokanarangan Thayaparan, Deborah Ferreira, and André Freitas. 2021. Hybrid autoregressive inference for scalable multi-hop explanation regeneration. In *36th AAAI Conference on Artificial Intelligence*.

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

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

Wenhan Xiong, Xiang Li, Srini Iyer, Jingfei Du, Patrick Lewis, William Yang Wang, Yashar Mehdad, Scott Yih, Sebastian Riedel, Douwe Kiela, and Barlas Oguz. 2021. Answering complex open-domain questions with multi-hop dense retrieval. In *International Conference on Learning Representations*.

Chen Zhao, Chenyan Xiong, Jordan Boyd-Graber, and Hal Daumé III. 2021. Multi-step reasoning over unstructured text with beam dense retrieval. In *North American Association for Computational Linguistics*.

A Appendix

A.1 Experiment Details

The IRGR-generator used the `T5-large`[1] model from HuggingFace library. The best models were chosen according to the best "Overrall All-Correct" metric on the validation set. During training, we used the following hyper parameters: learning rate: $3 \cdot 10^{-5}$, epochs: 15, training batch size: 4, validation batch size: 4, max number of input tokens: 512, max number of output tokens: 256, warm-up steps: 0, weight decay: 0.

The IRGR-retriever module uses the version `all-mpnet-base-v2`[2] from the Sentence-Transformers library. During training, we used the following hyper parameters: learning rate: $5 \cdot 10^{-5}$, epochs: 10, training batch size: 32, validation batch size: 32, loss function: *cosine similarity loss*, warm-up steps: 0, weight decay: 0.

A.2 Entailment Tree Evaluation Metrics Details

The alignment algorithm between the nodes from gold and predicted entailment trees can be roughly described as follows:

1. For each intermediate node $e_i \in \mathcal{E}$ and $e_j \in \mathcal{E}^*$, compute the set of leaf sentences in which the nodes are ancestors.

2. Align each intermediate node e_i to the first golden node e_j for which the Jaccard similarity of their respective set of associated leaf sentences is maximum. If any node e_i is associated with no gold nodes e_j (Jaccard similarity is zero), then e_i is aligned to a blank node (no conclusion).

Given the list of aligned nodes, the metrics "Intermediates (F1, All-Correct)" compute the similarity between two generated intermediate steps using $BLEURT$ (Sellam et al., 2020), a learned evaluation metric based on BERT. We use the BLEURT-Large-512 model to compute textual similarity scores. The prediction between intermediate nodes is considered correct if $BLEURT(e_i, e_j) > 0.28$ (this threshold was originally picked by Dalvi et al. (2021) using a subset of 300 manually labeled pairs).

A.3 Generator Input and Output Examples

The generation is done in multiple steps. Below are the input and output examples for a tree with two entailment steps (T=1 and T=2). In the following example, only the golden premises are used, which is how the generator is trained. At test time this format is extended to use the retrieved premises instead.

INPUT T=1: "`hypothesis: notebook paper can be recycled many times; sent1: recyclable means a material can be recycled / reused many times sent2: paper is recyclable sent3: notebook paper is a kind of paper;`"

OUTPUT T=1: "`sent2 & sent3 -> int1: notebook paper is recyclable;`"

INPUT T=2: "`hypothesis: notebook paper can be recycled many times; sent1: recyclable means a material can be recycled / reused many times; sent2 & sent3 -> int1: notebook paper is recyclable;`"

OUTPUT T=2: "`int1 & sent1 -> hypothesis;`"

Note that the input for **T=2** removed the premises used in the previous entailment step, i.e. "`sent2`" and "`sent3`", and added the generated entailment step from **T=1** to the end of the input.

[1] Model available in https://huggingface.co/t5-large

[2] Model available in https://huggingface.co/sentence-transformers/all-mpnet-base-v2