# Reasoning over Logically Interacted Conditions for Question Answering

**Anonymous ACL submission**

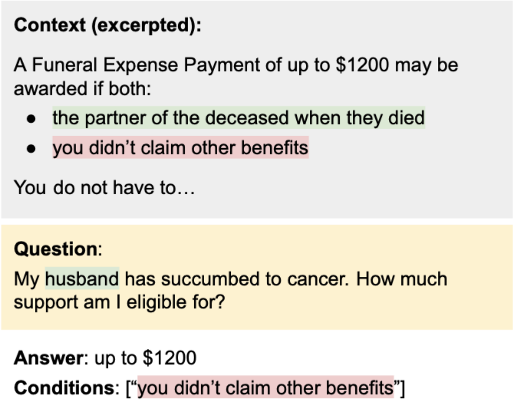
## Abstract

Some questions have answers that are correct only if certain conditions apply. Conditions are used to distinguish answers as well as to provide additional information to support them. To answer questions with conditions, models need to first find eligible answers and conditions from context and then perform logical reasoning to check whether conditions have been satisfied. We propose TReasoner to model this challenging reasoning process. In addition to finding answers, TReasoner can also identify unsatisfied conditions that are required to support the answers, as some answers are constrained by multiple conditions but only one or a subset of the conditions are satisfied. TReasoner consists of an entailment module, a reasoning module, and a generation module (if answers are free-form text spans). TReasoner achieves state-of-the-art performance on two benchmark QA datasets, outperforming the previous state-of-the-art by 3-10 points.[1]

## 1 Introduction

Recent work on QA has explored questions which have multiple possible answers, depending on conditions not explicitly given in the question (Min et al., 2020; Zhang and Choi, 2021; Dhingra et al., 2021; Chen et al., 2021). For example, "when was the first Covid vaccine approved" has different answers for different countries, so answers must be completed with implicitly assumed conditions (e.g. Dec 20th, 2020 [in the US]"). In this work, we follow this direction, but focus on a more challenging task, in which answers rely on multiple conditions that logically interact.

An example is shown in Figure 1. The span "up to $1200" is an eligible answer, associated with two conditions, "you are partner . . . of the deceased" and "you didn't claim other benefits". These two conditions interact, as the answer "up to $1200" is only valid if both conditions are satisfied. We say that those conditions are a condition group and the logical type of the group is "all" (as witnessed by the span "if both"). In addition to predicting eligible answers to the questions, QA in this context additionally requires models to perform the following two tasks. First, it must understand the document well enough to parse it into eligible answers, condition groups, and logical types; second, it must identify which conditions are entailed by the question (and the scenario), which are contradicted, and which are not mentioned but are required to support an eligible answer. With performing the two tasks, a model can produce an answer together with a description of when that answer is supported, i.e. the unsatisfied conditions.



**Context (excerpted):**

A Funeral Expense Payment of up to $1200 may be awarded if both:

- the partner of the deceased when they died
- you didn't claim other benefits

You do not have to…

**Question:**

My husband has succumbed to cancer. How much support am I eligible for?

**Answer**: up to $1200

**Conditions**: ["you didn't claim other benefits"]

Figure 1: An example of reasoning over *conditions*. The answer "up to $1200" is only correct if "both" conditions are true. The scenario suggests that the user is a partner of the deceased but there's no evidence suggesting the condition "you didn't claim any benefits". Answering this question requires not only finding probable answers but also identifying unsatisfied conditions.

One of the challenges in this task is to perform logical reasoning within condition groups to determine the entailment status of conditions. The entailment status of a condition is affected by two factors: the entailment status of itself, i.e. whether it is satisfied or contradicted by the provided evidence, and the entailment status of other conditions

[1] Codes and data will be released.

in the same condition group. Similar tasks have been studied in Clark et al. (2020b) where they constructed examples that contain groups of conditions and evidences for the conditions. Conditions in the group are either satisfied or contradicted by the evidences, so the task is referred to as *deductive reasoning* because all information needed to make a definite prediction is provided. We do not make such assumption, but instead only provide evidences for a subset of conditions in the group and ask models to identify unsatisfied conditions that need to be further checked. For example (Figure 1), we say "you didn't claim other benefits" is an unsatisfied condition because it is required by the candidate answer "up to $1200" but is not satisfied by the user's scenario. This task is commonly called *abductive reasoning*. Predicting unsatisfied conditions tests a model's ability in logical reasoning, including understanding logical operations, determining the entailment status of conditions in the logical groups, and finally determining whether an eligible answer is correct.

We propose TReasoner to tackle the task of reasoning with logically interacted conditions. TReasoner contains two modules: an entailment module and a reasoning module. The entailment module takes a condition in the context with the question and the provided evidence to predict its entailment status. The reasoning module takes the entailment module's outputs for all conditions and performs logical reasoning to identify unsatisfied conditions. If the answer is a free-form text span, TReasoner additionally uses a generation module to generate the answer span. The entailment module, reasoning module, and generation module are jointly trained. TReasoner shows excellent reasoning ability on a synthetic dataset and outperforms the previous state-of-the-art models on two Question Answering (QA) datasets, ConditionalQA and ShARC (Sun et al., 2021a; Saeidi et al., 2018), improving the state-of-the-art by 3-10 points on answer and unsatisfied condition prediction tasks.

## 2 Related Work

Models (Cohen, 2016; Cohen et al., 2020; Sun et al., 2020; Ren et al., 2020; Ren and Leskovec, 2020) have been developed for the deductive reasoning task with symbolic rules. Embedding-based methods (Sun et al., 2020; Ren et al., 2020; Ren and Leskovec, 2020) first convert symbolic facts and rules to embeddings and then apply neural network layers on top to softly predict answers. Recent work in deductive reasoning focused on tasks where rules and facts are expressed in natural language (Talmor et al., 2020; Saeed et al., 2021; Clark et al., 2020b; Kassner et al., 2020). Such tasks are more challenging because the model has to first understand the logic described in the natural language sentences before performing logical reasoning.

Different from deductive reasoning, the QA task proposed in this paper provides a list of conditions that if true would support an answer. (This is also referred to as abductive reasoning.) The ConditionalQA and ShARC dataset (Sun et al., 2021a; Saeidi et al., 2018) were proposed, where a question contains a user scenario that includes some background information that suggests the answer but is not enough to ensure its correctness. Similar examples were also seen in factual questions, e.g. AmbigQA (Min et al., 2020), where multiple answers are plausible given the facts asked in the question, but each answer is only correct under certain conditions. Answering such questions requires both finding the probable answers and identifying their underlying conditions.

Very limited work has explored abductive reasoning for QA. Previous work (Gao et al., 2020a,b; Ouyang et al., 2020) on the ShARC (Saeidi et al., 2018) dataset proposed to solve this problem by predicting a special label "inquire" if there was not enough information to make a definite prediction. The reasoning process was performed in the embedding space. Specifically, EMT and DISCERN (Gao et al., 2020a,b) computed an entailment vector for each condition and performed a weighted sum of those vectors to predict the final answer. DGM (Ouyang et al., 2020) additionally introduced a GCN-based model to better represent the entailment vectors. Even though these models were able to predict the answer labels as "inquire" when there were unsatisfied conditions, none of them could predict which conditions needed to be further satisfied. Furthermore, they simply concatenated the full context and the question into a single input and encode it with a Transformer with $O(N^2)$ complexity, making it not scalable to longer contexts.

## 3 Model

### 3.1 Task: QA with Conditions

We study the task of QA with logically interacted conditions. The model learns to find eligible answers to questions and additionally performs logical reasoning over conditions to check whether the

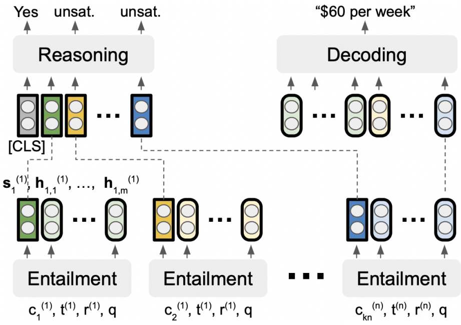


Figure 2: **TReasoner Overview**. The entailment module independently encodes each condition $c_j^{(i)}$, its associated results $r^{(i)}$ and logical types $t^{(i)}$, and the question $q$. The entailment module outputs a condition embedding $\mathbf{s}_j^{(i)}$ that will be input into the reasoning module to predict the answer labels and determine unsatisfied conditions, and a token embedding $\mathbf{h}_{k_i,p}^{(i)}$ that will be used by the decoding module to generate answer spans (if the question has a free-form answer).

eligible answers are correct. If the answers require additional conditions to be satisfied, the model identifies these unsatisfied conditions as well.

We consider a context that contains *results*, *conditions*, and *types*.[2] A *result* is a sentence that contains the answer, e.g. "You can get a Funeral Expense Payment of up to $1200 ...". Questions with yes/no answers also need result statements. For example, "You don't need to pay taxes if you ..." is the result statement for the question "Do I need to pay taxes?". A *condition* describes a requirement that needs to be meet for a result to be applicable, e.g. "physically or mentally disabled" in Figure 1. There could be multiple conditions for one result that interact under a logical *type*. For example, "if you're both:" requires both conditions to be satisfied. In this project, we consider four logical types that are commonly seen in QA tasks:

- "all": all conditions under this logical type should be satisfied in order to make the answer true. The logical type "if you're both:" in the example above is an example of this type.

- "any": only requires one of the conditions under the logical type "any" to be satisfied. For example, "if you satisfy at least one of the following conditions.". It doesn't matter whether other conditions have been satisfied, contradicted, or not mentioned in the question.

- "required": This is a special case of "all" / "any" when there is only one condition. Conditions with the logical type "required" must be satisfied. For example, "you must ... to get an up to $1200 payment."

- "optional": Conditions have the type "optional" if they are not relevant to the question. For example, "You will need to pay a $30 processing fee if you apply online".

Logical types are often not provided in real QA datasets, but can be inferred from the context. We discuss strategies to softly predict logical types to perform reasoning tasks for ConditionalQA in §4.2 and ShARC in §4.3. We will also discuss strategies to discover conditions and results from the context since they are often not labeled.

Formally, let the context $X$ consist of multiple results $r^{(1)}, \ldots, r^{(n)}$, and the result $r^{(i)}$ be constrained by a group of conditions $\{c_1^{(i)}, \ldots, c_{k_i}^{(i)}\}$ under the logical type $t^{(i)}$. We represent the context $X$ as a list of tuples $\{(\{c_1^{(1)}, \ldots, c_{k_1}^{(1)}\}, r^{(1)}, t^{(1)}), (\{c_1^{(2)}, \ldots, c_{k_2}^{(2)}\}, r^{(2)}, t^{(2)}), \ldots\}$. We learn a model that operates on context $X$ and question $q$ to predict the answer $a$ with a list of unsatisfied conditions $\hat{C} = \{\hat{c}_1, \ldots, \hat{c}_m\}$.[3]

### 3.2 Model

TReasoner consists of an *entailment module* and a *reasoning module*. The entailment module checks whether a condition has been satisfied. Practically, it takes a condition, a result, and a question as its input, and outputs a learned embedding. Each condition will be encoded independently. Embeddings of conditions will be passed to the reasoning module, which performs logical reasoning to predict an answer (if it is a multi-class label) and unsatisfied conditions. In cases where answers are text spans, we apply a *decoding module* to generate answer spans. All modules are jointly trained.

**Input** We independently encode each condition along with its associated result and the question. For condition $c_j^{(i)}$ in $\{c_1^{(i)}, \ldots, c_{k_i}^{(i)}\}$ with result $r^{(i)}$ under logical type $t^{(i)}$, we concatenate them and separate them with special prefixes.

$$s_j^{(i)} = \text{“condition:”} + \ c_j^{(i)} + \text{“type:”} + \ t^{(i)} \\ + \text{“result:”} + \ r^{(i)} + \text{“question:”} + \ q \qquad (1)$$

**Entailment Module** The entailment module encodes the concatenated input $s_j^{(i)}$ into a vector $\mathbf{s}_j^{(i)}$.

[2] In real cases, parsing context can be challenging. We do not explicitly parse the context but leave it as a sub-task for the entailment module. See below (§3.2 and §4.2)

[3] Some questions in the ConditionalQA dataset have multiple answers, but we do not handle these cases in this paper.

We initialize the parameters of the entailment module from pretrained LMs, which will be finetuned together with other modules.

$$\mathbf{s}_j^{(i)}, \mathbf{h}_{j,1}^{(i)}, \ldots, \mathbf{h}_{j,m}^{(i)} = \text{Entail}(s_j^{(i)}) \qquad (2)$$

The embedding of the first token of $s_j^{(i)}$ in Eq. 2 will be used as the embedding $\mathbf{s}_j^{(i)}$ for condition $c_j^{(i)}$ and will be used by the reasoning module to predict the answer label (for multi-class answers). $\mathbf{h}_{j,1}^{(i)}, \ldots, \mathbf{h}_{j,m}^{(i)}$ are the contextualized embeddings for the tokens in condition $s_j^{(i)}$. Token embeddings will not be used for reasoning but will be used for decoding if the answers are free-form answers.

At the entailment stage, each condition $s_j^{(i)}$ is encoded independently. The embedding output $\mathbf{s}_j^{(i)}$ is expected to have information about the entailment state of the condition, the logical operation, and whether the result is relevant to the question. Encoding each condition independently also reduces the encoding complexity of all conditions in the passage from $O(C^2)$ to $O(C)$ where $C$ is the number of conditions in the provided passage, and thus enables handling longer context with hundreds of conditions. This encoding strategy is motivated by FiD (Izacard and Grave, 2020)

**Reasoning Module** The reasoning module takes the embeddings of conditions from the entailment module and reasons over them to predict the answer label (if the answer is a multi-class label). We use a Transformer model as our reasoner because the self attention mechanism allows conditions $\{s_1^{(i)}, \ldots, s_{k_i}^{(i)}\}$ to attend to each other to perform reasoning steps. It is crucial for reasoning because, for example, if one of the conditions is satisfied and the operation type is "any", then other conditions will be implicitly satisfied, regardless of their real entailment status.

We prepend a learned vector $\mathbf{s}_0$ to the list of condition embeddings, which will be used as the [CLS] embedding to summarize the reasoning result. The output of the reasoning module, $\hat{\mathbf{s}}_0, \hat{\mathbf{s}}_1^{(1)}, \ldots, \hat{\mathbf{s}}_{k_n}^{(n)}$, will be used to predict the final label and unsatisfied conditions. Specifically, we use the first embedding $\hat{\mathbf{s}}_0$ to predict the answer label and use the subsequent embeddings $\hat{\mathbf{s}}_1^{(1)}, \ldots, \hat{\mathbf{s}}_{k_n}^{(n)}$ to predict unsatisfied conditions.

$$\begin{gathered}\hat{\mathbf{s}}_0, \hat{\mathbf{s}}_1^{(1)}, \ldots, \hat{\mathbf{s}}_{k_n}^{(n)} = \text{Reason}(\mathbf{s}_0, \mathbf{s}_1^{(1)}, \ldots, \mathbf{s}_{k_n}^{(n)}) \\ l_{\text{label}} = \text{softmax\_cross\_entropy}(\mathbf{W}_l^T \hat{\mathbf{s}}_0, \mathbb{I}_l) \\ l_{\text{cond}} = \text{softmax\_cross\_entropy}(\mathbf{W}_c^T \hat{\mathbf{s}}_j^{(i)}, \mathbb{I}_c)\end{gathered}$$

where $\mathbb{I}_l$ and $\mathbb{I}_c$ are one-hot vectors for class labels. The number of label classes is task-dependent, but in most cases, the final labels $\mathbb{I}_l$ are "yes", "no", and "irrelevant". The condition labels are "entailed", "contradicted", "not mentioned", "implied", and "to check". The first three classes are as they are named. The class "implied" means the entailment state of this condition is implied by other conditions with the same result, e.g. if one of the conditions with the logical type "any" has been satisfied, the rest of conditions are automatically "implied". The class "to check" means it is an unsatisfied condition. It is important to note that the condition loss $l_{\text{cond}}$ is an auxiliary loss and may not exist (or only exist for a subset of conditions) in real datasets.

For questions that have free-form answers, e.g. "up to $1200", the answers will be generated from the decoding module discussed in the next section. We will not supervise their class labels in training and can safely discard the predicted label in testing. In this case, only the predictions of the unsatisfied conditions will be kept. On the contrary, for questions that have multi-class answers, the reasoning module is trained to predict the correct label while the decoding module (discussed next) is trained to generate a special token [MULTI].

**Decoding Module** The decoding module takes token embeddings for all conditions $\mathbf{h}_{1,1}^{(1)}, \ldots, \mathbf{h}_{k_n,m}^{(n)}$ to generate answer spans. This module is mostly used when final answers are text spans. If an answer is a multi-class label, the decoding module should simply generate a special token [MULTI]. s We adopt the decoding strategy proposed by FiD (Izacard and Grave, 2020) with the T5 architecture (Raffel et al., 2019)[4], i.e. the token embeddings are concatenated for decoding even though the they are generated independently for each condition. The generation task is trained with teacher forcing. We do not write out the explicit expression for the teacher forcing decoding loss $l_{\text{decode}}$ here. Please refer to the T5 paper (Raffel et al., 2019) for more information. The decoded tokens $\hat{a}$ are taken as the predicted answer span.

$$\hat{a} = \text{Decode}(\mathbf{h}_{1,1}^{(1)}, \ldots, \mathbf{h}_{k_n,m}^{(n)}) \qquad (3)$$

**Loss Function** We jointly train the entailment module and reasoning module. We provide intermediate supervision on the entailment state of each condition, i.e. $\mathbf{s}_j^{(i)}$, if they are available. The final loss function is the sum of the answer loss $l_{\text{label}}$ and the

[4] The T5 encoder is used for the entailment module

condition entailment loss $l_{\text{cond}}$.

$$l = l_{\text{label}} + l_{\text{cond}}$$

If the answers contain text spans, we jointly train the decoding module as well. The loss function is the sum of the three losses:

$$l = l_{\text{label}} + l_{\text{cond}} + l_{\text{decode}}$$

### 3.3 Pretrained Checkpoints

The entailment module and decoding module (if any) load pretrained LM checkpoints and finetune the parameters for downstream tasks. For the dataset that has both multi-class answers and free-form answers, we initialize the entailment module and decoding module with the pretrained T5 encoder and decoder (Raffel et al., 2019). For a dataset that only has multi-class answers, the decoding module is not needed, so only the entailment module will be initialized. The entailment module can be initialized with T5 (encoder only) or any other pretrained LMs, e.g. BERT, RoBERTa, ELECTRA, BART, (Devlin et al., 2018; Liu et al., 2019; Clark et al., 2020a; Lewis et al., 2019), etc. We use ELECTRA for our entailment module if the decoding module is not needed (438M parameters), and T5 (873M parameters) otherwise.

The reasoning module is randomly initialized and jointly trained with the entailment and decoding modules. The number of Transformer layers for the reasoning module is a hyper-parameter. We choose the number of layers $l = 3$ or $l = 4$. Please see §4.1.2 for ablation study on the number of Transformer layers for the reasoning task.

## 4 Experiments

We experiment TReasoner with a synthetic dataset, CondNLI, and two benchmark QA datasets, ConditionalQA (Sun et al., 2021a) and ShARC (Saeidi et al., 2018), that require reasoning over conditions to predict the answers.[5]

### 4.1 CondNLI

#### 4.1.1 Task

The CondNLI dataset is constructed from the existing Natural Language Inference (NLI) dataset, MultiNLI (Williams et al., 2018). In the original NLI dataset, an example has a premise, a hypothesis, and a label, e.g. “entailed”, “contradicted” or “neutral”. Please see Appendix A for more information in dataset construction. Briefly, we treat the premise as context and the hypothesis as question, and make a few additional changes. First, each premise is paired with a list of conditions $c_j$'s that interact under a logical type $t$. Second, a context contains multiple premises, but at most one of the premises are relevant to the hypothesis.[6] The model should first identify the relevant premise and then check their conditions to predict labels and unsatisfied conditions. Third, each example is provided an additional list of known facts for checking the entailment status of the conditions. All premise, hypothesis, conditions, and facts are obtained from MultiNLI (Williams et al., 2018). Table 1 gives an example in CondNLI.

Context: If all [*“Aged 59 1/2 or older”*, *“Employed for two years”*],
then *“Get at least $60 a week”*.
If any [not *“Has two children”*, *“Has not applied before.”*],
then *“Waive the application fees”*.
Facts: [*“Tom is 65 years old”*, *“He has two sons”*, *“Rejected last year”*]
Question: Is *“Eligible for $60 a week”* correct?
Label: Yes, if *“Employed for two years”*

Table 1: An example in CondNLI. The question is about the first result *“Get at least $60 a week”* with only one of the conditions *“Aged 59 1/2 or older”*. *“Employed for two years”* is an unsatisfied condition in the answer.

| | Label (acc) | Conditions (F1) |
|---|---|---|
| (template) | | |
| FiD (concat) | 99.8 | 98.7 |
| FiD (TReasoner) | 99.6 | 99.2 |
| TReasoner | 99.8 | 99.2 |
| (with NLI) | | |
| FiD (concat) | 85.6 | 80.4 |
| FiD (TReasoner) | 86.7 | 82.8 |
| TReasoner | 95.0 | 91.3 |

Table 2: Experiment results on the CondNLI dataset in label accuracy and condition F1. FiD (concat) is run on the input that concatenates the question and context and is then chunked into smaller pieces. FiD (TReasoner) use the same input as TReasoner. “(template)” directly uses the templates with variable letters as inputs, while “(with NLI)” uses the examples that are instantiated with real NLI examples.

#### 4.1.2 Results

Previous work (Clark et al., 2020b) showed that Transformer-based Language Models, e.g. RoBERTa (Liu et al., 2019), have the ability to reason over multiple conditions to answer the reasoning question in the deductive reasoning setting, e.g. “if A and B then C” with facts on conditions A and B provided. We replace RoBERTa with FiD to

[5] ConditionalQA and ShARC are both released for research purposes.

[6] In some examples, none of the premises is relevant to the hypothesis. Such examples will be labeled as “Irrelevant”.

handle long contexts. FiD is trained to generate answer labels and a list of unsatisfied conditions. To simplify the generation task, we prepend a condition id to each condition and let the model generate the condition id instead. We train the model on two types of input, one using templates with variables in letters, and the other using examples where variables are instantiated with real NLI examples.

**Main Results** The experiment results are shown in Table 2. We measure both the accuracy of label prediction and the F1 of unsatisfied conditions. The results show that a plain Transformer-based sequence-to-sequence model (FiD) performs the logical reasoning task reasonably well if the context is simple, i.e. using the template with variables $A$, $B$, $\ldots$ as inputs. However, the FiD performs significantly worse on examples with real NLI examples. TReasoner still performs well on the CondNLI dataset with NLI examples.

**Generalization to More Conditions** The TReasoner is trained on templates with 6 conditions or fewer. To test TReasoner's ability to generalize to more conditions, we take a trained model and test it on the examples with more than 6 conditions. Figure 3 (left) shows the change of performance in both label classification and unsatisfied condition prediction tasks as the number of conditions increase.[7] We observe more decrease in performance in predicting unsatisfied conditions (probably because more conditions are unsatisfied), but it is still reasonable with 20 conditions.

**Number of Reasoning Layers** We additionally experiment with different numbers of layers in the reasoner module. The results are shown in Figure 3 (right). The Transformer-based reasoner module needs at least 3 layers to perform the reasoning task, especially for predicting unsatisfied conditions.

### 4.2 ConditionalQA

In the second experiment, we run TReasoner on a real QA dataset, ConditionalQA (Sun et al., 2021a) (CC BY-SA 4.0 License), that requires reasoning over long documents with much more conditions and more complex logical operations stated in natural language.

#### 4.2.1 Task

ConditionalQA is challenging because it requires the model to accurately locate relevant results and conditions from longer documents. Previous models, e.g. RuleTaker, DGM (Clark et al., 2020b; Ouyang et al., 2020) concatenate the inputs into a long sequence and then compute cross-attention over the concatenate input. The length of the input is constrained by the $O(N^2)$ complexity. Even if we adopt the Fusion-in-Decoder (Izacard and Grave, 2020) strategy to handle long sequences, performance is still limited (see Table 2).

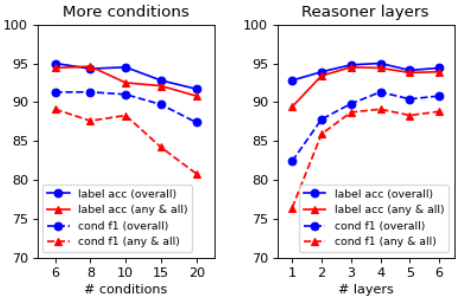


Figure 3: Left: Generalization results of reasoning over more conditions. Right: Results on the ablated model with different numbers of Transformer layers in the reasoning module. We report both label accuracy and F1 of unsatisfied conditions. "any & all" indicates that examples only have two types of logical operation: "any" and "all".

Another challenge in ConditionalQA is to identify logical operations for conditions. For example in Figure 1, the model should predict the logical operation "all" from the statement that "if you're both:". One could possibly provide intermediate supervision to predict logical operations. However, such labels are not provided in ConditionalQA and it is hard to find distant supervision labels. TReasoner encodes the logical operation in the condition's embeddings $\mathbf{s}_j^{(i)}$ (Eq. 2) and does not need additional supervision for the logical operation.

Furthermore, different from the CondNLI and ShARC datasets (§4.3), the ConditionalQA dataset contains questions with both yes/no answers and free-form answer spans. We apply the decoder module on the token embeddings $\mathbf{h}_{1,1}^{(1)}, \ldots, \mathbf{h}_{k_n,m}^{(n)}$ to generate the final answer spans (Eq. 3). Please see Appendix B for details on data preparation.

#### 4.2.2 Evaluation

The predictions are evaluated using two sets of metrics: EM/F1 and conditional EM/F1. EM/F1 are the traditional metrics that measures the predicted answer spans. The ConditionalQA dataset introduced another metric, *conditional EM/F1*, that jointly measures the accuracy of the answer span and the unsatisfied conditions. Please refer to the ConditionalQA paper (Sun et al., 2021a) for more information. Briefly, the conditional EM/F1 is the product of the original answer EM/F1 and the F1

[7] "any & all" indicates that the context only contains conditions under the logical operation "any" or "all".

| | Yes / No | | Extractive | | Conditional | | Overall | |
|---|---|---|---|---|---|---|---|---|
| | EM / F1 | w/ conds | EM / F1 | w/ conds | EM / F1 | w/ conds | EM / F1 | w/ conds |
| majority | 62.2 / 62.2 | 42.8 / 42.8 | – / – | – / – | – / – | – / – | – / – | – / – |
| ETC | 63.1 / 63.1 | 47.5 / 47.5 | 8.9 / 17.3 | 6.9 / 14.6 | 39.4 / 41.8 | 2.5 / 3.4 | 35.6 / 39.8 | 26.9 / 30.8 |
| DocHopper | 64.9 / 64.9 | 49.1 / 49.1 | 17.8 / 26.7 | 15.5 / 23.6 | 42.0 / 46.4 | 3.1 / 3.8 | 40.6 / 45.2 | 31.9 / 36.0 |
| FiD | 64.2 / 64.2 | 48.0 / 48.0 | 25.2 / 37.8 | 22.5 / 33.4 | 45.2 / 49.7 | 4.7 / 5.8 | 44.4 / 50.8 | 35.0 / 40.6 |
| TReasoner | **73.2 / 73.2** | **54.7 / 54.7** | **34.4 / 48.6** | **30.3 / 43.1** | **51.6 / 56.0** | **12.5 / 14.4** | **57.2 / 63.5** | **46.1 / 51.9** |

Table 3: Experimental results on ConditionalQA (EM / F1). The "EM/F1" columns reports the original EM/F1 metrics that are only evaluated on the answer span. The "w/ conds" is the conditional EM/F1 metric discussed in §4.2.2. Numbers of the baseline models are obtained from Sun et al. (2021a).

of the predicted unsatisfied conditions. The conditional EM/F1 is 1.0 if and only if the predicted answer span is correct and all unsatisfied conditions are found. If there's no unsatisfied condition, the model should predict an empty set.

### 4.2.3 Results

We compare TReasoner with a few baseline models, including ETC (in a pipeline) (Ainslie et al., 2020), DocHopper (Sun et al., 2021b), and Fusion-in-Decoder (FiD) (Izacard and Grave, 2020). The ETC pipeline first extracts possible answers from the context and then takes the question and extracted answers as input to find unsatisfied conditions. The answer extraction model and the condition prediction model are trained separately. DocHopper is a multi-hop attention system that iteratively attends to sentences to jointly predict the answers and unsatisfied conditions. The iterative process in DocHopper is updated in the embedding space so it is end-to-end differentiable. FiD is a encoder-decoder model based on T5. FiD improves over T5 by proposing to split long input sequences into short sequences, encode the short sequences independently, and jointly decode over all encoded embeddings to generate the outputs. For the ConditionalQA dataset, we train the FiD model to generate the answers followed by the list of unsatisfied conditions.

**Main Results** The experimental results are presented in Table 3. TReasoner achieves the state-of-the-art on both yes/no and extractive questions. TReasoner also significantly outperforms all the baselines on the questions with conditional answers with 166% and 148% relative improvement in the conditional EM/F1 metrics (w/ conds).

**Condition Accuracy** Since there's not a metric that directly measure the quality of predicted conditions, we additionally report the F1 of the predicted unsatisfied conditions (Table 2). The best baseline models, FiD, rarely predicts any conditions. This is likely because only a subset of the questions have unsatisfied conditions. Even though we train the FiD model only on the subset of questions that have conditional answers, its performance slightly improves but is still much lower than TReasoner by 16.5 points in condition F1.

| | Answer<br>(w/ conds) | Conditions<br>(P / R / F1) |
|---|---|---|
| FiD | 3.2 / 4.6 | 98.3 / 2.6 / 2.7 |
| FiD (conditional only) | 6.8 / 7.4 | 12.8 / 63.0 / 21.3 |
| TReasoner | **10.6 / 12.2** | **34.4 / 40.4 / 37.8** |

Table 4: Experimental results on the subset of questions in ConditionalQA (dev) that has conditional answers. Accuracy for the answers is evaluated using the conditional EM/F1 (w/ conds) metrics defined by Sun et al. (2021a). Conditions are evaluated in precision, recall and F1.

## 4.3 ShARC

We additionally experiment TReasoner with the ShARC (Saeidi et al., 2018) (CC BY 3.0 License) dataset. The ShARC dataset examples have shorter context, usually a few sentences or a short passage, but the logical operations between conditions are more complex, as is discussed below.

### 4.3.1 Task

The ShARC dataset has two subtasks: Decision Making and Question Generation. The decision making task asks the model to predict one of the following labels as the answer: "yes", "no", "inquire", and "irrelevant". The label "inquire" means that information provided by the question is not enough to make a definite prediction, i.e. there are unsatisfied conditions. In this case, the model should perform the Question Generation task to generate a followup question to clarify the unsatisfied conditions. The decision making task evaluates the predicted labels using micro and macro accuracy. The question generation task evaluates the BLEU scores of the generated question with the ground truth annotation. Note that some example could have multiple unsatisfied conditions, but only one of them will be annotated as ground truth followup

| | Decision (micro / macro) | Question (BLEU1 / BLEU4) |
|---|---|---|
| CM | 61.9 / 68.9 | 54.4 / 34.4 |
| BERTQA | 63.6 / 70.8 | 46.2 / 36.3 |
| UcraNet | 65.1 / 71.2 | 60.5 / 46.1 |
| Bison | 66.9 / 71.6 | 58.8 / 44.3 |
| E3 | 67.7 / 73.3 | 54.1 / 38.7 |
| EMT | 69.1 / 74.6 | 63.9 / 49.5 |
| DISCERN | 73.2 / 78.3 | 64.0 / 49.1 |
| DGM | 77.4 / 81.2 | 63.3 / 48.4 |
| TReasoner | **80.4 / 83.9** | **71.5 / 58.0** |

Table 5: Experimental results on the ShARC dataset. Numbers for the baseline models (Saeidi et al., 2018; Zhong and Zettlemoyer, 2019; Verma et al., 2020; Lawrence et al., 2019; Gao et al., 2020a,b; Ouyang et al., 2020) are borrowed from Ouyang et al. (2020).

question.[8] See Appendix C for data preparation.

### 4.3.2 Results

**Main Results** We compare TReasoner to a few strong baseline models, including the previous state-of-the-art model, e.g. DISCERN and DGM (Gao et al., 2020b; Ouyang et al., 2020). Different from the baseline models, which use separate models for label classification and unsatisfied condition prediction, TReasoner performs both tasks jointly.[9] The results are shown in Table 5. TReasoner outperforms the previous baselines by 3 points on the classification task and more than 8 points on the question generation task.

**Condition Accuracy** One problem with the current question generation task is that the ground-truth question only asks about one of the unsatisfied conditions, even though there could be multiple unsatisfied conditions. To further evaluate TReasoner's performance in predicting unsatisfied conditions, we manually annotate the logical operations in 20 passages that have more than one condition (857 data total),[10] and use the annotated logical operations to find all unsatisfied conditions. We report the F1 of the predicted unsatisfied conditions (see Table 6). Compared to the baselines (Gao et al., 2020b; Ouyang et al., 2020), TReasoner improves the F1 by 11.4.

**Label Accuracy v.s. Conditions** We additionally measure the accuracy versus the number of conditions in the context. We consider the number of all followup questions on each context as its number of conditions. Results in Table 7 show that the improvement in TReasoner's performance over the previous state-of-the-art model (DGM) mostly come from questions that have more than one conditions.

| | Decision (micro / macro) | Question (BLEU1 / 4) | Condition (F1) |
|---|---|---|---|
| DISCERN | 74.9 / 79.8 | 65.7 / 52.4 | 55.3 |
| DGM | 78.6 / 82.2 | **71.8 / 60.2** | 57.8 |
| TReasoner | **79.8 / 83.5** | **71.7 / 60.4** | **69.2** |

Table 6: Experiment results on the ShARC dataset (dev) compared to the baselines, DISCERN and DGM (Gao et al., 2020b; Ouyang et al., 2020). The Condition (F1) number is obtained by reruning their open-sourced codes.

| # conditions | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| DGM | 90.4 | 70.3 | 80.0 | 73.4 |
| TReasoner | 90.3 | 72.7 | 80.6 | 75.2 |
| *diff* | -0.1 | 2.4 | 0.6 | 1.8 |

Table 7: Ablation study on the label accuracy vs. the number of conditions in the context. Numbers of DGM (Ouyang et al., 2020) is obtained by reruning their open-sourced codes.

## 5 Conclusion

We study the problem of QA with answers that are constrained by a list of conditions that interact with each other under logical operations, such as "any" or "all". We propose a system, TReasoner, that contains an entailment module to check the entailment status of conditions and a jointly trained reasoning module that performs the logical reasoning to predict the final answers and the unsatisfied conditions. TReasoner shows excellent reasoning ability, and can easily generalize to more conditions on a synthetic dataset CondNLI. Furthermore, TReasoner achieves state-of-the-art performance on two challenging question answering datasets ConditionalQA (Sun et al., 2021a) and ShARC (Saeidi et al., 2018). However, reasoning over logically interacted conditions is still a very challenging task, and wrong predictions may lead to severe consequences in real world applications in professional domains. We advocate for more research in this direction.

## References

Joshua Ainslie, Santiago Ontañón, Chris Alberti, Philip Pham, Anirudh Ravula, and Sumit Sanghai. 2020. ETC: encoding long and structured data in transformers. *CoRR*, abs/2004.08483.

[8] To mitigate this issue in evaluation, we run an additional evaluation that measures the F1 of the predicted unsatisfied conditions. Please see results in Table 6.

[9] Previous models, e.g. DISCERN and DGM, additionally use a generation model to paraphrase the unsatisfied conditions into questions, similar to our generation process with T5.

[10] Each passage in ShARC has 32.9 data on average.

Wenhu Chen, Xinyi Wang, and William Yang Wang. 2021. A dataset for answering time-sensitive questions.

Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2020a. ELECTRA: pre-training text encoders as discriminators rather than generators. *CoRR*, abs/2003.10555.

Peter Clark, Oyvind Tafjord, and Kyle Richardson. 2020b. Transformers as soft reasoners over language. *CoRR*, abs/2002.05867.

William W Cohen. 2016. Tensorlog: A differentiable deductive database. *arXiv preprint arXiv:1605.06523*.

William W Cohen, Haitian Sun, R Alex Hofer, and Matthew Siegler. 2020. Scalable neural methods for reasoning with a symbolic knowledge base. *arXiv preprint arXiv:2002.06115*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Bhuwan Dhingra, Jeremy R. Cole, Julian Martin Eisenschlos, Daniel Gillick, Jacob Eisenstein, and William W. Cohen. 2021. Time-aware language models as temporal knowledge bases.

Yifan Gao, Chien-Sheng Wu, Shafiq Joty, Caiming Xiong, Richard Socher, Irwin King, Michael Lyu, and Steven C.H. Hoi. 2020a. Explicit memory tracker with coarse-to-fine reasoning for conversational machine reading. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 935–945, Online. Association for Computational Linguistics.

Yifan Gao, Chien-Sheng Wu, Jingjing Li, Shafiq R. Joty, Steven C. H. Hoi, Caiming Xiong, Irwin King, and Michael R. Lyu. 2020b. Discern: Discourse-aware entailment reasoning network for conversational machine reading. *CoRR*, abs/2010.01838.

Gautier Izacard and Edouard Grave. 2020. Leveraging passage retrieval with generative models for open domain question answering. *CoRR*, abs/2007.01282.

Nora Kassner, Benno Krojer, and Hinrich Schütze. 2020. Are pretrained language models symbolic reasoners over knowledge? *arXiv preprint arXiv:2006.10413*.

Carolin Lawrence, Bhushan Kotnis, and Mathias Niepert. 2019. Attending to future tokens for bidirectional sequence generation. 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 1–10, Hong Kong, China. Association for Computational Linguistics.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2019. BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *CoRR*, abs/1910.13461.

Jing Li, Aixin Sun, and Shafiq R Joty. 2018. Segbot: A generic neural text segmentation model with pointer network. In *IJCAI*, pages 4166–4172.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692.

Sewon Min, Julian Michael, Hannaneh Hajishirzi, and Luke Zettlemoyer. 2020. AmbigQA: Answering ambiguous open-domain questions. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5783–5797, Online. Association for Computational Linguistics.

Siru Ouyang, Zhuosheng Zhang, and Hai Zhao. 2020. Dialogue graph modeling for conversational machine reading. *CoRR*, abs/2012.14827.

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

Hongyu Ren, Weihua Hu, and Jure Leskovec. 2020. Query2box: Reasoning over knowledge graphs in vector space using box embeddings. *arXiv preprint arXiv:2002.05969*.

Hongyu Ren and Jure Leskovec. 2020. Beta embeddings for multi-hop logical reasoning in knowledge graphs. *Advances in Neural Information Processing Systems*, 33:19716–19726.

Mohammed Saeed, Naser Ahmadi, Preslav Nakov, and Paolo Papotti. 2021. Rulebert: Teaching soft rules to pre-trained language models. *arXiv preprint arXiv:2109.13006*.

Marzieh Saeidi, Max Bartolo, Patrick S. H. Lewis, Sameer Singh, Tim Rocktäschel, Mike Sheldon, Guillaume Bouchard, and Sebastian Riedel. 2018. Interpretation of natural language rules in conversational machine reading. *CoRR*, abs/1809.01494.

Haitian Sun, Andrew Arnold, Tania Bedrax Weiss, Fernando Pereira, and William W Cohen. 2020. Faithful embeddings for knowledge base queries. *Advances in Neural Information Processing Systems*, 33:22505–22516.

Haitian Sun, William W. Cohen, and Ruslan Salakhutdinov. 2021a. Conditionalqa: A complex reading comprehension dataset with conditional answers. *CoRR*, abs/2110.06884.

| | |
|---|---|
| (Template) | |
| Context: If all (A, B), then U. | |
| If any (not C, D), then V. | |
| Facts: a, c, not d. | |
| Question: Is u correct? | |
| Label: entailed, if B | |
| (Variables) | |
| A: Aged 59 1/2 or older. | a: Tom is 65 years old. |
| B: Employed for two years. | b: NOT_USED |
| C: Has two children | c: He has two sons. |
| D: Has not applied before. | not d: Rejected last year. |
| U: Get at least $60 a week | u: Eligible for $60 a week. |
| V: Waive the application fees | v: NOT_USED |

Table 8: An example of CondNLI. Variables $A$, $B$, . . . and $U$, $V$, . . . represent the conditions and premises. Variables $a$, $b$, . . . represent the known facts. $u$ is the question. Each pair of variables, e.g. ($A$, $a$), is instantiated with an NLI example.

Haitian Sun, William W. Cohen, and Ruslan Salakhutdinov. 2021b. End-to-end multihop retrieval for compositional question answering over long documents. *CoRR*, abs/2106.00200.

Alon Talmor, Oyvind Tafjord, Peter Clark, Yoav Goldberg, and Jonathan Berant. 2020. Leap-of-thought: Teaching pre-trained models to systematically reason over implicit knowledge. *Advances in Neural Information Processing Systems*, 33:20227–20237.

Nikhil Verma, Abhishek Sharma, Dhiraj Madan, Danish Contractor, Harshit Kumar, and Sachindra Joshi. 2020. Neural conversational QA: Learning to reason vs exploiting patterns. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7263–7269, Online. Association for Computational Linguistics.

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122. Association for Computational Linguistics.

Michael J. Q. Zhang and Eunsol Choi. 2021. Situatedqa: Incorporating extra-linguistic contexts into qa.

Victor Zhong and Luke Zettlemoyer. 2019. E3: Entailment-driven extracting and editing for conversational machine reading. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2310–2320, Florence, Italy. Association for Computational Linguistics.

## A CondNLI Dataset Construction

We first construct templates for the CondNLI examples and then replace the variables in the template with real NLI examples.

**Construct Templates** We use capital letter variables $U$, $V$, . . . to represent the premises in the context, and $A$, $B$, . . . to represent the conditions. As discussed above, a premise is paired with a list of conditions and a logical operation. We express the relationship between the premise and the conditions with the statement "if ... then ...". For example in Table 8, we say "If all ($A$, $B$), then $U$" to represent that the premise $U$ has the conditions $A$ and $B$, and the logical operation "all".

Since the question is only about one of the premises in the context, we randomly sample a premise, e.g. $U$, and take its corresponding hypothesis $u$ as the question. With the question $u$, only the conditions of the premise $U$ need to be satisfied.

We also provide a list of facts that are used to check the entailment state of the conditions. To construct the facts, we randomly sample a subset of the conditions from the context, e.g. $\{A, C, D\}$, and take the facts of the selected conditions, e.g. $\{a, c, d\}$. Furthermore, we randomly add the term "not" to a fact, e.g. not $d$, to indicate that the fact $d$ contradicts with its condition $D$.

With the question, e.g. $u$, and the list of facts e.g. $\{a$, $c$, not $d\}$, we can infer the answer label and identify unsatisfied conditions. We keep the label "entailed", "contradicted", and "neutral", and add an additional label "irrelevant" if none of the premise in the context is relevant to the document.

**Generate Examples** For a templates with variables $A$, $B$, $U$, $V$, . . . , $a$, $b$, $u$, $v$, . . . , we instantiate the variables with NLI examples to get the real data. We use the premises of original NLI examples for premises or conditions, i.e. capital letter variables, and the hypothesis for question and facts, i.e. lower-case variables. Note that sampling requires matching the entailment state of conditions, e.g. "not $d$" requires sampling from NLI examples with the original label "contradict".

We restrict the number of conditions in the context to 6 and randomly generate 65 distinct templates.[11] During training, we randomly pick a template and instantiate it with NLI examples to generate real training examples. This random generation process enables creating (almost) unlimited amount of training data. We randomly generate another 5000 examples for development and testing.

## B ConditionalQA Data Preparation

Examples in the ConditionalQA dataset provide a parsed web page as context, a question, and a user

[11] Restricting the number of conditions is only for the purpose of reducing training complexity. The experiment in Figure 3 (left) shows the model's capability of generalizing to more conditions.

scenario that describes some relevant information about the question. We parse the provided context into the format that contains a list of tuples $\{(\{c_1^{(1)}, \dots, c_{k_1}^{(1)}\}, r^{(1)}, t^{(1)}), \dots\}$ as in §3.1.

The context in ConditionalQA is provided as a list of HTML elements. We treat each element at the leaf of the DOM tree as a condition, and all its parents (from its direct parent to the root) as the result. Conditions under the same parent are considered to be in the same list $\{c_1^{(i)}, \dots, c_{k_i}^{(i)}\}$. As discussed before, the logical operations $t^{(i)}$ need to be inferred from the context. We drop the field "type:" in the input in Eq. 1 and ask the model to discover it from the context and implicitly encode it into the condition embeddings $\mathbf{s}_j^{(i)}$. The question $q$ is the combination of the question and scenario.

## C Sharc Data Preparation

Different from ConditionalQA, where each sentence in the context is treated as a condition, conditions in the ShARC dataset are shorter and are sometimes short phrases (sub-sentence). For example, the context "If you are a female Vietnam Veteran with a child who has a birth defect, you are eligible for ..." contains two conditions, "If you are a female Vietnam Veteran" and "with a child who has a birth defect".[12] In order to handle sub-sentence conditions, we follow the strategy proposed in two of the baseline models, DISCERN (Gao et al., 2020b) and DGM (Ouyang et al., 2020), that split a sentence into EDUs (Elementary Discourse Units) using a pretrained discourse segmentation model (Li et al., 2018). The discourse segmentation model returns a list of sub-sentences, each considered as a condition.

While we could treat each condition independently as we did previously for other datasets, the segmented EDUs are different in that they are not full sentences and may not retain their semantic meaning. Thus, we jointly encode all EDUs $s_j^{(i)}$ as a single passage and select embeddings at specific tokens in the sentence as the condition embeddings $\mathbf{s}_j^{(i)}$. We construct the input $s$ for the entailment module as followed.

$$s = \text{“condition:”} + c_1^{(1)} + \cdots + c_{k_n}^{(n)} + \text{“question:”} + q$$

Similar to ConditionalQA, we drop the "type:" argument because the logical operation is not provided and needs to be inferred from the context. We additionally drop the argument "result:" and let the model to implicitly select EDUs (with the prefix "condition:") as the result. The input $s$ is used to compute condition embeddings. The condition embedding $\mathbf{s}_j^{(i)}$ for the EDU $c_j^{(i)}$ is the embedding at the start of each condition $c_j^{(i)}$.

$$\mathbf{s}_1^{(1)}, \dots, \mathbf{s}_{k_n}^{(n)} = \text{Entail}(s)$$

For the question generation task, we use the same input $s$ as in decision making, except that we replace the prefix "condition:" with "unsatisfied condition:" for "unsatisfied" conditions. We fine-tune a T5 model for question generation.

## D Dataset Statistics

Dataset statistics are shown in Table 9.

|  | Train | Dev | Test |
|---|---|---|---|
| ShARC | 15581 | 1622 | 5866 |
| ConditionalQA | 2338 | 285 | 804 |

Table 9: Dataset statistics.

[12] It is arguable that this could be generally treated as one condition, but it is treated as two conditions with the logical operator "all" in the ShARC dataset.