Graph Guided Question Answer Generation for Procedural Question-Answering

Hai X. Pham[1*] **Isma Hadji**[1] **Xinnuo Xu**[1] **Ziedune Degutyte**[1] **Jay Rainey**[1]
Evangelos Kazakos[1†] **Afsaneh Fazly**[2] **Georgios Tzimiropoulos**[1] **Brais Martinez**[1]
[1]Samsung AI Center, Cambridge [2]Samsung AI Center, Toronto

Abstract

In this paper, we focus on task-specific question answering (QA). To this end, we introduce a method for generating exhaustive and high-quality training data, which allows us to train compact (e.g., run on a mobile device), task-specific QA models that are competitive against GPT variants. The key technological enabler is a novel mechanism for automatic question-answer generation from procedural text which can ingest large amounts of textual instructions and produce exhaustive in-domain QA training data. While current QA data generation methods can produce well-formed and varied data, their non-exhaustive nature is sub-optimal for training a QA model. In contrast, we leverage the highly structured aspect of procedural text and represent each step and the overall flow of the procedure as graphs. We then condition on graph nodes to automatically generate QA pairs in an exhaustive and controllable manner. Comprehensive evaluations of our method show that: 1) small models trained with our data achieve excellent performance on the target QA task, even exceeding that of GPT3 and ChatGPT despite being several orders of magnitude smaller. 2) semantic coverage is the key indicator for downstream QA performance. Crucially, while large language models excel at syntactic diversity, this does not necessarily result in improvements on the end QA model. In contrast, the higher semantic coverage provided by our method is critical for QA performance.

1 Introduction

Asking questions is a natural way for humans to understand how to perform a task. Questions that pertain to a given procedure (i.e., a structured task such as cooking a recipe) encompass both factual questions about a given step (e.g., what tools are used in a given step), as well as questions that span across multiple steps (e.g., the order of steps). A smart AI agent should be able to handle both types of questions to assist humans.

While GPT models and competing alternatives have shown impressive results on multiple applications, including QA, they require large amounts of cloud computing resources due to their extreme sizes, thus being inviable as the QA models behind a smart assistant. We show that it is possible to train *task-specific* small models (e.g. suitable for running on a mobile phone), that at the same time are as accurate and complete as GPT variants on the target task.

High-quality in-domain training data is however required but, unfortunately, most QA datasets focus on general text comprehension where the answers can be spans from the text (Rajpurkar et al., 2016; Dunn et al., 2017; Joshi et al., 2017; Yang et al., 2018), free-style answers about a specific context (Nguyen et al., 2016; He et al., 2018) or obtained from a conversation history in conversational QA (Reddy et al., 2019). Similarly, collecting high-quality QA data at scale requires expensive labeling efforts. This motivates our paper, where we propose a method that ingests large quantities of procedural instructions (e.g. cooking recipes) and automatically generates extensive Procedural QA (PQA) training pairs that can be used to fine-tune a well-performing small language model.

In particular, the goal is to automatically generate PQA pairs that elicit information both from single sentences (or steps) in a procedure as well as information that requires reasoning over multiple steps to understand the temporal aspect of a procedure. While there have been efforts to create multi-modal QA datasets from recipes that require alignment between vision and text (Yagcioglu et al., 2018; Pustejovsky et al., 2021a), to the best of our knowledge, our work is the first that specifically concentrates on extracting a rich set of QA pairs from procedural text. We focus on cooking recipes as a type of procedural text. In a cooking sce-

* Corresponding author (pham.xuan.hai@outlook.com)
† E. Kazakos contributed to this work while at SAIC-C.

Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics
Volume 1: Long Papers, pages 2501–2525

nario, single sentence-based questions span local concepts (e.g., quantities of ingredients, cooking times, and tools), while temporal questions cover multiple steps (e.g., order of actions, the contents of mixtures at certain steps).

To tackle this problem, we propose to leverage the highly-structured nature of procedural text and represent the semantics of the procedure as graphs from which we can automatically generate PQA pairs. Specifically, to cover all question types pertaining to individual steps in a recipe, we rely on Abstract Meaning Representation (AMR) graphs (Banarescu et al., 2013). We perform a controlled set of transformations on the AMR graph of a step to generate a number of question AMRs and then generate questions from those AMRs using a pre-trained AMR-to-text model. For temporal questions that span across multiple steps, we start by converting the recipe into an action flow graph (Momouchi, 1980; Hamada et al., 2000; Yamakata et al., 2020) using a neural graph parser (Donatelli et al., 2021). We then extract all potential temporal answers by traversing the graph, and generating temporal question templates in the AMR space. We then, once again, rely on AMR-to-text models to generate corresponding questions.[1] Optionally, our approach can take advantage of LLMs (e.g., GPT3 (Brown et al., 2020)) to increase the syntactic diversity and semantic coverage of the generated questions, either by improving the wording and paraphrasing the generated questions or via directly replacing the graph-to-text generation model with a GPT-based solution that relies on content selected with our graph-guided approach.

Extrinsic evaluation shows the usefulness of our generated data in training question-answering models, which outperforms all considered baselines. Our results highlight the importance of devising an approach dedicated to generating QA pairs from procedural text, as we show small models (e.g., T5-base with around 220M parameters) can compete with GPT3 and ChatGPT (175B params) when finetuned on specialized high-quality data. In addition, intrinsic evaluation of our generated data demonstrates its superiority in terms of diversity, coverage, and overall quality, compared to data generated using several baselines including GPT3-based methods.

[1]We use action flow graphs rather than more recent work on multi-sentence DocAMR (Naseem et al., 2022) as DocAMR does not consider the temporal nature of procedural text.

Contributions. In summary the contributions of our paper are threefold:

- We tackle the problem of task-specific QA from procedural text and show that small models can compete with strong LLM baselines when provided with high-quality and exhaustive training data.
- We introduce a novel graph-based method for question-answer generation from procedural text. We draw on existing graph semantic formalisms, such as Abstract Meaning Representations (AMRs), and also take advantage of Action Flow graphs to represent the temporal relations among recipe steps. This allows us to rely on existing text-to-graph parsers as well as graph-to-text generative models, alleviating the need for specialized annotations. Notably, we also show that our method can take advantage of pre-trained LLMs to increase syntactic diversity and semantic coverage.
- We empirically show that our generated QA pairs can be used to train compact question-answering models (e.g., 60M or 220M parameters) that can compete with strong GPT-based baselines. Additionally, we show that the proposed method results in QA pairs with great diversity and high coverage (compared to human-generated question-answer pairs).

2 Related Work

Question generation is an important topic within the natural language generation community (Rus et al., 2010), where given a source text (i.e., context) and a target answer, the task is to generate the corresponding question. The answer is either provided (Song et al., 2017; Zhou et al., 2017; Zhao et al., 2018; Chai and Wan, 2020; Chan and Fan, 2019; Wang et al., 2020) or automatically extracted from the context (Golub et al., 2017; Scialom et al., 2019; Pyatkin et al., 2021; Liu et al., 2020). Our work follows the latter approach, where we automatically extract answers and generate corresponding questions. Moreover, compared to ACS-QG (Liu et al., 2020), our question generation method does not require additional clue and style information extracted from the input text, as they are represented in the semantic graph of the text.

Existing methods either rely on hand-crafted rules and templates (Heilman and Smith, 2010;

Rakshit and Flanigan, 2021; Fabbri et al., 2020; Pustejovsky et al., 2021a), or use annotated data (in the form of text spans as answers, along with corresponding ground-truth questions) to learn to automatically generate QA pairs (Patil, 2020; Gong et al., 2023; Golub et al., 2017; Scialom et al., 2019; Pyatkin et al., 2021). Rule-based methods offer more control over the generated data, but are not easily scalable. Learning-based methods offer better scalability, but require costly annotations. Our graph-based approach combines the benefits of the two while addressing their short-comings. Specifically, our graph-guided content selection offers the desired control over the content extracted from procedural text (to form the answer), and draws on generic models for graph-to-text generation to generate questions (Jacob, 2020).

Recent work has shown that Large Language Models (LLMs) such as GPT3 can be used for generating QA pairs based solely on the input context texts (Wang et al., 2021; Yuan et al., 2022). However, these models are sensitive to the prompt used to generate the data, offer less control over the generated QA pairs, and are not cost-effective. In contrast, we provide evidence that our graph-controlled QA generation approach yields high-quality and diverse data that can be used for training question answering models that compete with LLMs while being orders of magnitude smaller.

3 Methodology

In this section, we present our approach to QA generation from procedural text. We introduce the AMR and flow graphs that our method relies on (§3.1), and detail our method for generating questions from single instructions (§3.2) and temporal questions spanning across multiple instructions (§3.3). Lastly, we propose to use LLMs to improve the language quality of generated questions (§3.4).

3.1 Preliminaries

Abstract Meaning Representation (AMR): The AMR abstracts away the syntactic idiosyncrasies of language and instead draws out the logical meaning of text entities and their relations in a sentence, following the conventions of common framesets (Banarescu et al., 2013). Figure 1 depicts a recipe instruction (see caption) and its AMR graph, generated by a text-to-AMR parser (Jacob, 2020), in PENMAN notation. As can be seen in this example, the AMR graph specifies all entities (ingredients, tools, cooking time) and their relations (location, duration, manner) in a sentence. We draw on this representation to *exhaustively* identify contents to ask questions about for each individual step in a recipe (see §3.2).

```
(c / cook-01
    :mode imperative
    :ARG0 (y / you)
    :ARG1 (a / and
        :op1 (c2 / chicken)
        :op2 (ii / ingredient
            :mod (o / other)))
    :location (p / pot)
    :duration (t / temporal-quantity
        :quant 20
        :unit (m / minute))
    :manner (h / heat-01
        :mod (m2 / medium))
    :purpose (p2 / prepare-01
        :ARG0 y
        :ARG1 (s / soup)))
```

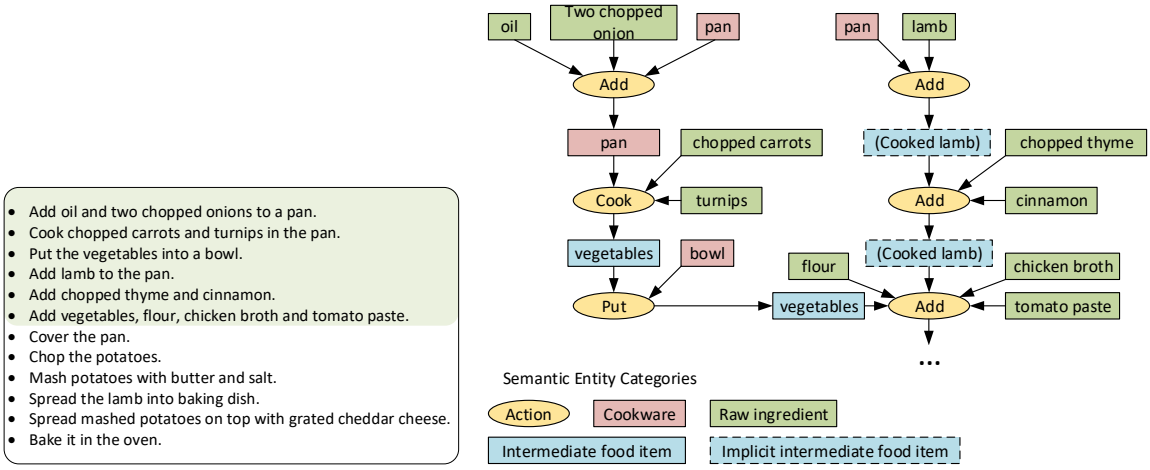
Figure 1: **AMR example.** Linearized AMR graph of the sentence *"Cook chicken and other ingredients in the pot over medium heat for 20 minutes to prepare the soup"*.

Flow Graphs: A flow graph (Momouchi, 1980; Hamada et al., 2000) is a directed acyclic graph containing actions, objects, other auxiliary entities (nodes) and their relations (edges), which provide essential information to complete a task. Importantly, flow graph relations encode the temporal order of actions and transformations (modifications/combinations) of objects. We draw on recent flow graph corpora and parsers (Yamakata et al., 2020; Donatelli et al., 2021) to generate action flow graphs such as the one shown in Fig. 2. We then use these graphs to generate questions that require understanding the temporal order of actions and object transformations over time (see §3.3).

3.2 Question generation from a single instruction

We extract the AMR graph for each sentence independently and generate three types of QA pairs from the graph; namely, role-specific, instruction-level, and polarity.

Role-specific QA. We begin by selecting the content that will serve as an answer from the AMR graph. The AMR graph consists of core and non-core roles. We select two main core roles (i.e.,



(a) "Shepherd pie" recipe (b) Semantic flow graph

Figure 2: **Flow graph example**. The action flow (sub-)graph of the highlighted text section in (a) is shown in (b) where word tokens are grouped together to form complete semantic entities belonging to one of the main categories. The semantic graph is further augmented with *implicit entities* to represent entities that are omitted from the text.

:ARG1, *:ARG2*) and several non-core roles (i.e., *:time*, *:duration*, *:location*, *:instrument*, *:mod*, *:domain*, *:purpose*, *:accompanier*, *:degree*, *:value* and *:quant*) to generate answers. Each role in the AMR consists of either a single concept (e.g., *:location* in Fig. 1) or a subgraph (e.g., *:ARG1*). For roles associated with a single concept, the concepts are used directly as our target answers, whilst for the latter, we use a pre-trained AMR-to-text model (Jacob, 2020) to convert the subgraph into a target answer.

To generate questions for each of the selected answers, we construct a corresponding *question* AMR. This is achieved by replacing the answer subgraph in the original AMR with the *amr-unknown* concept and transforming it into a proper AMR graph for natural question generation. The question is then generated using a graph-to-question model finetuned on generic question datasets (Rajpurkar et al., 2016; Pustejovsky et al., 2021b). The transformation algorithms for different roles as well as the graph-to-question generative model training are detailed in the appendix. Figure 3 shows examples of questions generated for different roles.

Instruction-level QA. Instruction-level questions are those for which the answer is the entire sentence. For example, given the instruction [*Slice the onion and coat in flour*] and the question [*How do I prepare the onion?*], the answer is the full instruction. In this category, we cover two types of questions: 1) "How do you [*do something*]?" and 2) "What do we do with [*something*]?". For the first type, the question AMR is created by adding a *:manner* role with *amr-unknown* concept to the original AMR. The second type requires transforming the original AMR into a new AMR in which all core roles (*:ARGx*) are grouped together into *:ARG2* to form [*something*], and the concept of *:ARG1* becomes *amr-unknown*. Once these transformations are applied we again use the AMR-to-text model to generate the questions.

Polarity "yes/no" QA. To generate questions with a "Yes" answer, we add a new node with the concept *amr-unknown* connected to the main verb root node with the *:polarity* role. To generate a question with a "No" answer from the same sentence, we further modify the resulting polarity question AMR by replacing one randomly chosen subgraph with a subgraph of the same semantic role sampled from another AMR.

3.3 Temporal question generation

We are also interested in questions about the transformation / composition of entities across time, as well as the temporal order of actions. These questions require content selection from multiple steps. We focus on three common types of temporal questions: 1) Composition of a mixture. For example in Fig. 2, one may ask about the ingredients that go into *vegetables*; 2) Next or preceding action. For example, in Fig. 2, one may want to know *what to do after putting vegetables into a bowl*. Note that in this case the fourth instruction is the start of another subtask, and the correct answer is the

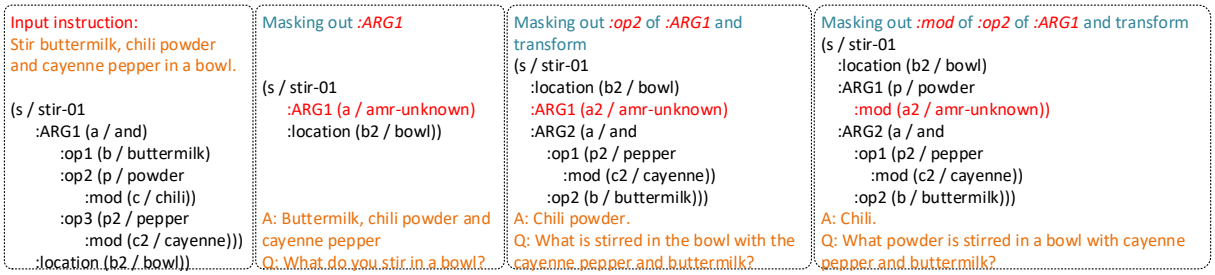


Figure 3: **Role-specific QA.** Three questions are created by targeting different roles in the input AMR.

sixth instruction. The correct answer is only clearly given the action flow graph; 3) The order of actions, e.g., *Is action A performed before/after action B?*.

To cover these question types, we propose a hybrid approach that relies on flow graphs for content selection and AMRs for question generation. More specifically, we create AMR-based templates for each of the question types, and traverse the flow graph to select answer contents (represented as AMR subgraphs extracted from related sentences) for each question type to fill in the templates. Finally, we use the AMR-to-text model to generate the questions. We adopt this hybrid strategy for two main reasons. First, temporal questions cannot be constructed directly by modifying the flow graph as was done for the sentence-based QA generation. Second, by composing the questions as AMR graphs, we can rely on the pre-trained AMR-to-text model to generate natural language questions. We now detail the approach adopted for each temporal question type.

Composition of mixtures QA. We design 12 question templates, represented as AMR graphs, that involve a [*mixture_name*], such as *"What are the ingredients of the **[mixture_name]**?"* In the example provided in Fig. 2b, mixture entities are indicated by cyan boxes in the flow graph. We only apply these templates on *named mixtures*, i.e. we ignore implicit items (dashed boxes in Fig. 2b) because it is not straightforward to assign names to such references. We then generate questions for each named mixture and traverse the flow graph to obtain the corresponding answer.

Next or preceding actions QA. We use a question AMR graph template equivalent to *"What do we do after/before **[action** A_j**]**?"* to generate questions. Then, at a particular action step A_j, the answer is either the next action in the flow graph, $A_k, k = \text{next}(j)$ for a "next" question, or the previous action in graph, $A_i, i = \text{pred}(j)$, for a "before" question. For example, in Fig. 2a given that $A_{j=7}$ is *"Cover the pan"*, the preceding action is $A_{i=6}$, *"Add vegetables, flour, chicken broth and tomato paste"*.

Order of actions QA. Here, we adopt two templates: *"Do we do A or do we do B first?"* that uses AMR *"or"* composition frame; or *"Doing A or doing B, which is first?"* which uses AMR *"amr-choice"* composition frame. We also swap A & B, so that for each pair of $\{A, B\}$ we can generate four questions. Once again, the answer is directly obtained by traversing the flow graph.

3.4 QA augmentation with LLMs

While the proposed graph-guided method offers a controlled solution to generate diverse QA pairs from procedural text with wide semantic coverage, it can nevertheless still benefit from strong LLMs. In particular, the language used in the QA pairs generated by the proposed method is tightly bound to the language used in the associated recipes. In contrast, humans tend to draw from their own vocabulary when posing questions. Thus, we also introduce two alternative methods to increase the syntactic diversity of the generated QA pairs using LLMs. We use the state-of-the-art GPT3 model.

Answer-based augmentation. As one of the strengths of our graph-guided approach is exhaustive content selection, one way to augment it with LLMs is to use answers generated with our approach and rely on GPT3 to generate corresponding questions (prompt details are provided in the appendix). However, we noticed that the questions generated with this approach sometimes do not semantically match the input answers. Therefore, we filter out the generated questions via round-trip consistency similar to (Alberti et al., 2019). In particular, we ask GPT3 to generate corresponding answers to the questions which it generated previously. We then compare the answer generated by GPT3 to the original answer in terms of ROUGE-1 metric and only keep the GPT3-based QA pair if

the score is > 0.25.

Paraphrasing-based augmentation. Another way to take advantage of LLMs' ability to generate diverse syntax is to directly task GPT3 with paraphrasing our graph-guided synthetic questions. Specifically, we paraphrase each question 5 times and filter out any duplicate questions.

4 Evaluation

We describe our experimental setup and question-answer generation baselines in §4.1, where we also introduce a new human-annotated PQA set (reference set) used for evaluation. §4.2 shows extrinsic evaluations, where question-answering models are trained with the generated data and evaluated on the aforementioned reference set. §4.3 describes intrinsic evaluations, assessing the distributions and quality of our generated QA pairs, which further amplifies the significance of our QA generation approach.

4.1 Experimental setup

Data. We use cooking recipes as a source of procedural text and randomly select 100 recipes from a popular food website (BBC). We use 70 recipes to generate QA pairs with our method and with the baselines. For evaluation purposes, we compile human-annotated QA pairs from the remaining 30 recipes and collected ~ 50 human-generated QA pairs per recipe. This yielded a PQA test set with 1857 QA pairs, where $\sim 30\%$ cover temporal questions that require reasoning over multiple steps in the recipe. We provide more details on the human data collection setup in the appendix.

Baselines. We propose a hybrid method for generating PQA pairs, relying on both graph-based logical rules, and trained deep generative models. We thus compare our work to state-of-the-art methods from each type of approach. Specifically, we compare to two rule-based methods (Pyatkin et al., 2021; Fabbri et al., 2020), and two learning-based methods for QG, including a T5-based model (Patil, 2020) finetuned on SQuAD (Rajpurkar et al., 2016) and a diffusion-based model (Gong et al., 2023). We also include comparisons to state-of-the-art LLMs. In a preliminary study, we found that GPT3 outperforms ChatGPT for the task of QA data generation so we use the former as a baseline. We consider two strategies, GPT3-sentence and GPT3-recipe, to generate QA pairs from sentences and entire recipes respectively. When evaluating the methods, including ours, we only consider questions that can be answered *solely* from the given context (i.e., recipe). A detailed description of all baselines considered is in the appendix.

4.2 Extrinsic evaluation

Question answering. We evaluate the usefulness of the generated data on the important application of question answering. We generate PQA pairs from the 70 recipes in the training set and use them to train a model for question answering. In particular, we target the application of open-ended QA, where given a question, q, and corresponding context, c, (i.e., recipe in this case), the goal is to generate the correct answer $a = \mathcal{F}(q, c)$. Here, $\mathcal{F}$ is a sequence-to-sequence model taking the concatenation of q and c as input and generating the answer, a. Since our goal is just to compare the different methods in terms of the quality of the training data generated, we use a T5-small model ($\sim 60M$ parameters), and finetune it with data obtained from each of the considered baselines. We also include a model finetuned on SQuAD (Rajpurkar et al., 2016), a widely-used large QA dataset, to illustrate performance when using generic QA data. Since we consider open-ended QA, we evaluate the generated answers using various language generation quality metrics, including BLEU, F1, ROUGE-L, and BLEURT (Sellam et al., 2020).

Results: The results summarized in Table 1 demonstrate the superiority of the data generated with our approach. Our method with only graph-to-text models (i.e. without LLM-based augmentations) outperforms all baselines on *all* metrics. Importantly, when used for question answering, this simplest variant of our method also outperforms baselines where GPT3 was used for data generation. These results suggest that the wide coverage of question types provided by our exhaustive content selection method plays a more significant role than the syntactic diversity of the generated language in the task of question answering. Finally, combining our graph-guided approach for improved coverage with GPT3, for improved language, yields the overall best results by a wide margin.

Question answering with larger models: Results summarized in Table 2 show that the generated data provides enough diversity and coverage to support the finetuning of a T5 model with up to 3B parameters, with performance gains consis-

	Generation method	BLEU	F1	ROUGE-L	BLEURT
	SQuADv2	6.2	23.8	23.4	34.0
Rule-based	Role-based QG (Pyatkin et al., 2021)	3.7	20.4	19.0	37.1
	Template-based QG (Fabbri et al., 2020)	6.8	25.4	25.1	35.7
Learning-based	Diffusion-based QG (Gong et al., 2023)	2.8	17.8	16.5	35.4
	T5-based QG (Patil, 2020)	5.9	27.6	27.0	38.6
GPT-based	GPT3-sentence	3.4	18.3	17.5	32.0
	GPT3-recipe	7.1	26.6	26.2	36.7
Ours	w/o augmentations	7.2	31.4	30.7	40.0
	w/ paraphrasing augmentation	7.3	33.5	32.6	42.0
	w/ answer-based augmentation	9.9	35.2	34.0	45.8

Table 1: **QA performance for different training data generation approaches**. A **T5-small** model was fine-tuned on different synthetic QA datasets and test results are computed on the human-annotated reference set.

Model	#params (B)	BLEU	F1	ROUGE-L	BLEURT
T5-small	0.06	7.2	31.4	30.7	40.0
T5-base	0.22	11.5	42.8	42.0	47.9
T5-large	0.77	13.9	45.0	44.0	47.9
T5-3B	3	16.7	45.5	44.7	51.2
FLAN-T5-XL		16.9	49.3	48.5	51.4
FLAN-T5-XL *(wP)*		21.8	54.0	53.1	56.3
FLAN-T5-XL *(wA)*		17.7	46.1	45.0	50.9
GPT-3	175	16.3	42.1	41.8	55.9
ChatGPT		17.3	41.6	41.0	56.2

Table 2: **QA performance for different model sizes and LLMs.** Different-sized T5 models, trained on our generated data *without LLM augmentations* unless explicitly mentioned: *(wP)* = w/ paraphrasing augmentation, *(wA)* = w/ answer-based augmentation. GPT3 and ChatGPT are the upper bound of the "generalist" QA approach. Performance is measured on the human-annotated reference set.

tently improving as a factor of the model's capacity. The results also show that smaller T5 models (e.g., T5-base with 0.22B parameters) already provide excellent performance, with the largest variants being competitive against (and even surpassing) the GPT3 and ChatGPT models, despite them being orders of magnitude larger and having been exposed to much larger amounts of data during training (including recipes that likely overlap with our test set). More interestingly, the FLAN-T5-XL model fine-tuned on our paraphrased data significantly outperforms GPT models, as well as the variant trained on answer-based augmented data. We attribute this substantial improvement to our proposed question graph transformations, which the LLM answer-based augmentation approach cannot benefit from. These transformations enrich the question pool diversity that models with larger capacity can effectively exploit, resulting in significant performance gain. More generally, we believe these results underscore the importance of devising approaches to generate domain-specific, high-quality data as proposed in this paper, especially when seeking a more favorable performance-vs-computational cost tradeoff on specific downstream tasks.

4.3 Intrinsic evaluation

Question diversity and coverage. We measure the diversity of generated questions in terms of *Dist-n*, the number of distinct n-grams (Li et al., 2016), and *n-gram Diversity*, calculated as $\frac{1}{N}\sum_{n=1}^{N}(\text{Dist-n}), N = 5$ (Wiher et al., 2022). We compute these metrics from the questions generated using the 70 recipes in the training split. On the other hand, *coverage* measures how well the human-generated questions in the reference set are

	Generation method	Dist-3 ↑	n-gram Div. ↑	Coverage ↑
Rule-based	Role-based QG (Pyatkin et al., 2021)	62.3	62.7	46.5
	Template-based QG (Fabbri et al., 2020)	81.1	80.4	40.5
Learning-based	Diffusion-based QG (Gong et al., 2023)	72.4	71.5	42.8
	T5-based QG (Patil, 2020)	74.7	74.3	45.6
GPT-based	GPT3 sentence	72.7	72.7	54.9
	GPT3 recipe	69.9	71.9	58.4
Ours	w/o augmentations	78.8	77.5	59.0
	w/ paraphrasing augmentation	78.3	77.6	67.3
	w/ answer-based augmentation	76.2	76.0	67.3

Table 3: **Intrinsic evaluation: question diversity & coverage.** Comparison of variants of our method and competing approaches in terms of diversity and coverage of the generated questions. Coverage is measured against the human-annotated reference set.

covered by questions automatically-generated from the same recipes. The coverage score is defined as $\frac{1}{N^{ref}} \sum_{i=1}^{N^{ref}} max_{j \in \tilde{N}} \rho(q_i^{ref}, \tilde{q}_j)$, where q_i^{ref} and $\tilde{q}_j$ denote the i^{th} question in reference set N^{ref} and j^{th} question in generated set $\tilde{N}$, respectively. We use BLEURT metric as the pair-wise scoring function ρ, thus the coverage score not only reflects semantic resemblance of generated questions w.r.t. human questions, but also their language naturalness and fluency.

Results: Table 3 shows our intrinsic experimental results and those from competing methods (a summary of all Dist-n scores, $n \in [1, 5]$, is included in the appendix). Even when relying solely on a simple graph-to-text generation model (i.e. no LLM augmentations), our method already far exceeds all QG baselines, including GPT3. In terms of diversity, our scores are slightly below those of Template-based QG (Fabbri et al., 2020). Different from all other methods including ours, in the Template-based QG approach, the questions are generated by shuffling parts of the original sentence to complete templates, therefore retaining most of the original n-gram diversity, however, the synthesized questions lack semantic adequacy and language fluency, reflected by low coverage score. Our method, with or without LLM augmetations, scores substantially better Dist-3 scores than other question generation methods, while also ensuring higher coverage, yielding overall best results. Notably, our method without any augmentation even has slightly better diversity scores than its LLM-augmented variants, indicating that our graph-based content selection approach, which is center amongst three variants, attributes to the richness and exhaustiveness of the synthetic questions. Paraphrasing augmentation, by fixing the language of generated text, further boosts the coverage score by 14% relative increase. Our variant with answer-based augmentation, although ensuring good coverage, has slightly lower diversity scores, because it lacks the question graph transformations employed by other two variants. In addition to the excellent QA performance of model trained on such data as demonstrated in our extrinsic evaluation, these results signify the quality of data generated by our proposed method.

Overall quality via human evaluation. We further conduct a human study to validate the quality of the generated questions, as well as the match between questions and answers. Specifically, we design a human annotation task where the rater assesses the generated questions in terms of grammatical correctness, adequacy, and answerability from the given context. Each entry was rated by 5 different raters, all of them native speakers. The corresponding answer is then revealed, and the rater is asked to judge the faithfulness and completeness of the answer. We assess all aspects on a 5-point Likert scale. Note that these metrics focus on the *quality* of the questions and answers, which is complementary to the diversity & coverage metrics. Details of the human annotation setup and process are in the appendix.

Results: We perform human study for three baselines, namely, the best rule-based and learning-based methods (according to Table 3) and a GPT-

Generation method	Q. Correct	Q. Adeq	Q. Answ	A. Faith	A. Compl
Role-based QG (Pyatkin et al., 2021)	2.65	2.58	2.98	3.15	2.63
T5-based QG (Patil, 2020)	2.95	3.10	2.63	2.73	1.68
GPT3 recipe	4.70	4.60	4.30	4.43	4.15
Ours w/o augmentations	3.35	3.58	3.40	3.63	2.83
Ours w/ anwer-based augmentation	4.73	4.60	4.55	4.43	4.08

Table 4: **Intrinsic evaluation: overall quality via human evaluation.** Questions are assessed for correctness (Correct), adequacy (Adeq), and answerability (Answ); Answers are assessed for faithfulness (Faith) and completeness (Compl). Scores are up to 5, higher is better.

based model, in addition to two variants of our method, one with graph-to-text generation, and one with GPT-based question generation. Table 4 shows that data generated with our approach fares well across all annotation aspects compared to other QG baselines. Methods leveraging GPT3, i.e., the "GPT3 recipe" and "Ours w/ answer-based augmentation", yield the highest quality by a wide margin, highlighting again the complementarity of the two components of our method.

5 Conclusion

In this paper we tackled task-specific QA from procedural text. To this end, we proposed a novel method for automatic generation of question-answer pairs from procedural texts in a comprehensive manner, both in their semantic content as well as syntactic diversity. We do so by exploiting the structured nature of the procedural data by using graph-based representations, and devise a systematic way of generating semantically-comprehensive question-answer pairs. We further enrich the syntactic correctness and diversity through the use of LLMs. We show that 1) using automatically-generated in-domain data to train a simple T5 model results in question-answering performance competitive with very large language models such as ChatGPT and GPT3. 2) our method results in excellent coverage of human-generated questions.

6 Limitations

Our proposed method heavily relies on AMRs and Flow Graph representations and thus our method is limited to the few languages supported. Multilingual support may become available once AMR sembank and flow graph corpus are expanded to support multiple languages. Furthermore, errors on the graph-parsing strategies are not mitigated within our method. Finally, we use a simple T5 with standard training to illustrate the performance for question-answering when training with data generated by our method. We believe there is room for further improvements by training more advanced models on our generated data.

Acknowledgements

We would like to thank the members of SAIC-Cambridge for their invaluable participation in the data collection process of the PQA reference set, as well as the human studies conducted in this work.

References

Chris Alberti, Daniel Andor, Emily Pitler, Jacob Devlin, and Michael Collins. 2019. Synthetic QA corpora generation with roundtrip consistency. In *Proc. of the Annual Meeting of the Association for Computational Linguistics*.

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2013. Abstract meaning representation for sembanking. In *Proc. of the Linguistic Annotation Workshop and Interoperability with Discourse*.

BBC. BBC good food. https://www.bbcgoodfood.com/about-bbc-good-food.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Conference on Neural Information Processing systems*.

Zi Chai and Xiaojun Wan. 2020. Learning to Ask More: Semi-Autoregressive Sequential Question Generation under Dual-Graph Interaction. In *Proceedings of the 58th Annual Meeting of the Association for*

Computational Linguistics, pages 225–237, Online. Association for Computational Linguistics.

Ying-Hong Chan and Yao-Chung Fan. 2019. A recurrent BERT-based model for question generation. In *Workshop on Machine Reading for Question Answering*.

Lucia Donatelli, Theresa Schmidt, Debanjali Biswas, Arne Köhn, Fangzhou Zhai, and Alexander Koller. 2021. Aligning Actions Across Recipe Graphs. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6930–6942, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Matthew Dunn, Levent Sagun, Mike Higgins, V. Ugur Güney, Volkan Cirik, and Kyunghyun Cho. 2017. Searchqa: A new Q&A dataset augmented with context from a search engine. *ArXiv*, abs/1704.05179.

Alexander Fabbri, Patrick Ng, Zhiguo Wang, Ramesh Nallapati, and Bing Xiang. 2020. Template-Based Question Generation from Retrieved Sentences for Improved Unsupervised Question Answering. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4508–4513, Online. Association for Computational Linguistics.

David Golub, Po-Sen Huang, Xiaodong He, and Li Deng. 2017. Two-Stage Synthesis Networks for Transfer Learning in Machine Comprehension. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 835–844, Copenhagen, Denmark. Association for Computational Linguistics.

Shansan Gong, Mukai Li, Jiangtao Feng, Zhiyong Wu, and Lingpeng Kong. 2023. DiffuSeq: Sequence to sequence text generation with diffusion models. In *International Conference on Learning Representations*.

Reiko Hamada, Ichiro Ide, Shuichi Sakai, and Hidehiko Tanaka. 2000. Structural analysis of cooking preparation steps in japanese. In *Proceedings of the Fifth International Workshop on on Information Retrieval with Asian Languages*, IRAL '00, page 157–164, New York, NY, USA. Association for Computing Machinery.

Wei He, Kai Liu, Jing Liu, Yajuan Lyu, Shiqi Zhao, Xinyan Xiao, Yuan Liu, Yizhong Wang, Hua Wu, Qiaoqiao She, Xuan Liu, Tian Wu, and Haifeng Wang. 2018. DuReader: a Chinese machine reading comprehension dataset from real-world applications. In *Proceedings of the Workshop on Machine Reading for Question Answering*.

Michael Heilman and Noah A. Smith. 2010. Good Question! Statistical Ranking for Question Generation. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 609–617, Los Angeles, California. Association for Computational Linguistics.

Brad Jacob. 2020. AMRLib: A python library that makes AMR parsing, generation and visualization simple. https://github.com/bjascob/amrlib.

Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. 2017. TriviaQA: A Large Scale Distantly Supervised Challenge Dataset for Reading Comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1601–1611, Vancouver, Canada. Association for Computational Linguistics.

Jiwei Li, Michel Galley, Chris Brockett, Jianfeng Gao, and Bill Dolan. 2016. A Diversity-Promoting Objective Function for Neural Conversation Models. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 110–119, San Diego, California. Association for Computational Linguistics.

Bang Liu, Haojie Wei, Di Niu, Haolan Chen, and Yancheng He. 2020. Asking questions the human way: Scalable question-answer generation from text corpus. In *Proceedings of The Web Conference 2020*, WWW '20, page 2032–2043, New York, NY, USA. Association for Computing Machinery.

Yoshio Momouchi. 1980. Control structures for actions in procedural texts and PT-chart. In *Conference on Computational Linguistics*.

Tahira Naseem, Austin Blodgett, Sadhana Kumaravel, Tim O'Gorman, Young-Suk Lee, Jeffrey Flanigan, Ramón Fernandez Astudillo, Radu Florian, Salim Roukos, and Nathan Schneider. 2022. DocAMR: Multi-sentence AMR representation and evaluation.

Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. 2016. MS MARCO: A human generated machine reading comprehension dataset. In *Conference on Neural Information Processing Systems - Workshops*.

Suraf Patil. 2020. Question generation using transformers. https://github.com/patil-suraj/question_generation.

James Pustejovsky, Eben Holderness, Jingxuan Tu, Parker Glenn, Kyeongmin Rim, Kelley Lynch, and Richard Brutti. 2021a. Designing multimodal datasets for nlp challenges. *arXiv:2105.05999*.

James Pustejovsky, Eben Holderness, Jingxuan Tu, Parker Glenn, Kyeongmin Rim, Kelley Lynch, and Richard Brutti. 2021b. Designing multimodal datasets for NLP challenges. *arXiv:2105.05999*.

Valentina Pyatkin, Paul Roit, Julian Michael, Yoav Goldberg, Reut Tsarfaty, and Ido Dagan. 2021. Asking It All: Generating Contextualized Questions for any

Semantic Role. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 1429–1441, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ Questions for Machine Comprehension of Text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.

Geetanjali Rakshit and Jeffrey Flanigan. 2021. ASQ: automatically generating question-answer pairs using AMRs. *arXiv:2105.10023*.

Siva Reddy, Danqi Chen, and Christopher D. Manning. 2019. CoQA: A conversational question answering challenge. *Transactions of the Association for Computational Linguistics*, 7:249–266.

Vasile Rus, Brendan Wyse, Paul Piwek, Mihai Lintean, Svetlana Stoyanchev, and Christian Moldovan. 2010. The first question generation shared task evaluation challenge. In *International Natural Language Generation Conference*.

Thomas Scialom, Benjamin Piwowarski, and Jacopo Staiano. 2019. Self-Attention Architectures for Answer-Agnostic Neural Question Generation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6027–6032, Florence, Italy. Association for Computational Linguistics.

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.

Linfeng Song, Zhiguo Wang, and Wael Hamza. 2017. A unified query-based generative model for question generation and question answering. *arXiv:1709.01058*.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford Alpaca: An instruction-following LLaMA model. https://github.com/tatsu-lab/stanford_alpaca.

Bingning Wang, Xiaochuan Wang, Ting Tao, Qi Zhang, and Jingfang Xu. 2020. Neural question generation with answer pivot. In *AAAI Conference on Artificial Intelligence*.

Shuohang Wang, Yang Liu, Yichong Xu, Chenguang Zhu, and Michael Zeng. 2021. Want to reduce labeling cost? GPT-3 can help. In *Conference on Empirical Methods in Natural Language Processing*.

Gian Wiher, Clara Meister, and Ryan Cotterell. 2022. On decoding strategies for neural text generators. *TACL*.

Semih Yagcioglu, Aykut Erdem, Erkut Erdem, and Nazli Ikizler-Cinbis. 2018. RecipeQA: A challenge dataset for multimodal comprehension of cooking recipes. In *Conference on Empirical Methods in Natural Language Processing*.

Yoko Yamakata, Shinsuke Mori, and John Carroll. 2020. English Recipe Flow Graph Corpus. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 5187–5194, Marseille, France. European Language Resources Association.

Antoine Yang, Antoine Miech, Josef Sivic, Ivan Laptev, and Cordelia Schmid. 2021. Just ask: Learning to answer questions from millions of narrated videos. In *International Conference on Computer Vision*.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380, Brussels, Belgium. Association for Computational Linguistics.

Xingdi Yuan, Tong Wang, Yen-Hsiang Wang, Emery Fine, Rania Abdelghani, Pauline Lucas, Hélène Sauzéon, and Pierre-Yves Oudeyer. 2022. Selecting better samples from pre-trained LLMs: A case study on question generation. *arXiv:2209.11000*.

Yao Zhao, Xiaochuan Ni, Yuanyuan Ding, and Qifa Ke. 2018. Paragraph-level Neural Question Generation with Maxout Pointer and Gated Self-attention Networks. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3901–3910, Brussels, Belgium. Association for Computational Linguistics.

Qingyu Zhou, Nan Yang, Furu Wei, Chuanqi Tan, Hangbo Bao, and M. Zhou. 2017. Neural question generation from text: A preliminary study. In *Natural Language Processing and Chinese Computing*.

A Summary

Here we provide all details. We begin by describing the PQA human data collection (i.e. reference set) and the human study setup in §B and §C, respectively. Next, we provide more detailed descriptions of the baselines considered in this work in §D. Finally, we provide further technical details about the proposed approach in §F.

B PQA reference dataset collection

Recall that we select 100 recipes from BBCfood (BBC), and randomly sample 70 recipes for training and 30 recipes are held off for the test set. For each recipe in the test (reference) set, we set up an interactive cooking simulation with two annotators, where one annotator asks questions that would help complete the task, and the other answers the questions. Both questions and answers are recorded and transcribed afterward. Question annotators are provided with detailed instructions, similar to the example shown in Listing 1. On the other hand, answer annotators are given the recipe text and ingredients and asked to interactively give the response to the question annotator. To ensure that QA pairs can be solely answered from the recipe, the answer annotator is instructed to reply with: "the recipe does not specify that", anytime a question asked cannot be answered from the recipe text alone.

C Human evaluation details

We design a human annotation task to evaluate the overall quality of questions and answers generated by a model. Specifically, we ask human raters to assess a set of generated questions with respect to grammatical correctness, adequacy, and answerability from a given context. For each question, the corresponding answer is then revealed to the rater, and they are asked to judge the faithfulness and completeness of the answer. Each question-answer pair is rated by 5 native English speakers. The human raters are provided with detailed annotation instructions shown in Listing 2.

D Question generation baselines

In this section, we provide further technical details for the methods used as baselines.

Rule-based QG (Pyatkin et al., 2021). Given a sentence, this method uses rules to generate questions for all semantic roles associated with a given entity, independently of the presence or absence of answers. Since this method is sentence-based, we use it to automatically generate questions for each step in a recipe. Also, since this method does not offer a solution for answer generation, we take the generated questions and accompanying recipe and use Alpaca (Taori et al., 2023) to automatically generate corresponding answers and filter out questions for which there is no answer in the recipe.

Template-based QG (Fabbri et al., 2020). A sentence is segmented into *[Fragment A]* + *[answer]* + *[Fragment B]* components. The *[answer]* component is replaced with the question *wh*-word, and re-combined with *[Fragment A]* and *[Fragment B]* in different orders to create questions. Then, similar to the role-based QG baseline, we use Alpaca to generate answers given the generated questions and context.

T5-based QG (Patil, 2020). Following previous work (Yang et al., 2021) we use a T5 model (Raffel et al., 2020) finetuned on the SQuAD dataset (Rajpurkar et al., 2016; Patil, 2020) for the task of question generation. Specifically, similar to previous work (Yang et al., 2021) we provide the finetuned model with the recipe text and let it automatically generate QA pairs.

Diffusion-based QG (Gong et al., 2023). We use a recent approach for non-autoregressive question generation based on text diffusion. We use Alpaca to generate corresponding answers.

GPT3-based QG (Brown et al., 2020). We consider GPT3 as an alternative method for question generation. We consider two variants: **(i)** GPT3-sentence, where we provide GPT3 with each step in the recipe independently and task it with generating *all* possible questions about its content. **(ii)** GPT3-recipe, where we give the entire recipe text as context and task GPT3 with generating *all* possible questions, with the goal of pushing GPT3 to ask temporal questions that span over multiple steps.

E Intrinsic evaluation

We provide all intrinsic evaluation scores in Table 5, including all Dist-n metrics, $n \in [1, 5]$, as a more complete version of Table 3 in the main paper. We also conduct paraphrasing experiments on all datasets generated by baseline methods. Notably, our method without any augmentation still

Listing 1 Example simulation instructions to elicit question-answer pairs for a recipe.

```
You are asked to cook using the following ingredients:

couscous, chargrilled artichokes, dijon mustard, olive oil, dill leaves,
↪  parsley leaves, lemon, watercress, sea bass fillets

There are 8 steps you need to finish.

Your task is to cook by interacting with the system.

You can ask any questions you have. For example:
    What ingredients do I need in the second step,
    Should I mix ingredient A with ingredient B?,
    Where should I put ...?", "How can I prepare ... ?,
    In step 3, I need to do ..., right?,
    How much ... do I need?,
    Do I need A or B?,
    When should I do ...?
    Why do I need to ...?
    Should I do A first or B first?,
    What ingredients do I need to prepare...?
    ...
The system will provide you with the necessary information.
You don't have to start with the first step, but to complete the task, you
↪  must receive a confirmation for each step of the recipe.

Note that:
    * Please imagine that you are in the kitchen, in front of all the
    ↪  ingredients and READY TO COOK.
    * You DO NEED detailed information for cooking, e.g. the order of putting
    ↪  ingredients, the place to put the ingredients, the amount of
    ↪  ingredients you need.
    * Try to ask different types of questions. For example, you are not
    ↪  encouraged to ask "what ingredients do I need for step N?"
    ↪  repetitively.
    * You can only ask general questions, like "what should I do next?", one
    ↪  time throughout the entire process.

```

Listing 2 Instructions provided to the human raters for assessing the overall quality of question and answer pairs.

```

RECIPE: {RECIPE_TEXT}
QUESTION: {QUESTION}
ANSWER: {ANSWER}

This task contains two phases. In the first phase, you need to read the RECIPE and the QUESTION above. You have to score the question on the basis of following three metrics. Note that, when scoring on the basis of one metric, please ignore the rest two entirely.

* Grammatical correctness: How well-phrased and grammatical is the question?
    - 1: absolutely grammatically incorrect
    - 2: mostly grammatically incorrect
    - 3: somewhat grammatically incoreect
    - 4: mostly grammatically correct
    - 5: absolutely grammatically correct

* Adequate: Does the question make sense in the context of the recipe?
    - 1: absolutely inadequate
    - 2: mostely inadequate
    - 3: somewhat adequate
    - 4: mostely adequate
    - 5: absolutely adequate

* Answerability: Is it possible to provide an answer to the question ONLY using the information provided in the recipe?
    - 1: absolutely not answerable
    - 2: mostely not answerable
    - 3: somewhat answerable
    - 4: mostely answerable
    - 5: absolutely answerable

Now, if all the numbers about are equal or more than 4, please continue the evaluation below. Otherwise, please click submit.

In the Second phase, you need to continuously read the ANSWER above. You have to score the answer on the basis of following two metrics. Note that, when scoring on the basis of one metric, please ignore the rest two entirely.

* Faithfulness: Does the answer ONLY contain the information provided in the recipe?
    - 1: absolutely not faithful
    - 2: mostely not faithful
    - 3: somewhat faithful
    - 4: mostely faithful
    - 5: absolutely faithful

* Answer's completeness: Does the provided answer completely address the question?
    - 1: completely fails to address the question
    - 2: mostely fails to address the question
    - 3: somewhat address the question
    - 4: mostely address the question
    - 5: completely address the question
```

	Generation Method	Dist-1	Dist-2	Dist-3	Dist-4	Dist-5	n-gram Div	Coverage
Rule-based	Role-based QG (Pyatkin et al., 2021)	98.7	81.1	62.3	43.9	27.5	62.7	46.5
	Role-based QG w/ paraphrasing	98.9	83.7	67.6	51.5	36.2	67.6	n/a
	Template-based QG (Fabbri et al., 2020)	95.4	90.3	81.1	72	63.2	80.4	40.5
	Template-based QG w/ paraphrasing	97.3	89.8	79.8	69.9	60.1	79.4	n/a
Learning-based	Diffusion-based QG (Gong et al., 2023)	95.2	85.9	72.4	58.7	45.4	71.5	42.8
	Diffusion-based QG w/ paraphrasing	97.1	87.4	75.2	62.9	50.6	74.6	n/a
	T5-based QG (Patil, 2020)	96.5	87.4	74.9	62.4	50	74.2	45.6
	T5-based QG w/ paraphrasing	98.2	88.1	76.4	64.6	52.9	76.0	50.9
GPT-based	GPT3 sentence	99.8	86.4	72.7	59.1	45.5	72.7	54.9
	GPT3 sentence w/ paraphrasing	99.2	87.7	75.4	63.1	50.9	75.3	57.8
	GPT3 recipe	99.6	94.9	69.9	54.9	40.1	71.9	58.4
	GPT3 recipe w/ paraphrasing	99.4	86.2	72.5	58.8	45.2	72.4	61.0
Ours	w/o augmentations	93.4	89.2	78.8	68.3	57.8	77.5	59.0
	w/ paraphrasing augmentation	96.7	89.0	78.3	67.5	56.7	77.6	67.3
	w/ answer-based augmentation	98.7	88.1	76.2	64.4	52.5	76.0	67.3

Table 5: **Intrinsic evaluation: question diversity and coverage.** This table shows all scores of all baseline generated datasets and their paraphrased supersets.

surpasses the paraphrased supersets of the baselines, showcasing the effectiveness of our proposed graph-based question generation method.

F Further technical details

F.1 Model learning

Graph-to-question generative model. While an off-the-shelf AMR-to-text generator (Jacob, 2020) works well on general sentences, it often fails to generate correct questions from question AMRs as we observed empirically. This problem may be due to the insufficiency of question data in the standard AMR datasets. To attenuate this issue and improve question generation performance, we fine-tune a T5-base model to generate questions from AMRs specifically. We parse questions taken from SQuAD (Rajpurkar et al., 2016) and R2VQ (Pustejovsky et al., 2021b) datasets into question AMRs and use the data to train our AMR-to-question model. We use the same training setting as AMRlib (Jacob, 2020), secifically we train the T5-base model for 8 epochs using AdamW optimizer, batch size $= 8$, starting learning rate $= 1e{-}4$ with linear schedule. The model was trained on a single 1080ti GPU in 20hrs.

Question-answering model training. We train one T5 models for each training dataset (QA pairs generated by one of the QG methods, including baselines and ours) using the same settings as follows: we train each model for 12 epochs using AdamW optimizer with $\beta = \{0.9, 0.999\}$, weight decay $= 0.01$, batch size $= 256$, starting learning rate $= 1e{-}5$ with cosine schedule. T5-small models were trained using eight 1080ti GPUs in under 3hrs. T5-base model was trained in one day using the same GPUs. T5-large and T5-3B/XL were trained for one and two days, respectively, using eight V100 GPUs.

F.2 GPT3 prompts details

We use GPT3 for several tasks: (i) as a baseline question generation model, with two variants of GPT3-sentence and GPT3-recipe as explained in §D above; and (ii) to augment our approach either as an alternative graph-to-text generation model (GPT3-QG), or as an added component to paraphrase the outputs of our graph-to-text generation module (GPT3-paraphrasing). For the GPT3-sentence and GPT3-recipe baselines, we follow up with a prompt to also elicit an answer (Answer Generation). We describe the prompts used for each case in Table 6.

F.3 Question generation from single instrutions

F.3.1 Role-specific QA

:ARG1 The general algorithm to generate ***all*** questions on different subgraphs under *:ARG1* is described in Algorithm 1.

:ARG1 splitting and regrouping. One of the limitations of graph-to-text generator is that, if the concept of *:ARG1* is a compound such as in the example of Figure 3 in the main text: "Stir buttermilk, chili powder and cayenne pepper in a bowl", then it is unable to generate question about a *:opX* role in the compound (e.g. :op1 (b/ buttermilk):

(s / stir-01 :ARG1 (a / and) :op1 (b / buttermilk) :op2 (p / powder :mod (c / chili)) :op3 (p2 / pepper :mod (c2 / cayenne))) :location (b2 / bowl))

Task	Prompt
GPT3-sentence	Sentence: {CONTEXT} Instruction: Read the above sentence, and ask {N_PAIR} different questions that can only be answered by referring to the given sentence.
GPT3-recipe	Recipe: {CONTEXT} Instruction: Read the recipe above, and ask {N_PAIR} different questions that can only be answered by referring to the given recipe.
GPT3-paraphrasing	Rewrite this sentence: {QUESTION_SENTENCE}
GPT3-QG	Context: {CONTEXT} Instruction: Read the above context, and ask {N_PAIR} different questions that can be answered as "{ANSWER}". Do not generate answers.
Answer Generation	Answer the question using information in the preceding background paragraph. If there is not enough information provided, answer with "The recipe does not specify"

Table 6: Prompts used to generate questions, answers, or paraphrases for the various GPT3-based models.

Our proposed solution is to transform the above AMR into

(s / stir-01 :ARG1 (b / buttermilk) :ARG2 (a / and) :op1 (p / powder :mod (c / chili)) :op2 (p2 / pepper :mod (c2 / cayenne))) :location (b2 / bowl))

then ask question about *:ARG1* by replacing "buttermilk" with *amr-unknown*. We can gradually apply a similar transformation for *:op2* and *:op3* in the above example. If *:ARG2* exists in the original sentence, we check whether *:ARG1* and *:ARG2* are semantically equivalent. If they are equivalent, the remaining *:opX* roles in *:ARG1* are merged with *:ARG2*. Otherwise, we convert the original *:ARG2* role into *:instrument* or *:location*, and then split *:ARG1*.

:ARG2 Directly placing *amr-unknown* on *:ARG2* would not work most of the time. We empirically found that, the graph-to-text generator is unable to generate correct question if the *amr-unknown* concept is placed directly on *:ARG2* role. This limitation may originate from AMR data that the model was trained on, which does not contain questions on *:ARG2*. Furthermore, in order to prepare question data to finetune the generator, we used the text-to-graph parser to parse questions in the R2VQ dataset into question AMRs, from which we observed that there was not any questions on *:ARG2*. Thus, we proposed a solution that swaps *:ARG1* and *:ARG2*, turning *:ARG2* into *:ARG1*, from which we can generate questions about *:ARG2* (now in the form of *:ARG1*). However, there are two major problems with swapping: 1) Whether the concept of *:ARG2* is a food item. If *:ARG2* describes a tool then swapping will invalidate the original sentence. 2) Whether the concepts of *:ARG1* and *:ARG2* are swappable - in other words, are they of equivalent roles?. For example, the sentence "Mix chicken with spices" and its transformed version "Mix spices with chicken" are semantically equivalent, but "Add spices to chicken" and "Add chicken to spices" are not.

To address the first problem, we create a filter to check if *:ARG2* is a tool or not. We do so by first gathering all *:instrument* concepts from the YouCook2 dataset, and during QA generation, we check if the concept of *:ARG2* is an instrument among the list, in that case we convert *:ARG2* core role to :instrument role and ask question on *:instrument* instead. To solve the second problem, we devise a set of rules to determine if *:ARG2* and *:ARG1* are semantically equivalent. Firstly, we check the verb if it implies moving direction or not. Such verbs include "add", "put", "pour", etc.. Secondly, because *:ARG2* typically follows a preposition, we check if the preposition is directional, ie. it's among "in", "on", "to", "into", "over". In such cases, we do not carry out swapping and instead convert *:ARG2* into *:location*, and ask question about :location as usual.

One example is shown in Figure 4.

:time The procedure is shown in Algorithm 2. Notably, in order to overcome the limitation of the AMR-to-text generator, we first remove all non-core roles from the AMR graph, except for *:time*.

Quantity (-quantity) concepts. This section applies to all quantity concepts, except *temporal-quantity* which often appears in :duration role. The procedure is shown in Algorithm 3. The key idea is to search for the *:quant* role within the subgraph of *-quantity* concept, and replace its concept (or

Algorithm 1 Generate questions about *:ARG1*

```
procedure GENERATE_ARG1_ATTRIBUTE_QUESTION
    ret = { }
    for role in :ARG1 concept sub-roles do
        if role = :mod then
            if exist_role(:quant) then:
                remove_role(:quant)
            end if
            ret.add(replace_amr_unknown(:mod))
        else if role = :quant then
            ret.add(replace_amr_unknown(:quant))
        end if
    end for
    return ret
end procedure
```

```
procedure GENERATE_ARG1_QUESTIONS
    ret = { }
    ret.add(replace_amr_unknown(:ARG1))
    if :ARG1 concept is single entity then
        ret.add(generate_ARG1_attribute_question())
    else if :ARG1 concept is compound then
        Split entities in :ARG1
        for each entity do
            Set entity as concept of :ARG1, other entities form :ARG2
            ret.add(replace_amr_unknown(:ARG1))
            ret.add(generate_ARG1_attribute_question())
        end for
    end if
    return ret
end procedure
```

- Original sentence:
We mix salt and chicken.
(m / mix-01
    :ARG0 (w / we)
    :ARG1 (s / salt)
    :ARG2 (c / chicken))
- Directly replace concept of ARG2 with amr-unknown:
(m / mix-01
    :ARG0 (w / we)
    :ARG1 (s / salt)
    :ARG2 (a / amr-unknown))
How much salt do we mix?
- Swap concepts of ARG1 and AGR2:
(m / mix-01
    :ARG0 (w / we)
    :ARG1 (a / amr-unknown)
    :ARG2 (s / salt))
What do we mix with salt?

Figure 4: Example questions generated for the concept of *:ARG2* role.

Algorithm 2 Gennerate questions about *:time*

```
Remove all roles except :ARG1, :ARG2, :time
if concept of :time starts with "until" then
    replace :time with :extent
    return replace_amr_unknown(:extent)
else
    return replace_amr_unknown(:time)
end if
```

rather, value) with *amr-unknown*. Note that, due to limitations of the text generator, we are unable to generate the correct question on ":quant" in a large graph, hence before that, we must simplify the graph.

Algorithm 3 Generate questions about *quantity*.

```
Remove all roles except :ARG1, :ARG2, :location, and the role in question.
return replace_amr_unknown(:quant)
```

Other roles The other supported roles are: *:duration*, *:location*, *:instrument*, *:mod*, *:domain*, *:purpose*, *:accompanier*, *:degree*, *:value* and *:quant*. Their questions are generated as described in Algorithm 4. With a few exceptions, we can simply replace the concept of a target role with *amr-unknown* to generate a question about that role.

Algorithm 4 Direct question generation for AMR roles.

```
if role = :mod then
    remove_role(:quant)
    return replace_amr_unknown(:mod)
else if role = :quant then
    Remove all roles except :ARG1, :ARG2, :location, and the target role.
    return replace_amr_unknown(:quant)
else
    return replace_amr_unknown(role)
end if
```

F.3.2 Instruction-level questions

There are two types of questions to ask *"how to do something"*:

- How do you [do something]?
- What do we do with [something]?

How do you [do something]? The procedure is described in Algorithm 5. The goal is achieved by adding the *:manner* role with *amr-unknown* concept. To overcome the limitation of AMR-to-text generation, we remove all non-core roles from the AMR graph before generating the question.

Algorithm 5 *"How"* question generation

```
Remove all non-ARG roles and corresponding concepts
return add_role(:manner, amr-unknown)
```

What do we do with [something]? We generate questions for the whole set of original entities in *:ARGx*, as well as every single entity. The procedure is described in Algorithm 6, and is summarized here:

- Grouping all *:ARGx* into *:ARG2*. This step basically combines all food items into one single compound defining *"something"*.
- Adding *:ARG1* with *amr-unknown* concept, to enable question generation.
- Replacing the current verb frame with *"do-02"* frame.

Algorithm 6 *"What do we do with [something]?"* question generation

```
ret = { }
Remove all non-ARG roles and corresponding concepts
if :ARG2 exists then
    merge all :ARGx into :ARG1
end if
replace_concept(graph_top, "do-02")
Rename :ARG1 → :ARG2
//generate one question about the entire compound in :ARG2
ret.add(add_role(:ARG1, amr-unknown)
//generate question about every single entity in :ARG2
Split entities in :ARG2
for each entity do
    Set entity as the sole concept of :ARG2 → new_graph
    ret.add(new_graph)
end for
return ret
```

Some examples are shown in Listing 3.

Listing 3 Examples of generating "What do we do...?" questions.

```
- The original instruction:
"Fry the coated chicken wings in oil at 350 degrees for 3-5 mins."

- Original graph:
(f / fry-01
      :mode imperative
      :ARG0 (y / you)
      :ARG1 (w / wing
            :part-of (c / chicken)
            :ARG1-of (c2 / coat-01))
      :ARG2 (o / oil)
      :location (t / temperature-quantity
            :quant 350
            :scale (c3 / celsius))
      :duration (b / between
            :op1 (t2 / temporal-quantity
                  :quant 3
                  :unit (m / minute))
            :op2 (t3 / temporal-quantity
                  :quant 5
                  :unit (m2 / minute))))

- Simplifying the graph:
(f / fry-01
      :mode imperative
      :ARG0 (y / you)
      :ARG1 (w / wing
            :part-of (c / chicken)
            :ARG1-of (c2 / coat-01))
      :ARG2 (o / oil))

- Question on all food items:
(f / do-02
    :ARG0 (y / you)
    :ARG2 (a / amr-unknown
        / and
        :op1 (w / wing
            :part-of (c / chicken)
            :ARG1-of (c2 / coat-01))
        :op2 (o / oil))
    :ARG1 a)
What do you do with a coated chicken wing and oil?

- Question on single entity:
(f / do-02
    :ARG0 (y / you)
    :ARG2 (w / wing
        :part-of (c / chicken)
        :ARG1-of (c2 / coat-01))
    :ARG1 (a / amr-unknown))
What do you do with a chicken's coated wings?
--------------------------------
(f / do-02
    :ARG0 (y / you)
    :ARG2 (o / oil)
    :ARG1 (a / amr-unknown))
What do you do with oil?
```

F.3.3 Polarity "yes/no" questions

The procedure is described in Algorithm 7. The key idea is to add a new node to the original AMR, with the concept of *amr-unknown* connected to the main verb node with the *:polarity* role.

F.4 Temporal question generation

F.4.1 Instructions & action graph

Figure 5 shows an example of a cooking recipe and its corresponding flow graph. As can be seen in the flow graph, the dependencies among actions and other cooking entities (e.g., ingredients and intermediate food items) do not necessarily follow the sequential order of the steps in the recipe.

F.4.2 Temporal question templates and examples

Composition of mixture. We design 12 question templates, listed below:

1. "What are the ingredients of the **{mixture_name}**?"
2. "What are the ingredients to prepare the **{mixture_name}**?"
3. "What are the ingredients required for the **{mixture_name}**?"
4. "What are the ingredients required to prepare the **{mixture_name}**?"
5. "What are the ingredients needed for the **{mixture_name}**?"
6. "What are the ingredients needed to prepare the **{mixture_name}**?"
7. "What is in the **{mixture_name}**?"
8. "What ingredients are in the **{mixture_name}**?"
9. "What ingredients go into the **{mixture_name}**?"
10. "What ingredients are for the **{mixture_name}**?"
11. "What ingredients make the **{mixture_name}**?"
12. "What do I need for the **{mixture_name}**?"

We only apply these question templates with a ***named mixture***, and ignore implicit mixtures and pronouns (such as "it" and "them"). The procedure is described in Algorithm 8. Some examples of questions generated from the graph in Figure 5 are shown in Listing 4.

Questions about preceding/next action. In this task we employ two templates:

- "What do we do before A_i?"
- "What do we do after A_i?".

The algorithm to generate "next" action is given in Algorithm 9. Notice in this algorithm, we limit A_k to those with $k > i$. Some examples are shown in Listing 5 and 6. To generate "before" question, we will find the previous action instead of the next one in the flow graph.

Questions about the order of actions. We adopt two templates:

- "Do we do A or do we do B first?": using AMR "or" composition frame.
- "Doing A and doing B, which is first?": using AMR "amr-choice" composition frame.

We also swap A & B, so for each pair of $\{A_i, A_j\}$ we can generate four questions. Full examples are shown in Listing 7.

Algorithm 7 "Yes/No" question generation

```
remove_role(:mode)
add_role(:ARG0, choice({"I","we","you"}))
add_role(:polarity, amr-unknown)
sample_and_replace(role) for role in orginal_AMR # for "No" question
```

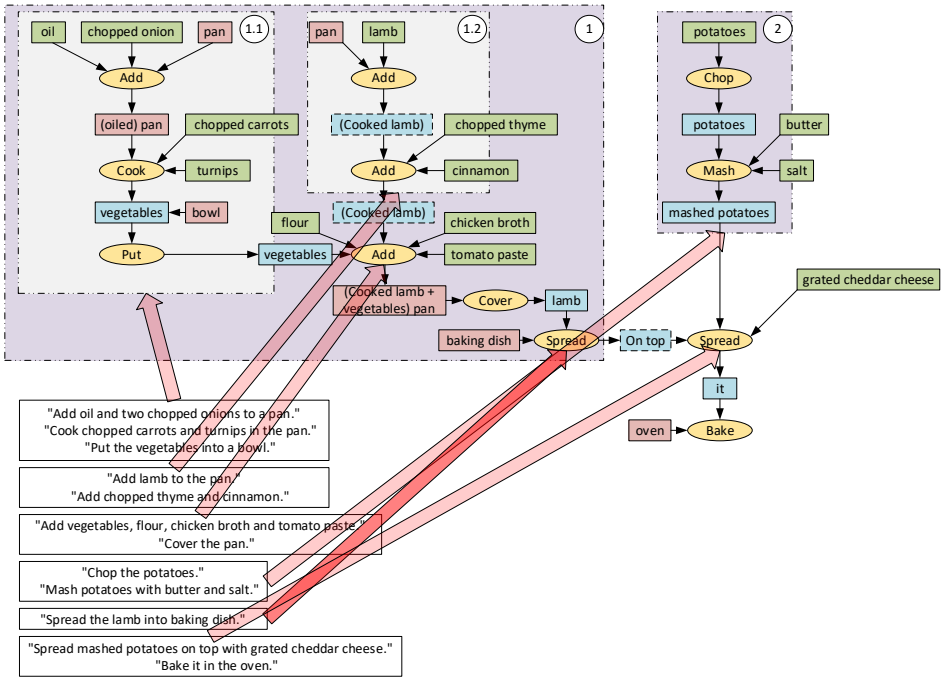


Figure 5: An example of a cooking recipe (divided into subtasks, each containing several instructions), and the corresponding flow graph (divided into subgraphs corresponding to each subtask). We can see that the recipe may be followed in a different order than the sequential ordering of the steps in the written recipe.

Algorithm 8 Generate questions about "mixture".

```
procedure GET_INGR_OF_MIXTURE(graph, mixture)
    prev_act_id = graph[mixture].prev_act_id
    action = graph[prev_act_id]
    ret = {}
    for ingr ∈ action.input do
        if action.input[ingr] < 0 then
            ret.add(ingr)
        else
            others = get_ingr_of_mixture(graph, ingr)
            if len(others) > 0 then
                ret.add(others)
            end if
        end if
    end for
    return ret
end procedure
procedure GENERATE_MIXTURE_QUESTION(graph, templates)
    ret = {}
    for action ∈ graph.action_with_mixtures() do
        for mixture ∈ action.mixtures() do
            ingrs = get_ingr_of_mixture(graph, mixture)
            answer = create_answer(ingrs)
            for template ∈ templates do
                question = create_question(template, mixture)
                ret.add((question, answer))
            end for
        end for
    end for
    return ret
end procedure
```

Listing 4 Examples of generating questions about a "mixture".

```
The original instruction:
"Put the vegetables into a bowl."

Q: What is the ingredient in vegetable preparation? (type 2)
(ii / ingredient
    :domain (a / amr-unknown)
    :purpose (p / prepare-01
        :ARG1 (v / vegetable)))

Q: What ingredients are required to prepare vegetables? (type 4)
(r / require-01
    :ARG1 (ii / ingredient
        :domain (a / amr-unknown))
    :purpose (p / prepare-01
        :ARG1 (v / vegetable)))


A: Chopped carrots and turnips.
(c / chop-01
    :ARG1 (a / and
        :op1 (c2 / carrot)
        :op2 (t / turnip)))
```

Algorithm 9 Generate questions about the next action.

procedure GENERATE_NEXT_ACTION_QUESTION(graph, templates)
  $ret = \{\}$
  for $action \in graph.actions()$ **do**
    $next\_actions = \{\}$
    if $action.next\_action \neq NULL$ **then**
      next_actions.add(action.next_action)
      $other\_actions = find\_prev\_actions(graph, action.next\_action)$
      for $a \in other_actions$ **do**
        if $a.id > action.id$ **then**
          next_actions.add(a)
        end if
      end for
    end if
    if $len(next\_actions) > 0$ **then**
      questions = create_question(templates, action)
      for $a \in next\_actions$ **do**
        answer = get_action(graph, a)
        for $question \in questions$ **do**
          ret.add((question, answer))
        end for
      end for
    end if
  end for
  return ret
end procedure

Listing 5 Examples of generating questions about the next action (from recipe in Fig. 5 above).

```
The instruction in focus (#7):
"Chop the potatoes."

Q: What will we do next?
(d / do-02
    :ARG0 (w / we)
    :ARG1 (a / amr-unknown)
    :time (n / next))

Q: What do we do after chopping potatoes?
(d / do-02
    :ARG0 (w / we)
    :ARG1 (a / amr-unknown)
    :time (a2 / after
        :op1 (c / chop-01
            :ARG1 (p / potato))))

A: Mash potatoes with butter and salt.
(m / mash-01
    :mode imperative
    :ARG0 (y8 / you)
    :ARG1 (p8 / potato)
    :accompanier (a10 / and
        :op1 (b4 / butter)
        :op2 (s / salt)))
```

Listing 6 Examples of generating questions about preceding action (from recipe in Fig. 5 above).

```
The instruction in focus (#10):
"Spread mashed potatoes on top with grated cheddar cheese."

Q: What do we do before spreading mash potatoes on top with grated cheddar
↪  cheese?
(d / do-02
    :ARG0 (w / we)
    :ARG1 (a / amr-unknown)
    :time (b / before
        :op1 (s / spread-01
            :ARG1 (p / potato
                :ARG1-of (m / mash-01))
            :ARG2 (t / top)
            :accompanier (c / cheese
                :mod c
                :mod (c2 / cheddar))
            :ARG1-of (g / grate-02))))

A: Mash potatoes with butter and salt.
```

Listing 7 Examples of generating "which is first" questions.

```
Instruction #0: Add oil and two chopped onions to a pan.
Instruction #1: Cook chopped carrots and turnips in the pan.

Question 1: "First, do we add oil and 2 chopped onions to the pan,
             or do we cook the chopped carrots and turnips in the pan?"
(o3 / or
    :op1 (a / add-02
        :ARG0 (w / we)
        :ARG1 (a2 / and
            :op1 (o / oil)
            :op2 (o2 / onion
                :quant 2
                :ARG1-of (c / chop-01)))
        :ARG2 (p / pan))
    :op2 (c4 / cook-01
        :ARG1 (a3 / and
            :op1 (c2 / carrot
                :ARG1-of (c3 / chop-03))
            :op2 (t / turnip))
        :location (p2 / pan)
        :ARG0 w)
    :polarity (a4 / amr-unknown)
    :ord (o4 / ordinal-entity
        :value 1))

Question 2: "First, add oil and 2 chopped onions to the pan,
             or cook the chopped carrots and turnip in the pan?"
(a / amr-choice
    :op1 (a3 / add-02
        :ARG1 (a2 / and
            :op1 (o / oil)
            :op2 (o2 / onion
                :quant 2
                :ARG1-of (c / chop-01)))
        :ARG2 (p / pan))
    :op2 (c4 / cook-01
        :ARG1 (a4 / and
            :op1 (c2 / carrot
                :ARG1-of (c3 / chop-03))
            :op2 (t / turnip))
        :location (p2 / pan))
    :ord (o3 / ordinal-entity
        :value 1))

Answer: "First, add oil and 2 chopped onions to the pan."
(a3 / add-02
    :ARG1 (a2 / and
        :op1 (o / oil)
        :op2 (o2 / onion
            :quant 2
            :ARG1-of (c / chop-01)))
    :ARG2 (p / pan)
    :ord (o3 / ordinal-entity
        :value 1))
```