Generating QA Pairs from Unstructured Text

Okarango Data Technologies, Chennai TN

Version History

Date	Version	Revision	Description	Author
March 11, 2020	1.0	1	Documentation for AQG *(AQG: Automatic Question Generator)*	Shubham Singh
March 12, 2020	2.0	0	Documentation for AQG *(AQG: Automatic Question Generator)*	Shubham Singh

Table of contents

1. Introduction

The text generation task that converts an input document into a model-generated hierarchy of question-answer (QA) pairs arranged in a top-down tree structure. Questions at higher levels of the tree are broad and open-ended, while questions at lower levels ask about more specific factoids. An entire document has multiple root nodes ("key ideas") that unfold into a forest of question trees.

2. Data preprocessing

We leverage the abundance of reading comprehension QA datasets (SQuAD, QuAC, CoQA) to train a pipelined system for the question generator. One major challenge is the lack of labeled hierarchical structure within existing QA datasets; to tackle this issue the question taxonomy of Lehnert (1978) is used to classify questions in these datasets as either GENERAL or SPECIFIC (question specificity). Then a neural question generation system is conditioned on these two classes, which enables the generation of both types of questions from a paragraph.

2.1 Rules for specificity classification

What makes one question more specific than another? The scheme for classifying question specificity maps each of the 13 conceptual question categories defined by Lehnert (1978) to three coarser labels: GENERAL, SPECIFIC, or YES-NO. As a result of this mapping, SPECIFIC questions usually ask for low-level information (e.g., entities or numerics), while GENERAL questions ask for broader overviews (e.g., "what happened in 1999?") or causal information (e.g, "why did..."). Many question categories can be reliably identified using simple templates and rules; A complete list is provided in the following Table.

Conceptual class	**Specificity**	**Question asks for...**	**Sample templates**
Causal Antecedent, Goal Oriented, Enablement, Causal, Consequent, Expectational	GENERAL	the reason for the occurrence of an event and the consequences of it	*Why...,* *What happened after / before...,* *What was the cause/reason/purpose...,* *What led to ...*
Instrumental	GENERAL	Procedure/Mechanism	*How* question with VERB parent for *How* in the dependency tree
Judgemental	GENERAL	a listener's opinion	Words like *you*, *your* present
Concept Completion, Feature Specification	GENERAL *or* SPECIFIC	Fill-in-the-blank information	*Where / When / Who...* ("SPECIFIC" templates)

Quantification	SPECIFIC	an amount	*How many / long...*
Verification, Disjunctive	YES-NO	Yes-No answers	The first word is VERB
Request	N/A	an act to be performed	(absent in datasets)

Table 1: The 13 conceptual categories of Lehnert (1978) and some templates to identify them and their specificity.

2.2 Classifying questions not covered by templates

Roughly half of all questions were classified with the templates and rules (Table 1) (`custom-squash/data/question_rules.py`); for the remaining half, a data-driven approach is taken. First, 1000 questions were manually labeled in QuAC using the *specificity* labels. This annotated data is then fed to a single-layer CNN binary classifier (Kim, 2014) using ELMo contextualized embeddings (Peters et al., 2018) (`custom-squash/data/question-classifier`). On an 85%-15% train-validation split, a high classification accuracy of 91% is achieved. The classifier also transfers to other datasets: on 100 manually labeled CoQA questions, with a classification accuracy of 80%. To obtain the final dataset (Table 2), the rule-based approach is used for all questions in SQuAD 2.0, QuAC, and CoQA and apply the classifier to label questions that were not covered by the rules.

Dataset	**Size**	**GENERAL**	**SPECIFIC**	**YES-NO**
SQuAD	86.8k	28.2%	69.7%	2.1%
QuAC	65.2k	34.9%	33.5%	31.6%
CoQA	105.6k	23.6%	54.9%	21.5%
All	257.6k	28.0%	54.5%	17.5%

Table 2: Distribution of classes in the final datasets.

In all datasets, all the unanswerable questions and questions whose answers span multiple paragraphs were removed. A few very generic questions (e.g. “what happened in this article?”) were manually identified and removed from the training dataset. Some other questions (e.g., “where was he born?”) are duplicated many times in the dataset; such questions were down-sampled to a maximum limit of 10 (`custom-squash/data/filter_dataset.py`).

3. Approach

The pipelined system (Figure 1) that takes a single paragraph as input and produces a hierarchy of QA pairs as output; for multi-paragraph documents, we process each paragraph independently of the rest. At a high level, the pipeline consists of five steps:

1. Answer span selection

2. Question generation conditioned on answer spans and specificity labels
3. Answering generated questions
4. Filtering out bad QA pairs
5. structuring the remaining pairs into a GENERAL-to-SPECIFIC hierarchy

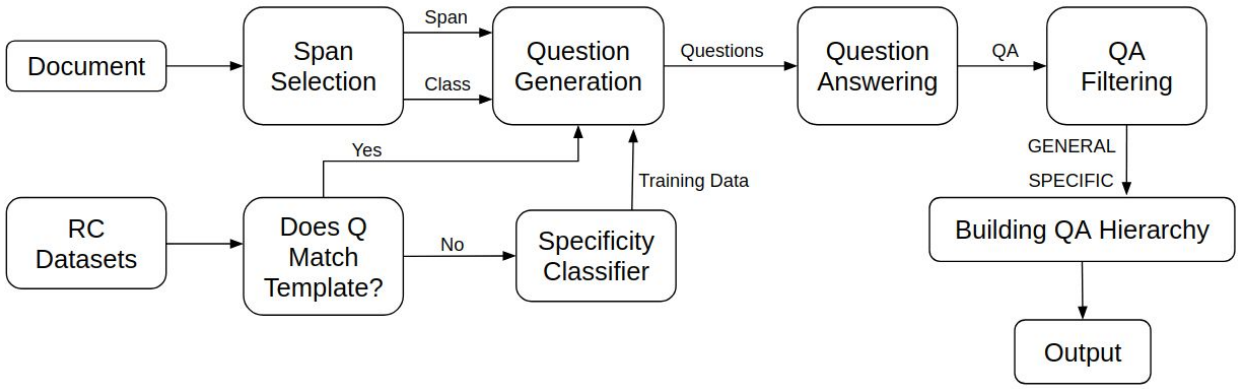

Figure 1: An overview of the process by which we generate a pair of GENERAL-SPECIFIC questions, which consists of feeding input data ("RC" is Reading Comprehension) through various modules, including a question classifier and a multi-stage pipeline for question generation, answering, and filtering.

3.1 Answer span selection

The pipeline begins by selecting an answer span from which to generate a question. To train the system, we can use ground-truth answer spans from the labeled datasets, but at test time how do we select answer spans? The solution is to consider all individual sentences in the input paragraph as potential answer spans (to generate GENERAL and SPECIFIC questions), along with all entities and numerics (for just SPECIFIC questions).

```
def get_answer_spans(para_text):
    para_nlp = nlp(para_text)
    sentences = [(x.text, x.start_char) for x in para_nlp.sents]

    entities = []
    entity_dict = {}
    for x in para_nlp.ents:
        if x.text in entity_dict:
            continue
        entity_dict[x.text] = 1
        entities.append((x.text, x.start_char))

    return sentences, entities
```

3.2 Conditional question generation

The question generation system is fine-tuned from a pretrained GPT-2 small model (Radford et al., 2019). This modified system is based on Wolf et al. (2019) and uses their code base as a starting point. To train the question generation model using the paragraph and answer as a language modeling context. For GENERAL questions, input sequence looks like "`<bos> ..paragraph text.. <answer-general> ..answer text.. <question-general> ..question text.. <eos>`" and equivalently for SPECIFIC questions. In addition, we leverage GPT-2's segment embeddings to denote the specificity of the answer and question. Each token in the input is assigned one out of five-segment embeddings (paragraph, GENERAL answer, SPECIFIC answer, GENERAL question and, SPECIFIC question). Finally, answer segment embeddings were used in place of paragraph segment embeddings at the location of the answer in the paragraph to denote the position of the answer in the paragraph. This is illustrated in the following figure. This codebase is forked from huggingface/transfer-learning-conv-ai: 🦄 State-of-the-Art Conversational AI with Transfer Learning.

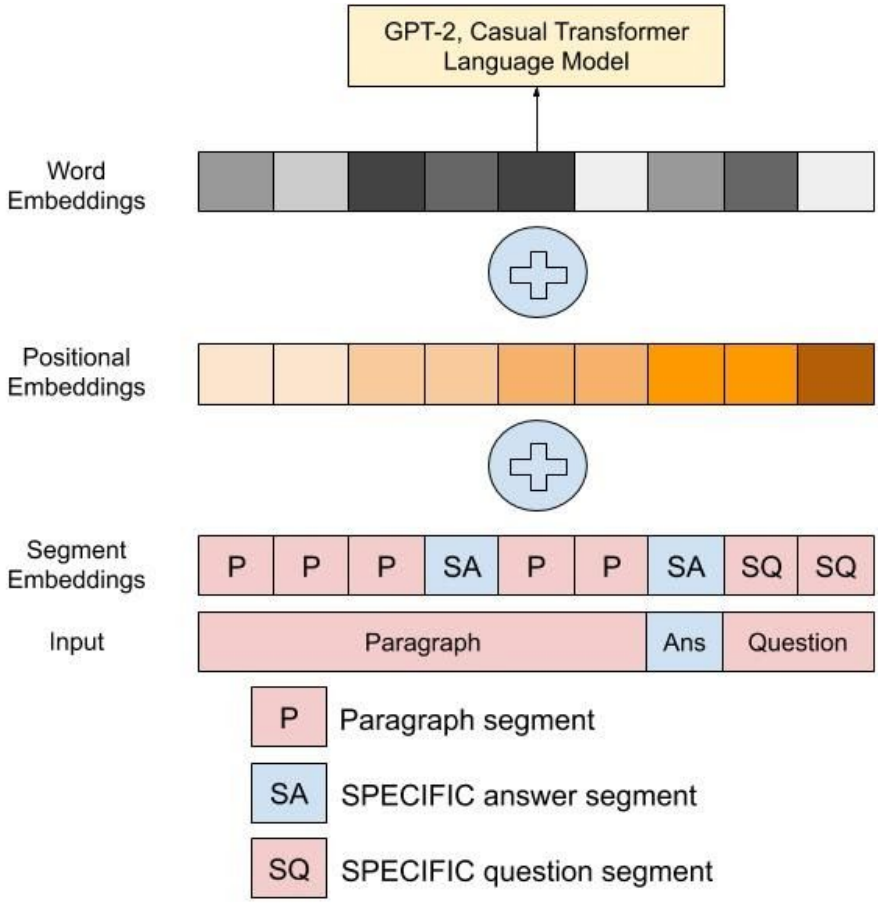

Figure 2: An illustration of the model used for generating a SPECIFIC question. A paragraph (context), answer and question and concatenated and the model is optimized to generate the

question. Separate segment embeddings are used for paragraphs, GENERAL answers, GENERAL questions, SPECIFIC answers and, SPECIFIC questions. Note that the answer segment embedding is also used within the paragraph segment to denote the location of the answer. (custom-squash/question-generation/train.py)

Model parameters:

Argument	Type	Default value	Description
dataset_path	`str`	`""`	`Path or url of the dataset. If empty download from S3.`
dataset_cache	`str`	`'./dataset_cache.bin'`	`Path or url of the dataset cache`
model	`str`	`"openai-gpt"`	`Path, url or short name of the model`
train_batch_size	`int`	`4`	`Batch size for training`
valid_batch_size	`int`	`4`	`Batch size for validation`
gradient_accumulation_steps	`int`	`8`	`Accumulate gradients on several steps`
lr	`float`	`6.25e-5`	`Learning rate`
lm_coef	`float`	`1.0`	`LM loss coefficient`
mc_coef	`float`	`1.0`	`Multiple-choice loss coefficient`
max_norm	`float`	`1.0`	`Clipping gradient norm`
n_epochs	`int`	`3`	`Number of training epochs`
device	`str`	`"cuda" if torch.cuda.is_available() else "cpu"`	`Device (cuda or cpu)`
fp16	`str`	`""`	`Set to O0, O1, O2 or O3 for fp16 training (see apex documentation)`
local_rank	`int`	`-1`	`Local rank for distributed training (-1: not distributed)`

Following is the script used to train the question-generator model.

custom-squash/schedulers/schedule_gpt2_corefs.sh

```
python -m torch.distributed.launch \
    --nproc_per_node=4 question-generation/train.py \
    --eval_before_start \
    --n_epochs 4 \
    --model_checkpoint gpt2 \
    --dataset_path data/temp_dataset/instances_corefs \
    --dataset_cache data/temp_dataset/cache_corefs \
    --output_dir question-generation/gpt2_corefs_question_generation
```

3.3 Answering generated questions

While the question generation model is conditioned on pre-selected answer spans, the generated questions may not always correspond to these input spans. Sometimes, the generated questions are either unanswerable or answered by a different span in the paragraph. By running a pre-trained QA model over the generated questions, we can detect questions whose answers do not match their original input spans and filter them out. The predicted answer for many questions has partial overlap with the original answer span; in these cases, the predicted answer span is displayed during the evaluation, as a qualitative inspection shows that the predicted answer is more often closer to the correct answer. For question answering BERT-based question answering module (Devlin et al., 2019) is being used which is trained on SQuAD 2.0 (Rajpurkar et al., 2018). We have used an open-source PyTorch implementation to train this model. The question answering module is a BERT-based model trained on SQuAD 2.0, forked from huggingface/transformers: 🤗 Transformers: State-of-the-art Natural Language Processing for TensorFlow 2.0 and PyTorch. The official code base for BERT question-answering model can be found here google-research/bert: TensorFlow code and pre-trained models for BERT.

Model parameters:

Argument	Type	Default Value	Description
bert_model	str	None	Bert pre-trained model selected in the list: bert-base-uncased, bert-large-uncased, bert-base-cased, bert-large-cased, bert-base-multilingual-uncased, bert-base-multilingual-cased, bert-base-chinese.

output_dir	str	None	The output directory where the model checkpoints and predictions will be written.
train_file	str	None	SQuAD json for training. E.g., train-v1.1.json
predict_file	str	None	SQuAD json for predictions. E.g., dev-v1.1.json or test-v1.1.json
max_seq_length	int	384	The maximum total input sequence length after WordPiece tokenization. Sequences longer than this will be truncated, and sequences shorter than this will be padded.
doc_stride	int	128	When splitting up a long document into chunks, how much stride to take between chunks.
max_query_length	int	64	The maximum number of tokens for the question. Questions longer than this will be truncated to this length.
do_train	str	None	Whether to run training.
do_predict	str	None	Whether to run eval on the dev set.
train_batch_size	int	32	Total batch size for training.
predict_batch_size	int	8	Total batch size for predictions.
learning_rate	float	5e-5	The initial learning rate for Adam.
num_train_epochs	float	3.0	Total number of training epochs to perform.
warmup_proportion	float	0.1	Proportion of training to perform linear learning rate warmup for. E.g., 0.1 = 10% of training.
n_best_size	int	20	The total number of n-best predictions to generate in the nbest_predictions.json output file.
max_answer_length	int	30	The maximum length of an answer that can be generated. This is needed because the start and end predictions are not conditioned on one another.

verbose_logging	-	-	If true, all of the warnings related to data processing will be printed. A number of warnings are expected for a normal SQuAD evaluation.
no_cuda	-	-	Whether not to use CUDA when available
seed	int	42	random seed for initialization
gradient_accumulation_steps	int	1	Number of updates steps to accumulate before performing a backward/update pass.
do_lower_case	-	-	Whether to lower case the input text. True for uncased models, False for cased models.
local_rank	int	-1	local_rank for distributed training on gpus
fp16	-	-	Whether to use 16-bit float precision instead of 32-bit.
loss_scale	float	0	Loss scaling to improve fp16 numeric stability. Only used when fp16 set to True. 0 (default value): dynamic loss scaling. Positive power of 2: static loss scaling value.
version_2_with_negative	-	-	If true, the SQuAD examples contain some that do not have an answer.
null_score_diff_threshold	float	0.0	If null_score - best_non_null is greater than the threshold predict null.
server_ip	str	''	Can be used for distant debugging.
server_port	str	''	Can be used for distant debugging.

3.4 Question filtering

After generating candidate questions from a single answer span, we use simple heuristics to filter out low-quality QA pairs. We remove generic and duplicate question candidates and pass the remaining QA pairs through the multistage question filtering process described below. Rules for question filtering is defined in `paragraph` class of `custom-squash/squash/filter.py` module.

Irrelevant or repeated entities: Top-k random sampling often generates irrelevant questions; we reduce their incidence by removing any candidates that contain nouns or entities unspecified in the passage. As with other neural text generation systems (Holtzman et al., 2018), we commonly observe repetition in the generated questions and deal with this phenomenon by removing candidates with repeated nouns or entities.

```
def remove_exact_duplicate_questions(self, input_list):
"""
Within each paragraph, ensure that every GENERAL/SPECIFIC predicted
question is unique.
"""
```

```
def remove_exact_duplicate_answers(self,
                                   Input_list,
                                   question_type="specific"):
"""
Within each paragraph, ensure that every GENERAL/SPECIFIC predicted answer
is unique.
"""
```

Unanswerable or low answer overlap: All candidates marked as “unanswerable” by the question answering model were removed, which prunes 39.3% of non-duplicate question candidates. These candidates are generally grammatically correct but considered irrelevant to the original paragraph by the question answering model. Next, we compute the overlap between the original and predicted answer span by computing word-level precision and recall (Rajpurkar et al., 2016). For GENERAL questions generated from sentence spans, we attempt to maximize recall by setting a minimum recall threshold of 0.3 Similarly, we maximize recall for SPECIFIC questions generated from named entities, with a minimum recall constraint of 0.8. Finally, for SPECIFIC questions generated from sentence spans, we set a minimum precision threshold of 1.0, which filters out questions whose answers are not completely present in the ground-truth sentence.

```
def calculate_overlap_scores(self):
    """
    For answers of each generated questions this function calculates the
    overlap between them.
    """
    for qa in self.original_qas:
        ans = qa['predicted_answer']
        gt_ans = qa['answers'][0]['text']
        # store all word overlap metrics between predicted answer and
```

```
original answer
        qa['exact_match'] = exact_match_metric(ans, gt_ans)
        qa['f1_match'] = f1_metric(ans, gt_ans)
        qa['recall_match'] = recall_metric(ans, gt_ans)
        qa['precision_match'] = precision_metric(ans, gt_ans)
        qa['unanswerable'] = qa['predicted_answer'] == ''
```

Low generation probability: If multiple candidates remain after applying the above filtering criteria, then the most probable candidate is selected for each answer span. SPECIFIC questions generated from sentences are an exception to this rule: for these questions, the ten most probable candidates selected, as there might be multiple question worthy bits of information in a single sentence. If no candidates remain, in some cases we use a fallback mechanism that sequentially ignores filters to retain more candidates.

3.5 Forming a QA hierarchy

The output of the filtering module is an unstructured list of GENERAL and SPECIFIC QA pairs generated from a single paragraph. To form a meaningful hierarchy First, a parent for each SPECIFIC question is chosen by maximizing the overlap (word-level precision) of its predicted answer with the predicted answer for every GENERAL question. If a SPECIFIC question's answer does not overlap with any GENERAL question's answer (e.g., "Dagobah" and "destroy the Sith"), then it is mapped to the closest GENERAL question whose answer occurs before the SPECIFIC question's answer ("What happened in the battle ...?").

4. Evaluation

The pipeline is evaluated on documents from the QuAC development set using a variety of crowdsourced experiments. Concretely, we evaluate the quality and relevance of individual questions, the relationship between generated questions and predicted answers, and the structural properties of the QA hierarchy.

Experiment	Generated		Gold	
	Score	Fleiss κ	Score	Fleiss κ
Is this question well-formed?	85.8%	0.65	93.3%	0.54
Is this question relevant?	78.7%	0.36	83.3%	0.41
(among *well-formed*)	81.1%	0.39	83.3%	0.40
Does the span *partially* contain the answer?	85.3%	0.45	81.1%	0.43
(among *well-formed*)	87.6%	0.48	82.1%	0.42
(among *well-formed and relevant*)	94.9%	0.41	92.9%	0.44
Does the span *completely* contain the answer?	74.1%	0.36	70.0%	0.37
(among *well-formed*)	76.9%	0.36	70.2%	0.39
(among *well-formed and relevant*)	85.4%	0.30	80.0%	0.42

Table 3: Human evaluations demonstrate the high individual QA quality of our pipeline's outputs. All inter-annotator agreement scores (Fleiss κ) show "fair" to "substantial" agreement (Landis and Koch, 1977).

4.1 Individual question quality and relevance

The first evaluation measures whether questions generated by the system are well-formed (i.e., grammatical and pragmatic). Ask crowd workers whether or not a given question is both grammatical and meaningful (As "meaningful" is potentially a confusing term for crowd workers, they were asked only for grammatical correctness and achieved very similar results). For this evaluation, we acquire judgments for 200 generated QA pairs from the QuAC validation set (with an equal split between GENERAL and SPECIFIC questions). The first row of Table 3 shows that 85.8% of generated questions satisfy this criterion with a high agreement across workers. Question relevance: How many generated questions are actually relevant to the input paragraph? While the percentage of unanswerable questions that were generated offers some insight into this question, we removed all of them during the filtering pipeline (Section 3.4). Hence, an input paragraph and generated question displayed to crowd workers (using the same data as the previous well-formedness evaluation) and asked whether or not the paragraph contains the answer to the question. The second row of Table 3 shows that 78.7% of the questions are relevant to the paragraph.

4.2 Individual answer validity

Is the predicted answer actually a valid answer to the generated question? In the filtering process, we automatically measured the answer overlap between the input answer span and the predicted answer span and used the results to remove low-overlap QA pairs. To evaluate answer recall after filtering, a crowdsourced evaluation is performed on the same 300 QA pairs as above by asking crowd workers whether or not a predicted answer span contains the answer

to the question. We also experiment with a more relaxed variant (partially contains instead of completely contains) and report results for both task designs in the third and fourth rows of Table 3. Over 85% of predicted spans partially contain the answer to the generated question, and this number increases if we consider only questions that were previously labeled as well-formed and relevant.

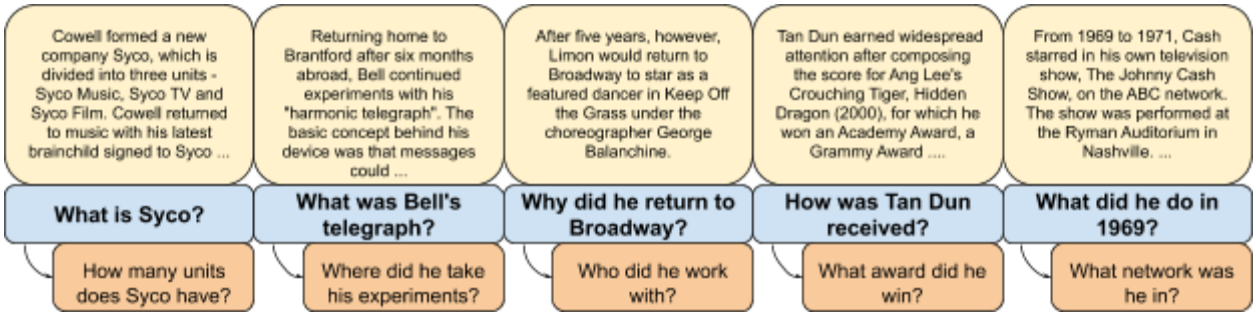

Figure 4: Question hierarchies generated by the system with reference snippets. Questions in the hierarchy are of the correct specificity class (i.e., GENERAL , SPECIFIC).

4.3 Structural correctness

To examine the hierarchical structure of generated QA pairs, three experiments were conducted.

How faithful are output questions to input specificity? First, it is investigated whether the model is actually generating questions with the correct specificity label. We run the specificity classifier (Section 2) over 400 randomly sampled questions (50% GENERAL, 50% SPECIFIC and, no duplicates) and obtain a high classification accuracy of 91%. This automatic evaluation suggests the model is capable of generating different types of questions.

A**re GENERAL questions more representative of a paragraph than SPECIFIC questions?** To see if GENERAL questions really do provide higher-level information, we sample 200 GENERAL-SPECIFIC question pairs grouped together as described in Section 3.5. For each pair of questions (without showing answers, crowd workers were asked to choose the question which, if answered, would give them more information about the paragraph. As shown in Table 4, in 89.5% instances the GENERAL question is preferred over the SPECIFIC one, which confirms the strength of specificity-controlled question generation system.

Experiment	Score	Fleiss κ
Which question type asks for more information?	89.5%	0.57
Which SPECIFIC question is closer to GENERAL QA?		
different paragraph	77.0%	0.47
same paragraph	64.0%	0.30

Table 4: Human evaluation of the structural correctness of the system. The labels "different/same paragraph" refer to the location of the intruder question. The results show the accuracy of specificity and hierarchies.

In a pilot study, asking workers "Which question has a longer answer?", observed a higher preference of 98.6% for GENERAL questions.

How related are SPECIFIC questions to their parent GENERAL question? Finally, the effectiveness of the question grouping strategy was investigated, which binds multiple SPECIFIC QA pairs under a single GENERAL QA pair. We show crowd workers a reference GENERAL QA pair and ask them to choose the most related SPECIFIC question given two choices, one of which is the system's output and the other an intruder question. We randomly select intruder SPECIFIC questions from either a different paragraph within the same document or a different group within the same paragraph. As shown in Table 4, crowd workers prefer the system's generated SPECIFIC question with higher than random chance (50%) regardless of where the intruder comes from. As expected, the preference and agreement is higher when intruder questions come from different paragraphs, since groups within the same paragraph often contain related information (Section 5.2).

5. Qualitative Analysis

In this section, we analyze outputs (Figure 4, Figure 5) of the pipeline and identify its strengths and weaknesses.

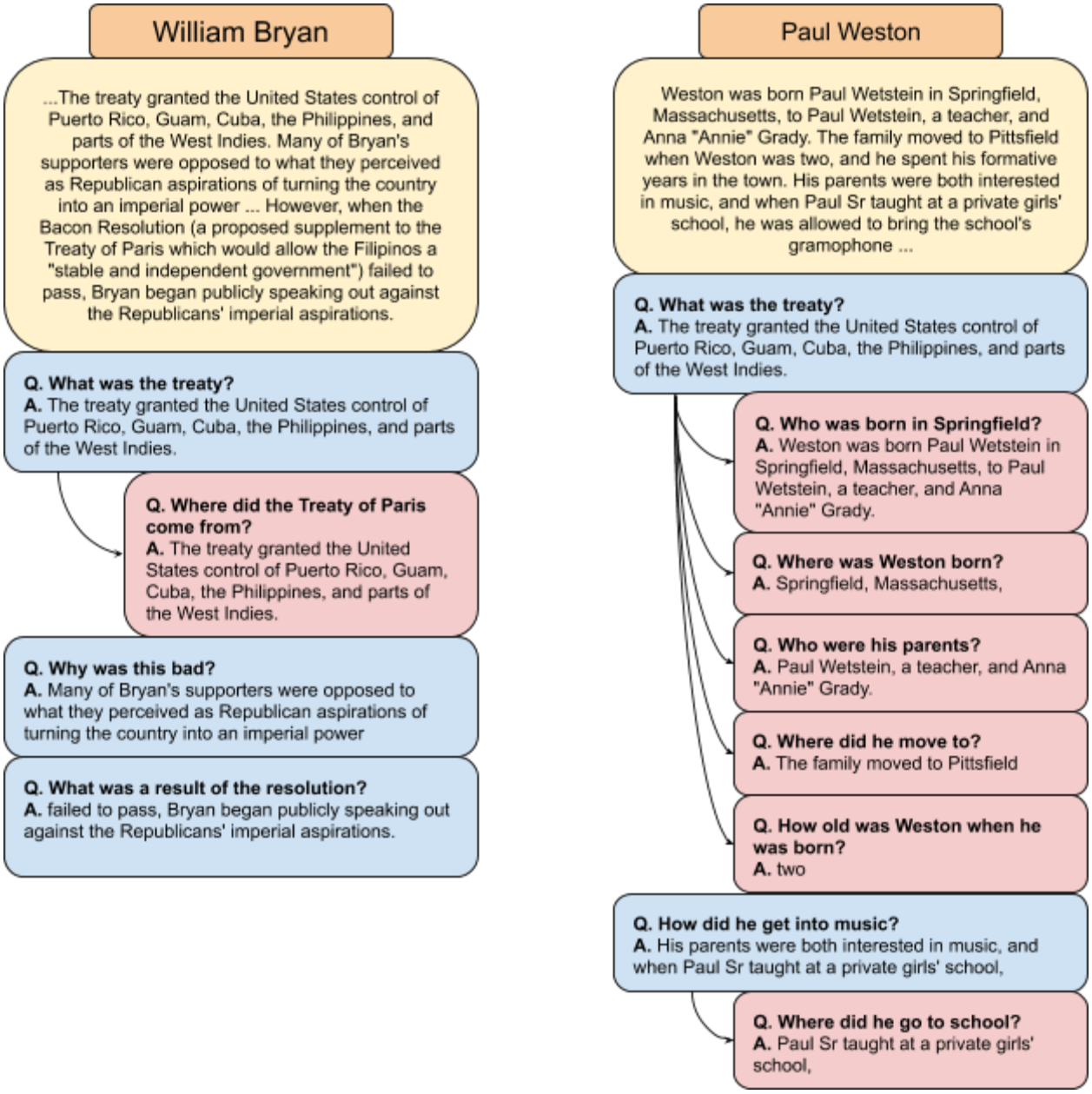

Figure 5: Two outputs generated by the system. The William Bryan example has interesting GENERAL questions. The Paul Weston example showcases several mistakes the model makes.

5.1 What is the pipeline good at?

Meaningful hierarchies: The method of grouping the generated questions (Section 3.5) produces hierarchies that clearly distinguish between GENERAL and SPECIFIC questions;

Figure 4 contains some hierarchies that support the positive results of the crowdsourced evaluation.

Top-k sampling: Similar to prior work (Fan et al., 2018; Holtzman et al., 2019), we notice that the beam search often produces generic or repetitive beams (Table 5). Even though the *top-k* scheme always produces lower-probable questions than beam search, the filtering system prefers a *top-k* question **49.5%** of the time.

	*"In **1942**, Dodds enlisted in the US army and served as an anti aircraft gunner during World War II."*
B	In what year did the US army take place? In what year did the US army take over? In what year did the US army take place in the US?
T	What year was he enlisted? When did he go to war? When did he play as anti aircraft?

Table 5: Beam Search (B) vs Top-k sampling (T) for SPECIFIC question generation. Top-k candidates tend to be more diverse.

5.2 What kind of mistakes does it make?

We describe the various types of errors the model makes in this section, using the Paul Weston output in Figure 5 as a running example.

Reliance on a flawed answering system: Pipeline's output is tied to the quality of the pretrained answering module, which both filters out questions and produces final answers. QuAC has long answer spans (Choi et al., 2018) that cause low-precision predictions with extra information (e.g., "Who was born in Springfield?"). Additionally, the answering module occasionally swaps two named entities present in the paragraph.

Redundant information and lack of discourse: In the system, each QA pair is generated independently of all the others. Hence, Sometimes the outputs lack an inter-question discourse structure. The system often produces a pair of redundant SPECIFIC questions where the text of one question answers the other (e.g., "Who was born in Springfield?" vs. "Where was Weston born?"). These errors can likely be corrected by conditioning the generation module on previously-produced questions (or additional filtering).

Lack of world knowledge: The models lack commonsense knowledge ("How old was Weston when he was born?") and can misinterpret polysemous words. Integrating pretrained contextualized embeddings (Peters et al., 2018) into the pipeline is one potential solution.

Multiple GENERAL QA per paragraph: System often produces more than one tree per paragraph, which is undesirable for short, focused paragraphs with a single topic sentence. To improve the user experience, it might be ideal to restrict the number of GENERAL questions we show per paragraph. While we found it difficult to generate GENERAL questions representative of entire paragraphs, a potential solution could involve identifying and generating questions from topic sentences.

Coreferences in GENERAL questions: Many generated GENERAL questions contain coreferences due to the contextual nature of the QuAC and CoQA training data ("How did he get into music?"). Potential solutions could involve either constrained decoding to avoid beams with anaphoric expressions or using the CorefNQG model of Du and Cardie (2018).

5.3 Which models did not work?

This includes:

1. End-to-end modelling to generate sequences of questions using QuAC.
2. Span selection NER system.
3. Generation of GENERAL questions representative of entire paragraphs
4. Answering system trained on the combination of QuAC, CoQA, and SQuAD.

6. Related work

This work is related to research in three broad areas: question generation, information retrieval, and summarization.

Question Generation: This work builds upon neural question generation systems (Du et al., 2017; Du and Cardie, 2018). The pipeline conditions generation on specificity, similar to difficulty conditioned question generation (Gao et al., 2018). QA pair generation has previously been used for dataset creation (Serban et al., 2016; Du and Cardie, 2018). Joint modeling of question generation and answering has improved the performance of individual components (Tang et al., 2017; Wang et al., 2017; Sachan and Xing, 2018) and enabled visual dialog generation (Jain et al., 2018).

Information Retrieval: The hierarchies generated are related to interactive retrieval setting (Hardtke et al., 2009; Brandt et al., 2011) where similar webpages are grouped together. The pipeline is also related to exploratory (Marchionini, 2006) and faceted search (Yee et al., 2003).

Summarization: This work is related to query-focused summarization (Dang, 2005; Baumel et al., 2018) which conditions an output summary on an input query. Hierarchies have also been applied to summarization (Christensen et al., 2014; Zhang et al., 2017; Tauchmann et al., 2018.

7. References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In Proc. International Conference on Learning Representations (ICLR).

Tal Baumel, Matan Eyal, and Michael Elhadad. 2018. Query focused abstractive summarization: Incorporating query relevance, multi-document coverage, and summary length constraints into seq2seq models. arXiv preprint arXiv:1801.07704.

Christina Brandt, Thorsten Joachims, Yisong Yue, and Jacob Bank. 2011. Dynamic ranked retrieval. In Proceedings of the fourth ACM international conference on Web search and data mining.

Orkut Buyukkokten, Hector Garcia-Molina, and Andreas Paepcke. 2001. Seeing the whole in parts: text summarization for web browsing on handheld devices. In Proceedings of the 10th international conference on World Wide Web.

Eunsol Choi, He He, Mohit Iyyer, Mark Yatskar, Wen tau Yih, Yejin Choi, Percy Liang, and Luke Zettlemoyer. 2018. Quac: Question answering in context. In Proc. Conference on Empirical Methods in Natural Language Processing (EMNLP).

Janara Christensen, Stephen Soderland, Gagan Bansal, et al. 2014. Hierarchical summarization: Scaling up multi-document summarization. In Proc. Association for Computational Linguistics (ACL).

Hoa Trang Dang. 2005. Overview of duc 2005. In Document Understanding Conferences.

Thomas H Davenport, David W De Long, and Michael C Beers. 1998. Successful knowledge management projects. Sloan management review, 39(2):43–57.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In Proc. Conference of the North American Chapter of the Association for Computational Linguistics – Human Language Technologies (NAACL HLT).

Xinya Du and Claire Cardie. 2017. Identifying where to focus in reading comprehension for neural question generation. In Proc. Conference on Empirical Methods in Natural Language Processing (EMNLP).

Xinya Du and Claire Cardie. 2018. Harvesting paragraph-level question-answer pairs from wikipedia. In Proc. Association for Computational Linguistics (ACL).

Xinya Du, Junru Shao, and Claire Cardie. 2017. Learning to ask: Neural question generation for reading comprehension. In Proc. Association for Computational Linguistics (ACL).

Angela Fan, Mike Lewis, and Yann Dauphin. 2018. Hierarchical neural story generation. In Proc. Association for Computational Linguistics (ACL).

Yifan Gao, Jianan Wang, Lidong Bing, Irwin King, and Michael R Lyu. 2018. Difficulty controllable question generation for reading comprehension. arXiv preprint arXiv:1807.03586.

Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson Liu, Matthew Peters, Michael Schmitz, and Luke Zettlemoyer. 2018. Allennlp: A deep semantic natural language processing platform. arXiv preprint arXiv:1803.07640.

Tom Gruber. 2008. Collective knowledge systems: Where the social web meets the semantic web. Web semantics: science, services and agents on the World Wide Web, 6(1).

Kai Hakkarainen and Matti Sintonen. 2002. The interrogative model of inquiry and computer-supported collaborative learning. Science & Education, 11(1):25–43.

David Hardtke, Mike Wertheim, and Mark Cramer. 2009. Demonstration of improved search result relevancy using real-time implicit relevance feedback. Understanding the User-Logging and Interpreting User Interactions in Information Search and Retrieval (UIIR-2009).

Michael Heilman and Noah A Smith. 2010. Good question! statistical ranking for question generation. In Proc. Conference of the North American Chapter of the Association for Computational Linguistics – Human Language Technologies (NAACL HLT).

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. 2018. Deep reinforcement learning that matters. In Proc. Association for the Advancement of Artificial Intelligence (AAAI).

Jaakko Hintikka. 1981. The logic of informationseeking dialogues: A model. Werner Becker and Wilhelm K. Essler Konzepte der Dialektik, pages 212–231.

Jaakko Hintikka. 1988. What is the logic of experimental inquiry? Synthese, 74(2):173–190.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. Neural computation.

Ari Holtzman, Jan Buys, Maxwell Forbes, Antoine Bosselut, David Golub, and Yejin Choi. 2018. Learning to write with cooperative discriminators. In Proc. Association for Computational Linguistics (ACL).

Ari Holtzman, Jan Buys, Maxwell Forbes, and Yejin Choi. 2019. The curious case of neural text degeneration. arXiv preprint arXiv:1904.09751.

Matthew Honnibal and Ines Montani. 2017. spacy 2: Natural language understanding with bloom embeddings, convolutional neural networks and incremental parsing. To appear.

Unnat Jain, Svetlana Lazebnik, and Alexander G Schwing. 2018. Two can play this game: visual dialog with discriminative question generation and answering. In Proc. IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR).

Yanghoon Kim, Hwanhee Lee, Joongbo Shin, and Kyomin Jung. 2018. Improving neural question generation using answer separation. arXiv preprint arXiv:1809.02393.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In Proc. Conference on Empirical Methods in Natural Language Processing (EMNLP).

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. In Proc. International Conference on Learning Representations (ICLR).

J Richard Landis and Gary G Koch. 1977. The measurement of observer agreement for categorical data. biometrics, pages 159–174.

Wendy G Lehnert. 1978. The process of question answering: A computer simulation of cognition, volume 978. Lawrence Erlbaum Hillsdale, NJ.

Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective approaches to attention-based neural machine translation. In Proc. Conference on Empirical Methods in Natural Language Processing (EMNLP).

Gary Marchionini. 2006. Exploratory search: from finding to understanding. Communications of the ACM, 49(4):41–46.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch. In NIPS 2017 Autodiff Workshop.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In Proc. Conference of the North American Chapter of the Association for Computational Linguistics – Human Language Technologies (NAACL HLT).

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. In Empirical Methods in Natural Language Processing, pages 2383– 2392.

Siva Reddy, Danqi Chen, and Christopher D. Manning. 2018. Coqa: A conversational question answering challenge. arXiv preprint arXiv:1808.07042.

Mrinmaya Sachan and Eric Xing. 2018. Self-training for jointly learning to ask and answer questions. In Proc. Conference of the North American Chapter of the Association for Computational Linguistics – Human Language Technologies (NAACL HLT).

Roger C Schank. 1972. Conceptual dependency: A theory of natural language understanding. Cognitive psychology, 3(4):552–631.

Abigail See, Peter J Liu, and Christopher D Manning. 2017. Get to the point: Summarization with pointergenerator networks. In Proc. Association for Computational Linguistics (ACL).

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In Proc. Association for Computational Linguistics (ACL).

Iulian Vlad Serban, Alberto García-Durán, Caglar Gulcehre, Sungjin Ahn, Sarath Chandar, Aaron Courville, and Yoshua Bengio. 2016. Generating factoid questions with recurrent neural networks: The 30m factoid question-answer corpus. In Proc. Association for Computational Linguistics (ACL).

Manfred Stede and David Schlangen. 2004. Information-seeking chat: Dialogues driven by topic-structure. In Proceedings of Catalog (the 8th workshop on the semantics and pragmatics of dialogue; SemDial04).

Duyu Tang, Nan Duan, Tao Qin, Zhao Yan, and Ming Zhou. 2017. Question answering and question generation as dual tasks. arXiv preprint arXiv:1706.02027.

Christopher Tauchmann, Thomas Arnold, Andreas Hanselowski, Christian M Meyer, and Margot Mieskes. 2018. Beyond generic summarization: A multi-faceted hierarchical summarization corpus of large heterogeneous data. In Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC).

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In Proc. Neural Information Processing Systems (NIPS).

Christian Wagner. 2004. Wiki: A technology for conversational knowledge management and group collaboration. Communications of the association for information systems, 13(1):19.

Christian Wagner and Narasimha Bolloju. 2005. Supporting knowledge management in organizations with conversational technologies: Discussion forums, weblogs, and wikis. Journal of Database Management, 16(2).

Tong Wang, Xingdi Yuan, and Adam Trischler. 2017. A joint model for question answering and question generation. arXiv preprint arXiv:1706.01450.

Thomas Wolf, Victor Sanh, Julien Chaumond, and Clement Delangue. 2019. Transfertransfo: A transfer learning approach for neural network based conversational agents. CoRR, abs/1901.08149.

Mark Yatskar. 2019. A qualitative comparison of coqa, squad 2.0 and quac. Proc. Human Language Technology/Conference of the North American Chapter of the Association for Computational Linguistics (HLT/NAACL).

Ka-Ping Yee, Kirsten Swearingen, Kevin Li, and Marti Hearst. 2003. Faceted metadata for image search and browsing. In Proceedings of the SIGCHI conference on Human factors in computing systems.

Amy X. Zhang, Lea Verou, and David Karger. 2017. Wikum: Bridging discussion forums and wikis using recursive summarization. In Conference on Computer Supported Cooperative Work and Social Computing (CSCW).