semiPQA: A Study on Product Question Answering over Semi-structured Data

Xiaoyu Shen, Gianni Barlacchi, Marco del Tredici, Weiwei Cheng and Adrià de Gispert
[1]Amazon Alexa AI
{gyouu, gbarlac, mttredic, weiweic, agispert}@amazon.com

Abstract

Product question answering (PQA) aims to automatically address customer questions to improve their online shopping experience. Current research mainly focuses on finding answers from either *unstructured* text, like product descriptions and user reviews, or *structured* knowledge bases with pre-defined schemas. Apart from the above two sources, a lot of product information is represented in a *semi-structured* way, e.g., key-value pairs, lists, tables, json and xml files, etc. These semi-structured data can be a valuable answer source since they are better organized than free text, while being easier to construct than structured knowledge bases. However, little attention has been paid to them. To fill in this blank, here we study *how to effectively incorporate semi-structured answer sources for PQA* and focus on *presenting answers in a natural, fluent sentence*. To this end, we present semiPQA: a dataset to benchmark PQA over semi-structured data. It contains 11,243 written questions about json-formatted data covering 320 unique attribute types. Each data point is paired with manually-annotated text that describes its contents, so that we can train a neural answer presenter to present the data in a natural way. We provide baseline results and a deep analysis on the successes and challenges of leveraging semi-structured data for PQA. In general, state-of-the-art neural models can perform remarkably well when dealing with seen attribute types. For unseen attribute types, however, a noticeable drop is observed for both answer presentation and attribute ranking.

1 Introduction

Product question answering (PQA) is playing an increasingly important role in e-commerce platforms. It is able to greatly improve the online shopping experience since customers do not need to traverse over the detailed web pages to seek information themselves. Traditional approaches built *structured* knowledge bases for product attributes and mapped customer questions into executable queries(Frank et al., 2007; Tapeh and Rahgozar, 2008; Hui et al., 2013; Li et al., 2019). In recent years, with the rapid progress of large-scaled pre-trained neural models, many research works have achieved promising results by leveraging only *unstructured* text, like product descriptions, user reviews and community answers (Cui et al., 2017; Gupta et al., 2019; Gao et al., 2019; Zhang et al., 2020). Lying between these two source types, a lot of product information is often organized in a semi-structured form, e.g., key-value pairs, lists and tables from product web pages, json and xml files from internal databases, etc. These semi-structured data can be a valuable answer source since they are better organized and more precise than free text, while being much cheaper to maintain than structured knowledge bases. Nonetheless, few research works have ever considered them and there is no public available dataset for its study. This paper aims to fill in this blank and study *how to effectively incorporate semi-structured answer sources for PQA and present answers in a natural sentence*. To this end, we construct a dataset to benchmark this study. It contains 11,243 product questions about json-formatted semi-structured data [1]. The data contains 320 unique attribute types (size, material, color, etc) spanning a diverse set of semi-structured forms like key-value pairs, lists and hierarchies. Each data is paired with manually annotated text that describes its contents. Table 1 shows some examples from the dataset. Given a question, there are two steps we need to get an answer: (1). **Attribute ranking**: selecting the proper attribute that contains the information to answer the question. Modern pre-trained neural models and QA datasets mainly focus on plain text, so they may not gen-

[1]As json is a standard format for storing data with arbitrary types/schemata, other representations (such as tables or xml files) can be easily mapped to it.

eralize well to ranking semi-structured attributes, especially with limited training data. (2). **Answer presentation**: presenting the answer in a fluent sentence. It is not user-friendly to directly present the semi-structured data to customers, especially for applications like voice assistants. We apply data-to-text generation models to convert these data into fluent text.

For attribute ranking, we build our model upon state-of-the-art pre-trained language models. Due to the small size of our training data, we follow the common practice of pre-finetuning the attribute ranker on four large-scale QA datasets: Natural Questions (Kwiatkowski et al., 2019), AmazonQA (McAuley and Yang, 2016), NewsQA (Trischler et al., 2017) and Squad (Rajpurkar et al., 2016). Since these are all based on unstructured text, we also experiment with converting semi-structured attributes into text before being passed to the ranker. Our results show that text-based QA models are quite robust to semi-structured data representations, and can rank attributes correctly with only keyword matching without the extra order information.

For answer presentation, we consider a *question-independent* answer presenter, which is less risky than question-dependent presentation while being more flexible than span extraction or multi-choice selection. We evaluate both a template-based system and a neural sequence-to-sequence generation model. Each template is one or more sentences with gaps that can be filled with pre-defined rules (Deemter et al., 2005). However, semi-structured data does not follow any unified schema, so designing rules to cover all possible data forms or unseen attributes is infeasible. Our neural generation models are initialized with Bart (Lewis et al., 2020) and T5 (Raffel et al., 2020), two representative pretrained models for generative tasks, and fine-tuned on a small set of annotated examples. Compared with the template system, we show the neural approach improves not only the fluency, but also the faithfulness of presented answers.

Finally, we discuss and analyse the challenge of generating factually-correct sentences without hallucinate information, as well as the difficulty of handling unseen attributes in both ranking and answer presentation.

2 Dataset

The data collection contains 3 stages: semi-structured attribute collection, text annotation and question sourcing. This section will explain these three stages in order then present the statistics.

Attribute Collection We obtain the semi-structured attributes of product information from our internal database. These attributes are aggregated from different providers with varied schema. We select 320 unique attribute types from it, filter out information only for internal use and indicator tags containing no actual information like "language_tag", "attribute_id" etc. For each of the 320 attribute types, we randomly sample 20 products containing such attribute from 5M products sold in the US market (The 5M products are randomly sampled from different categories), then extract their attribute instances. After removing duplicate ones, we get 3,316 unique attribute instances in the end. We then preprocess them to lower-case all characters, remove emojis and normalize all floats to contain at most 2 decimals.

Text Annotation After obtaining the semi-structured attributes, we hire annotators from Amazon Mechanical Turk to write a natural sentence for each attribute instance. We restrict to US-based annotators who completed > 500 tasks, out of which more than 97% had been accepted. Before the formal annotation, we did a pilot study with 100 samples. Without extra information, we find 16% of attributes are not understandable to humans, which indicates proper context is necessary to understand the meanings of attributes. Therefore, we also provide the product image and title in the second round of pilot study. By adding the extra information, only 4% of them are not understandable. We then continue with this setting and get all attributes annotated. We also remove all attributes that are not understandable to annotators (usually those that rely on other information to interpret), and end up with 3,191 attribute instances annotated with their description text.

Question Sourcing We collect questions on Mechanical Turk by present annotators with the image, title and rating of the product plus one of its associated attribute instances. Annotators are asked to imagine themselves as potential customers, and their task is to ask four questions about this attribute, which means that the attribute contains the information to answer their question. We explicitly add three criteria that annotators must follow: ques-

Table 1: Examples of question-data-text triples in the dataset. The data features diverse forms of semi-structures like key-value pairs, lists and hierarchies.

Question:	Is the body made out of nylon?
Data (key-value):	fabric_type:{ value:"Body: Nylon/spandex; cup linings:100% polyester;cup pad:100% polyurethane."}
Text:	The body is made of nylon and spandex, the linings in ...
Question:	What kind of devices fit in this?
Data (list):	compatible_devices: {value:“apple ipad mini 4"}; {value:“apple ...
Text:	The product is compatible with apple ipad mini 4, apple ipad air...
Question:	Is this metal?
Data (hierarchy):	blade:{material:[{value:“Plastic"}],length:[{unit:inches,value:3.0}]}
Text:	The blade on this measures 3 inches and is plastic.

tion must be (1) Meaningful, having a reasonable chance of being asked in daily shopping, and not parroted, rigid questions like "what is the [attribute name]"; (2) Diverse, so the three questions must not be paraphrase each other, and (3) Answerable by the attribute, ensuring that the attribute contains the information to answer the questions. After getting these questions, we lower case them and remove duplicate questions about the same products.

Dataset Split and Statistics The dataset will be used to train and evaluate the (1) attribute ranker and (2) answer presenter. For both, we have two test scenarios, one containing only seen attributes with unseen values, and the other containing only unseen attributes to test the model generalization capability. For the unseen scenario, we randomly sample 30 attribute types from all 320 types. We sample 58 instances from them and add into the dev set, while the rest are used as test set. For the seen scenario, we randomly sample 440 instances from the remaining 290 attribute types. 220 of them are added into the dev set and the rest serve as the test set. We use one fixed dev set containing both seen (220) attributes and unseen (58) attributes. All remaining instances serve as training set. Due to the small data size, we perform cross validation to get more reliable results. We repeat the above process ten times with different seeds to get 10 different splits, then train/evaluate on them and average the results. For each question asking about one attribute, we treat all other attribute instances belonging to the same product as negative candidates. The candidate positive-negative ratio is 1:17.89.

3 Attribute Ranking

Attribute ranking aims to select the proper attribute that contains the information to answer the user-posed question.

We start from a tf-idf baseline, which has been shown a strong baseline for sentence matching tasks (Arora et al., 2017). We count the frequency based on the attribute instances on the training set. At test time, we convert question and answer candidate into tf-idf vectors based on the counted frequency, then compute their cosine similarity as the ranking score.

Following the common practice, we also tried concatenating the question and candidate attribute into one sequence then feeding into the Roberta-base encoder (Liu et al., 2019), a Transformer-based neural model pretrained on billions of text. The final classifier is built on top of the representation of the first [CLS] token. The multi self-attention layers of the encoder makes sure each token is able to interact with all other tokens to capture the dependency relations. The model is trained to maximize the likelihood of the positive candidates and minimize that of the negative candidates. As for the input form of the semi-structured attribute, we experimented with 5 forms: (1) **name-only**: only use the attribute name as input. (2) **value-only**: only use the attribute value as input. (3) **linearized**: use the linearized json which concatenates the attribute name and value as input. (4) **template**: use the template system to generate its corresponding text, then use the generated text as input. (5) **neural**: use the neural generator to generate its corresponding text, then use the generated text as input.

Due to the limited size of our training data, we follow a two-step setting (Garg et al., 2020) where the Roberta-base model is first fine-tuned on a large-scale QA dataset, then fine-tuned on our semiPQA training data. This has been shown to improve performance in the low-resource setting (Hazen et al., 2019; Garg et al., 2020). We consider 4 datasets: (1) **NQ**: the Google Natural Questions dataset. We use its sentence selection version (Garg et al., 2020), where its negative samples are cate-

gorized into 4 classes to improve the robustness of the model. It contains 61,186 questions from the Google queries for training. (2) **AmazonQA**: QA pairs from the Amazon community QA website (Gupta et al., 2019). We remove answers containing "I don't know", "I'm not sure" etc, and filter questions more than 32 words and answers more than 64 words. Negative candidates are answers about different questions under the same product. It contains 1,065,407 community questions for training. (3) **NewsQA**: QAs about news articles (Trischler et al., 2017). We convert it into sentence selection and drop the span label. For each question, we sample 5 negative sentences not labeled as correct for training. The training dataset contains 75,473 questions. (4) **Squad**: QAs about wikipedia paragraphs (Rajpurkar et al., 2016). We treat sentences containing the ground-truth span as positive and other sentences in the same paragraph as negative. The training dataset contains 87,599 questions. Notably, *all answers in the above 4 datasets are in form of unstructured sentences.*.

We analyzed the performance under three settings: (1) zeroshot where the model is applied directly to the testsets without using our training data, (2) performance on seen attributes after finetuning on the training data and (2) performance on unseen attributes after finetuning on the training data [2]. Precision@1 results are shown in Figure 1. We also computed other metrics like MAP, MRR and HIT@5, but they show a similar trend and are omitted for space limit.

Zeroshot Performance The zeroshot results are visualized in the upper part of Figure 1, where we apply the rankers finetuned on different datasets to directly test on our data. As can be seen, when only the attribute name or value is available, the performance is significantly lower than the others, both for neural models and the tf-idf baseline. This suggests we need information from both the attribute name and value to rank attributes properly. Neither of them are sufficient by its own. *Neural models finetuned on unstructured text can generally adapt well to semi-structured data (linerized form), except for the one finetuned on NQ which performs poorly compared with others*. One reason could be that the negative samples from NQ are finer-grained. It must learn to differentiate between sentences containing correct answer spans

[2]In the zeroshot setting, we only evaluate on the seen attribute split since there is no concept of "seen" or "unseen" for zeroshot evaluation.

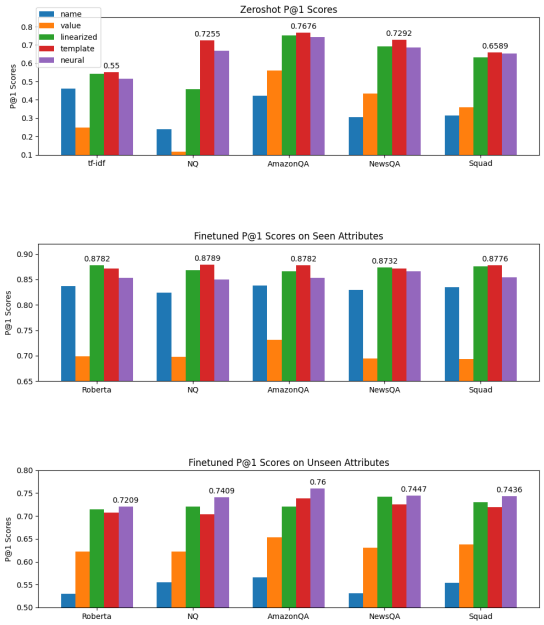

Figure 1: p@1 in zeroshot/finetuned settings.

but talking about irrelevant things, and correct sentences. Therefore, it must rely on the sentence structure to infer the meaning and decide whether it is relevant or not (Garg et al., 2020). When directly tested on semi-structured jsons, it cannot easily interpret non-natural sentences. When finetuned on other datasets like NewsQA, AmazonQA and Squad, negative samples are randomly sampled and hardly contain the correct answer span, so the model might only rely on span detection and do not need well-formed sentences. Using template-generated text leads to the best zeroshot performance for all models, next come the neural-generated text and linearized json which perform slightly worse. Among all datasets used for finetuning, AmazonQA adapts best for all input formats. This is not surprising considering that it is also about product questions and has the largest data size for finetuning.

Finetuned Performance on Seen Attributes The finetuned results on seen attributes is visualized in the middle of Figure 1. "Roberta" indicates the model is initialized with the Roberta-based checkpoint without being finetuned on any other QA datasets in advance. "NQ" indicates that the Roberta-based model is first finetuned on NQ, then finetuned on our training data, same for "AmazonQA", "NewsQA" and "Squad". Similarly to the zeroshot setting, using only the attribute name or

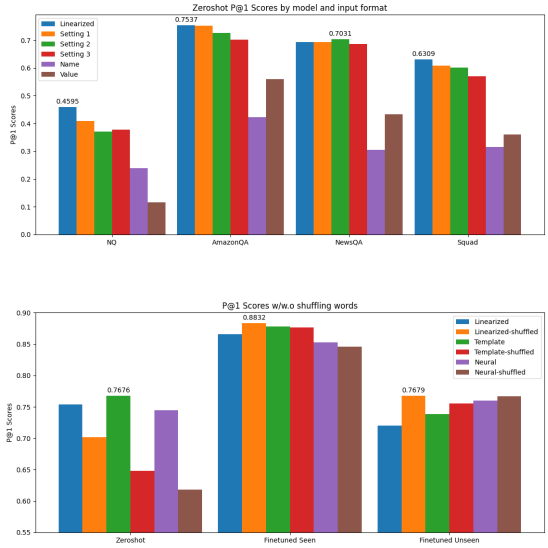

Figure 2: p@1 with varying input formats.

value leads to significantly worse results, although attribute names seem to be more important when finetuning on the training data. *Using the linearized json format and the template-generated text have the best overall performance, achieving a precision score of over 85%. Using more natural and fluent text does not help the ranking performance on seen attributes*. Although neural generated text are of higher-quality according to human evaluations (to be shown in Section 4), this does not make the ranking task easier and leads to performance drop, suggesting that presentation is not a requirement for ranking and can be addressed separately. *Pre-finetuning on large-scale text-based QA datasets also does not help the performance on seen attributes*, as the Roberta result already achieves similar performance. The model is able to quickly learn the correspondence between questions and seen attributes even with the limited training data.

Finetuned Performance on Unseen Attributes In the bottom of Figure 1, we show the finetuned performance when testing on unseen attributes. As expected, a significant performance drop is observed for all models, especially when using attribute names only as this is mostly equivalent to classification over unseen labels. *Using neural-generated text as input achieves the best performance in all settings*. We hypothesise that the neural-generated texts are less rigid and more diverse than template-generated or linearised json data, which prevents the model from overfitting. Finetuning from Roberta directly performs the worst on average, and finetuning first on AmazonQA generally leads to a smaller performance drop with respect to seen attributes. The large amount of questions in AmazonQA, though not helpful for seen attributes, do improve the model robustness over unseen attributes.

Analysis As shown above, directly using the linearized json format performs well in the zeroshot setting, which indicates that models finetuned on QA datasets are able to learn to generalize to the json format when finetuning on the sentence format. To investigate this surprising finding, we perform an ablation study in the following settings:

1. Remove all quotation marks plus curly braces from the json.

2. On top of (1), further remove all colons from the json.

3. On top of (2), further shuffle the word order in json.

By gradually removing the structural features of the representation, we aim to evaluate whether the model needs this json structure for attribute ranking. The zeroshot p@1 scores obtained are reported in Figure 2. We also do the same to text generated from the template and neural models.

As can be seen, *removing the json structure does not have a great effect on performance*. Even after shuffling the word orders completely, the performance drop is within 5% for most models. However, removing either the attribute name or value does lead to significant performance drops, which indicates that *the model relies more on semantic matching against both attribute name and value for prediction, rather than on the structure or word order information*.

Finally, the bottom figure shows that in the zeroshot setting, shuffling the word order reduces the performance for both the linearized, template and neural format. The drop is more for template and neural format but less for the linearized json format. This implies the pretrained QA models are more sensitive to word orders in the sentence format than the structured json format. When finetuned on the training data, however, word orders loses importance. Interestingly, when testing on unseen attribute, shuffling the word order even improves model performance. This further confirmed that *for this task, the model does not need to rely on*

the word order to make predictions, shuffling the word order can even improve the model robustness on generalizing to unseen attributes.

4 Answer Presentation

The first approach we consider for answer presentation is to use handcrafted templates. However, defining a perfect template for each attribute is challenging due to the lack of a standard schema and templates cannot scale to unseen attributes. With this concern, we also experiment with training a neural data-to-text generator trained with annotated text as the target.

Template System When designing the template system, we aim to capture general rules across different attribute types so that one template can be reusable to other similar attributes. We define each template should contain (1) a precondition specializing when to apply the template, (2) one or several corresponding text with gaps to fill, and (3) a set of rules defining how to fill in the gaps. For example, the following is a template defined from the attribute type *ARE_BATTERIES_REQUIRED*:

```
Precondition: applies if the POS
tag of the attribute
name follows the pattern of
be_NOUN_VERBed.
Rule: (1) If the value is "Y" or
"yes" or "True":
output "It VERBs the NOUN".
(2) Otherwise: output "It does
not VERB the NOUN".
```

where VERBs and VERBed mean the third person singular and past particle form of the verb. For *ARE_BATTERIES_REQUIRED*, VERBs would be "requires" and VERBed is "required". It can also apply to other attribute types following the same pattern like "is_assembly_required" and "is_software_included".

During template construction, we maintain a template bank starting from empty. As we see more attribute types, we check if any template from the bank can be applied, and if so, whether it generates the correct text or whether we need to manually update the template. Otherwise, we create a new template for this attribute type. This process is repeated until we go over all the 320 attribute types three times, to refine, merge and fix the template bank and rules. After these rounds, we end up with a total of 23 distinct templates.

Nevertheless, during the construction process, we realize it is nearly impossible to devise a template system to cover all cases well, even for the limited 320 attribute types that we focus on. The difficulty lies in the following two diversities in the data: (1) linguistic diversity: The attribute values do not follow any strict rule. They can be free text as long as it conveys the meaning, which makes it hard to design general rules even for a single attribute type. (2) structural diversity: The json format is a loose structure. The same semantic meaning can be organized in different ways and hierarchies. Applying one rule for different structures can easily lead to parsing errors. Table 2 shows some examples of different values for the same attribute type. We can see that even for one single attribute, it requires many verbalizing rules to handle different structures and attribute values, let alone extending the template rules to multiple attribute types.

Attribute value	Text
{ value:"gas-powered"}	The product is gas-powered.
{ value:"batteries"}	It runs on batteries.
{ value:"Manual" }	This doesn't have power.
{ value:"NA" }	This doesn't run on any power.

Table 2: Different instances of the attribute type "power_source_type" and human annotated text.

Neural Generator To avoid pre-defined rules and to generalise to unseen attributes, we train a neural generator model initialized either with Bart (Lewis et al., 2020) or T5 (Raffel et al., 2020), two state-of-the-art generative models pretrained on large amount of web text with self-supervised objectives. As input, we feed the linearized json-formatted data [3] and the output is the annotated text.

We further normalize the numbers in both the attribute and text to keep them in a consistent form, to help the model learn their correspondence in the generation task. For example, we turn forms like "1.", "1.0" and "1.00" into 1, and normalize words to numeric values ("one" → "1" etc).

To minimize the changes of hallucination in the generation, we also delexicalize words in the an-

[3] We also tried other input formats like flattening the hierarchical structure, adding instruction prompts (Schick and Schütze, 2020; Liu et al., 2021) etc, but did not find significant improvements.

Model	BLEU	chrF	PARENT-F1	#PARAMs	Faith	Cov	Flu
Performance on Seen Attributes							
Template	-	-	-	-	0.9612	0.9546	3.203
Reference	-	-	-	-	0.9574	0.9728	3.532
Bart-Base	0.3704	0.571	0.36445	139M	-	-	-
Bart-Large	0.3917	0.615	0.38748	406M	-	-	-
T5-Small	0.3267	0.542	0.33128	60M	-	-	-
T5-Base	**0.4061**	0.601	0.38214	220M	-	-	-
T5-Large	0.4060	**0.616**	**0.40806**	770M	**0.9731**	**0.9776**	**3.657**
T5-L (delex)	0.3911	0.604	0.39277	770M	0.9620	0.9640	3.632
Performance on Unseen Attributes							
Reference	-	-	-	-	**0.9401**	**0.8050**	3.520
Bart-Base	0.3386	0.553	0.31463	139M	-	-	-
Bart-Large	0.3541	0.586	0.33292	406M	-	-	-
T5-Small	0.3187	0.512	0.31379	60M	-	-	-
T5-Base	0.3293	0.544	0.33113	220M	-	-	-
T5-Large	**0.3869**	**0.610**	**0.37365**	770M	0.9125	0.9231	**3.610**
T5-L (delex)	0.3696	0.597	0.35275	770M	-	-	-

Table 3: Automatic Metric and human evaluation Results for Answer Presentation

notated text that match with the attribute values, replacing them by a tag in the input attribute, a common technique used in data-to-text generation (Wen et al., 2015; Ferreira et al., 2019; Chang et al., 2020b,a). The tag is the linearized path from the root node (attribute name) to the tag of the value. For example, for the second sentence in Table 1, the text "The product is compatible with ..." will be delexicalized into "The product is compatible for (concatenated) [value]." In the testing phase, after the model decodes the delexicalized text, the tag is then replaced to the corresponding value in the input attribute. While this can provide the model with a clear correspondence between input and output, it also adds the risk of losing the linguistic information like tense, singular/plural after delexicalization.

Automatic Evaluation For the automatic metrics, we report the BLEU (Papineni et al., 2002), chrF (Popović, 2015) and PARENT-F1 (Dhingra et al., 2019) score. The results of automatic metrics are shown in Table 3, where we try different sizes of models and list their number of model parameters (#PARAMs). Generally all the three metrics correlate well with each other. As expected, larger models tend to perform better than smaller models, with a larger difference on unseen versus seen attributes, which suggests that *larger models generalize better than smaller models on unseen attributes*. This could be because larger models are encoded with more language knowledge, which makes them less likely to overfit to the attributes in the training data.

T5-large achieves the best performance across all metrics. Therefore, we train with the delexicalized text as mentioned in Section 4 based on T5-large to see if the delexicalization can improve the performance further (T5-L (delex) in the table). All scores are evaluated on the lexicalized text output, which means that all delexicalized parts have been replaced with the input attribute values so that we can have a fair evaluation.

Delexicalization, unfortunately, does not help with the performance. It lowers down the scores over all metrics compared with directly using the original text as the target. The reason could be that T5 is pretrained with natural text itself. It has no delexicalized slots in its training corpus. Therefore, it fails to adapt well to the format of delexicalized text. Indeed, we find that T-5 sometimes generates text with slot names that do not exist in the input attribute which affects its performance. For future research, it would be interesting to see how to adapt pretrained generative models to delexicalized text, or even directly pretraining large-scaled generative models on delexicalized text.

Human Evaluation We conduct a human evaluation of the generated texts, focusing the following three dimensions: (1) **Faithfulness**, whether the text is faithful to the attribute (binary). (2) **Coverage**, whether the text covers all contents in the attribute (binary). (3) **Naturalness**, whether the text is a natural sentence rather than a machine-generated rigid one. 4-ary score from 1(rigid), 2(slightly rigid), 3(slightly natural) to 4(natural) On seen attributes, we evaluate the T5-large and T5-large with delexicalized text (T5-L (delex)), plus the template system and the annotated reference.

Attribute	From Template	From T5-large
allergen_information: { value:gluten_free }; { value:dairy_free }	allergen warning: the product contains gluten free,dairy free.	this product is gluten free and dairy free.
team_name: { value:"null" }	the team name of the product is null.	this does not have a team name.
speaker_type: {value:"portable bluetooth speakers"}	the product has a portable bluetooth speakers speaker.	this is a portable bluetooth speaker.
installation_type: value:"driver side"	the product is installed using the driver side.	this is installed on the driver side.

Table 4: Example of template-generated texts that are labeled as unfaithful.

Attribute	Reference	From T5-large
size_per_pearl: { value:"iphone" }	it is an iphone.	the product has an iphone size pearl.
switch_type: { value:"rotary switch" }	this has a switch that turns.	the product has a rotary switch.
target_species: { value:"Dog" }	for dogs.	this is for dogs.
installed_size:[{unit: unknown_modifier, value:32.}]	its cache memory installed_size:[{unit: unknown_modifier, value:32.}]	the product has a cache memory of 32 units.

Table 5: Example references which are labeled as unfaithful(first two rows) or unnatural (last two rows).

On unseen attributes, we only evaluate T5-large and the reference since handcrafted templates cannot be applied to unseen attributes at all. From each of 10 data splits, we randomly sample 50 attributes from it such that each model has 500 attribute-text pairs being evaluated. Each pair is evaluated by three annotators. The final scores are averaged over the 500 pairs for each model. We show the results and the agreement score among annotators in Table 3 and Table 6 respectively.

Faithful	**Coverage**	**Natur-4class**	**Natur-2class**
0.97762	0.97402	0.80499	0.92569

Table 6: Agreement Score for Answer Presentation.

Overall, the evaluation has a rather high agreement score. Naturalness has the lowest agreement since it is 4-ary. We also calculate the binary score for naturalness by combining natural and slightly natural into one bucket, and combining rigid and slightly rigid into the other bucket. The agreement score grows to over 0.92 by this means. We then manually checked and corrected all attribute-text pairs that do not have an agreement score of 1 for faithfulness and coverage. For naturalness, as it is a rather subjective metric anyway, we do not correct it. We also manually verified the faithfulness and coverage for the attribute *nutritional_info*, which we find especially hard to be evaluated correctly due to its complexity.

Overall all models have high scores on both faithfulness and coverage, and differences are small. For naturalness, as expected, templates have the lowest score. Using delexicalization underperforms the standard neural model, which is consistent with the findings from the automatic metric scores. We observe two interesting phenomena: (1) Neural models outperform templates even for faithfulness and (2) Neural models outperform human references for faithfulness and naturalness. In Table 4 and 5, we list examples of text generated from templates/references that are labeled as unfaithful/unnatural to the attribute. As can be seen, errors in template-generated texts usually occur because *the templates designed for certain values do not apply to a new value*. Errors in humans references are due to annotation noise, which is usually inevitable. The T5 model outperforms the reference, suggesting that it is able to round up these few annotation errors and learn the general pattern from the most correct references.

5 Conclusion

In this work, we study how to effectively leverage semi-structured data for product question answering. As there is no public datasets for this problem, we collect a dataset containing manually annotated questions together with description text about semi-structured attributes from our internal database. We present empirical results and findings about two key challenges of this problem: attribute ranking and answer presentation . Experiments show that neural models can provide superior text than template systems and perform well for ranking seen attributes, albeit there is still a noticeable drop when it comes to unseen attributes for both ranking

and generation.

References

Sanjeev Arora, Yingyu Liang, and Tengyu Ma. 2017. A simple but tough-to-beat baseline for sentence embeddings. In *5th International Conference on Learning Representations, ICLR 2017*.

Ernie Chang, David Ifeoluwa Adelani, Xiaoyu Shen, and Vera Demberg. 2020a. Unsupervised pidgin text generation by pivoting english data and self-training. *arXiv preprint arXiv:2003.08272*.

Ernie Chang, Jeriah Caplinger, Alex Marin, Xiaoyu Shen, and Vera Demberg. 2020b. Dart: A lightweight quality-suggestive data-to-text annotation tool. In *Proceedings of the 28th International Conference on Computational Linguistics: System Demonstrations*, pages 12–17.

Lei Cui, Shaohan Huang, Furu Wei, Chuanqi Tan, Chaoqun Duan, and Ming Zhou. 2017. Superagent: A customer service chatbot for e-commerce websites. In *Proceedings of ACL 2017, System Demonstrations*, pages 97–102.

Kees van Deemter, Mariët Theune, and Emiel Krahmer. 2005. Real versus template-based natural language generation: A false opposition? *Computational linguistics*, 31(1):15–24.

Bhuwan Dhingra, Manaal Faruqui, Ankur Parikh, Ming-Wei Chang, Dipanjan Das, and William Cohen. 2019. Handling divergent reference texts when evaluating table-to-text generation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4884–4895.

Thiago Castro Ferreira, Chris van der Lee, Emiel van Miltenburg, and Emiel Krahmer. 2019. Neural data-to-text generation: A comparison between pipeline and end-to-end architectures. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 552–562.

Anette Frank, Hans-Ulrich Krieger, Feiyu Xu, Hans Uszkoreit, Berthold Crysmann, Brigitte Jörg, and Ulrich Schäfer. 2007. Question answering from structured knowledge sources. *Journal of Applied Logic*, 5(1):20–48.

Shen Gao, Zhaochun Ren, Yihong Zhao, Dongyan Zhao, Dawei Yin, and Rui Yan. 2019. Product-aware answer generation in e-commerce question-answering. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pages 429–437.

Siddhant Garg, Thuy Vu, and Alessandro Moschitti. 2020. Tanda: Transfer and adapt pre-trained transformer models for answer sentence selection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 7780–7788.

Mansi Gupta, Nitish Kulkarni, Raghuveer Chanda, Anirudha Rayasam, and Zachary C Lipton. 2019. Amazonqa: A review-based question answering task. *arXiv preprint arXiv:1908.04364*.

Timothy J Hazen, Shehzaad Dhuliawala, and Daniel Boies. 2019. Towards domain adaptation from limited data for question answering using deep neural networks. *arXiv preprint arXiv:1911.02655*.

Kai Hui, Bin Gao, Ben He, and Tie-jian Luo. 2013. Sponsored search ad selection by keyword structure analysis. In *European Conference on Information Retrieval*, pages 230–241. Springer.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. 2019. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880.

Feng-Lin Li, Weijia Chen, Qi Huang, and Yikun Guo. 2019. Alime kbqa: Question answering over structured knowledge for e-commerce customer service. In *China Conference on Knowledge Graph and Semantic Computing*, pages 136–148. Springer.

Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. 2021. Gpt understands, too. *arXiv preprint arXiv:2103.10385*.

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

Julian McAuley and Alex Yang. 2016. Addressing complex and subjective product-related queries with customer reviews. In *Proceedings of the 25th International Conference on World Wide Web*, pages 625–635.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318.

Maja Popović. 2015. chrf: character n-gram f-score for automatic mt evaluation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 392–395.

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*, 21:1–67.

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.

Timo Schick and Hinrich Schütze. 2020. Few-shot text generation with pattern-exploiting training. *arXiv preprint arXiv:2012.11926*.

Ali Ghobadi Tapeh and Maseud Rahgozar. 2008. A knowledge-based question answering system for b2c ecommerce. *Knowledge-Based Systems*, 21(8):946–950.

Adam Trischler, Tong Wang, Xingdi Yuan, Justin Harris, Alessandro Sordoni, Philip Bachman, and Kaheer Suleman. 2017. Newsqa: A machine comprehension dataset. In *Proceedings of the 2nd Workshop on Representation Learning for NLP*, pages 191–200.

Tsung-Hsien Wen, Milica Gasic, Nikola Mrkšić, Pei-Hao Su, David Vandyke, and Steve Young. 2015. Semantically conditioned lstm-based natural language generation for spoken dialogue systems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1711–1721.

Wenxuan Zhang, Yang Deng, Jing Ma, and Wai Lam. 2020. Answerfact: Fact checking in product question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2407–2417.