Howdy Y’all: An Alexa TaskBot

Majid Alfifi, Xiangjue Dong, Timo Feldman, Allen Lin, Karthic Madanagopal, Aditya Pethe, Maria Teleki, Zhuoer Wang, Ziwei Zhu, James Caverlee

Texas A&M University College Station, TX, USA

{majid, xj.dong, feldman.timo, al001, karthic11, apethe100}@tamu.edu
{mariateleki, wang, zhuziwei, caverlee}@tamu.edu

Abstract

In this paper, we present Howdy Y’all, a multi-modal task-oriented dialogue agent developed for the 2021-2022 Alexa Prize TaskBot competition. Our design principles guiding Howdy Y’all aim for high user satisfaction through friendly and trustworthy encounters, minimization of negative conversation edge cases, and wide coverage over many tasks. Hence, Howdy Y’all is built upon a rapid prototyping platform to enable fast experimentation and powered by four key innovations to enable this vision: (i) First, it combines a rules, phonetic matching, and a transformer-based approach for robust intent understanding. (ii) Second, to accurately elicit user preferences and guide users to the right task, Howdy Y’all is powered by a contrastive learning search framework over sentence embeddings and a conversational recommender for eliciting preferences. (iii) Third, to support a variety of user question types, it introduces a new data augmentation method for question generation and a self-supervised answer selection approach for improving question answering. (iv) Finally, to help motivate our users and keep them engaged, we design an emotional conversation tracker that provides empathetic responses to keep users engaged and a monitor of conversation quality.

1 Introduction

Conversational AI is advancing rapidly, with considerable efforts targeted at improving social chatbots (Adiwardana et al., 2020), intelligent assistants like Alexa and Siri, and conversational search and recommendation (Zhang et al., 2018; Jannach et al., 2021), among others. In this paper, we describe the design of Howdy Y’all, a *task-oriented* conversational system that helps users complete tasks as part of the 2021-2022 Alexa TaskBot Challenge. Compared to open-domain dialogue systems – which aim to chat with users about a variety of topics to maximize user engagement – task-oriented systems aim to help users find the right task, guide the user through the steps of completing the task, and help the user if they face (unexpected) difficulties. Hence, key challenges include correctly understanding the user need, finding the right task from a collection of candidate tasks, assisting the user in accomplishing the task, and monitoring the flow of the conversation as it evolves. In the following, we first describe the overall system architecture of Howdy Y’all before moving on to the key research innovations.

1.1 System Overview

Howdy Y’all is implemented using the Cobot framework provided by the Alexa Prize team on AWS services. A user invokes our Alexa skill which sends a multimodal request to our AWS Lambda function which in turn performs the following operations: (i) It performs global intent handling using the multimodal input; (ii) It dispatches parallel requests passing the text input to evaluate multiple NLP components remotely, including Coreference Resolution, Noun Phrase Extraction, and Named Entity Recognition; (iii) It dispatches parallel requests to run the following components remotely:

- *Domain Classifier.* This component classifies the text input into one of the following categories: DIY, RECIPE, MEDICAL, PROFESSIONAL, FINANCIAL, LEGAL, DANGER, OTHERS.
- *Intent Classifier.* This is a transformer-based model that classifies user utterances into several intents, as discussed in Section 2.
- *Task Search.* For DIY tasks, we rely on a BERT-based model finetuned on a large WikiHow dataset that is served using FAISS, a library for efficient similarity search and clustering of dense vectors Johnson et al. (2019). For recipe tasks, we follow a similar approach (finetuning instead on a WholeFood dataset), plus an additional sentence embedding based on contrastive learning Gao et al. (2021) over a large recipe dataset Majumder et al. (2019).
- *Question Answering.* To answer questions, we have a suite of question answering modules, as detailed in Section 4

and (iv) it passes the values to response generators (RGs) for Recipes, DIY, Recommendation, Chitchat, Empathy, Functional capabilities (like set a timer), and Utilities. The Utility RGs are Fallback, Help, and Transition RGs. When all other non-utility RGs fail to produce a suitable response, the system first checks the Help RG to see whether there are help messages based on user state. For example, if the user has difficulty in navigating the steps, the Help RG will respond with "You can navigate the steps by saying: next, previous, repeat or go to step number". If there are no help messages for that state, we resort to the Fallback RG with a generic fallback (e.g. "Say: help, and I'll read you the available options!").

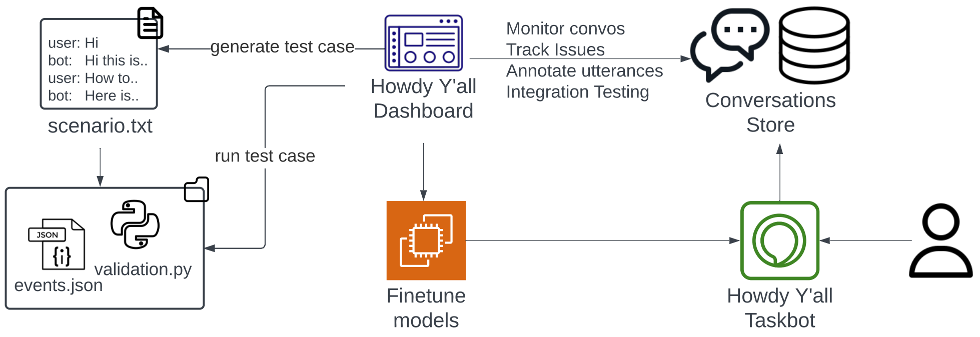

Figure 1: Howdy Y'all Rapid Prototyping Platform. The dashboard is used to monitor the conversations in real time and to provide annotations on user utterances. This labeled dataset is later used to fine-tune models (like for intent classification and question answering). In addition, the platform makes it easy to generate integration test cases whenever a change is made to the code base.

1.2 Rapid Prototyping Platform

The overall system architecture is part of a custom rapid prototyping platform we have developed to enable fast experimentation and automated integration testing as illustrated in Figure 1.

Howdy Y'all dashboard. The dashboard stores all of the conversations along with their details (e.g., the user utterance, bot response, which response generator was selected, the intent label, and so on). Further, the dashboard supports annotation so that our team can generate labeled training data for our core components (e.g., labeling the correct intent for each utterance). This supports rapid experimentation, where we can train new models that are then tested over this labeled data. The dashboard also supports conversation labeling, so we can identify conversations that faced difficulties.

Automated integration testing framework. We also designed a complementary testing framework that enables automated integration tests to ensure bug fixes and new features do not break the functionality of the system. The framework extends the Cobot SDK by (i) providing a script generate_test.sh that takes a directory (e.g. testcase_select_by_title) which has a required file called scenario.txt that contains pairs of user utterances and bot responses. The framework will automatically generate two more files in this directory namely events.json and validation.py. The whole directory becomes a single test case and the only thing required by the developer is to design and maintain the scenario. These test cases are then run every time a change is made to the code base and the results of these test cases inform the decision to deploy to production.

2 Intent Understanding

Given a user utterance, what is the user's underlying *intent*? This intent understanding is a critical component of a TaskBot; should we search for a recipe? start a timer? or go back a step? Our approach combines rules, Soundex, and a transformer-based intent classifier trained over thousands of examples from our rapid prototyping platform.

Rule-Based Intent Classification. In our initial design, we relied upon regex-matching techniques combined with the state manager and previous system responses to classify a user's utterance. There are two types of intents: the basic ones (YesIntent, NoIntent, etc.) and high-level ones (newSearch, Recommendation, etc.). The same user utterances under different dialogue state are classified as different intents. For example, "ok" is classified as "YesIntent" when the user is answering the confirmation question asked by our bot and as "NextIntent" when the user is navigating inside the task.

Soundex-Based Intent Classification. After deploying Howdy Y'all, we found many cases where the utterance text provided by the Automatic Speech Recognition (ASR) module is often noisy and has errors, in that it is often *phonetically similar* to contextually expected utterances as shown in Table 1. Hence, we adopted the well-known Soundex approach for identifying phonetically similar words as part of our rule-based intent classifier.[1]

Contextually Expected User Utterance	Output Examples from ASR Module
start cooking	*star cooking*
next	*best, let's, list*
ingredient	*a gradient*
start task	*start cast, fart task, start disk*

Table 1: Examples of ASR Errors

Transformer-Based Intent Classification. Naturally, these rule-based approaches provide a baseline foundation for intent classification but face difficulty in handling complex paraphrases and context information as illustrated in Figure 2.

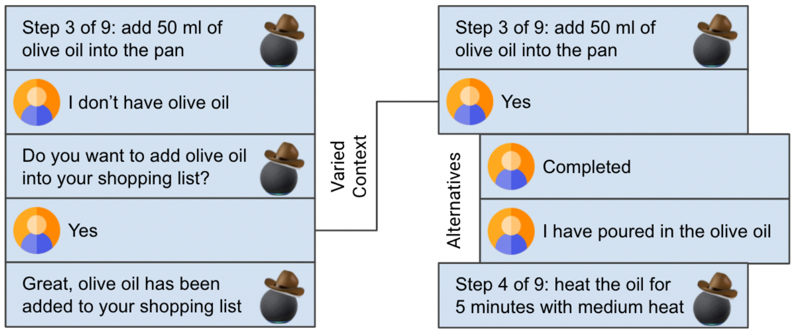
Step 3 of 9: add 50 ml of olive oil into the pan
I don't have olive oil
Do you want to add olive oil into your shopping list?
Yes
Great, olive oil has been added to your shopping list
Varied Context
Step 3 of 9: add 50 ml of olive oil into the pan
Yes
Alternatives
Completed
I have poured in the olive oil
Step 4 of 9: heat the oil for 5 minutes with medium heat

Figure 2: Different expressions of the same intent and the same expression of different intents

Traditionally, intent classifiers of chatbots and dialogue systems map the user utterance to a specific task (such as PlayMusic, GetWeather, BookRestaurant, etc.) of the system [Coucke et al. (2018); Hemphill et al. (1990)]. We investigated the transcripts of our user-bot conversations and observed that the user may not always have a clear intent. Hence, we extend our intent classifier to a *reaction-based* intent classification model capable of choosing the best responder of our system when the user utterance does not yield a clear intent. For instance, 1) the intent classifier may detect that the user does not have a clear DIY or cooking task in mind, so the TaskBot can invoke the conversational recommender to make a *Recommendation*; 2) the intent classifier may also label the user utterance

[1]For example: *b*, *f*, *p*, *v* are all mapped to the number 1, because they are all pronounced from the front of the mouth.

as *Ambiguous* when the semantic meaning is not clear so that the TaskBot can ask a clarification question.

Based on our rapid prototyping platform, we identified 27 intent classes as listed in Table 2 for our TaskBot. The 27 intent classes can be further categorized into 14 categories associated with the *General*, *PreTask*, or *InTask* stage of the conversation. After a training and trial annotation session, six team members inspected 4,819 turns of conversation. We carefully selected 3,169 labeled utterances as our training data based on the quality of the conversations. To ensure a golden test set, three additional team members further reviewed the labels of 634 user utterances.

Stage	**Level-1 Intent**	**Level-2 Intent**	**Expected TaskBot Reaction**
PreTask	Search	WikiHow	Searching WikiHow articles
		WholeFood	Searching Whole Food recipes
		MoreResults	Providing more search results
	SearchResultsQA	-	Answering questions regarding the search results
	Recommendation	-	Performing conversational task recommendation
	Select	Default	Starting a task with default selection
		n/x	Starting task with the n-th option or name x
InTask	Navigation	Next	Reading the next content
		Previous	Reading the previous content
		N	Reading step n
		Deails/ListAll	Reading details of current step or list all ingredients
		SwtichMode	Switching of ingredient/step mode
	Question	Subsitutes	Prompting subsitutes or shopping list management
		QA	Answering task-related questions
General	Functional	Pause	Pause the TaskBot and waits for instructions
		Cancel	Cancel the current search or task
		Stop	Terminate the conversation
	SeekingHelp	-	Providing help message based on the context
	Repeat	-	Repeating what the TaskBot just said
	AlexaFunctionalities	Shopping List	Managing the user's shopping list
		Timer	Managing the timer
		AdjustVolume	Managing the volume of the device
		Others	Apologizing for unsupported Alexa functionalities (making records for future support consideration)
	Greetings/ChitChat	-	Making conversational response
	Ambiguous	-	Asking clarification question based on the context
	Dissatisfaction	-	Apologizing to and calming down the user
	NotSupported	-	Responding to input that is sensitive, offensive, or out of our domain

Table 2: List of Intents

Based on the annotated data, we fine-tuned a RoBERTa-BASE model transformer introduced by Liu et al. (2019) with a linear sequence classification layer on top of the pooled output. Besides the incoming user input, the model also considers the context of the conversation, which includes the previous turns of user inputs, TaskBot responses, and classified intention labels (up to 5 turns). We report the model's performance under four different settings in Table 3.

Model	**Accuracy**	**Macro Averaged**		
		Precision	**Recall**	**F-1**
Fine-tuned RoBERTa-BASE	88.84	67.25	63.32	63.99
- reduced amount of context(up to 3 turns)	86.28	65.18	59.78	60.94
- w/o classified intention labels	85.33	62.88	55.94	56.57
- w/o context	81.72	61.29	53.83	55.13
Popularity Baseline	36.57	2.61	7.14	3.83
Stratified Random Baseline	20.83	7.04	7.12	7.04

Table 3: Evaluation Results of the Intent Classification Model

The evaluation results suggest that utilizing context information significantly boosts the model's performance. Our qualitative analysis also suggests the model's superior performance in capturing the meaning of unseen language expressions that typically would overwhelm a rule-based system. However, our analysis of the best-performing model also reveals that inconsistent training labels and the lack of training data limit its performance on tail classes. Hence, we are refining our intent annotation guidelines, cross-validating with existing annotations, actively labeling new data, exploring data augmentation methods, and experimenting with more powerful pre-trained language models.

3 Improving Search and User Preference Elicitation

Guiding the user to the right task is a critical challenge. Here, we introduce our approach for both *search*, when the user has a clear task in mind (like "Help me make a chocolate cake") and *elicitation*, when we must uncover user preferences to find the right task (like "I'm interested in dessert.").

3.1 Improving Search

In our initial deployment of Howdy Y'all we discovered that the existing search APIs work reasonably well with simple keyword-based searches, but sometimes returns irrelevant results when the exact combination of words used in the user's query does not appear in the indexed content. As a first step we experimented with improving search results by query expansion – by rewriting the user query with other lexical variants and synonymous words. This method worked well but is not robust to new terms and requires considerable manual effort to maintain. Hence, we turned to a dense retrieval approach inspired by recent successes in using pre-trained language models for search, e.g., Reimers and Gurevych (2019).

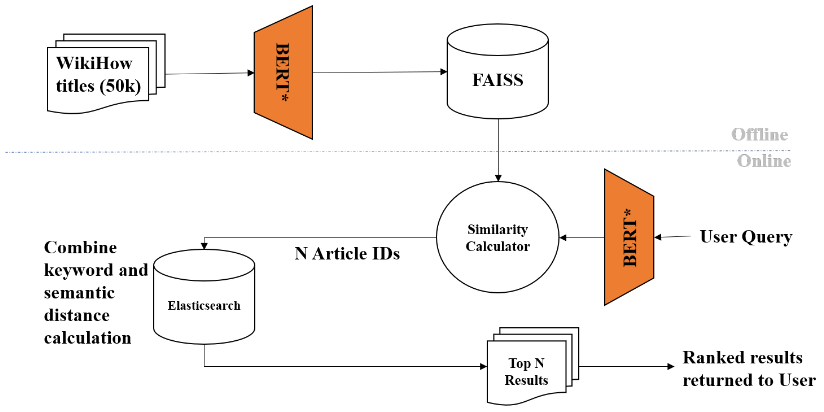

Figure 3: Howdy Y'all Search Framework

In our case, we not only wanted to encode text into a vector representation, but also need an efficient mechanism to compute a similarity score between the user's query and document title. Sentence-BERT (SBERT) Reimers and Gurevych (2019) is a type of BERT model that uses a twin network to generate semantically meaningful sentence embeddings and calculate a similarity score that shows how two sentences are semantically related. We used existing pre-trained SBERT models and further pre-trained using the WikiHow dataset. We converted title and description from each WikiHow document into its vector form using SBERT once and used it to compute similarity against a user query at runtime.

Since the WikiHow Koupaee and Wang (2018) and WholeFoods datasets have more than 100,000 documents, finding relevant documents by computing similarity scores at runtime is time-consuming. Hence, we adopt an in-memory index called FAISS Johnson et al. (2019), which compresses and stores vectors of large dimensions using a technique called product quantization. FAISS also allows searching for multiple queries at a time which is ideal for our TaskBot.

Based on our logged conversations, we created a small evaluation dataset for DIY tasks and compared the performance of our search against the default WikiHow search. We report the search performance using three evaluation metrics (See Table 5) : (i) Normalized Discounted Cumulative Gain at 10 (NDCG@10); (ii) Mean Ranking Position (MR); and (iii) Hit Rate @ 10 (Hit@10).

Apart from finding relevant documents, this approach is also able to recommend articles when exact results are not available. For recipe tasks, we follow a similar approach (finetuning instead on a WholeFood dataset), plus an additional sentence embedding based on contrastive learning Gao et al. (2021) over a large recipe dataset Majumder et al. (2019).

User Query: How to clean my restroom	
Elasticsearch	**Ours**
How to Overcome Public Restroom Embarrassment	How to clean a bathroom
How to use a public restroom	How to clean bathroom grout
How to avoid germs in public restrooms	How to clean a bathroom sink
How to setup a restroom trailer	How to clean a bathroom sink drain
How to clean bathroom tile	How to clean a jetted tub

User Query: How to color my wall	
Elasticsearch	**Ours**
How to pick a color for an accent wall	How to paint a wall
How to choose paint color for an bedroom	How to paint an interior wall
How to paint a concrete wall	How to paint walls near a ceiling
How to change your eye color	How to change your eye color
How to change your eye color	How to paint textured walls

Table 4: Comparison of search results for a set of sample queries

Evaluation Metric	**Elasticsearch**	**Ours**
NDCG@10	0.285	**0.618**
MR	7.614	**4.552**
Hit@10	0.441	**0.824**

Table 5: Preliminary Evaluation Results.

3.2 User Preference Elicitation

In many cases, a user may not have a specific DIY task or recipe in mind. For example, a user asking for a cake recipe may simply say "Alexa, help me find a cake recipe." Searching for cake recipes will yield 100s of potential matches, overwhelming the user. Hence, we pair our search component with *user preference elicitation* in cases like this to help guide the user to a recipe (or DIY task) best matching the user's underlying needs and preferences.

Approach. Our approach views the TaskBot as an instance of a Conversational Recommender System (CRS). Unlike a traditional recommender system, a CRS utilizes natural language to directly converse with the user to explicitly elicit the user's preferences towards a certain item (either a recipe or a WikiHow article). On one hand, the elicited preferences can help Howdy Y'all quickly narrow down the candidate item space thus recommending only the most relevant items; on the other hand, the elicited preferences can be used as human interpretable justifications on why certain items are recommended (or not recommended). For the aforementioned 'cake recipe example,' an example conversation is shown in Figure 4.

CRS Challenges and Approach. While a CRS can achieve accurate recommendations by directly conversing with the user, it faces two major challenges – user impatience and exploitation vs. exploration (deciding when/what to ask and when to recommend). Unlike a traditional recommender system that gives one-shot recommendations to the user, a CRS interacts with the user to gradually learn the user's detailed preferences and eventually guide her to relevant recommendations. However, this interaction process can be tedious and might cause the user to quit the system due to impatience. Therefore, when designing a CRS, it is crucial to take user experience into consideration. On one hand, we want the system to learn as many preferences as possible (exploitation) from the user, so the recommendations could be highly personalized; on the other hand, we must make recommendations (exploration) before the user leaves due to impatience.

At each turn we can either prompt the user for more preferences or recommend a small set of items based on the user preferences that the system has learned so far. Therefore, it is crucial for a CRS to develop or learn an action policy that: (1) decides when to recommend versus when to ask more preferences; and (2) if the system decides to acquire more preferences from the user, which attribute from the candidate attribute set should be asked (e.g., for a cake recipe, such candidate attribute set might include cook time, servings, dietary constraints, and occasion). To address the aforementioned two challenges, we adopt a simple yet effective heuristic-based attribute selection approach. For deciding whether to prompt the user for more preferences or recommend items based on the preferences learned so far, the system chooses to recommend with the probability of $\frac{n}{max(|V|,n)}$ where V is the current candidate set and n is a hyperparameter that can be tuned. The intuition is that the confidence of recommendation grows when the candidate size is smaller. For choosing

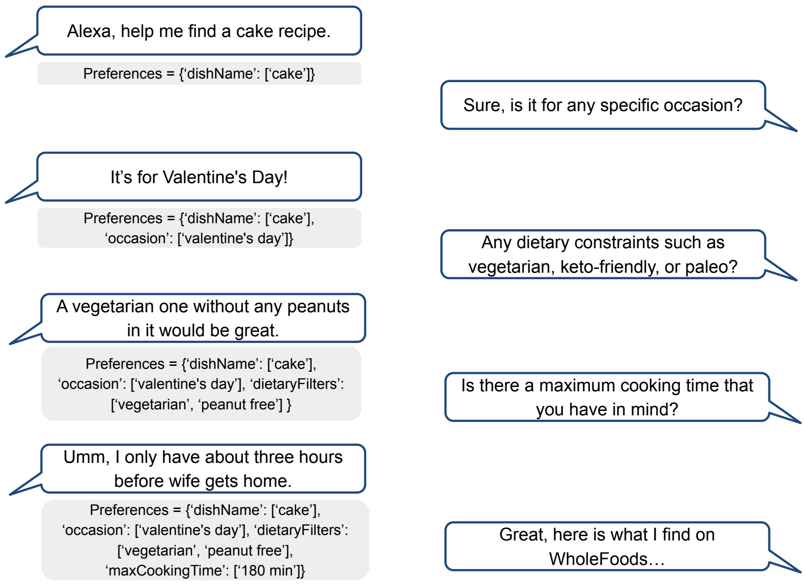
Alexa, help me find a cake recipe.
Preferences = {'dishName': ['cake']}
Sure, is it for any specific occasion?
It's for Valentine's Day!
Preferences = {'dishName': ['cake'], 'occasion': ['valentine's day']}
Any dietary constraints such as vegetarian, keto-friendly, or paleo?
A vegetarian one without any peanuts in it would be great.
Preferences = {'dishName': ['cake'], 'occasion': ['valentine's day'], 'dietaryFilters': ['vegetarian', 'peanut free'] }
Is there a maximum cooking time that you have in mind?
Umm, I only have about three hours before wife gets home.
Preferences = {'dishName': ['cake'], 'occasion': ['valentine's day'], 'dietaryFilters': ['vegetarian', 'peanut free'], 'maxCookingTime': ['180 min']}
Great, here is what I find on WholeFoods…

Figure 4: Example Conversation to Elicit User Preferences

which attribute to prompt to the user, we calculate the entropy for all the remaining attributes in the candidate set and choose the attribute with the maximum entropy. More specifically, we calculate the entropy of each candidate attribute as follows:

$$Entropy(p, V_{cand}) = -prob(p)log_2(prob(p)), prob(p) = \frac{|V_{cand} \cap V_p|}{|V_{cand}|}$$

In practice, we have found that this elicitation component is quite challenging to effectively incorporate into the TaskBot for two reasons. First, it requires identifying a consistent set of attributes for candidate tasks. Since DIY tasks vary so greatly, we are focusing on recipes where there are some common attributes (like cook time and dietary restrictions). Second, many users do not have clearly defined preferences, so repeated questioning can lead to poor engagement versus directly making recommendations at the beginning of the conversation.

4 Improving Question Answering

Our users ask many questions, so Howdy Y'all relies on several question answering (QA) components. As a baseline, we rely on the Amazon EVI QA API to answer *factoid* questions, with a QA classification model that decides whether the response is relevant or irrelevant to the question. Further, we have built a database of tasks and recipes where we record attributes like the quantity of ingredients, oven temperature, cooking time. In this way, we can directly answer *extractive* questions like "How much oil do I need?". We augment this database with a common set of substitutes, so we can answer user needs like "I don't have any olive oil."

Beyond factoid and extractive QA, Howdy Y'all also supports richer *abstractive* QA. As a baseline, we used 20,202 question-answer pairs from the community Q&A section under each WikiHow article to fine-tune a BART (Lewis et al., 2020) model as our base abstractive QA model. Example pairs can be found in Table 6. While a vanilla BART model provides a good first step, we propose to learn a better sentence representation to provide higher-quality answers.

Concretely, we propose a new question generation model which uses contrastive learning and answer reconstruction. We use ROUGE scores as our automatic metrics and (1) fluency: whether the questions are grammatically correct and fluent; (2) relevance: whether the question is related to the answer; and (3) correctness: whether the question can actually be answered by the provided answer as determined through human evaluation. This generation model achieves better results than the vanilla BART-base model by 2.0% on ROUGE-L score. In addition, we randomly sample 100 QA pairs and ask three annotators to rate the generated questions from the baseline model and our best model shows 1.3% and 2.6% improvement on relevance and correctness in human evaluation.

Title: How to Prepare a Healthy Meal for Your Pet Dog.
Summary: To prepare a healthy meal for your dog, choose lean meat with the bones and fat removed, like chicken or beef . . .
Q: What can I feed my dog if I have run out of dog food?
A: In the short term, any bland human food such as chicken or white fish with rice or pasta is just fine . . .
Q: How much homemade dog food do you feed your dog?
A: Great question because it highlights one of the problems of feeding home prepared foods . . .
Q: What should I not feed my dog?
A: There are many human foods that are toxic to dogs. Top of the list of foods NOT to give are . . .

Table 6: Question-answer pairs from WikiHow.

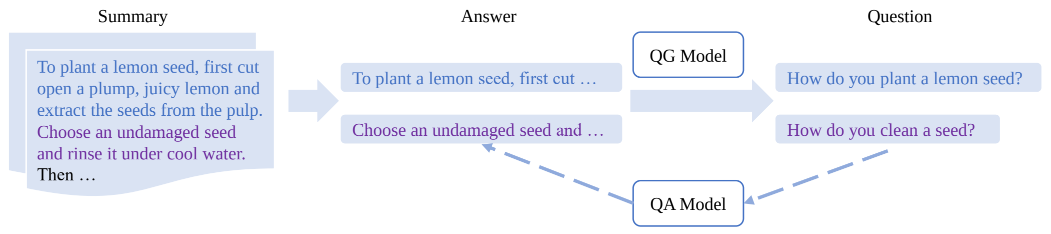
Summary
Answer
Question
To plant a lemon seed, first cut open a plump, juicy lemon and extract the seeds from the pulp. Choose an undamaged seed and rinse it under cool water. Then . . .
To plant a lemon seed, first cut . . .
Choose an undamaged seed and . . .
QG Model
QA Model
How do you plant a lemon seed?
How do you clean a seed?

Figure 5: An overview of synthetic data generation framework and generated question examples from our QG model.

This question generation can then be used as part of a data augmentation approach to improve the quality of answers we generate. Each sentence from the summary of a WikiHow article is regarded as an answer, and then we use our proposed question generation to generate corresponding questions. The generated 200K synthetic QA pairs can then be fed into our previous QA model to perform pre-training. Experimentally, we find the new QA model achieves better results than the vanilla BART-base model by 3.6% on ROUGE-L score.

User Utterance	**Vanilla BART-base**	**Ours**
How to find spring fashion trends?	Spring is a great way to find spring fashion trends online.	You can find spring fashion trends by checking out fashion sites like fashionweek and amazon.
What is mozzarella cheese?	Mozzarella cheese is a type of cheese that is made from fresh, grated mozzarella. It can be made from different types of cheeses, such as cream cheese, cheese curds, and cream cheese substitutes.	Mozzarella cheese is a type of cheese that is made from fresh, ground mozzarella. It is a great addition to pizza or other casual dishes.
What is Chinese strainer?	Chinese strainer is a type of strainer that is made of bamboo. It is used to make a strainer from bamboo.	A chinese strainer is a type of strainer that is used to remove food from dishes in chinese cuisine. it can be used for soups, stews, soups and stews.
Can we work on the last topic when I was here what we were doing?	Yes, you can definitely work on the last topic when you're back in class.	Yes, you can work on the last topic if you wish.

Table 7: Example questions and answers.

To illustrate, we show in Table 7 several example questions and the answers generated by a Vanilla BART model and our approach.

5 Emotional Conversation Tracking

We believe that an important part of interfacing with the user in a conversational setting is understanding and addressing their emotions, and we believe that connecting with the user can be an opportunity to help move the user towards task completion. Hence, we have designed a *empathic responder* and a *negative conversation monitor*.

Empathic Responder. There are two main approaches we take to create a conversational tone: a *proactive approach* where we take the initiative to create a conversational tone can help encourage the user to engage more with the TaskBot, and a *reactive approach* where we respond to the user's conversational utterances. To illustrate the proactive approach, we conversationalize existing steps in a task (or recipe). For example, at the halfway step we prepend "We're halfway there!" to the step, as an encouragement to the user. Similarly, we congratulate the user upon completion of a task. To illustrate the reactive approach, we have custom responders designed for (i) when the user wants to chat, so we can engage in a some banter before returning to the task; (ii) when the user tells the TaskBot their name or emotional state, so we can personalize our follow-on conversations; and (iii) when the user thanks the TaskBot.

Identifying Negative Conversations. In many conversations, users become frustrated if the system repeatedly misunderstands them. This type of situation is a negative user experience. Therefore, we must find a way to recover and provide a better user experience. The first step to recovering from a negative user experience is to detect this type of situation as it is occurring. To do this, we predict the final rating of an in-progress conversation based on features such as length and the number of times the Fallback Response Generator is called. We use these features to compare the current conversations to previous conversations with similar features. We then use the final ratings of these similar conversations to predict the rating of the current conversation. If this predicted rating is below a threshold, we know that the current conversation is going poorly, and we can act to recover from this situation. Once we detect that a conversation is going poorly, we apologize to the user in an empathetic manner to deescalate the situation.

6 Future Work

In our ongoing work, we are continuing to refine and improve the core components introduced above. And with considerable new conversations in the semi-final event, we have a much larger set of annotated conversations to train our models. In one exciting direction, we are exploring the potential of *learning to conversationalize*. Much of the core recipe and DIY-task specific instructions are based on articles that were originally designed to be read (e.g., the many steps on WikiHow for how to paint a room). These instructions are fundamentally a mismatch with a naturalistic conversation. Hence, we aim to learn new methods to transform these instructions into conversations: to *conversationalize* them. Our hypothesis is that this can be helpful in promoting user engagement by improving our proactive approach to conversation. There are two main approaches we plan to explore in conversationalizing instruction steps: a rule-based modifier and neural style transfer. To further improve the question answering capability of our TaskBot, we are also working on a self-supervised answer selection method for question generation. The method aims at building models that can knock out backbone tokens while preserving candidate answer tokens of an input passage.

The rule-based modifier is similar to the approach taken by our Empathic Responder, but applied to all instruction steps. By prepending transition text to our instruction steps, we hope to soften the tone of our TaskBot and make it more conversational. For example, the speech output, "Take the cake out of the oven," can become "Nicely done! Next, take the cake out of the oven." This aims to make our TaskBot smoother in transition, while also acknowledging the work that the user has done to this point. The neural style transfer approach builds on recent work in text style transfer to control attributes in text like politeness, emotion, formality, and so on Jin et al. (2020). With a parallel conversational version of our instruction steps, we could generate our desired text with our target style using standard sequence to sequence models, but it is difficult to find a quality parallel dataset for DIY and recipe instructions containing a conversational tone. Instead, we will investigate prototype editing methods to transfer from source text x with attribute a to its counterpart x' with attribute a'. This method has the advantage of explicitly retaining content attributes of a sentence while changing style attributes. This is particularly important in our TaskBot as editing the content itself could result in deteriorated instruction quality for users. Using prototype editing, our target style a' could be also changed depending on what is most conducive to user engagement, resulting in a "polite" or "funny" TaskBot.

References

Adiwardana, D., Luong, M.T., So, D.R., Hall, J., Fiedel, N., Thoppilan, R., Yang, Z., Kulshreshtha, A., Nemade, G., Lu, Y., et al., 2020. Towards a human-like open-domain chatbot. arXiv preprint arXiv:2001.09977 .

Coucke, A., Saade, A., Ball, A., Bluche, T., Caulier, A., Leroy, D., Doumouro, C., Gisselbrecht, T., Caltagirone, F., Lavril, T., et al., 2018. Snips voice platform: an embedded spoken language understanding system for private-by-design voice interfaces. arXiv preprint arXiv:1805.10190 .

Gao, T., Yao, X., Chen, D., 2021. Simcse: Simple contrastive learning of sentence embeddings. arXiv preprint arXiv:2104.08821 .

Hemphill, C.T., Godfrey, J.J., Doddington, G.R., 1990. The atis spoken language systems pilot corpus, in: Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27, 1990.

Jannach, D., Manzoor, A., Cai, W., Chen, L., 2021. A survey on conversational recommender systems. ACM Computing Surveys (CSUR) 54, 1–36.

Jin, Jin, Hu, V., Mihalcea, 2020. Deep learning for text style transfer: A survey. arXiv preprint arXiv:2011.00416 .

Johnson, J., Douze, M., Jégou, H., 2019. Billion-scale similarity search with gpus. IEEE Transactions on Big Data 7, 535–547.

Koupaee, M., Wang, W.Y., 2018. Wikihow: A large scale text summarization dataset. arXiv preprint arXiv:1810.09305 .

Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., Zettlemoyer, L., 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, pp. 7871–7880.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., Stoyanov, V., 2019. Roberta: A robustly optimized bert pretraining approach. arXiv preprint arXiv:1907.11692 .

Majumder, B.P., Li, S., Ni, J., McAuley, J., 2019. Generating personalized recipes from historical user preferences. arXiv preprint arXiv:1909.00105 .

Reimers, N., Gurevych, I., 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. arXiv preprint arXiv:1908.10084 .

Zhang, Y., Chen, X., Ai, Q., Yang, L., Croft, W.B., 2018. Towards conversational search and recommendation: System ask, user respond, in: Proceedings of the 27th acm international conference on information and knowledge management, pp. 177–186.