
Making Something out of Nothing: Building Robust Task-oriented Dialogue Systems from Scratch

Zekun Li*, Hong Wang*, Alon Albalak, Yingrui Yang, Jing Qian, Shiyang Li, Xifeng Yan[†]

Department of Computer Science
University of California, Santa Barbara
Santa Barbara, CA 93106

{zekunli, hongwang600, alon_albalak, yingruiyang, jing_qian, shiyangli, xyan}@ucsb.edu

Abstract

In this paper, we introduce Gauchobot, a task-oriented dialogue system developed for the Alexa Prize Taskbot Challenge. We identified two main obstacles to building better conversational AI assistants in real-world applications. The first is great human efforts needed in data annotation and engineering a dialogue system that provides service in a new domain from scratch. The high cost, often ignored by existing research work, has blocked the broad deployment of dialogue systems. The second is a lack of robustness when facing undesirable situations during a conversation in real scenarios. The existing paradigm, which pre-defines dialogue flows and confines the users to a box with restricted options, makes dialogue systems easily stumped by complex conversations. To solve these two issues, we invent a methodology that can automatically generate data with minimum human efforts to train a unified framework capable of handling various sub-tasks in completing a task-oriented conversation. Feedback from real-world users can be easily incorporated into the model by automatically generating more training data and thus improving the model over time. Besides, we integrate multiple generative-based and retrieval-based response generation models into our Gauchobot, making it capable of handling not only task-oriented commands, but also QA, chit-chat, and other out-of-domain cases. As a result, Gauchobot can not only help complete tasks with rich user experience, but also provide a general framework of building robust task-oriented dialogue systems quickly from scratch.

1 Introduction

Conversational AI has been a long-standing area of interest in Computer Science. With recent advancements in Natural Language Processing [5], it has gained more attention from both academia and industry. One important yet challenging direction is to build task-oriented dialogue (TOD) systems, where a conversational AI assistant is used to assist customers in completing complex tasks.

The typical pipeline of a task-oriented dialogue system usually consists of four essential modules: the natural language understanding (NLU) module [27, 26, 4] to parse user utterances into predefined user intentions, the dialogue state tracking (DST) module [28, 25, 7, 12] to track and store the dialogue state along with the dialogue history, the dialogue policy management (DP) module [17, 16, 22] to determine the next dialogue action based on current dialogue state, and the natural language generation (NLG) module [24, 18] to map the selected action to its surface form and generate the response in natural language. Following this pipeline there are, in general, two approaches to building a TOD system. The first approach is a carefully designed and programmed rule-based system, where

*Equal contribution and team leads.

[†]Faculty advisor.

the developers have to define the *intent* and *dialogue policy* for all possible cases, usually in the form of dialogue flows. This approach gives the developers full control of how to handle tasks and interactions with users, allowing them to incrementally add new features. The second approach is to build neural conversational agents [8], which require large quantities of annotated dialogues to train each component of the system. These annotated dialogues are often collected and annotated via crowd-sourcing in the Wizard-of-Oz setup [9].

Cold Start Each of the aforementioned approaches, however, has crucial drawbacks when the system needs to be developed and deployed from scratch within a short time period. Rule-based systems require **massive engineering**, making it impossible to build a working system within limited competition time and human resources. On the other hand, neural-based systems require **massive training data** in order to achieve good performance, which was not available at the beginning of the competition. Actually many dialogue projects face the same kind of bootstrapping dilemma, which has blocked the broad deployment of dialogue systems in real-world applications. In this competition, rather than engineering a solution specialized for cooking/DIY tasks we aim to develop a general framework, minimizing both human annotation and engineering efforts, so that it can be reused to quickly build robust task-oriented dialogue systems. We design a pipeline that can generate training data for each sub-task from a small number of configuration files, avoiding large-scale, tedious user annotation. Various data augmentation techniques are incorporated to increase the size and variety of training data. Generally, modules in the pipeline (e.g. NLU, DP) are handled by separate models, limiting the scalability of a dialogue agent. We leverage transfer learning among different tasks by unifying all the sub-tasks to a text-to-text format and use a T5 [20] model to perform multi-task learning. Leveraging our pipeline, we can automatically generate a large amount of diverse data by simply writing a few configuration files to train a model capable of performing all the sub-tasks in a task-oriented dialogue system.

Robustness A complex dialogue system consists of multiple components and errors accumulated from previous modules can be propagated to the later ones [13]. Therefore, each component in the dialogue system should be robust. Automatic speech recognition (ASR) is the first step of a spoken dialogue system. To avoid potential ASR errors, we develop a contextual ASR error detection and correction method. To have an accurate understanding of the user’s intent, we design multiple tasks including intent classification, slot filling, and intent entailment check. We also propose a data augmentation method based on GPT-3 [2] to increase the size and diversity of training data for these tasks. As for dialogue policy learning, to handle the variation and exception in dialogue flows, we made efforts from two aspects: (1) develop automatic data generation and data augmentation methods to cover as many dialogue flows as we can; and (2) integrate multiple response generation models to generate responses when encountering unexpected cases, trying to recover from unknown intents.

User Engagement Keeping users engaged is crucial as better user engagement leads to a better user experience. To improve user engagement, we design several methods to help the users find the tasks that they are interested in. To make the task completion more interesting and engaging, we utilize large language models to incorporate content about the task, such as fun facts, stories, and tips.

In conclusion, our principle is to design a robust dialogue system that can provide a good user experience with minimal development cost and effort. In the following sections, we will discuss how to implement it in detail.

2 System Overview

Figure 1 illustrates a simplified overview of our Gauchobot’s system design. A dialogue turn starts with the Amazon ASR model converting user speech into text and passing it to our bot. ASR Error Correction module tries to detect and correct the potential ASR errors, and passes corrected text into the Hierarchical Intent Detection module, which chooses one of the following three pipelines:

- **Task-oriented Dialogue (TOD):** This pipeline is the most basic and important one. It will interpret the user’s task-oriented goal and then respond by providing information and performing system actions such as querying a task and navigating the steps.
- **Question Answering (QA):** In addition to the commands that are provided by the system action or database, users may ask questions during the task completion process. In this case, the question answering pipeline will be invoked to deal with different kinds of questions.

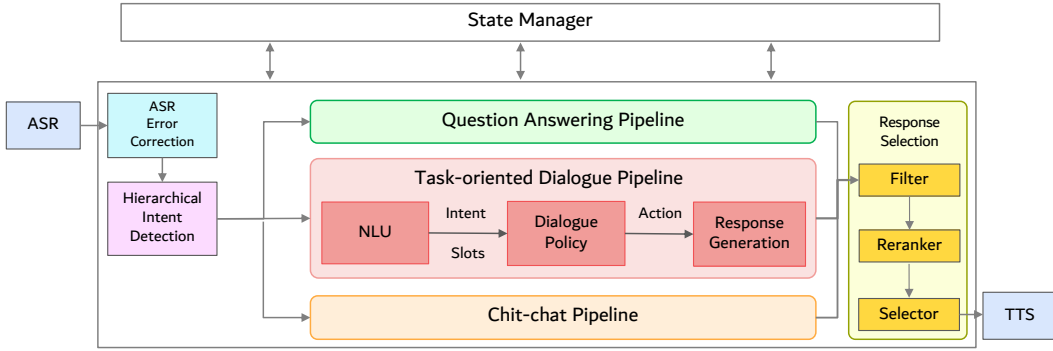


Figure 1: A simplified overview of Gauchobot's system design.

- Chit-chat: Sometimes the users' intention may not be explicit or task-oriented, such as in social chit-chat. To handle this kind of dialogue turns and provide a smooth transition between task-oriented and non-task-related turns, the chit-chat pipeline will be invoked. Note that any unknown out-of-scope (OOS) intent will be handled by this pipeline.

Once the candidate responses are generated by the invoked pipeline, they will be passed to the response selection module, where a suite of response filters, the response reranker, and the response selector are used to select the best response from the candidates based on the user query, the current dialogue state, and the conversation context. The output is then returned to Amazon's Text-to-Speech model, which converts the generated text to audio response and delivers it to the user. Throughout the whole process, we use state manager, a reliable database provided in the cobot codebase, to maintain the dialogue state.

3 Task-oriented Dialogue Pipeline

A general process of a task-oriented dialogue is to first find a task instruction based on the user's query and then guide the user to finish the task by following the instruction step by step. To achieve this, we mainly have three core components (1) Natural Language Understanding (NLU), which aims to understand the user's intents and extract the value of our pre-defined slots. (2) Dialogue Policy Learning, which maps the user's intent to a predefined dialogue action. (3) Response Generation, which converts the output of these actions (usually) into natural language and surfaces it to the user. In the following parts, we mainly talk about the NLU and Dialogue Policy modules in Section 4. We will give a brief description of their functions and then introduce how we use the unified framework to implement these two modules. Besides, we will also discuss how to make the generated response more user-engaged in Section 7.

4 A Unified Framework for Building TOD from Scratch

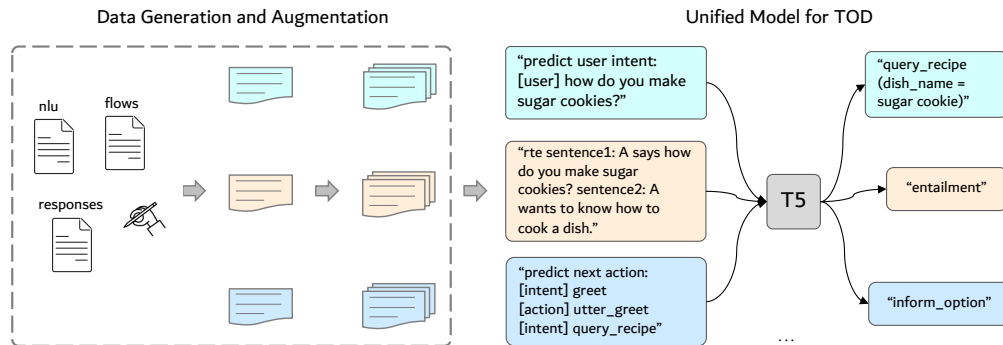


Figure 2: The unified framework of developing the task-oriented dialogue pipeline from scratch.

Here we introduce the unified framework we use to build a task-oriented dialogue system pipeline. As shown in Figure 2, we can modify a few human-readable YAML configuration files, based on which the data of different sub-tasks can be generated automatically. Using this small set of generated data as seed, we perform various data augmentation methods to expand the scale and diversity of training data. We integrate different dialogue modules (NLU and DP) into a unified model and train the model through multi-task learning. The input and output of each sub-tasks of TOD are formatted into text and a unified T5 model is used to predict both intent and action as a text-to-text task [20].

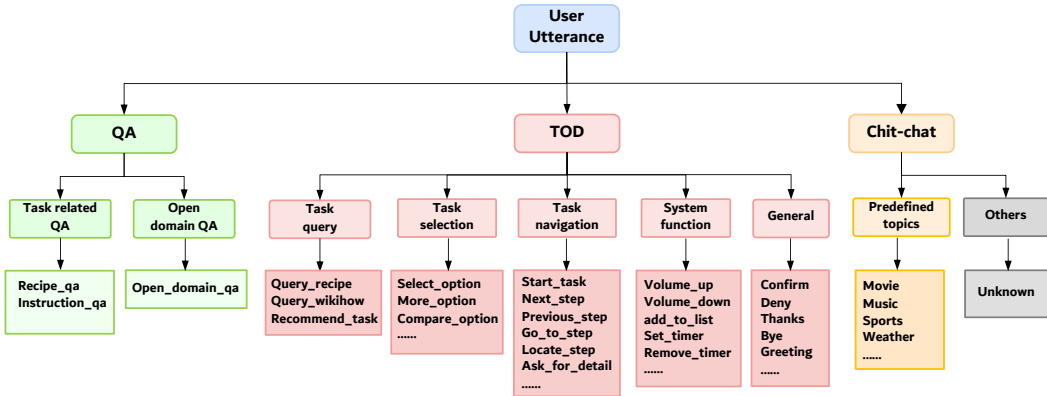


Figure 3: A hierarchical schema of dialogue intents (only frequent intents are listed).

4.1 NLU Module

Different from the coarse intent classification module mentioned in Section 2, in the TOD pipeline we classify the user’s query into more fine-grained task-oriented intents, as shown in Figure 3. In addition to the task-oriented intents, we also further categorize the QA and Chit-chat intents, in order to select the corresponding dialogue actions. Any out-of-domain intent will finally fall into the Chit-chat → Others → Unknown category and invoke the chit-chat pipeline to generate the response. Developing a model to accurately detect users’ intents needs to deal with the **surface form** problem (different choice of words to convey the same intent). Therefore, it is critical to use a large amount of diverse training data to train the model.

4.1.1 Data Generation from Configuration Files

Inspired by Rasa [1], we use human-readable YAML files to store the configuration information for NLU and DP. Figure 4 illustrates an example of the format of NLU configuration files.

Intent Classification & Slot Filling These two tasks are usually performed together. Given a user’s query utterance, we aim to predict the user’s intent and also extract the value of mentioned slots. To generate training data, we randomly select possible slot values and fill them in the corresponding slot placeholders in the templates (the orange rectangle in Figure 4). However, there are no fixed patterns or templates for QA intents. Therefore, we use the questions in Natural Question (NQ) [11] dataset as the training examples for the `open_domain_qa` intent. As for the `instruction_qa` intent, which is a question about a specific task, we use a question generation model [14] to generate question-answer pairs for each recipe and task in wikihow. The questions are used as training examples for `instruction_qa` intent to train the NLU model, while the question-answer pairs are used to train the QA model in the QA pipeline as introduced in Section 5.

Intent Entailment Check This task focuses on Recognizing Textual Entailment (RTE), whose goal is to identify whether a hypothesis sentence (the description of the predicted intent) can be entailed by a premise sentence (the user’s utterance). We illustrate an example in the green rectangle in Figure 4. For a user’s utterance whose intent is recognized as `query_recipe`, we train the intent entailment check model to identify whether the hypothesis of the user’s intent being `query_recipe` can be entailed by the user’s utterance. As we fine-tune a unified T5 model that has been pre-trained on the RTE task, the knowledge from RTE can be easily transferred to the intent entailment check task. To generate training data, we concatenate the user utterance of each intent with its intent description

as the positive samples (entailment), and the description of other intents as the negative samples (not_entailment). If an intent predicted by the model cannot pass the intent entailment check, the intent will be changed to unknown.

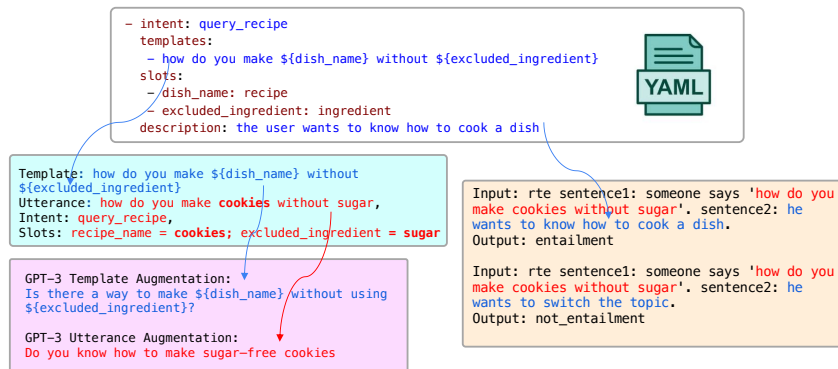


Figure 4: The generated data of various sub-tasks in the NLU module.

4.1.2 Data Augmentation Using GPT-3

Despite that we can generate a large amount of data by filling in different slot values, the generated samples are still in the same pattern, and lack linguistic diversity. To deal with this problem, we use various data augmentation methods. In addition to using the methods provided in the NLPAUG package [15], we introduce a method that prompts a large pretrained language model GPT-3 [2] to generate new examples that have the same meaning but different patterns with the existing data. We found that without any fine-tuning, the generated example data are of high quality and lead to a rise in accuracy of the intent classification model trained with the augmented data. We design two ways to perform data augmentation based on the data generated from configuration files using GPT-3.

Template Augmentation As shown in Figure 4, we use GPT-3 to generate similar templates as the specified ones. By this means, we can obtain many new training samples by filling in the new templates. However, the pattern within the slot value is still fixed.

Utterance Augmentation We can also fill in the slot placeholders to generate utterance samples first and then prompt GPT-3 to generate similar utterances. As shown in the purple rectangle in Figure 4, the data generated in this way are more diverse and can be seen as a paraphrase of the original examples. The model trained with these rephrased utterances is more robust in extracting the slot value. However, it incurs a higher cost to generate each individual sample separately. We mainly use this method to generate training examples for the query_recipe and query_wikihow intents, as we need to extract the queried task instead of entities from the user utterances.

4.2 Dialogue Policy Module

The Dialogue Policy Learning module provides the function to predict a dialogue action based on the detected intent and current dialogue states. For some intents, the policy is easy to learn as they have a one-to-one mapping from intent to action. For example, the intent of start_cooking is always mapped to the action that initializes the cooking process. We thus maintain a rule-based policy management part to handle these simple and fixed dialogue flows. This can also help us to ensure that the system is stable and some critical business logic rules are followed. However, as the rules can not cover any possible dialogue flows, we also try to generate data covering as many dialogue flows as possible, and encode the dialogue flows into the unified model.

4.2.1 Data Generation from Configuration Files

The configuration files for the dialogue policy module contain the dialogue flow templates (a.k.a. stories). For each turn, we will specify intent and slot information, and also the corresponding action. To facilitate reuse, we group some fixed collocation of turns as a sub-story, and a complete dialogue flow consists of a sequence of sub-stories. We specify the following four attributes for a sub-story in a dialogue flow template.

- **Probability**: as some sub-stories like greeting may not happen in every dialogue, we specify the probability of each sub-story happening in the current dialogue flow.
- **Insertable**: some sub-stories can be inserted in multiple positions in a dialogue, but some cannot. For example, asking the bot to hold on may happen at any time during a conversation, but a greeting usually happens in the beginning. Therefore, we specify whether a sub-story is insertable. If so, we will insert the sub-story at the possible positions in a dialogue.
- **Ordered**: this attribute indicates whether the multiple turns in a sub-story need to be ordered.
- **Repeatable**: some sub-stories can happen repeatedly such as step navigation, while some only happen once such as option selection. If a sub-story is repeatable, we will repeat it for a random time when generating possible dialogue flows.

By specifying these attributes in a dialogue flow template, we can generate a large number of possible dialogue flows, based on which we can simulate numerous dialogues. We train the model to predict the dialogue action given the intent and action of previous turns and the predicted intent of the current turn. Figure 2 shows the format of the constructing data for dialogue action prediction.

4.2.2 Constructed Dataset

The statistics of the constructed dataset (Gauchobot Data) for each task are shown in Table 1. On average we write only 21 templates for each intent, and by using our data generation and augmentation method we can generate around 1020 utterances (training examples) for each intent. As for DP, we generate 10000 dialogue flows based on 42 sub-stories. Note that we can control the size of the generated dataset. As the current data size has already achieved a satisfying accuracy so that we don't further generate more data. The dataset is split into train/validation/test sets by an 8:1:1 ratio.

Table 1: Statistics of the automatically constructed dataset for each task.

Task	Statistics	
Intent Classification & Slot Filling	Intents	54
	Slots	20
	Utterances	55063
	Avg. # template per intent	21
	Avg. # utterance per intent	1020
Intent Entailment Check	Positive training examples	55063
	Negative training examples	55063
Dialogue Policy	Sub-stories	42
	Generated dialogue flows	10000

4.2.3 Experiments

We evaluate the performance of the latest version of our taskbot on user feedback. We randomly select 251 conversations with 2376 turns covering all the defined intents (internal Alexa Prize testers' feedback has been excluded).

Intent type	#Intents	#Slots	#Templates	#Utterances	Feedback	
					Intent	Slot
Task query	3	12	344	26920	95.25	92.16
Task selection	4	1	57	1072	88.76	78.72
Task navigation	20	5	332	6112	93.76	83.67
System function	7	1	32	693	87.50	66.67
General	8	0	125	375	91.13	-
Task-oriented QA	2	1	134	9540	85.48	86.11
Open domain QA	1	0	-	10000	82.26	-
Predefined topic	9	0	117	351	62.50	-
OOD (Unknown)	-	-	-	-	43.28	-
Overall	54	20	1141	55063	87.21	89.75

Table 2: The summary of training data, and NLU performance for each intent type. The metrics are intent classification accuracy and slot filling F1.

The results are shown in Table 2. We can see that with most templates and training samples, the model achieves good accuracy in detecting the task query intent and extracting the query from user utterances. However, the performance of detecting the QA intents is not satisfying, as it is challenging to distinguish which type of QA is correct from the user’s query. Most of the failed cases come from the OOD intent. As we don’t have training data for OOD intent, many OOD utterances will be classified as in-domain intents. This is also the motivation for us to propose the intent entailment check task to check whether the predicted intent is correct. As for the dialogue policy learning, we find that system can recover any possible dialogue flows with an accuracy of 100%. Incorporated with the rule-based policy management module, we can maintain the system stability and make sure the crucial business logic rules are followed.

5 Blending Question Answering and Chit-chat with TOD

Table 4: An example dialogue (not with real users), containing engaging content (**task description**, **fun facts**, **tips**), **QA turns**, **chit-chat turns**, and **ASR errors**.

#	Utterance	Invoked Pipeline	Complementary
1	User: alexa, let’s work together. Bot: Hi, this is an Alexa Prize Taskbot. I’m happy to help you with cooking or any diy tasks.	Launch	Start of conversation.
2	User: how to make apple crumble? Bot: Most cultures have some form of baked apple dessert. Here is a great match from Whole Foods Market: simple apple crumble. Do you want to go ahead?	TOD	We retrieve related description of the query recipe from open source.
3	User: sure. Bot: simple apple crumble is a great choice! To begin with, you will need to prepare 8 ingredients. If you are ready, say start cooking.	TOD	
4	User: do you know the serving of this recipe? Bot: This recipe for simple apple crumble has 8 servings.	QA	We retrieve the document associated with the recipe to find the answer.
5	User: I don’t like cinnamon, can i use something else? Bot: I have found these substitution for cinnamon: powder ginger, cardamom, ginger, nutmeg, ground ginger, allspice, green cardamom, and cassia.	QA	We search external knowledge source to find the substitution and answer the question.
6	User: I got ginger, let’s start. Bot: No problem! Let’s start cooking! There are 8 steps in total. Let’s start with Step 1: Preheat oven to 180 degrees celsius. A small tip is that preheating the oven in advance will help ensure more even cooking.	TOD	We use GPT-3 to generate a cooking tip for this step.
7	User: what is 180 degrees celsius in Fahrenheit? Bot: 180 degrees Celsius is 356 degrees Fahrenheit.	QA	This is an open-domain question and is answered by calling <code>factoid_qa</code> API.
8	User: ok, what’s next? Bot: Step 2: prepare the fruit by peeling and chopping into even pieces. Did you know that apple crumble is a dessert that originated in Britain during World War II for food rationing?	TOD	We use GPT-3 to generate a fun fact about this recipe.
9	User: I didn’t know that before. By the way, what is the oven temperature? Bot: 180 degrees celsius.	QA	The answer is generated by CoQA model, taken the summarized dialogue context and user utterance as input.
10	User: go to step bot . Bot: Ok, let’s move to step 4: meanwhile, Rob flour salt and cold butter together to form a breadcrumb. There are 4 steps left. Keep it up!	TOD	The ASR error is corrected by matching the transcribed text with common command phrases.
11	User: how many bot butter do i need? Bot: You need 125 g cold butter.	QA	The ASR error is detected and corrected by matching mentioned entities in context.
12	User: alexa go to the last step. Bot: Step 8: Bake for 30 minutes until golden brown. This is the last step. Good job! If you have finished, just say stop.	TOD	
13	User: I have cooked 30 minutes and it is still not brown. Bot: you can bake it for 5-10 minutes longer. Remember to check regularly to make sure it does not overcook.	Chit-chat	The intent of this turn isn’t detected and thus handled by chit-chat pipeline.
14	User: thank you. i have finished. Bot: Well done! i’m so happy we finished it together. Come back and visit us next time! Bye!	TOD	End of conversation

5.1 Question Answering

Users may have a wide variety of questions throughout the task completion process. Therefore, question answering is a crucial element in better assisting the users with their tasks. User questions can be mainly categorized into the following four types. We illustrate an example conversation containing each type of question in the **cyan** part in Table 3.

QA about the task To handle these kinds of questions, we need to first detect what information the user is requesting. For example, in Table 3 the user asks for the serving of the selected recipe in turn #4, and the quantity of needed ingredients in turn #10. Then, we can find the answers in the structured document associated with the recipe/task.

QA with conversation context This kind of question answering is also referred to as Conversation QA (CoQA). In CoQA, the conversational assistant has to understand the conversation context and find the answer within the conversation. For example, in turn #9, the user asked about the oven temperature, which was mentioned in the previous turn #6. To handle these kinds of questions, we first train a model to generate question-answer pairs based on wikihow articles. Then, the generated question-answer pairs are automatically filtered, revised, and labeled by hand to ensure quality. These high-quality question-answer pairs are used to train the unified T5 model.

QA with domain knowledge This kind of questions is also about the instruction but requires domain knowledge to answering it. For example, in turn #13, the user met problems when following the instruction to cook. Recently, it is proven that large-scale pretrained language models trained on large corpora encapsulate a large volume of external knowledge and can be used as a knowledge base [19]. We thus utilize GPT-J with 6B parameters [23], to answer the questions. To prompt the model to generate a better answer without being misled by the unrelated information, we utilize a dialogue summarization method to compress the dialogue history by only keeping the task-related information.

Open domain QA A `factoid_qa_function` is provided in the cobot codebase to answer the open-domain questions. As a backup, we also use the Bing search engine to search the answers. We use a suite of rule-based filter and neural model to calculate the relevance between the candidate answers with the question. The one with highest relevance score will be returned as the final answer.

5.2 Chit-chat

A user-friendly dialogue agent should be able to handle both task-oriented dialogue and chit-chat and provide a smooth transition between these two types of conversation. We categorize chit-chat into the following two types and use different methods to generate their responses.

Pre-defined topics As shown in Figure 3, we pre-define some chit-chat topics that are talked about. As the main focus of the task-oriented dialogue system is to assist the users in completing the task, we try to avoid the response that will lead to long follow-ups. Instead, we will try to lead the conversation back to the task-related content. To achieve that, when the user talks about these topics, we will respond with the curated templates and recommend a task that is related to this topic.

Others Any other topic or intents that our system couldn't detect will fall into this category. To respond to this kind of user request, we utilize two response generation models: (1) **BlenderBot** [21], and (2) **GPT-J** [23]. BlenderBot is specifically designed for chit-chat and is thus suitable to deal with the case when the user wants to have a social chit-chat with our bot but the topic cannot be detected. GPT-J is a large pre-trained language model that is able to handle any task that involves understanding or generating natural language or even code. Therefore, when the user's intent can not be detected and they are not trying to have chit-chat, the GPT-J model can usually generate a decent response. To select the best option from the response candidates, we apply a rule-based filter and a **DialogRPT** [6] for ranking. The responses that are more likely to get upvotes, more relevant to the dialogue context, and will not lead to a long follow-up, are preferred.

6 Automatic Speech Recognition Error Correction

To detect and correct the potential ASR errors, we propose several approaches based on phonetic similarity. The key idea is to find an existing entity that sounds like the transcribed text. Specially, we calculate the phonetic similarity between the transcribed text with existing entities using pyphonetics [10]. If the similarity with an existing entity is higher than the threshold, we will correct the transcribed text into that entity. There are several different sources to construct the candidate entities.

- **Mentioned entity**: we found that some ASR errors happen when the user asks questions about the entities that have been mentioned in the previous conversation. An example can be seen in turn #9 of the conversation in Table 3. By comparing the entity with existing entities in context, we can correct this kind of ASR error and respond correctly to the user's query.
- **Common command**: some ASR errors happen in the frequently used command phrases. An example can be seen in turn #8 of the example conversation, where the user's real intent is to "go to step four". By matching the transcribed text with some predefined commands, this type of ASR errors can be detected and corrected.z

- **Task query:** when the user queries some uncommon tasks, it is hard for the ASR model to correctly recognize them. For example, a user’s query “*how to join timber beams*” might be wrongly transcribed as “*how to join timber beans*”. To handle this, we store a local database of task names and match the ASR transcribed query with these local task names.

7 User Engagement

It is very important to keep users engaged during the task completion process instead of just reading the recipe or task instructions. To achieve this, we mainly made efforts from two aspects: (1) finding the tasks that users want, and (2) enriching the tasks to make them more interesting.

7.1 Finding the Tasks

Recipe Recommendation Sometimes the users don’t know what to cook or which recipe to choose when interacting with our taskbot. In this case, we will recommend some recipes to the users by asking their preference on the attributes of recipes, and also provide the requested recipe attributes to help users make a decision. The setting is similar to MultiWOZ [3], which is a task-oriented dialogue dataset designed for multi-domain tasks like booking restaurants, hotels and so on. The ontology that defines all attributes called slots and possible slot values is given by the search recipe API. In general, the slots can be divided into *informable* and *requestable* slots. *Informable* slots are attributes that we will ask the users to constrain the search, such as *cuisine*, *dietary*, and so on. *requestable* slots are the attributes of recipes that we can provide to the users such as *serving*, *cooking time*, *nutrition* and so on. Some informable slots and requestable slots are overlapped.

7.2 Enriching the Tasks

It is quite boring to simply read the instruction to the users. We thus try to enrich the task with interesting, knowledgeable and engaging content. For each queried DIY task or recipe, we retrieve interesting and related descriptions, aiming to attract the user in the beginning. Besides, we utilize GPT-3 [2] to generate the following three types of content to further enrich the task instruction:

- **Fun fact:** The fun fact is usually about the task/recipe or an entity in the instruction.
- **Story:** in addition to the fun fact, we also utilize GPT-3 to generate a short story for a recipe/task. It can be a story of the origin of a recipe like an apple crumble, a fruit like an apple, or a production like glass.
- **Tip:** the instructions for DIY tasks or recipes are usually not very detailed, and the users may come across many problems when doing the tasks. To better help the users to finish the task, we use GPT-3 to generate some tips for each step of instruction. We summarize some frequently asked questions as a prompt and use GPT-3 to generate tips that can answer this kind of questions.

As we only use the task or instruction to prompt the GPT-3 for generation, no user information will be passed to the GPT-3 API. An example of the generated engaging content is shown in Table 3.

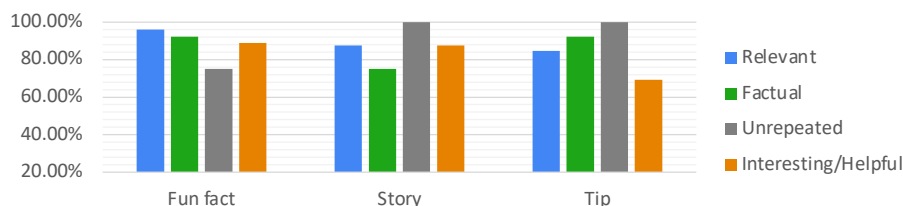


Figure 5: Human evaluation results of the generated engaging content.

Human evaluations We conduct human evaluations on generated engaging content. To simulate the final events, we randomly select 10 recipes and 10 diy tasks from the given task list. We manually check whether the percentage of generated content that is (1) **relevant** to the task instruction; (2) **factual**; (3)**unrepeated** in a task; and (4)**interesting** as fun facts/stories or **helpful** as tips. The results

are shown in Table 5. As there is few entities mentioned in some tasks and the generated facts are about the same entity, which may sounds repeated. Some stories are not factual which is reasonable as long as they are interesting. As for the generated tips, some instructions are not very informative. Therefore it is hard for not only the model but also human experts to give a helpful tip. Overall, the generation performance is satisfying but it is important to determine which kind of engaging content should be generated for an instruction.

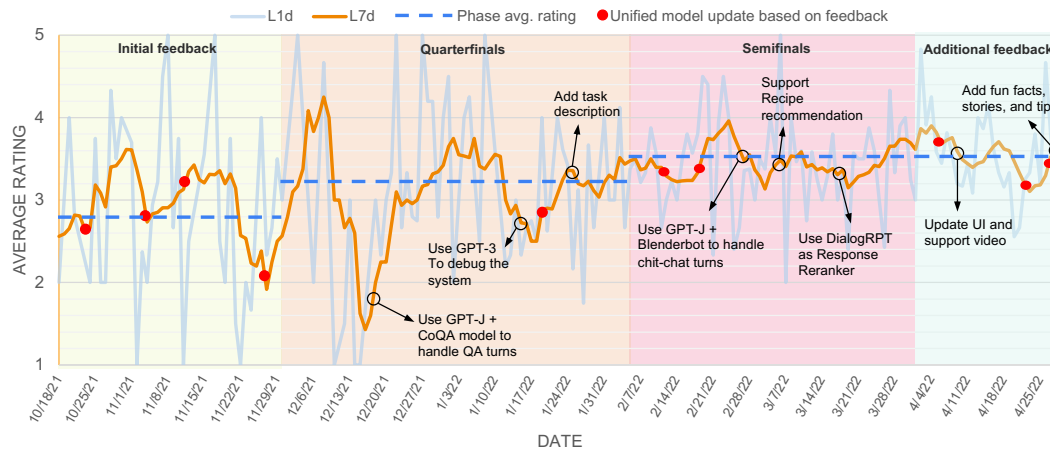


Figure 6: The average feedback rating of Gauchobot throughout the competition. L1d denotes the average daily rating, while L7d denotes the average of ratings received in the last 7 days.

8 Analysis and Discussion

In Figure 6, we illustrate the user feedback ratings and also the major changes we made throughout the competition. Although the daily rating L1d seems unsteady, the overall trend is positive. With our constant improvement, the average rating of each phase is higher than the previous one. As mentioned before, it is easy to update the unified model based on the current user feedback, and after each time of update, we can observe a rise in rating. By adding the QA pipeline, the weekly average rating increased from 1.8 to 3.7 in half a month, which proves the importance of QA in conversation towards completing a task. Likewise, after adding the Chit-chat pipeline and response reranker, the average rating increased to 3.8 by the end of the semifinals.

9 Conclusion

In this paper, we discussed the challenges when building a taskbot from scratch. To alleviate the cold-start problem, we utilize data generation and augmentation techniques to produce a diverse set of training data, which is used to train a unified model to handle various sub-tasks. In this way, we build a dialogue system from just a few configuration files, which are easy to maintain and update. We explore different approaches to improve the model robustness under ASR noise and unexpected situations. We found that with an intent classification model that is accurate enough in detecting in-domain intents, and a response generation model to handle the other out-of-domain situations, the dialogue system is robust enough to handle any real-world conversations. To improve user engagement, we utilize GPT-3 to generate interesting, helpful, and related content about the task. With successful and failed trials, the current system takes roughly 2 Ph.D. students 10 months to build. We are now in the process of further shortening the development time of similar taskbots.

Acknowledgments

Thanks to Professor William Wang for all of his comments and suggestions. We thank Amazon.com, Inc. and Alexa Prize organizers for their technical and financial support of the team. We would also like to thank Tom Zu, Carina Quan, Qiru Hu, and other colleagues for their help, feedback and advice.

References

- [1] Tom Bocklisch, Joey Faulkner, Nick Pawlowski, and Alan Nichol. Rasa: Open source language understanding and dialogue management. [arXiv preprint arXiv:1712.05181](#), 2017.
- [2] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.
- [3] Paweł Budzianowski, Tsung-Hsien Wen, Bo-Hsiang Tseng, Inigo Casanueva, Stefan Ultes, Osman Ramadan, and Milica Gašić. Multiwoz—a large-scale multi-domain wizard-of-oz dataset for task-oriented dialogue modelling. [arXiv preprint arXiv:1810.00278](#), 2018.
- [4] Qian Chen, Zhu Zhuo, and Wen Wang. BERT for joint intent classification and slot filling. *CoRR*, abs/1902.10909, 2019.
- [5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. [arXiv preprint arXiv:1810.04805](#), 2018.
- [6] Xiang Gao, Yizhe Zhang, Michel Galley, Chris Brockett, and Bill Dolan. Dialogue response ranking training with large-scale human feedback data. In *EMNLP*, 2020.
- [7] Michael Heck, Carel van Niekerk, Nurul Lubis, Christian Geishauer, Hsien-Chin Lin, Marco Moresi, and Milica Gasic. Trippy: A triple copy strategy for value independent neural dialog state tracking. In Olivier Pietquin, Smaranda Muresan, Vivian Chen, Casey Kennington, David Vandyke, Nina Dethlefs, Koji Inoue, Erik Ekstedt, and Stefan Ultes, editors, *Proceedings of the 21th Annual Meeting of the Special Interest Group on Discourse and Dialogue, SIGdial 2020, 1st virtual meeting, July 1-3, 2020*, pages 35–44. Association for Computational Linguistics, 2020.
- [8] Ehsan Hosseini-Asl, Bryan McCann, Chien-Sheng Wu, Semih Yavuz, and Richard Socher. A simple language model for task-oriented dialogue. In Hugo Larochelle, Marc’Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.
- [9] John F Kelley. An iterative design methodology for user-friendly natural language office information applications. *ACM Transactions on Information Systems (TOIS)*, 2(1):26–41, 1984.
- [10] Ilias Koutsakis. Python library for phonetic algorithms. <https://github.com/Lilykos/pyphonetics>, 2020.
- [11] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466, 2019.
- [12] Shiyang Li, Semih Yavuz, Kazuma Hashimoto, Jia Li, Tong Niu, Nazneen Rajani, Xifeng Yan, Yingbo Zhou, and Caiming Xiong. Coco: Controllable counterfactuals for evaluating dialogue state trackers. In *International Conference on Learning Representations*, 2020.
- [13] Bing Liu and Ian Lane. End-to-end learning of task-oriented dialogs. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Student Research Workshop*, pages 67–73, New Orleans, Louisiana, USA, June 2018. Association for Computational Linguistics.
- [14] Luis Enrico Lopez, Diane Kathryn Cruz, Jan Christian Blaise Cruz, and Charibeth Cheng. Transformer-based end-to-end question generation. [arXiv preprint arXiv:2005.01107](#), 4, 2020.
- [15] Edward Ma. Nlp augmentation. <https://github.com/makcedward/nlpaug>, 2019.

- [16] Baolin Peng, Xiujun Li, Jianfeng Gao, Jingjing Liu, and Kam-Fai Wong. Deep dyna-q: Integrating planning for task-completion dialogue policy learning. In Iryna Gurevych and Yusuke Miyao, editors, Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers, pages 2182–2192. Association for Computational Linguistics, 2018.
- [17] Baolin Peng, Xiujun Li, Lihong Li, Jianfeng Gao, Asli Celikyilmaz, Sungjin Lee, and Kam-Fai Wong. Composite task-completion dialogue policy learning via hierarchical deep reinforcement learning. In Martha Palmer, Rebecca Hwa, and Sebastian Riedel, editors, Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017, pages 2231–2240. Association for Computational Linguistics, 2017.
- [18] Baolin Peng, Chenguang Zhu, Chunyuan Li, Xiujun Li, Jinchao Li, Michael Zeng, and Jianfeng Gao. Few-shot natural language generation for task-oriented dialog. In Trevor Cohn, Yulan He, and Yang Liu, editors, Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020, volume EMNLP 2020 of Findings of ACL, pages 172–182. Association for Computational Linguistics, 2020.
- [19] Fabio Petroni, Tim Rocktäschel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, Alexander H Miller, and Sebastian Riedel. Language models as knowledge bases? arXiv preprint arXiv:1909.01066, 2019.
- [20] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. arXiv preprint arXiv:1910.10683, 2019.
- [21] Stephen Roller, Emily Dinan, Naman Goyal, Da Ju, Mary Williamson, Yinhan Liu, Jing Xu, Myle Ott, Kurt Shuster, Eric M Smith, et al. Recipes for building an open-domain chatbot. arXiv preprint arXiv:2004.13637, 2020.
- [22] Ryuichi Takanobu, Runze Liang, and Minlie Huang. Multi-agent task-oriented dialog policy learning with role-aware reward decomposition. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault, editors, Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020, pages 625–638. Association for Computational Linguistics, 2020.
- [23] Ben Wang and Aran Komatsuzaki. GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model. <https://github.com/kingoflolz/mesh-transformer-jax>, May 2021.
- [24] Tsung-Hsien Wen, Milica Gasic, Nikola Mrksic, Pei-hao Su, David Vandyke, and Steve J. Young. Semantically conditioned lstm-based natural language generation for spoken dialogue systems. In Lluís Màrquez, Chris Callison-Burch, Jian Su, Daniele Pighin, and Yuval Marton, editors, Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015, pages 1711–1721. The Association for Computational Linguistics, 2015.
- [25] Chien-Sheng Wu, Andrea Madotto, Ehsan Hosseini-Asl, Caiming Xiong, Richard Socher, and Pascale Fung. Transferable multi-domain state generator for task-oriented dialogue systems. In Anna Korhonen, David R. Traum, and Lluís Màrquez, editors, Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers, pages 808–819. Association for Computational Linguistics, 2019.
- [26] Kaisheng Yao, Baolin Peng, Geoffrey Zweig, Dong Yu, Xiaolong Li, and Feng Gao. Recurrent conditional random field for language understanding. In IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2014, Florence, Italy, May 4-9, 2014, pages 4077–4081. IEEE, 2014.
- [27] Kaisheng Yao, Geoffrey Zweig, Mei-Yuh Hwang, Yangyang Shi, and Dong Yu. Recurrent neural networks for language understanding. In Frédéric Bimbot, Christophe Cerisara, Cécile Fougerson, Guillaume Gravier, Lori Lamel, François Pellegrino, and Pascal Perrier, editors, INTERSPEECH 2013, 14th Annual Conference of the International Speech Communication Association, Lyon, France, August 25-29, 2013, pages 2524–2528. ISCA, 2013.

- [28] Jianguo Zhang, Kazuma Hashimoto, Chien-Sheng Wu, Yao Wang, Philip S. Yu, Richard Socher, and Caiming Xiong. Find or classify? dual strategy for slot-value predictions on multi-domain dialog state tracking. In Iryna Gurevych, Marianna Apidianaki, and Manaal Faruqui, editors, Proceedings of the Ninth Joint Conference on Lexical and Computational Semantics, *SEM@COLING 2020, Barcelona, Spain (Online), December 12-13, 2020, pages 154–167. Association for Computational Linguistics, 2020.