T-REX: Task-oriented REtrieval-focused eXtendable chatbot

Amit Hattimare[*] **Arkin Dharawat, Yelman Khan, Yen-Chieh Lien, Chris Samarinas, George Z. Wei, Yulin Yang, Hamed Zamani**[†]

Center for Intelligent Information Retrieval (CIIR)
Manning College of Information and Computer Sciences
University of Massachusetts Amherst
Amherst, MA 01003
`{ahattimare, adharawat, yelmankhan, ylien, csamarinas, gzwei, yulinyang, zamani}@cs.umass.edu`

Abstract

We present Maruna Bot, a Task-Oriented Dialogue System (TODS) that assists people in cooking or Do-It-Yourself (DIY) tasks using either a speech-only or multi-modal (speech and screen) interface. Building such a system is challenging, because it touches many research areas including language understanding, text generation, task planning, dialogue state tracking, question answering, multi-modal retrieval, instruction summarization, robustness, and result presentation, among others. Our bot lets users choose their desired tasks with flexible phrases, uses multi-stage intent classification, asks clarifying questions to improve retrieval, supports in-task and open-domain Question Answering throughout the conversation, effectively maintains the task status, performs query expansion and instruction re-ranking using both textual and visual signals.

1 Introduction

This work discusses the technical contributions of the Maruna team from the University of Massachusetts Amherst throughout the development of Maruna Bot for participation at the Amazon's Alexa Prize TaskBot Challenge in 2021/2022. During the challenge, a user would be connected to one of the ten university TaskBots at random by using a certain invocation phrase (e.g. *let's work together* or *assist me*). The goal of each bot is to help the user complete their desired task, limited to cooking and Do-It-Yourself (DIY) home improvement tasks. After the user has completed their interaction, they can leave a rating and verbal feedback.

To provide a helpful user experience, a task-oriented dialogue system needs to excel at language understanding, language generation, conversational planning, question answering, retrieval, summarization, result presentation among other traits. Specifically, we had to incorporate aspects from various disciplines such as Natural Language Processing (NLP), Dialogue Systems, Information Retrieval (IR), Computer Vision (CV), and Human-Computer Interaction (HCI). Considering the expertise of our team, Maruna Bot mainly focuses on the IR and NLP aspects of the challenge. Our main goals include:

Robustness: Our system should be able to handle all kinds of utterances. Given the free-form nature of customer utterances, it is important to gracefully recover from errors and be robust to unexpected inputs.

[*]Team Lead
[†]Faculty Advisor

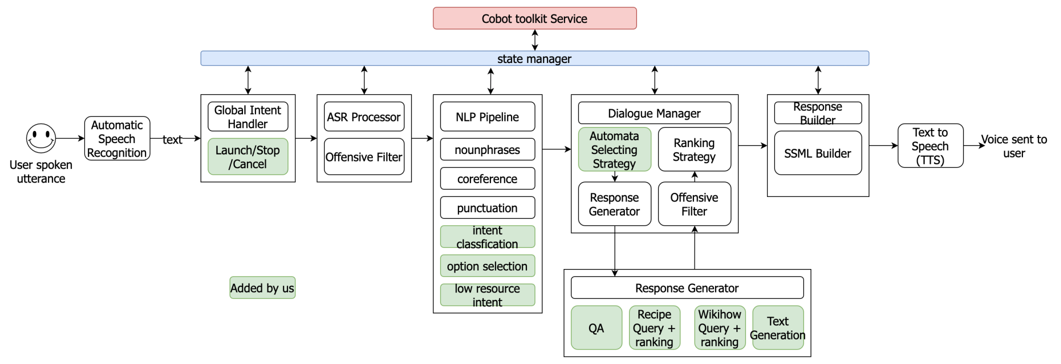


Figure 1: High level system architecture for Maruna bot.

Scalability: Each model deployed in our system should be parameter efficient, exhibit fast inference times, and have low memory requirements. Additionally, all the components should be horizontally scalable to handle all taskbot interactions.

Easily extendable to open-domain: Even though the focus of the challenge is on helping customers with cooking or DIY tasks, there is no reason to constrain research and development to just these domains. It is imperative that the system can generalize to the many others domains with minimal modifications.

Mixed-initiative interaction: Sometimes, customer's are unsure of how to interact with the bot or the utterances are unclear. Additionally, it is difficult and tiring for a customer to always drive the conversation forward. A taskbot ought to take initiative and preemptively ask clarifying questions in order to improve recommendations when appropriate.

Multi-modal understanding: For cooking and DIY tasks, the visual modality is a crucial information carrier for instruction presentation. We aim to design a lightweight multi-modal retrieval framework and exploit external visual data for our task.

Finally, it is paramount that the taskbot gets the job done. While interacting with a task-oriented dialogue system, the customer should be satisfied with the assistance given by the bot.

2 The Maruna Bot Overview

The system architecture of Maruna Bot is shown in Figure 1. It is built on top of the CoBot service framework [7]. The components in green are developed by the Maruna team and the rest are the ones provided by CoBot. The framework is divided into multiple levels to reduce coupling and parallelize component development. At every turn, the user utterance is converted to text using Amazon's automatic speech recognition (ASR) system and passed to our system. It is first handled by the *Global Intent Handler* which covers high-level actions like starting and stopping a conversation. Then offensive user content is identified and filtered using a filtering module. Filtered text goes to the *NLP pipeline* which consists of multiple NLP modules arranged in a directed acyclic graph (DAG) where the output of one module can be the input of other. We added three additional modules, namely, intent classification (§4.1), low-resource intent classification (§4.1) and option selection (§3.4), which run in parallel. The output of all the NLP modules is passed to the Dialogue Manager, which is responsible for creating an output for the user, both textual and visual. It consists of *Automata selecting strategy*, which is our Dialogue State Tracking (DST) component (§4.2) and decides which Response Generator should be used for getting the response. Maruna Bot consists of four Response Generators: QA (§6), Recipe Query with Re-ranking (§3.1), WikiHow Query with Re-ranking (§3.2), and Text Generation. At each turn, only one of the response generators is used depending on the output of the DST module. Each Response Generator also receives the necessary state variables to get more context about the ongoing conversation. The final created response then again goes through a filtering module to remove offensive responses before getting converted to speech using Amazon's Text to Speech system.

3 Retrieval Methods

Each participating team was provided two retrieval APIs: a *WholeFoods API* to retrieve recipes and a *wikiHow API* to retrieve DIY task instructions. Both these APIs use *ElasticSearch* as their backend and thus support only term-matching search. We developed multiple components to improve both precision and recall of the retrieval step in both settings.

3.1 Recipe Retrieval

We utilize the WholeFoods API to retrieve recipes based on the user query. Although the API did return relevant results in many cases (high recall), their ordering was not satisfactory. Therefore, we decided to add a re-ranking and filtering layer. To achieve this, we trained a doc2vec model on a publicly available dataset, Recipe1M+ [14]. Using this model we rank the retrieved recipes based on their inner product similarity with the given recipe query. Results with score below a minimum threshold are filtered out.

3.2 Retrieval for DIY Questions

For DIY tasks, we noticed that there are several instances where the API does not return any relevant results (low recall) or returns many non-relevant results in addition to the relevant result (low precision). We addressed these two issues by implementing a result filtering pipeline (see Figure 2).

Query Expansion. We implemented a pseudo-relevance feedback query expansion method to deal with the low recall of the wikiHow API. Using a publicly-available dataset of wikiHow tasks [8], we built a dense approximate nearest neighbor index of titles using a pre-trained BERT-base dual-encoder [22]. Query expansion is done by getting terms from the top k retrieved titles. Once we get a list of terms, we use the expansion terms and the initial query terms to retrieve results using the API.

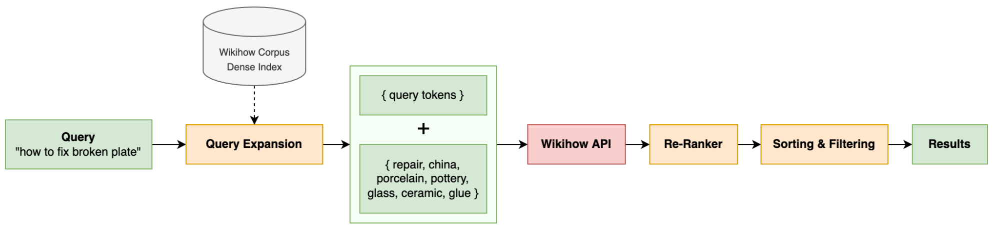

Wikihow Corpus Dense Index
Query "how to fix broken plate"
Query Expansion
{ query tokens }
+
{ repair, china, porcelain, pottery, glass, ceramic, glue }
Wikihow API
Re-Ranker
Sorting & Filtering
Results

Figure 2: wikiHow Query Expansion pipeline

Re-ranking. To improve precision-oriented metrics, we use a DistilBERT [24] binary classification model fine-tuned on the Quora Question Pair dataset [3]. We then use a pointwise re-ranker to sort and filter out the irrelevant results.

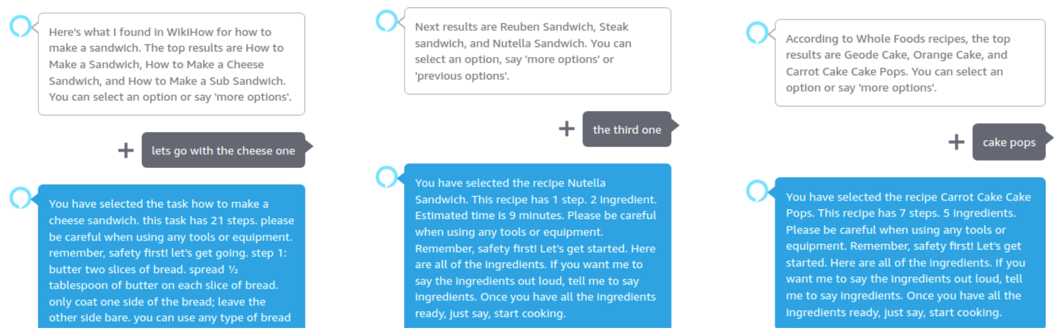
3.3 Multi-Modal Retrieval.

Our current retrieval framework is built on text-only data. To incorporate visual information into the system at a low cost, we consider image captions as the anchor between images and text retrieval models. A good caption could extract the important visual features from the image, which provides information for the retrieval models.

We aim to utilize generated captions as augmented data for text retrieval. Considering captions could be collected and generated offline, we can pre-process all images in the dataset and store them as additional textual data for the future use, instead of handling images online by image processing components. Besides, as textual data, the existing text retrieval system does not need reformulation to handle the augmentation.

Formally, consider a query q, a document d, and retrieval model M such that the score $s = M(q, d)$, if we have related image set $I^q = (i^q_1, i^q_2, ..., i^q_n)$ and $I^d = (i^d_1, i^d_2, ..., i^d_n)$, which are related to q and

[3] https://www.kaggle.com/c/quora-question-pairs


Here's what I found in WikiHow for how to make a sandwich. The top results are How to Make a Sandwich, How to Make a Cheese Sandwich, and How to Make a Sub Sandwich. You can select an option or say 'more options'.

lets go with the cheese one

You have selected the task how to make a cheese sandwich. this task has 21 steps. please be careful when using any tools or equipment. remember, safety first! let's get going. step 1: butter two slices of bread. spread ½ tablespoon of butter on each slice of bread. only coat one side of the bread; leave the other side bare. you can use any type of bread

Next results are Reuben Sandwich, Steak sandwich, and Nutella Sandwich. You can select an option, say 'more options' or 'previous options'.

the third one

You have selected the recipe Nutella Sandwich. This recipe has 1 step. 2 ingredient. Estimated time is 9 minutes. Please be careful when using any tools or equipment. Remember, safety first! Let's get started. Here are all of the ingredients. If you want me to say the ingredients out loud, tell me to say ingredients. Once you have all the ingredients ready, just say, start cooking.

According to Whole Foods recipes, the top results are Geode Cake, Orange Cake, and Carrot Cake Cake Pops. You can select an option or say 'more options'.

cake pops

You have selected the recipe Carrot Cake Cake Pops. This recipe has 7 steps. 5 ingredients. Please be careful when using any tools or equipment. Remember, safety first! Let's get started. Here are all of the ingredients. If you want me to say the ingredients out loud, tell me to say ingredients. Once you have all the ingredients ready, just say, start cooking.

(a) Selection using long phrase (b) Selection using position (c) Selection using short phrase

Figure 3: Option Selection Examples in Maruna Bot.

d respectively, we can directly utilize the following equation to include visual information without modifying M:

$$s = M(q + C(i_1^q) + C(i_2^q) + ... + C(i_n^q), d + C(i_1^d) + C(i_2^d) + ... + C(i_n^d)), \quad (1)$$

where $C(i)$ is the caption of the i^{th} image and the summation represent string concatenation.

We conducted an initial analysis on OK-VQA [15, 19], an open-domain visual question-answering benchmark, to validate if captions improve the retrieval performance. We trained a caption generator on the MSCOCO dataset [11], an image source of OK-VQA, and augmented the queries by pseudo-captions. As a result, Mean Reciprocal Rank improves from 13.13 to 14.84, and Recall@5 improves from 19.30 to 21.13. The improvement shows that generated captions can bring useful visual information into the text retrieval model.

3.4 Option Selection

The retrieval stage returns relevant results/options based on the user's initial query. Once the options are displayed or read out to the user, the next step is to allow the user to select the option they want. We define this task as *Option Selection*. We want the user to be free in choosing an option in a way they prefer. Therefore, we broadly categorized the option categories into *positions*, *numerical values*, and *phrases*. Position could either be first, second/middle, or third/last depending on the number of relevant results. Numerical values are similar but identified as number one, two, or three. For phrases, we use a 2-stage method for selecting an option. Figure 3 shows some examples. We use a semantic similarity approach between the user utterance and the options. This allows us to match acronyms (e.g. BBQ and barbecue) and synonym terms among others. For semantic textual similarity, we use a BERT-based pre-trained dual-encoder model [22]. If we do not find a good match at the first stage, we use TF-IDF at the bigram level. We determine the quality of the matches based on threshold values. These thresholds have been empirically selected based on the different test cases from user conversations. We also do some pre-processing before going through the aforementioned stages. We remove stopwords from the utterances in order to effectively select the correct option.

4 Dialogue Management

The dialogue manager consists of two main components: *Intent Classification* and *Dialogue State Tracking*. Together they help us to keep the conversation with the user coherent and make incremental progress towards task completion.

4.1 Intent Classification

A major part of any Task-Oriented dialogue system (TODS) or taskbot is Intent Classification. It needs to identify which requests should be supported and which should be politely refused with an appropriate reason. A correctly classified user intent makes it easy for subsequent models (like QA, clarification, and result presentation) to produce accurate responses, reduce back-tracking and customer frustration. Most commercial taskbots systems are designed to help with few specific tasks and therefore have their custom intents. Even for the TaskBot challenge, we have a specific goal of helping customers with DIY or cooking tasks. We, however, do not have any pre-defined intents or an existing dataset of example dialogues to train a model. It is completely open-ended and we are free to model our dialogues in any way.

4.1.1 BERT-based Intent Classification

Intent Classification Data We use the CLINC150 [9] dataset, which focuses on multi-domain TODS, to create our intent classification model. Other existing datasets do not focus on multi-domain TODS, consist of very few intents (for e.g., seven in Coucke et al. [4]), do not explicitly support out-of-scope queries, or do not support colloquial language. CLINC150 has 22,500 in-scope queries covering 150 intents, which can be grouped into 10 general domains. It also has 1,200 out-of-scope queries making the total query count 23,700. To support better cooking versus DIY task identification, we create 110,000 additional labeled examples each from Recipe1M+ [14] titles and wikiHow [8] titles using question templates.

Custom Label Identification The dataset contains queries from 151 intents including out-of-scope intent. They are, however, different from what we need for the taskbot. We looked into the supported intent set of Switchboard dialogue act corpus (SwDB) to understand what types of intents are seen in human conversations. Saha et al. [23] use 17 intents from SwDB to make their classifier, which helps us in narrowing down the set. We identified custom intents that can enable some simple workflows specific to the challenge and revised this set based on actual user conversations. We fixed 17 custom intents to support our bot and mapped all 150 intents of CLINC150 to these 17 intents.

Handling Class Imbalance The mapped data has high class imbalance due to uneven mapping. We used multiple approaches to deal with this including upsampling, downsampling, and balanced sampling with weighted loss function where a combination of both up and down sampling is used and final data has limited class imbalance.

Our Intent Classifier We used the modified CLINC150 dataset described above to fine-tune a pre-trained `DistilBert` model [24] with a classification head. We use this over the vanilla BERT model because it uses 40% less memory and is 60% faster while retaining 97% of its language understanding capabilities. We got best test accuracy using balanced sampling with weighted loss function. But even after getting a high test accuracy, the model did not cover many real-world use-cases well like *"how to make a puppet"* and *"help me in washing clothes"*. To improve our wikiHow task classification, we added 110k DIY task titles from wikiHow dataset and retrained on this model. Our test accuracy was again above 95 with improved DIY classification. To make the recipe query identification more robust and not classify queries like *"how to make fake nails"* as recipe, we created a two-stage classifier where the first stage handles *wikihow*, *recipe* and *other* classes. We have 110k+ examples for each of these intents. If the intent is *other*, we use the fine-grained classifier with 17 intents.

Fine-tuning with Crowd-sourced Data To further improve the performance on real examples, we collected over 500 conversation dialogues and labeled user utterances along with conversation history using *crowd-sourcing*. After removing trivial utterances, we were able to collect only 660 labeled examples, which was not sufficient for training purpose. The dataset was also heavily skewed towards some classes, for example half of the data has *out-of-scope* label and 179 examples have *recipe* label. This is due to the nature of the taskbot. Most of the conversations stayed in the task-selection stage, involved navigation and lacked variety. We eventually used the collected data for only test evaluation.

4.1.2 Intent Classification using Paraphrase Identification

As mentioned in §4.1.1, for many intents there is insufficient data. We identify such intents as *low-resource intents*. To tackle the problem of low-resource intent classification, our first approach utilizes a pre-trained paraphrase identification model [17]. Based on user conversations, we identified a few intents as *low-resource* and created a small set of ground truth examples for such intents. Then we used the paraphrase identification model to score each example-user utterance pair. The intent associated with the highest score is classified as the intent to be used. Based on various test cases, we determine a threshold value, and if our score threshold is not satisfied, we do not pick any intent.

This approach works well for the small set of ground truth examples but is limited in its usage. We discuss the limitations in the following section.

4.1.3 Low-Resource Intent Classification with Contrastive Training

The paraphrase identification method we described above has some major issues. Firstly, it requires very careful selection of ground truth examples that can cover all the possible paraphrases. The second and most important issue is the paraphrase identification approach **does not scale**, classifying a single user utterance requires classification of the pairs of the given utterance with all the ground truth examples. If the set of ground truth examples grows in size, that method becomes very inefficient. To solve these issues, we developed a dense encoder model for user utterances, trained with contrastive loss inspired by metric learning [27] and some recent work on few-shot intent classification [31, 32]. We used a BERT-based encoder and learned its parameters by minimizing the following loss:

$$\sum_{i}^{C}\sum_{j}^{k}\left(\sum_{x_p\in P_{x_{i,j}}} dist(x_{i,j},x_p) - \sum_{x_n\in N_{x_{i,j}}} dist(x_{i,j},x_n)\right) \tag{2}$$

For every training mini-batch, we sample a number of C classes, k examples per class, and a set of positives $P_{x_{i,j}}$ and negatives $N_{x_{i,j}}$. The positive examples are other utterances that belong to the same class, and negative examples are utterances that belong to different classes. $dist(a,b)$ is a distance metric between a, b, such as *cosine distance*. The general idea behind this approach is to train a dense encoder that learns close representations for utterances of the same class and distant representations for utterances that belong to different classes. For training we used the CLINC150 dataset, and in addition we pre-trained the encoder using Natural Language Inference data from SNLI [2], MultiNLI [28] and ANLI [16].

The learned representations of this encoder can be used directly for intent classification using a linear transformation and minimization of the cross-entropy loss, or using a k-Nearest-Neighbor (kNN) approach. By adopting the kNN approach, we are able to define a sufficiently large set of ground truth examples that cover all the user cases and perform classification efficiently using Approximate Nearest Neighbor (ANN) search. For ANN we use the FAISS library [6]. Apart from the scalability benefit, the representations are more robust and generalizable compared to the paraphrase identification model, because they are adapted for the intent classification domain.

4.2 Dialogue State Tracking

Dialogue State Tracking (DST) refers to estimating the dialogue states at each turn. It is usually modeled as a slot filling problem where we have pre-defined number of slots that need to be filled. It could be single- or multi-domain problem. It generally takes two forms–first, fixed vocabulary where we have a fixed number of candidate words for each slot, and second, open vocabulary where the slot values are unbounded for example area name, date, time, movie, etc. For e.g., the sentence *“I would like to have roasted chicken for dinner”* has values ‘roasted chicken’ and ‘dinner’ for slots *‘recipe_name’* and *‘meal_type’*, respectively. Datasets like MultiWOZ-2.0 and MultiWOZ-2.1 [30] are used to train and verify multi-domain DST models. For our TaskBot, we do not have any training data, slots identified, or possible values to fill in slots. Therefore, we cannot use any existing SOTA DST model.

To tackle this problem, we first manually identify *states* that are relevant for this task. Some states along with brief description is mentioned in Table 1. For each state, we identify intents that can create a transition to another state. We model this graph as a pushdown automata which also allows us to keep track of history in *stack*. This gives us some flexibility in handling the conversation flow, identify valid intents based on the conversation state, and support things like not allowing the user to change an in-progress task, question answering, asking clarifying questions, giving out detailed error responses to users, and suggesting possible things a user could do if they get stuck.

Table 1: Some states that we identified for the TaskBot along with their brief descriptions.

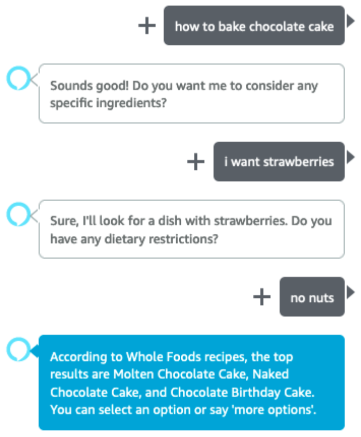
State	Description
launch request	The conversation starts from this state. It handles bot introduction and picking a task (recipe/DIY) from the user.
recipe query	Presents multiple recipes based on the query and helps select one.
show ingredients	Allows navigating through a recipe's ingredients and answer any questions on them.
show steps	Allows navigating through the steps of a task and answer any questions.

5 Clarifications

5.1 Clarifying Questions for Recipe Queries

Users often fail to define well their information need in a single query, and system initiated clarifying questions can be useful in many cases. The first type of clarifying questions we introduced in our bot was for recipes. Previous works have shown that even asking one good question can greatly boost retrieval performance [1]. Given the closed-domain nature of recipes, we have limited our questions to only ask if the user wants any specific ingredients or has any dietary restrictions.

We currently have a list of pre-defined templates to ask the user about ingredients and dietary restrictions since these two requirements are common across all recipes. This removes the need to generate questions via a language model. We use a BERT NER model fine-tuned on ingredient-extraction datasets such as FoodBase [18] and Food.com [13]. Our model is also fine-tuned to detect negations of ingredients, e.g., if the user says "no eggs" then we present recipes without any eggs. We use a low-resource paraphrase identification model to map to similar dietary needs, e.g: *"I want no meat"* gets mapped to the restriction *"vegetarian"*. We also try to detect these aspects when the user first makes the query. If the user says, *"I am looking for a vegan salad"* we avoid asking the user about their dietary needs. This way, we reduce friction by not asking for information that has already been provided. After we have extracted the relevant data, we query the WholeFoods API and then sort the result by relevance, before presenting them to the user.



how to bake chocolate cake

Sounds good! Do you want me to consider any specific ingredients?

i want strawberries

Sure, I'll look for a dish with strawberries. Do you have any dietary restrictions?

no nuts

According to Whole Foods recipes, the top results are Molten Chocolate Cake, Naked Chocolate Cake, and Chocolate Birthday Cake. You can select an option or say 'more options'.

Figure 4: clarifying question example

5.2 Open-Domain Clarifying Questions

Generation of open domain clarifying questions is challenging and cannot be effectively done using query templates and entity recognition models. DIY tasks fall into this category because they can be related to multiple different domains. This task first requires some understanding of the various query aspects and subtopics. For example, for the query *desserts* we would have facets such as *without sugar* or *low calorie*. For the query *buy watches*, some relevant facets would be *for men*, *cheap*, or *automatic*. For this purpose, we first focused on developing models for supervised open-domain **query facet extraction**. The extracted query facets can be used to help the user in result exploration and as input to a clarifying question generation model. We have experimented with three types of models on the MIMICS dataset [29].

Sequence Labeling. The first model is a sequence labeling model that classifies every token of a given document as part of a facet span or not. We adopted the BIO tagging format as seen in multiple NER tasks. We tokenize document d_{ij} and assign a label `B`, `I`, or `O` to each token. `B` if w_x is the beginning token for a facet, `I` if w_x is a facet token other than the beginning token and `O` otherwise.

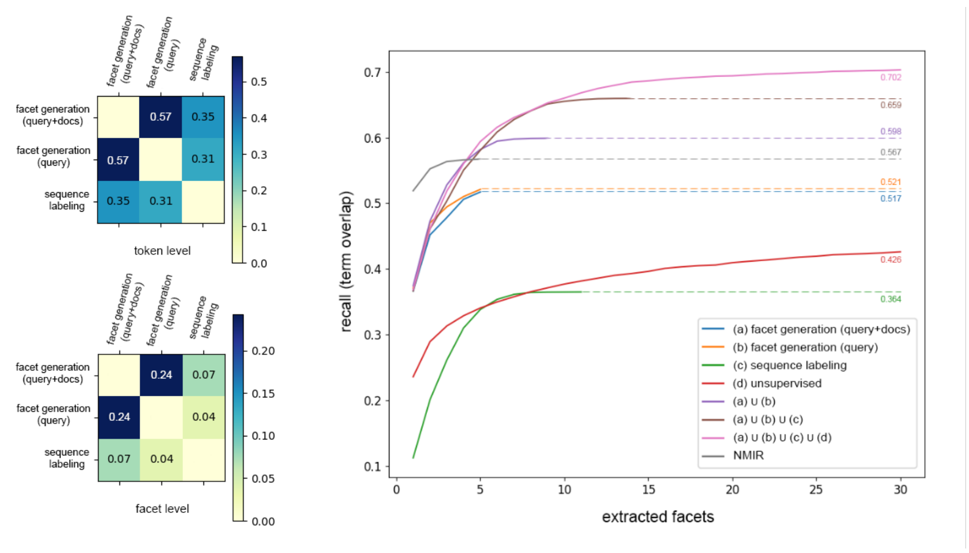


Figure 5: (Left) Average overlap coefficient between the outputs of the supervised models. (Right) Comparison of term overlap recall for the various extraction and generation models.

We adapt the parameters of a RoBERTa-base model using the following objective:

$$\theta^*_{\text{ext}} = \arg\min_{\theta_{\text{ext}}} \sum_{i=1}^{n} \sum_{j=1}^{k} \frac{1}{|d_{ij}|} \sum_{x=1}^{|d_{ij}|} -\log p(y_{ijx} | M_{\theta_{\text{ext}}}(q_i, d_{ij})_x) \tag{3}$$

where q_i is the given query, $d_{i,j}$ is a document and $M_{\theta_{\text{ext}}}$ are the learnable model parameters, which is a RoBERTa-base model with classification head.

Facet Generation. The second model we developed is an autoregressive facet generation model which tries to generate the list of relevant facets given a query and a list of documents. We used BART [10], a Transformer-based encoder-decoder model for text generation. We trained two variations of this generative model: (1) one variation only takes the query tokens and generates the facets, and (2) another variation that takes `[CLS] query tokens [SEP] doc 1 tokens [SEP] doc 2 tokens [SEP] ... doc r tokens [SEP]` as input and generates facet tokens one by one. Therefore, we use the following training objective:

$$\theta^*_{\text{gen}} = \arg\min_{\theta_{\text{gen}}} \sum_{i=1}^{n} \frac{1}{|y'_i|} \sum_{x=1}^{|y'_i|} -\log p(y'_{ix} | v, y'_{i1}, \cdots, y'_{ix-1}) \tag{4}$$

where v is the BART encoder's output.

Frequent ngrams. The last model we used for facet extraction is a simple yet effective unsupervised method that extracts frequent ngrams from the returned documents. We filtered ngrams that start or end with verbs, prepositions, determiners, symbols or pronouns using the average perceptron tagger of the NLTK library. We sort the remaining ngrams using the following scoring function:

$$s(q_i, f') = (1 + \alpha * \text{overlap}(q_i, f')) \times \text{freq}(f', D_i) \tag{5}$$

where f' is a facet candidate ngram for query q_i, the function overlap(q_i, f') returns the percentage of words from the query appearing in f', freq(f', D_i) computes the frequency of ngram f' in the top retrieved documents, and α is a hyperparameter.

Facet Aggregation We performed an analysis on the output of the supervised models using the average overlap coefficient on the token and facet level and we discovered that their outputs have low overlap (as seen in Figure 5, Left), thus combining their results can increase the recall of facets significantly.

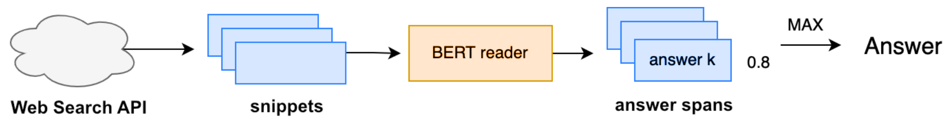


Figure 6: Our open-domain QA pipeline.

We have experimented with three different ranking and aggregation methods to combine the outputs of the models: point-wise and pair-wise ranking models based on BERT, Maximal Marginal Relevance diversification (MMR) [3] and Round-Robin Merging. Our experiments showed that the naive Round-Robin Merging method gives the best results in terms of recall. Figure 5 (right) shows the token level recall for our models and their Round-Robin aggregated lists. Our complete aggregated list improves the recall by a big margin compared to the previous SOTA model, NMIR [5].

6 Question Answering

Throughout the challenge, we noticed two particular trends that lead to questions being asked about the task at hand. First, customers are **forgetful** when interacting with conversational agents due to the lack of permanence on speech-only devices or performing the task with screen supported devices. Second, questions that are **broadly about** the task, stem from the lack of domain knowledge.

6.1 In-Task Question Answering

For a selected task, there are steps, which are split if they would cause too much cognitive load for the customer, as well as ingredients if the task selected is a recipe. The keys here are that these in-task questions are factoid and that the answers exist in the task text by assumption. This is the exact setup for span selection reading comprehension tasks, such as SQuAD [20, 21] or NewsQA [26]. More concretely, let s represent the step the customer is currently at where $0 \leq s < n$ and n represent the number of steps (with internal splitting) of the currently selected task and t_i represent the text at step i. Then for a given in-task question q that is answerable from the previously navigated text, we build the context $c_s = \{t_1; t_2; \ldots; t_s\}$. We use a pretrained RoBERTa model [12] fine-tuned on SQuAD [20, 21] to answer q with context c_s. If the confidence of the answer span is sufficiently high (i.e. $> 1e-5$), then we return it as the next TaskBot utterance.

6.2 Open-Domain Question Answering

Apart from in-task QA, we also implemented a pipeline for open-domain QA. Figure 6 summarizes the steps we follow for answering an open-domain question. For retrieval, we use the Bing API to retrieve the top 10 results for the given query. In the second step we download the web pages, extract the main body of text, and create a set of snippets. The set of extracted snippets is then passed to a BERT reader model, similar to the one used for in-task QA. The reader model identifies the most probable answer span for every query-snippet pair, and we then pick the answer span with the highest confidence. The confidence scores are normalized with softmax using the selected answer logits across the different snippets.

7 Lessons Learned

This project is vast in its scope with multiple research and engineering problems that cannot be completely solved by a small team in few months. Over the course of this challenge, we have learned many valuable lessons which we share below.

Team Management Lessons First, the TaskBot project involves multiple open-ended research problems and each of them could take a month or more of research and experimentation. A bigger team size is required so that many components could be worked on in parallel and students are not side-tracked due to engineering issues. Second, to efficiently use students' time, the work done towards this project would better be integrated as part of a course. Students should have well-defined roles and act as product owners for their assigned tasks. Third, the competition has multiple phases;

certification, quarter finals qualification, semifinals and then finals, and success in one does not necessarily mean success in others as the leader board is refreshed after each checkpoint. The team should continue identifying customer issues and work towards resolving them.

Engineering Lessons A major part of this project is dedicated to make research outputs available to the customer by deploying them in the TaskBot. Moving a model from research environment to production takes time which we underestimated. Multiple tests are required to be done in the Beta version before pushing to Production. After working on research, similar or more time is required to make it part of the pipeline. That should be accounted for in time estimation. Second, a team will greatly benefit if they have a member who works only on the infrastructure maintenance part, fixing engineering issues with deployed models, testing and deployments. Communicating with AWS support, adding and maintaining monitors, handle CPU/memory issues, etc. take time. Third, without a dedicated person to help in integrating vision research and making UI improvements, it is difficult to make multi-modal advancements reach customers which was expected in this challenge. Lastly, considerable development effort is required to make an initial version of the bot which supports a simple dialogue flow. It has minimal research requirements but without it we cannot support customers and cannot collect data points for improvements.

Research Lessons One big issue we faced during our research was that experiments that are done in isolated research environments often do not reflect the real world scenario when the bot is deployed. It takes more time to make a model 'customer-ready'. For some tasks, such as intent classification, there is not enough annotated data and we need models that can easily adapt in a low-resource setting. Related to this, creating a data collection pipeline is very time-consuming and the quality of data collected depends on many other moving parts of the bot. It is possible to collect some data for test purposes but extremely difficult to collect large-scale data for training purposes. Another big challenge is measuring the impact of new changes in our bot, which often depends on the amount of relevant traffic that we get. Last but not least, the ASR issues lead to ungrammatical or incorrect sentences that cause issues in Intent Classification, which in turn causes cascading effects on other parts of the bot. Robustness of models towards ASR issues is difficult but necessary.

8 Open Issues and Future Directions

One of the most important core components of our bot, is the intent classifier. Having an accurate and robust intent classification method is crucial for the flow of interactions with a user. Our BERT-based model for this task can be improved by using multi-task learning or using more diverse externally available data. In addition, the new efficient low-resource intent classification model can be compared with the current methods and be deployed for more efficient and robust classification. Recipe retrieval could be improved by using our doc2vec model as a first stage process. This would allow us to get semantic results without relying only on term matching. Since we do not have direct access to the Wholefoods corpus, we will need to build an index using an existing large-scale corpus of recipes or our own corpus crawled from the web. Regarding our multi-modal retrieval model, we can augment target datasets by transferring caption generation models trained on external datasets and improve our current re-ranker by including visual modality. For QA we can expand the ground truth answer spans to include noun phrases with either preprocessing or postprocessing. This should encourage the model to predict longer answer spans, which in SQuAD are generally shorter. We can generate more human-like clarifying questions by deploying our facet extraction model. We can use the facets generated along with a language model to generate these questions [25]. Further, we can use conversation history to predict user behavior and use that to resolve ambiguities (e.g., during ASR issues) or make better suggestions to the user. In our plans, we also aim to improve result presentation by using extractive and abstractive summarization techniques to shorten very long instruction descriptions.

We will release an open-source version of our contributions, named OpenMarunaBot, to foster research in this area.

References

[1] M. Aliannejadi, H. Zamani, F. Crestani, and W. B. Croft. Asking clarifying questions in open-domain information-seeking conversations. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, jul 2019. doi: 10.1145/3331184.3331265. URL https://doi.org/10.1145%2F3331184.3331265.

[2] S. R. Bowman, G. Angeli, C. Potts, and C. D. Manning. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 632–642, Lisbon, Portugal, Sept. 2015. Association for Computational Linguistics. doi: 10.18653/v1/D15-1075. URL https://aclanthology.org/D15-1075.

[3] J. Carbonell and J. Goldstein. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '98, page 335–336, New York, NY, USA, 1998. Association for Computing Machinery. ISBN 1581130155. doi: 10.1145/290941.291025. URL https://doi.org/10.1145/290941.291025.

[4] A. Coucke, A. Saade, A. Ball, T. Bluche, A. Caulier, D. Leroy, C. Doumouro, T. Gisselbrecht, F. Caltagirone, T. Lavril, M. Primet, and J. Dureau. Snips voice platform: an embedded spoken language understanding system for private-by-design voice interfaces, 2018.

[5] H. Hashemi, H. Zamani, and W. B. Croft. Learning multiple intent representations for search queries. *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, 2021.

[6] J. Johnson, M. Douze, and H. Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2019.

[7] C. Khatri, B. Hedayatnia, A. Venkatesh, J. Nunn, Y. Pan, Q. Liu, H. Song, A. Gottardi, S. Kwatra, S. Pancholi, et al. Advancing the state of the art in open domain dialog systems through the alexa prize. *arXiv preprint arXiv:1812.10757*, 2018.

[8] M. Koupaee and W. Y. Wang. Wikihow: A large scale text summarization dataset. *arXiv preprint arXiv:1810.09305*, 2018.

[9] S. Larson, A. Mahendran, J. J. Peper, C. Clarke, A. Lee, P. Hill, J. K. Kummerfeld, K. Leach, M. A. Laurenzano, L. Tang, and J. Mars. An evaluation dataset for intent classification and out-of-scope prediction, 2019.

[10] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *ACL*, 2020.

[11] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.

[12] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. RoBERTa: A robustly optimized BERT pretraining approach. *ArXiv*, abs/1907.11692, 2019.

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

[14] J. Marin, A. Biswas, F. Ofli, N. Hynes, A. Salvador, Y. Aytar, I. Weber, and A. Torralba. Recipe1m+: A dataset for learning cross-modal embeddings for cooking recipes and food images. *IEEE transactions on pattern analysis and machine intelligence*, 43(1):187–203, 2019.

[15] K. Marino, M. Rastegari, A. Farhadi, and R. Mottaghi. Ok-vqa: A visual question answering benchmark requiring external knowledge. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[16] Y. Nie, A. Williams, E. Dinan, M. Bansal, J. Weston, and D. Kiela. Adversarial NLI: A new benchmark for natural language understanding. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2020.

[17] A. Nighojkar and J. Licato. Improving paraphrase detection with the adversarial paraphrasing task. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 7106–7116, Online, Aug. 2021. Association for Computational Linguistics. URL https://aclanthology.org/2021.acl-long.552.

[18] G. Popovski, B. K. Seljak, and T. Eftimov. FoodBase corpus: a new resource of annotated food entities. *Database*, 2019, 11 2019. ISSN 1758-0463. doi: 10.1093/database/baz121. URL https://doi.org/10.1093/database/baz121. baz121.

[19] C. Qu, , H. Zamani, L. Yang, W. B. Croft, and E. Learned-Miller. Passage Retrieval for Outside-Knowledge Visual Question Answering. In *SIGIR*, 2021.

[20] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas, Nov. 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1264. URL https://aclanthology.org/D16-1264.

[21] P. Rajpurkar, R. Jia, and P. Liang. Know what you don't know: Unanswerable questions for SQuAD. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 784–789, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-2124. URL https://aclanthology.org/P18-2124.

[22] N. Reimers and I. Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019. URL http://arxiv.org/abs/1908.10084.

[23] T. Saha, D. Gupta, S. Saha, and P. Bhattacharyya. Emotion aided dialogue act classification for task-independent conversations in a multi-modal framework. *Cognitive Computation*, pages 1–13, 2020.

[24] V. Sanh, L. Debut, J. Chaumond, and T. Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, 2020.

[25] I. Sekulić, M. Aliannejadi, and F. Crestani. *Towards Facet-Driven Generation of Clarifying Questions for Conversational Search*, page 167–175. Association for Computing Machinery, New York, NY, USA, 2021. ISBN 9781450386111. URL https://doi.org/10.1145/3471158.3472257.

[26] A. Trischler, T. Wang, X. Yuan, J. Harris, A. Sordoni, P. Bachman, and K. Suleman. NewsQA: A machine comprehension dataset. In *Proceedings of the 2nd Workshop on Representation Learning for NLP*, pages 191–200, Vancouver, Canada, Aug. 2017. Association for Computational Linguistics. doi: 10.18653/v1/W17-2623. URL https://aclanthology.org/W17-2623.

[27] X. Wang, X. Han, W. Huang, D. Dong, and M. R. Scott. Multi-similarity loss with general pair weighting for deep metric learning. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5017–5025, 2019.

[28] A. Williams, N. Nangia, and S. Bowman. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122. Association for Computational Linguistics, 2018. URL http://aclweb.org/anthology/N18-1101.

[29] H. Zamani, G. Lueck, E. Chen, R. Quispe, F. Luu, and N. Craswell. Mimics: A large-scale data collection for search clarification. In *Proceedings of the 29th ACM International on Conference on Information and Knowledge Management*, CIKM '20, 2020.

[30] X. Zang, A. Rastogi, S. Sunkara, R. Gupta, J. Zhang, and J. Chen. Multiwoz 2.2: A dialogue dataset with additional annotation corrections and state tracking baselines. *arXiv preprint arXiv:2007.12720*, 2020.

[31] J. Zhang, K. Hashimoto, W. Liu, C.-S. Wu, Y. Wan, P. S. Yu, R. Socher, and C. Xiong. Discriminative nearest neighbor few-shot intent detection by transferring natural language inference. In *EMNLP*, 2020.

[32] J. Zhang, T. Bui, S. Yoon, X. Chen, Z. Liu, C. Xia, Q. H. Tran, W. Chang, and P. Yu. Few-shot intent detection via contrastive pre-training and fine-tuning. *ArXiv*, abs/2109.06349, 2021.