
Conditia: A State Machine Like Architecture for Multi-Modal Task Bots

Jerome Ramos *

University College London
London, UK
jerome.ramos.20@ucl.ac.uk

To Eun Kim *

University College London
London, UK
to.kim.17@ucl.ac.uk

Zhengxiang Shi

University College London
London, UK
zhengxiang.shi.19@ucl.ac.uk

Xiao Fu

University College London
London, UK
xiao.fu.20@ucl.ac.uk

Fanghua Ye

University College London
London, UK
fanghua.ye.19@ucl.ac.uk

Yue Feng †

University College London
London, UK
yue.feng.20@ucl.ac.uk

Aldo Lipani ‡

University College London
London, UK
aldo.lipani@ucl.ac.uk

Abstract

Conversational Artificial Intelligence (AI) has been a long-standing area of exploration in the research community and has now penetrated both academia and industries with products such as Siri and Alexa. In this work, we present **CO**oking-**aNd-DI**y-**TA**sks-based (Conditia) ChatBot, a task-oriented dialogue system, for the 2021 Alexa Prize TaskBot Challenge. Conditia provides an engaging multi-modal agent that assists users in cooking and home improvement tasks. Our main goal was to generate engaging dialogue and provide an effortless user experience in order to create a memorable, enjoyable experience for users. In this paper, we discuss Conditia's state machine like architecture and analyze the various conversational strategies implemented that allowed us to achieve excellent performance throughout the competition.

1 Introduction

In recent years, there has been growing interest in the field of conversational AI and, in particular, task-oriented conversational agents. Unlike open-ended dialogue systems, which deal with open-domain knowledge without any pre-defined tasks Huang et al. (2019), task-oriented dialogue systems are designed with specific domains and tasks in mind. Our **CO**oking-**aNd-DI**y-**TA**sks-based (Conditia) Bot, which means seasoned in Italian, is a multi-modal task-based dialogue system designed for the 2021 Alexa Prize TaskBot Competition to assist users with tasks related to cooking and Do-It-Yourself (DIY) tasks. Our goal in building this TaskBot was to generate engaging dialogue and provide easy to understand instructions in order to make a memorable user experience.

Conditia uses a state machine like architecture in order to keep track of the state of the conversation. The state machine like architecture lends itself particularly well to task based dialogue systems

* denotes first author

† denotes the team leader

‡ denotes the faculty advisor

because each step in a set of instructions can be mapped to its corresponding state in the state machine. After mapping a user utterance to a particular intent, the state machine transitions to the next state. After each state transition, the system will update the user with the next set of possible utterances.

Condito also takes advantage of the multi-modal experience of Alexa devices with screens. Using the Alexa Presentation Language (APL) API, our team heavily customized the user interface to improve aesthetics and usability. Additionally, many Alexa users communicated with Condito using a headless device without any screen. To account for the absence of visuals to convey extra information, we edited response prompts in headless mode to return more detailed text. Condito achieved admirable scores, particularly towards the end of the competition. In this work, we analyze the strengths of Condito as well as areas of improvement for future work.

2 System Architecture

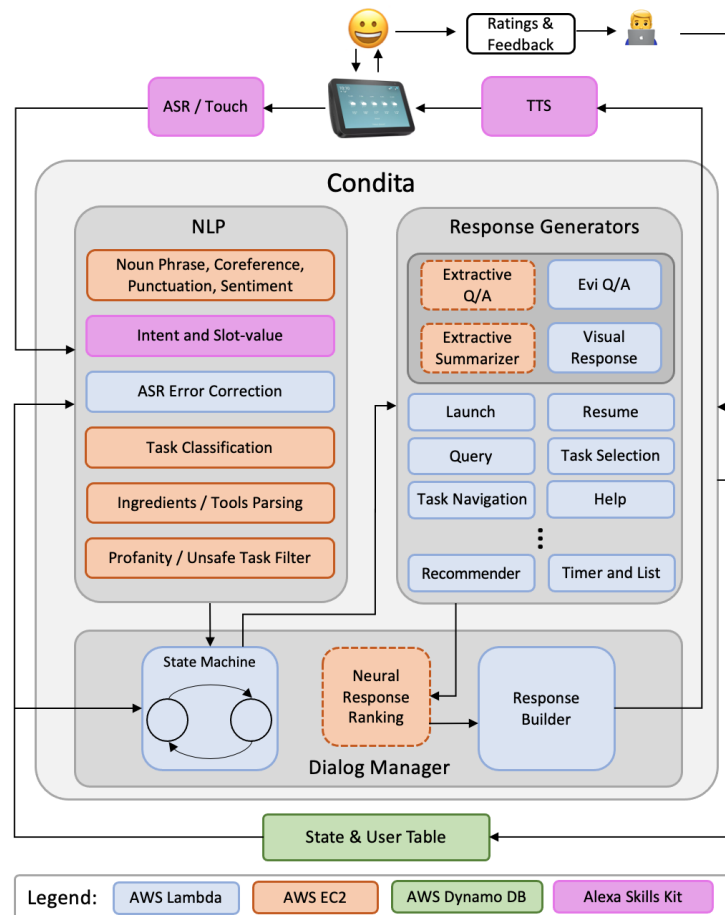


Figure 1: System Architecture of Condito. 1) Not all response generators are depicted in the Figure. 2) Extractive Q/A and Summarizer, Evi Q/A, and Visual response generators are modules that interact with each response generator. 3) Modules with dotted box are experimented within the system but not deployed in the production. 4) After every conversation, user can rate the session on a scale of 1 to 5 with textual feedback. User ratings and feedback pipeline is constructed for continuous development. GitHub Workflows streamlines the fetching of ratings from AlexaPrize S3 bucket, followed by a table joining with state table. The joined table is used not only for continuous development but also for creating response ranking data set.

Our system design is built on top of the CoBot framework Khatri et al. (2018), as can be seen in Figure 1. Each turn consists of the following steps:

1. The user either speaks a spoken utterance, which is then transcribed by Alexa's Speech Recognition (ASR) service, or uses a touch input.
2. The user's input is then sent to an AWS Lambda endpoint. Because Lambda is a stateless function, all information of the bot's state is held within AWS DynamoDB.
3. The input is ingested into the natural language processing and understanding pipeline (see Section 3) to produce NLP and task related annotations.
4. The state machine will then select several possible response generators to run based on the user's input and the current state of the bot (see Section 5.1)
5. The chosen response generators all run in parallel and return their generated output. Because running response generators can affect the information stored in the state model, any updates to state are first stored in temporary copies of the state model.
6. The neural response ranker then ranks all responses generated and selects the best response (see Section 5.2). The bot then updates any information changed including the state based on the selected response and discards all other temporary information. The state and user tables in AWS DynamoDB are updated accordingly.
7. The response builder then post-processes the output text and sends the bot utterance to Alexa's Text To Speech (TTS) Service to say the bot's response to the user. Additionally, if the user has a screen device, the response builder will also return a visual response built on top of the Alexa Presentation Language (APL).

3 Natural Language Processing

3.1 Initial NLP Modules

The *Neural Coreference* module extracts the coreference clusters from the previous and current turn. It returns the pronouns referenced along with the noun it points to. As the name suggests, the *Nounphrase Extraction* module provides all the nounphrases detected from the current user utterance. Along with these modules, information about *punctuation* and *sentiment* from the users' utterance are stored in the state table. *Nounphrase Extraction* is heavily used when filling the slot values that were missed from the Alexa Skills Kit (see Section 3.2). Low sentiment level of the utterance is used for triggering a sensitive responder. These modules are also used when correcting ASR errors.

3.2 Intent Classification and Slot Filling

Alexa Skills Kit (ASK) is used to define necessary intents and slots for understanding users' utterance. Custom defined intents are: TaskRequestIntent (DIYTask, RecipeTask, Food, and Drink as slots) TimerManagementIntent, ListManagementIntent, CompleteIntent, UserNameIntent, DietaryPreferencesIntent, SkipIntent, RecommendationsIntent. Other than these, Amazon built-in intents are used, such as Cancel, Stop, Resume, Yes, No, Select, Next, Previous, Help, and Fallback intent.

Since ASK does not guarantee robust performance of intent detection and slot-filling, *Task Classification* layer and *Nounphrase Extraction* module are used to complement the performance of ASK.

3.3 Task Classification

Understanding the topic of users' question is important for a seamless conversation in task oriented conversation. Therefore, we introduce an extra *Task Classification* layer within the NLP pipeline.

For model training, we have collected questions related to Cooking⁴ and DIY⁵ from <https://stackexchange.com/questions>. 24,954 and 50,000 task-specific questions are collected for cooking and DIY, respectively. The corpus is vectorized based on the frequency of the tokens,

⁴<https://cooking.stackexchange.com/questions/>

⁵<https://diy.stackexchange.com/questions/>

then fed into the multinomial Naive Bayes text classifier. It showed 98% test accuracy, with high F1-scores (0.97 in cooking task, 0.99 in DIY task). After the deployment to the EC2 instance, authors have not seen any failing cases from this model.

3.4 Parsing Ingredients and Tools

In this section we first introduce the proposed ingredient parsing model, as shown in Figure 2 and then present the experimental results of our model.

Model ⁶ comprises of three major components: text encoder, attention-based recurrent module, and an output layer. The encoder learns to represent each token given an ingredient phrase and its corresponding predicted tag label from the NLTK toolkit Bird and Loper (2004). The recurrent module is trained to learn contextualized embeddings of these tokens. The output layer contains a linear projection of trainable parameters and a sigmoid function that is used to map each word embedding into a scalar score.

Encoder. Given an ingredient phrase, $\{w_i\}_{i=1}^s$, where w_i is the i -th word and s is the number of tokens in this sequence, we use the text encoder to encode each word in this sequence into a vector, $\{u_i\}_{i=1}^s$, where u_i is the d -dimension vector representation for i -th token. To provide additional signals for the model, we also classify words into their parts of speech and labeling them accordingly via part-of-speech tagging in the NLTK toolkit (*ibid.*). Each token has one tag label, which will be converted into a d -dimensional vector via a trainable look-up table. Then token vectors, $\{u_i\}_{i=1}^s$ will be updated by adding these tag vectors to their corresponding token vectors, as shown in the Figure 2.

Recurrent Module will leverage the recurrent module to learn contextualized embedding. This consists of N layers, where each layer contains one self-attention module and one feed-forward layer. Each feed-forward layer is further comprise of a linear transformation layer, a dropout layer Srivastava et al. (2014) and a normalization layer Ba et al. (2016).

Output Layer. After obtaining contextualized word embeddings, where words' syntactic and semantic representation are learned in a given ingredient phrase, the output layer is used to map each word embedding into a scalar score.

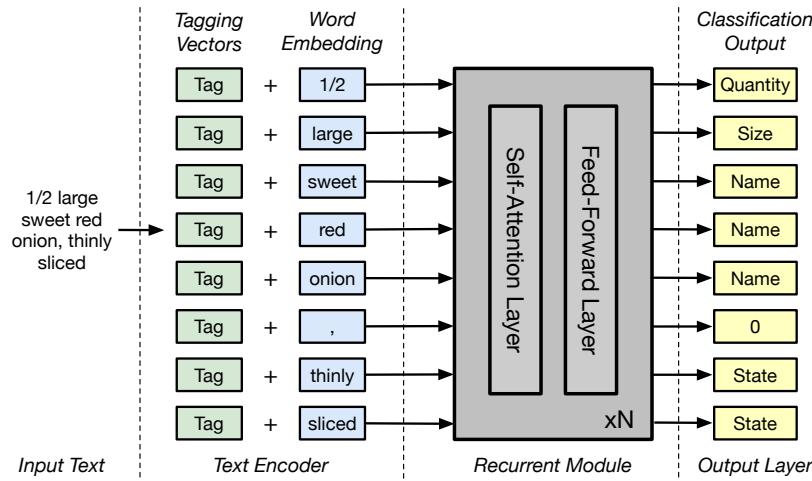


Figure 2: Ingredient Parsing model architecture. The model comprises three major components: a text encoder, an attention-based recurrent module, and an output layer. In the text encoder part, given one token from the input text, each green cell represents the encoded vector for its predicted NLTK tagging label and each blue cell stands for the encoded vector for this token. The output layer will predict one class for each token (yellow cell) after going through the recurrent module. N represents the number of layers in the recurrent module.

⁶<https://github.com/ZhengxiangShi/IngredientParser>

Materials. Two annotated datasets from Diwan et al. (2020), AllRecipes and Food.com, are used for experiments. In the datasets, each ingredient phrase includes ingredients used in the recipe and their corresponding attributes. Attributes are:

- **Name:** Name of the ingredient, e.g. salt or pepper;
- **State:** Processed method of the ingredient, e.g. ground or thawed;
- **Unit:** Measuring unit of the ingredient, e.g. gram, cup, or, tablespoon;
- **Quantity:** Quantity of the ingredient, e.g. 1 or 1/2;
- **Size:** e.g. small or large;
- **Temperature:** Temperature of the ingredient prior to cooking, e.g. hot or cold;
- **Dry/fresh:** Whether the ingredient is dry or fresh, e.g. freshly;
- **O:** Punctuation in the ingredient phrase, e.g. ", ".

Table 1: Experiment Results: Test Accuracy on either/both AllRecipes and/or FOOD.com dataset

Model	Testing	Training		
		AllRecipes	FOOD.com	Both
Ours	AllRecipes	96.98	94.38	97.30
Ours	FOOD.com	87.51	91.45	93.15
Ours	Both	89.62	92.28	93.64

Experimental Results. Table 1 presents the performance of our ingredient parsing model on the AllRecipes and Food.com dataset. The model performs better when trained on both datasets. It is notable that all trained models perform worse on the Food.com dataset than on the AllRecipes dataset, thus leading the test results on the combination dataset decrease compared to the results on the AllRecipes dataset. Overall, our model achieve over 90% accuracy rate over all datasets when both datasets are used for training.

3.5 ASR Error correction

After the utterances go through the previous NLP-steps, substantial amount of information is stored in a state table. This information can not only be utilized during response generation but also be referred to when attempting ASR correction. ASR correction is done in a rule-based continuous-development fashion. User Feedback pipeline (see Figure 1) can alert developers to look into conversational logs if low ratings are given by users continuously. Once ASR error is detected by developers from the logs, rule-based logic can be added to the ASR Error Correction module within the NLP pipeline. The logic reads to the previously stored NLP information from the state table and makes a correction. For example, if user is in a TASK_SELECTION State and said "I want option tree", it is highly likely that ASR processor made error in catching a word "three".

3.6 Avoiding dangerous and sensitive conversation

In accordance with the Alexa Prize guidelines, any conversation that is considered dangerous must be stopped immediately. Dangerous tasks are collected through Amazon Mechanical Turk⁷ crowd-sourcing. Paid workers were given both dangerous and safe tasks and told to decide if they would allow their adult children to do the task unattended or not. This allowed us to have a dataset with binary labels (safe and unsafe). Additionally, if the user asks a sensitive question, the bot must say that they cannot answer this question and offer to continue working on the current task. Examples of sensitive topics include legal, financial, and medical domains. To classify an utterance as dangerous or sensitive, we use regular expressions to check for banned words from a list of violation keywords provided by the Alexa Prize team. If the conversation is deemed safe and non-sensitive, the utterance is then passed along to the dialogue management module.

⁷<https://www.mturk.com/worker>

4 Response Generations

4.1 Stateful Responders

Stateful responders change the information stored in the state table. Refer to Section 5.1 for a description of each stateful responder.

4.2 Stateless Responders

Additionally, there are stateless responders that do not make any state changes when called. If the system selects one of these responders, the system remains in its current state for the next turn in the conversation.

- **Help Responder:** Sometimes, users are unsure what the system is expecting as input. In order to alleviate this confusion, the help responder returns example utterances that the user can say. At any point in the conversation, users can say, "help" to hear available commands.
- **Evi Open domain QA Responder:** While working on a task, users may have related questions. For example, a user might ask "How many teaspoons are in a tablespoon?" To answer these questions, we implemented the Evi QA API into Condito to answer these questions. Additionally, if Evi is unable to return an answer, it will return a fallback prompt saying that it is unable to respond to the user's utterance.
- **Dietary Preferences:** Users can set their dietary preferences as described in Section 4.3.2.
- **Sensitive Responder:** The sensitive responder is run for every utterance to check if an utterance is classified as sensitive. If the utterance is sensitive, the bot will inform the user that it cannot talk about that particular subject and asks the user if they would like to continue with the current task.
- **List Responder:** Users can create shopping lists on their Alexa app or add/remove items from existing shopping lists.
- **Timer Responder:** Timers can be useful when cooking or working on a DIY task. Users can ask Alexa to set up timers while they are completing a task.

4.3 Sub-modules for Response Generators

4.3.1 Extractive QA and Summarizer

Extractive QA was developed to handle questions that are asked during the steps of a task. Although we have Evi Question Answering API, it is not useful when the answer should be found within a specific passage. For example, Extractive QA module can be used when user asks how many/much a certain ingredient is needed for a recipe given a recipe information. RoBERTa-Base model Liu et al. (2019) pretrained on SQuAD dataset Rajpurkar et al. (2018) is used and deployed as one of the remote service on EC2.

Extractive Summarizer is used to prevent from providing users with lengthy task description. Lengthy task information are usually retrieved in DIY task. Once this description is outputted to the users without any summarization or truncation, users tended to give low ratings after the conversation. Therefore, within a response generator, we had extractive summarization model to first summarize the description before passing it to the response builder. Pretrained Distilled-BERT Sanh et al. (2019) model is used for faster computation.

4.3.2 Filter by Dietary Preference

To accommodate for users with dietary preferences, the dietary preferences filter was designed to allow users to exclude certain ingredients when searching for recipes. For example, users can say, "add peanuts to my dietary preferences" to filter out any recipes that list peanuts as an ingredient. This user-related information can be saved in the user table.

In order to advertise this feature to users, the dietary preference feature was added in the "Help" prompt. Whenever the bot fails to understand a user's request, the bot informs the user that they

can say "help" to hear additional options. If the user responds with "help", the system replies with instructions on how to use the dietary preferences feature.

4.3.3 Fun Facts

Many TaskBots can feel rigid to interact with because of the cut-and-dry responses. To make the bot feel more human-like and interesting to interact with, we manually scraped fun facts from cooking websites such as <https://facts.net/cooking-facts/>. All facts were manually scraped to prevent any inappropriate facts from being added. The collection of fun facts were stored in a key-value database with the food name as a key and fun fact as the value. When a user asks for a recipe, the fun fact database is queried for the user's requested recipe and appends the relevant fact to the systems output. If the user's recipe is not found in the fun fact database, a general fun fact about cooking is returned instead.

5 Dialogue Management

5.1 State Machine

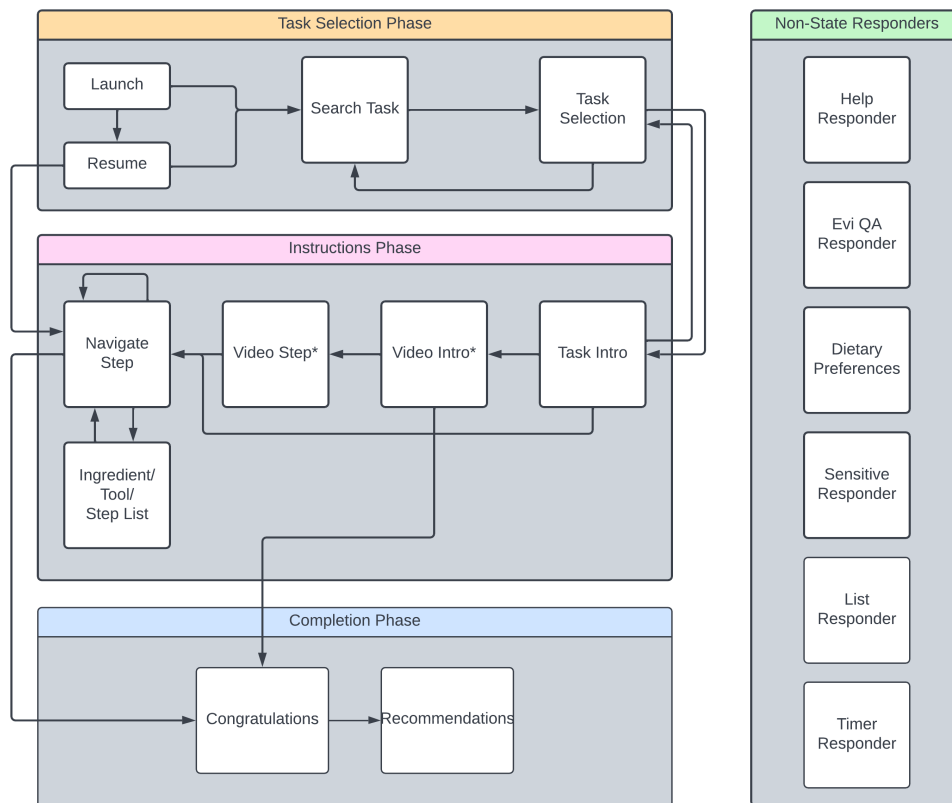


Figure 3: State machine architecture of Conditia. The user begins at the Launch State. Steps with an asterisk (*) can only be reached if the user is using a screen device.

Users can converse with an Alexa Prize TaskBot by saying, "assist me". They are then routed to one of several task-bots in the competition. Conditia users select a task to work on by walking through the Task Selection Phase. Users start at the *Launch State* and transition to the next states as described below:

- **Launch:** On start, the bot will check if whether the user is still working on a task from a previous conversation. If they have a task in progress, the bot will move to the Resume state. Otherwise, the bot will move to the Search Task State.

- **Resume:** The bot will ask users if they would like to continue working on their task. If the user responds yes, they will move to the Navigate Step state and will continue working on the step they were last working on. If the user responds no or asks for a new task, they will move to the Search Task state.
- **Search Task:** After the user requests for results for a certain task, the bot will call the WikiHow and Whole Foods Market APIs to search for and return relevant results.
- **Task Selection:** Users can select tasks by either saying the title of the task, uttering a phrase with an ordinal (e.g. the third one), or by touching the corresponding image if they have a screen device. Additionally, the can say, "show me more" to view additional tasks or "go back" to hear the previous tasks again.

After selecting a task, the user enters the Instructions Phase. The Instructions Phase consists of the following states:

- **Task Intro:** Before starting the task, there is an introductory state where the bot explains several key metrics of the task, such as user ratings, estimated time of completion, ingredients needed, etc. Users can also choose to go back to the task selection state if they wish to select another task.
- **Video Intro (screen device only):** Several WikiHow articles include videos on how to perform the task. The bot will ask the user if they would like to view the video if there is a video available. If the user responds yes, the bot will then begin playing the video. If the user says no, the state then moves to the Task Intro step. If there is no screen present, the bot will skip to the navigate steps state.
- **Video Step (screen device only):** Using the APL Video Template, Conditia plays the video. Users can control the video either through touch input or through voice. At the end of the video or at user request, the bot asks if they would like to continue to the step by step instructions or if they are finished with the task. The user will then be routed to the Task Intro state and Congratulations step, respectively.
- **Navigate Step:** In the Navigate Step state, the bot instructs the user about the current step of the task. Users can say "next" or "previous" to see the next or previous state respectively.
- **Ingredient List:** The Ingredient List state lists all of the ingredients in the recipe. If the user has a screen device, the system will return a list view of the ingredient list. Note that this step is only usable if the task is classified as a cooking recipe.
- **Tools List:** The Tools List state lists all of the tools in the task. For example, it may list the cooking utensils needed in a recipe or the equipment needed for a DIY task. If the user has a screen device, the system will return a list view of the tools list.
- **Step List:** The Step List state shows all of the steps in list view form if the user has a screen device. Otherwise, it reads out every step in the task.

Once a user finishes a task, they move on to the Completion Phase. The Completion Phase consists of the following two states:

- **Congratulations:** Once the user completes the final step, the user is congratulated for finishing the task.
- **Recommendations:** After the user is congratulated for finishing the task, the bot recommends two tasks that the user can do for next time. Due to the rules of Alexa Prize, the user cannot start a new task after task completion. Thus, the bot informs the user that they can start a new conversation with an Alexa Prize TaskBot and ask about this task.

Detected intents/slots, selected responders, and state changes are annotated with the conversations in Table 3.

5.2 Neural Response Selection

To achieve a better performance in response selection, ranking strategy has been developed based on a neural response ranker, BERT-FP Han et al. (2021). BERT-FP focuses on a short conversation

context and achieves the state-of-the-art results in many datasets. We have fine-tuned this model with two datasets and obtained reasonably good performance.

5.2.1 Dataset

Stack Exchange is a popular web-forum for general questions. Each question has a thread of answers and comments. Here we take the question and the its comments as the context and answers as the targets. Besides the targets, we randomly pick the other two unrelated documents as the wrong answers. This process was done both in cooking-related and DIY-related questions from *stackexchange.com*. 96,691 and 189,477 questions are collected for cooking and DIY, respectively. For each target, we generated five wrong responses. These 286,168 cases were then divided into train, test, and validation set by 3:1:1 ratio.

Real user data is also used to fine-tune the model as the TaskBot has been deployed. With 3,984 conversations between our bot and users, we generated 5,808 cases in this dataset. We used the same ratio when splitting the dataset for training.

5.2.2 Performance

Table2 shows that BERT-FP has a strong potential to handle the response ranking task in cooking and DIY domain on production.

Dataset	Stage	MAP	MRR	P@1	R@1	R@2	R@5
Stack Exchange data	test	0.9935	0.9935	0.9871	0.9017	1	1
Stack Exchange data	validate	0.9766	0.9908	0.9846	0.4193	0.7042	0.9816
Real user data	test	0.9767	0.9767	0.9534	0.9534	1	1
Real user data	validate	0.9496	0.9688	0.9483	0.6336	0.9267	0.9784

Table 2: Performance of BERT-FP model on Stack Exchange data and the real user data. Mean Average Precision, Mean Reciprocal Rank, Precision at one, and Recall are denoted as MAP, MRR, P@1, and R@k, respectively.

6 Multimodal Customer Experience

One of the main features of Condita is the multi-modal user experience for users with an Alexa screen device. Using the Alexa Presentation Language (APL), we developed an easy-to-use interface to enhance the user experience. By showing information visually, we reduce the verbosity of prompts. This resulted in higher user satisfaction and longer user engagement. Figure 4 shows a few examples of Condita’s visual responses.

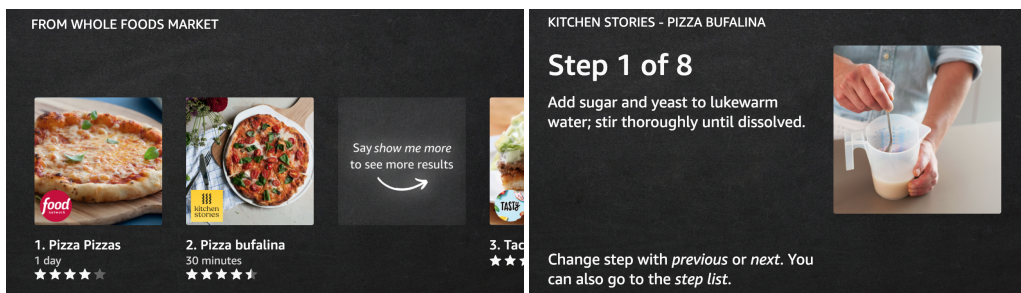


Figure 4: UI for the Task Selection (left) and Navigate Step (right) states.

Due to the absence of a screen in headless mode, information that would normally be shown visually must be communicated verbally. For example, a recipe’s rating and estimated time of completion can easily be seen on the screen device. However, users would not know this information if they are unable to see it on the screen. To alleviate this issue, we added additional information in the responses of headless prompts. For example, in the bot tells users the task’s rating and estimated time of completion during the task intro state in headless mode.

Utterance	Intent	Slot	Response Generator	Commentary
User: Alexa, open Conditia Conditia: Hi, this is an Alexa Prize TaskBot. I can help with cooking or home improvement.				
How can I help?		Invocation	Launch	LAUNCH -> SEARCH_TASK No previous ongoing task is found
User: Alexa, open Conditia Conditia: Last time we were working on chocolate cake.				
Would you like to continue?		Invocation	Resume	LAUNCH -> RESUME Previous ongoing task is found. Transition to SEARCH_TASK afterwards if yes.
User: How do I cook pasta ? Conditia: Here's what I found in Whole Foods Market for pasta. The first is X, second is Y, and the last is Z. The highest rated option is the last one and the recipe with the shortest completion time is the first one.				
Which option would you like?	TaskRequestIntent	RecipeTask	Query	SEARCH_TASK -> TASK_SELECTION Checks if the task is sensitive or not. Can show more results upon request. Interactive visual response.
User: How do I fix a leaky faucet in my bathtub? Conditia: Here's what I found in WikiHow for fixing a leaky faucet...	TaskRequestIntent	DIYTask	Query	SEARCH_TASK -> TASK_SELECTION
User: I do not want any options there. Conditia: Sorry about that. Any other task you want to ask?	CancelIntent		Cancel	TASK_SELECTION -> SEARCH_TASK Go back to SEARCH_TASK if a task is not chosen yet.
User: I want the third option. Conditia: Good choice! I'll tell you 8 ingredients you need. If you want to skip, say 'next'...	SelectIntent	ListPosition	Task_selection	TASK_SELECTION -> TASK_INTRO Introduction to tools and ingredients needed for the task.
Conditia: I found a summary video related to the task. Would you like to watch? User: Yes, I want to watch the video	YesIntent		Video	TASK_INTRO -> VIDEO_INTRO Only available to screen device users.
User: Pause the video Conditia: (Visual Response)			Video	VIDEO_INTRO -> VIDEO_STEP Pause, play, rewind, fast forward, and starting over are available. Transition to NAVIGATE state if user agrees to continue after the video is ended.
User: Continue to the steps. Conditia: Step 1 out of 7.				
Preheat the oven to 350 degrees Fahrenheit.	NextIntent		Show_steps	TASK_INTRO -> NAVIGATE Next, start, continue, I'm ready, etc. are available. Able to go back to the selection phase.
User: Go back to previous step. Conditia: Step 4 out of 7.				
Drain well and transfer to a large heatproof bowl.	PreviousIntent		Show_steps	NAVIGATE -> NAVIGATE Remind users with useful commands during the steps. E.g. Ingredient list, help responder.
User: Let's jump to step five . Conditia: Sure. Step 5 out of 7. Gently stir in tomatoes, vegetables, and squash purée.	SkipIntent	StepNumber	Show_steps	NAVIGATE -> NAVIGATE
User: What is squash purée? Conditia: According to a recipe from Epicurious, squash purée is a dish commonly prepared with calabaza or butternut squash, olive oil, fresh lime juice, and nutmeg.	FallbackIntent		QA_responder	NAVIGATE -> NAVIGATE Evi general QA responder and extractive QA is selected and run in parallel, followed by a reselection from neural reranker.
User: Show me the list of ingredients again? Conditia: For this recipe you need 8 ingredients. Please feel free to interrupt while I say all the ingredients to you. You need 1 pound of whole grain pasta...				
To continue your task, please say 'continue'	FallbackIntent		Ingredient_list	NAVIGATE -> INGREDIENTS_LIST Tools list can be shown if tools exist, especially for DIY tasks. Helpful visual aid can be provided depending on the device type. Can navigate through the list.
User: Help Conditia: Move the steps by 'next' and 'previous'. Also, skip to the specific step by 'skip to the third step' for example.	HelpIntent		help	Help responder can be activated at any states.
User: Add cheese into my shopping list Conditia: Sure, cheese is added to your shopping list	ListManagementIntent	item	list_management	Can add and remove items from the shopping list.
User: Set timer for 10 minutes Conditia: Timer is created for 10 minutes!	TimerManagementIntent	duration	timer_management	Can set, read, stop, resume, and cancel the timer.
User: Go next Conditia: Bravo! There are no more steps. If you would like some follow up recommendations, please say 'recommendations' Otherwise,				
let's complete this task!	NextIntent		congratulations	NAVIGATE -> CONGRATS If user navigates to the next step but if it was the last step, congratulate user for finishing the task and ask if she wants follow up recommendations.
User: I want some recommendations Conditia: Definitely! After all the work you've done, what do you think about fruit juice or apple pie? User: Let's go for a juice. Conditia: Nice choice! I'll be more than happy to assist you again.	RecommendationIntent		recommendations	CONGRATS -> RECOMMENDATIONS End the conversation for a new session with the recommended task.

Table 3: Conversations annotated with detected intent/slot, selected response generator, and state changes.

7 Conclusion

In this work, we have described the architecture design for Conditia, a state-machine based, task-oriented dialogue system that assists users with cooking or home improvement tasks. Our focus on improving the multi-modal customer interface and generating engaging dialogue helped create an easy-to-use, memorable user experience.

References

- Huang, M.; Zhu, X.; Gao, J. Challenges in Building Intelligent Open-domain Dialog Systems. 2019.
- Khatri, C. et al. Advancing the State of the Art in Open Domain Dialog Systems through the Alexa Prize. 2018.
- Bird, S.; Loper, E. NLTK: the natural language toolkit. 2004.
- Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research* **2014**, *15*, 1929–1958.
- Ba, J. L.; Kiros, J. R.; Hinton, G. E. Layer normalization. *arXiv preprint arXiv:1607.06450* **2016**,
- Diwan, N.; Batra, D.; Bagler, G. A named entity based approach to model recipes. 2020 IEEE 36th International Conference on Data Engineering Workshops (ICDEW). 2020; pp 88–93.
- Liu, Y.; Ott, M.; Goyal, N.; Du, J.; Joshi, M.; Chen, D.; Levy, O.; Lewis, M.; Zettlemoyer, L.; Stoyanov, V. RoBERTa: A Robustly Optimized BERT Pretraining Approach. 2019.
- Rajpurkar, P.; Jia, R.; Liang, P. Know What You Don't Know: Unanswerable Questions for SQuAD. 2018.
- Sanh, V.; Debut, L.; Chaumond, J.; Wolf, T. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. 2019.
- Han, J.; Hong, T.; Kim, B.; Ko, Y.; Seo, J. Fine-grained Post-training for Improving Retrieval-based Dialogue Systems. Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. Online, 2021; pp 1549–1558.