DiWBot: A Cooking and DIY Conversation Guidance System

Richard Magnotti[*]
Rutgers University
richard.magnotti@rutgers.edu

Denson George[*]
Rutgers University
denson.george@rutgers.edu

Zelong Li[†]
Rutgers University
zl359@cs.rutgers.edu

Jianchao Ji[†]
Rutgers University
jj635@cs.rutgers.edu

Hyungjung Joo
Rutgers University
hyunjung.joo@rutgers.edu

Jiaxing Yu
Rutgers University
jy729@linguistics.rutgers.edu

Ramitha Ravishankar
Rutgers University
rsr146@scarletmail.rutgers.edu

Lina Moe
Rutgers University
mkm213@scarletmail.rutgers.edu

Baber Khalid
Rutgers University
baber.khalid@rutgers.edu

Yongfeng Zhang
Rutgers University
yongfeng.zhang@rutgers.edu

Matthew Stone
Rutgers University
mdstone@cs.rutgers.edu

Abstract

As conversational agents become sophisticated, they are becoming more prevalent in the daily lives of people. They have the ability to help users accomplish daily tasks and help in their day-to-day lives. In this report we summarize our findings and development of our taskbot: *Do it With Bot (DiWBot)* built to help people in cooking and DIY tasks for the 2023 Alexa Prize TaskBot Challenge [1]. We present the engineering techniques employed to build our bot and an analysis of the conversations between our taskbot and its human users. This analysis reveals what conversation behaviors and taskbot features are preferred by the human users and which behaviors impact the ratings in a negative manner. We present this report to aid in the development of similar taskbots.

1 Introduction

Smart agents like Alexa are becoming more prevalent in the daily lives of people and help them in several daily tasks: from controlling their appliances to reading books. Currently, these agents

[*]Equal Contribution
[†]Equal Contribution

2nd Proceedings of Alexa Prize TaskBot (Alexa Prize 2023).

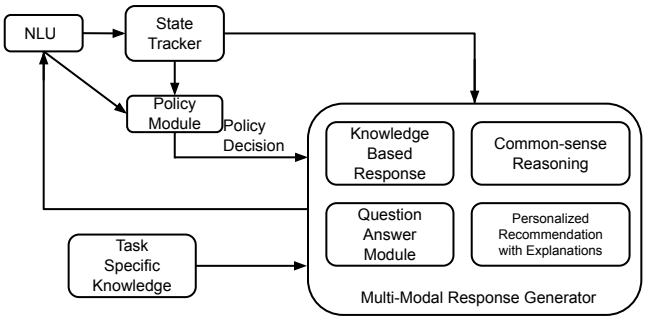

Figure 1: Architecture pipeline for our conversation system. The above diagram shows how different components of our dialogue system come together to make a complete system.

assume that human commands embody the entire context in their instructions and do not interact with their users to refine the criteria. However, these have the potential to help humans accomplish their daily tasks like cooking, cleaning and chores around the home by interacting with them actively. In this report, we present the design of an Alexa-based taskbot designed to guide users through cooking and DIY tasks. Our designed taskbot interacts with users through their Alexa devices and helps them choose a cooking or DIY task they want to accomplish. It then enables users to explore the steps required to finish the ongoing task.

There are several directions we could have taken to design a conversation agent for our approach. End-to-end neural models are a possible way forward but they are harder to get to conform to specific behaviors. In addition, latency needs to be minimized when interacting with humans. Due to that, an end-to-end neural model may end up being computationally expensive. Another direction is to design every dialogue module using manually designed heuristics. However, this approach will suffer from the requirements of huge amounts of manual effort to get any reasonable coverage. We chose a middle ground where several individual modules served specific purposes and utilized relatively light-weight neural solutions where appropriate along with manually designed heuristics.

An architecture diagram for our approach is shown in Figure 1. Our architecture involves several modules. When users speak an utterance it goes through a Natural Language Understanding (NLU) module which extracts the necessary information from it. A state tracker embeds the extracted information in a data structure which a policy module utilizes to make high-level dialogue decisions. These decisions are passed to the respective responders to generate an appropriate textual or multi-modal response. We explain the details of our architecture in detail in the following sections.

2 Taskbot Design

2.1 Extracting NLU Information

We found information the system extracts from user utterances strongly depends on which phase they are engaged in. During the search phase the user can opt for a criteria-based recommendation, ask to be surprised, and initiate or continue their explicit search for a task of interest. We instrumented our domain classifier to do the initial heavy lifting in this regard. The user searches for a task, and the system classifies their utterance as either pertaining to DIY, cooking, or in the case they initiate an unrelated sub-dialogue - general domain. While we initially used key-phrase matching for this purpose, we promptly substituted it for Amazon's domain classifier once it was made available.

The number of possible intents a user can express is substantial, even in well-defined task-oriented dialogue, e.g., requesting the next step in a task, initiating a sub-dialogue on a previous utterance, ending the conversation, etc. So, the responsibility of inferring a user's intent is distributed between carefully selected heuristics and a trained classifier. Heuristics include detecting key navigation phrases such as go home, last step, or resume task. Similarly, the system identifies key phrases with respect to task progression, e.g., different task, could you repeat that step, done with ingredients, show more options, etc. On the other hand, our model is BERT- base, trained on the Amazon Wizard of Task dataset to do multi-class classification. Given a user utterance, it outputs the expressed intent as *ask-question-ingredients-tools*, *chitchat*, *request-next-step*, *ask-question-recipe-steps*, *misc*, or *stop*.

Of particular note, during the search phase a DIY query must be handled differently than for cooking. The Amazon WikiHow API request structure is more relaxed in contrast to cooking. The recipes API requires a strict format where fields must be explicitly filled, e.g., *cookingMethod*, *dietaryFilters*, *dishName*, *ingredients*, etc. So, given a user utterance the system first strips superfluous language leaving only recipe-related information. Using an off the shelf Natural Language Processing package, the residual utterance is decomposed into noun-phrases and verb-phrases. Each component is then checked against the Wholefoods Snapshot dataset since the fields required by the API and dataset are almost one-to-one, e.g. *ingredients:ingredients*, *cookingMethods:cookingMethods*, *dishName:displayName*, *mealCourse:courses*, etc. If afforded the opportunity to progress to semifinals, our next step would have been to train an NER model to extract important information like recipe names, ingredients, and quantities expressed in an utterance.

2.2 Dialogue Policy and Response Generation

Given a task manual T, the job of the system is to guide users through all the steps in the task T. In our system the bulk of the logic to manage the dialogue flow lives in *SELECTING STRATEGY*, which has adopted what logic would otherwise be in *RANKING STRATEGY* as in the default TaskBot convention. We structured *SELECTING STRATEGY* to act primarily as an intent classifier, triggering and dispatching pertinent dialogue details to response generators, which we conceptualize as specialized functions which handle those intents.

First the system assesses which phase the user is in. This dictates the type of functionality that is available. During the search phase a user can supply a question or query, in which case they will be presented with a selection of tasks that meet their search criteria produced by *DIY QUERY RESPONDER* or *RECIPE QUERY RESPONDER*. Similarly, a user can ask a question, request a recommendation, or ask to be presented more results triggering *QA RESPONDER*, *RECOMMENDATION RESPONDER*, and *MORE RESULTS RESPONDER* respectively, the former two of which employ an LLM for their purposes. Note, users who exit their task during the execution phase without completion will be greeted upon their next interaction with an option to resume their previous interaction or begin anew. Users can choose their desired task flexibly through partial name matching (e.g. "braid"), order selection (e.g. "second option"), or interfacing with an on-screen option (see Figure 2). Which responder gets triggered depends on the domain of a user's utterance. Once the desired task is chosen, a corresponding state variable is set to the relevant domain. This simplifies the subsequent intent classification process by reducing the possible interpretations to the execution phase of a particular task and domain. Phase and task domain inform system decisions such as between responders like *RECIPE QUERY RESPONDER* and *RECIPE SHOW STEPS RESPONDER*. During the execution phase, a user can freely navigate steps, phases, and supplemental functionality like *repeat* where the system will restate the current step and *start over* for returning to the search phase. Note, if a user's intention is uncertain the system will default to the *QA RESPONDER* as a backup.

One key challenge was implementing *back* functionality, i.e. returning to the immediately previous state. Ideally the system would maintain carefully selected state variables that track the minimum necessary details to represent a well-defined state. The rigid dialogue flow imposed by the default TaskBot architecture limits what, and most importantly how much, information can be stored. The default cache persists only cursory information such as user utterance, ASR confidence, user intent, system output, and NLP pipeline output. The system should also track evidence of transitions such as previously triggered response generators, chosen tasks, selection of on-screen options, current step, if a recommendation was requested, operating domain, and API search criteria. Persistent variable caching is limited in measure by the datastore DynamoDB, constraining the total possible stack size.

2.3 Query Response Generator

Once *SELECTING STRATEGY* determines a user's utterance contains a task query, *QUERY RESPONDER* extracts the query information and funnels it to either the recipe or DIY API.

User: Show me a recipe for instant pot zuccini pizza caserole.
Query: $\text{DishName} - \text{zuccini pizza caserole}$, $\text{Instrument} - \text{instant pot}$

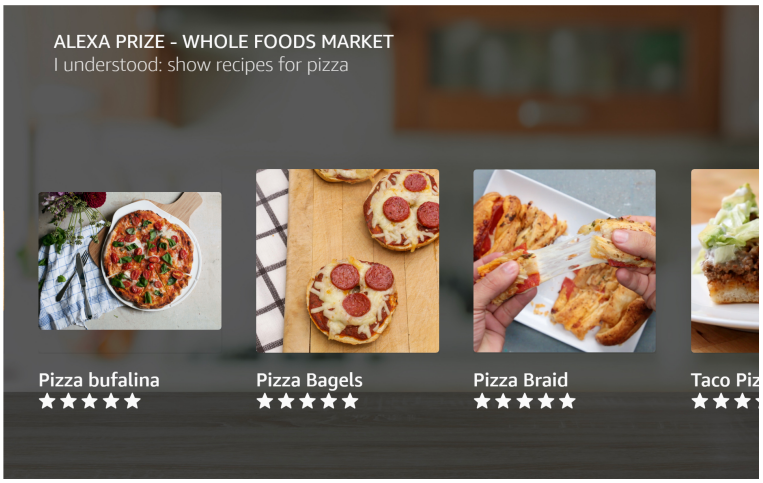

Figure 2: Example of results from a search for a recipe.

The results of the API search are returned to the responder, and organized into speech output and APL document for a user to select from.

2.4 Show Steps Response Generator

When a user chooses a task, they start in the ingredients stage if recipe, and step 1 if DIY. Once a user indicates they're ready to move on to the recipe proper, navigation relative to DIY and cooking tasks is symmetrical. Every next step they system navigates the API response data structure to extract the new information, and build a new APL document and speech output. Each turn the state is persisted in the event that a user disconnects or exits the task prematurely, so they can resume at their exit point.

Handling General Q/A: With the recent and rapid development of Large Language Models (LLMs), we explore the utilization of their strong generative capabilities to provide appropriate responses to customers' general questions in recipe and DIY-related interactions. As we lack direct interaction data from customers, we simulate general question conversations. Taking recipe inquiries as an example, we classify general questions into four types: ingredient introduction, cuisine introduction, recipe background, and recipe-related ingredient inquiry. For each category, we initially design two prompts for ingredient introduction, five prompts for cuisine introduction, four prompts for recipe background, and six prompts for recipe-related ingredient inquiry, manually. Additionally, we enlist linguistic experts to contribute more prompts based on similar applied scenarios, encompassing varied vocabulary and prompt paraphrasing. Table 1 presents examples of prompts for each category.

Question Category	Prompt Example Input
Ingredient introduction	Provide an introduction the ingredient: {ingredient}.
Cuisine introduction	Give me some background information on {cuisine} cuisine.
Recipe background	Where does {recipe_name} originate from?
Recipe-related ingredient inquiry	Can you list the essential ingredients for making "{recipe_name}"?

Table 1: General Q/A Prompt

For backbone model selection, we conduct tests on simulated conversations using various open-source LLMs, such as T5-series models[3] [3], T0-series models[4] [4], and LLaMA-series models[5] [9]. Generally, LLMs with a greater number of parameters exhibit stronger generative performance but also result in slower response latency. Considering the trade-off between response latency and Q/A

[3]https://huggingface.co/docs/transformers/model_doc/t5
[4]https://huggingface.co/bigscience/T0
[5]https://huggingface.co/docs/transformers/main/model_doc/llama

performance, we opt for T0_3B[6] [4], an open-source LLM over 3 billion parameters, as the first version of our backbone model for general Q/A. Utilizing the backbone model, we employ diverse inputs with the manually designed prompts and, with the guidance of linguistic experts, identify the top ten prompts for each category through evaluations.

The open-source LLMs can offer general responses to recipe and DIY questions, but these responses may lack specificity to the cooking and DIY domain. This is because open-source LLMs are pretrained on diverse corpora, rather than being focused on specific expert fields with conversational data. To enhance the LLM's performance, fine-tuning it with specific field datasets becomes necessary. However, this process is limited by the quantity and quality of manually simulated conversation datasets, and the budget constraints are also a consideration.

Fortunately, our team was allowed to access the Amazon LLM *Alexa Teacher Model*, specifically AlexaTM-20B model [6], which has been fine-tuned using conversation datasets. This model demonstrates superior generation performance for our tasks, and it effectively addresses the response latency issues often encountered when using larger LLMs, thanks to the utilization of the internal API of AlexaTM models.

One issue to address is that our previous setting only considered single-round conversations, meaning an immediate response to the latest user utterance. However, crucial information for accurately answering users' questions may be hidden in previous rounds of conversations or the anchor article related to the recipe or DIY topic. Fortunately, with the support of AlexaTM in-context learning, we can incorporate multi-round conversations, including the article contexts as previous system responses, leading to improved and more relevant responses. To ensure efficient response generation and focus on the latest rounds of conversations, we restrict the context of conversations to five utterances. Additionally, if the response is deemed irrelevant to the question by the QA classifier API provided by Amazon, we will only use the latest user utterance to regenerate the response.

In addition to exploring deep and large models, we also implement a set of forbidden words using controllable non-neural-network methods. These words act as filters to eliminate inappropriate outputs, thereby fulfilling specific quality test requirements.

Recommendations Recommendation systems play an important role in areas such as recipe suggestions and DIY task searches. Because in some cases users may lack a precise understanding of their needs, possessing only broad or vague criteria. Therefore, it becomes essential to have a recommendation model that can deduce potential user requirements from their initial search query.

However, within conversational tasks, the typical search queries are often in informal, everyday language. It is difficult for traditional models in understanding these queries and subsequently generating appropriate responses. Traditional models often struggle to fully comprehend the intricacies and nuances of such natural, colloquial language [2]. This problem is alleviated with the help of transformer-based models, specifically Large Language Models (LLMs). These models are designed to understand natural language more effectively, thereby making them more adept at interpreting conversational search queries. In turn, they can produce more accurate and appropriate responses based on the user's input.

In this project, we deployed AlexaTM-20B [7] as our principal recommendation model. AlexaTM-20B is a Large Language Model (LLM) that has been pre-trained on data with a strong emphasis on conversation. It has been expressly designed to handle tasks that involve chat-like interactions, making it appropriate for recommendation purposes within our project's context.

The model is activated specifically when the terms "recommendation" or "recommend" are present in the user's query, triggering the model to generate a relevant output. However, there are inherent challenges to using an LLM in search tasks. The outputs produced by the LLM might lack the degree of specificity that a search task needs. While the LLM might generate broad categories or generic terms, a search task often requires pointed and specific results within a given dataset.

For instance, if the output from the LLM is "Pie," the practical need in a search scenario would be a more exact match within the dataset such as "Apple Pie." This introduces the need for an additional layer of precision matching, which is accomplished using the Best Match 25 (BM-25) algorithm in our project.

[6]https://huggingface.co/bigscience/T0_3B

BM-25 [8] is an information retrieval function that ranks documents based on the query terms appearing in each document. This algorithm is highly efficient for matching tasks, aiding in identifying specific entries in large datasets that are the most relevant to the output generated by the LLM. Therefore, it serves as a critical bridge between the LLM's broad outputs and the highly specific search results required in recommendation contexts, thereby enhancing the effectiveness of AlexaTM-20B in providing precise and relevant recommendations. With the assistance of the LLM and information retrieval function, the model can generate highly relevant recipes or DIY tasks for the user.

2.5 Generating Multi-Modal Responses

Our initial design for Multi-Modal responses was created with the goal to allow users to quickly get through long WikiHow articles, or long recipes. Our approach of this was displaying the information with the "AlexaTextList Template" which allows for displaying summarized versions of each step on one screen (the first sentence of a step), with automatic scrolling for whenever the user requests the next or previous step. For headless devices we wanted quickly navigate to the WikiHow articles that was deemed most appropriate to the user's query, instead of reading out multiple long WikiHow article names. The goal was to allow the user to extract as much information as possible as quickly as possible. However, this goal was complicated by poor Speech Recognition which confused headless device users when they reach a WikiHow article that was nothing like the utterance they initially said. To remedy this, more steps were added to the flow of the conversation to ensure that what the user wanted matched what the bot understood. For devices with APL capabilities, a new APL template was created that displayed the system's understanding of the user's utterance at the top of the screen for each step and for all conversation flows, and more correction paths to our flow in the event that the user's utterance was accidentally misinterpreted.

3 Evaluation

3.1 Engineering Dialogue Features

Part of our process to understand how users judge their conversations with our system involved constructing low level criteria segmented into three broad categories in character with Siro 2022 [5]: i) Task Completion, ii) Understanding, and iii) Efficiency. Our aim was to engineer dialogue features and train a regression model to predict user score. Below we discuss our features and evaluation results.

Task Completion, which we segment into low-level features *completed task* which indicates if a user has made it to and performed the final step, *resuming* if a conversation was taking place during a user's previous interaction.

Understanding, with constituents *system misunderstanding errors*, i.e. number of times the system does not properly understand user intent, *median ASR confidence score* across all utterances, technical or *system errors* or coding errors on the back end, and finally the total *number of QA* trigger events. Our dialogue policy sends user-inquiries to the QA responder, Amazon TM. However, we also send utterances to be addressed by the same module when they don't fit the dialogue paths of our other responders.

Efficiency, consists of firstly *efficiency*, which we define as a Boolean whether a user found a task they were interested in engaging with within the first three turns of entering the search phase [5]. In addition, the total number of *user interactions*, the *total duration* of the conversation in seconds, the *average latency*, *minimum latency*, and *maximum latency* between utterances and system responses in the session. Finally we tracked the *number of failed searches* where the API call did not return any results.

Other ancillary features include *device type* whether APL-supported or headless.

3.2 Log File Data Extraction

The information we needed to extract in order to fulfill the engineered features comes from mining the CloudWatch log files. However, by default the system cache is printed at every turn alongside our own logs which act as a simplified representation of the cache in addition to tracking other variables useful

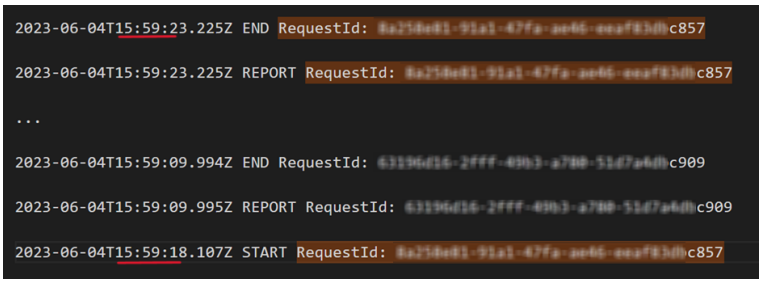

Figure 3: Example of log output not being chronologically ordered.

to the dialogue policy such as *phase*, *intent*, and *domain*. Moreover, our logs also trace dialogue paths, viz. which services and response generators are triggered along with their output.

3.3 Challenges of Parsing Log Files

While some information was simple to tease out, such as the number of user-intent misunderstandings or the device type, others were substantially more difficult because of the sometimes unreliable structure of the logs. One such challenge arises when multiple interactions happen to take place in close temporal proximity. Logs of the two conversations overlap and interweave, disrupting the flow of information. While there are often nearby anchors such as *conversationId* and *RequestId*, the fact that this is not always the case makes sorting information more complicated. In addition to occasional scrambling, sometimes details are also not chronologically sorted. Although the *START* marker should come before *END* then *REPORT* given some *RequestId*, cases such as Figure 3 demonstrate that logs don't always adhere to proper ordering.

3.4 Qualitative Analysis of Logs

Throughout the competition, qualitative analysis of logs was performed to examine features causing users' difficulty. This helped with the discovery and prioritization of new features, which was decided based on which issues users frequently encountered. During the competition our priority was to focus on rated conversation logs, and then look at the remaining logs without ratings. After the competition, we went through 160 rated conversations more thoroughly from part of the initial feedback period - our date range was 5/10/2023-6/11/2023. We noticed that roughly 20% of our ratings were users rating the conversation poorly (1 or 2 stars) when nothing seems to have gone wrong, or users giving ratings despite never interacting beyond the initial home screen, and users giving high ratings (4-5 stars) despite the bot performing poorly (by not guiding the users through their task due to a code error, query search came back with no results, or a speech recognition failure caused the user to go down a different path). We noticed that low ratings happened mostly from code errors, flow failures (conversation paths that weren't accounted for at the time) and users attempting chit chat with the bot, and a small percentage from ASR failures; however, many high rated conversations had instances of chit chat, flow failures and ASR failures as well. We also noticed that conversations under 40 seconds seemed to be inconsistent (some were rated poorly, some were rated highly) despite many of them never interacting with the bot.

Class	Mean RMSE
Task completion + understanding + efficiency + misc	2.390 (0.199)
Task completion + understanding + efficiency + misc [conversations over 40 sec]	2.390 (0.199)
Task completion + understanding	2.352 (0.114)
Ask completion + efficiency	2.424 (0.092)
Understanding + efficiency	2.387 (0.115)

Table 2: Regression Model Error Across Classes

Feature Name	Importance
failsearch	0.04198757
tot_utts	0.05106385
num_idk	0.05614763
num_tech_errs	0.05727179
efficiency	0.06847101
numQA	0.032577
mindur	0.06538291
maxdur	0.04865347
medianasr	0.07002685
isapl	0.08323045
avgdur	0.07873612
ConversationDurationInSeconds	0.10783626
Is Task Completed?	0.18983898
Is Resume task	0.04877608

Table 3: Feature Importance

3.5 Regression Model

Once information was curated from rated conversations, we sorted the data according to time stamp. This was to account for the fact that the data is time series, and that not all data points are necessarily from the same distribution. The state of the system at one point in time is likely different than another, given that development progresses and features advance. We opted to use the Gradient Boosted Tree model *XGBoostRegressor*, and trained it on 427 ratings from 5/10/2023-6/11/2023. It produced a root-mean-square error of *2.390*, indicating that the average difference between its predicted values and actual user scores is 2.390. We then trained the model on different combinations of classes, all producing similar results (Table 2). Given the nature of rated conversations under 40 seconds, we similarly trained the model on the pruned dataset - producing a similarly high error. Finally, we extracted the feature importance, indicating that *task completion* and *conversation duration* are the most impactful on user score (Table 3).

The evaluation results show our model is a poor indicator of actual user score, possibly for a number of different reasons. It is possible that user scores are perhaps not as reliably consistent as one might believe. It is also possible that, given our relatively small pool of dialogue features, they simply did not capture the correct user rating criteria.

4 Conclusion

Qualitative analysis does not quite align with our trained regression model. For example, qualitative analysis suggests that system errors and ASR failures tend to correlate negatively with user ratings, but the regression model predicts those features to be relatively inconsequential. However, note that ASR confidence in our case is *median*, and so does necessarily not catch the instances of exceptionally low ASR. It is most likely that our dialogue features do not fully capture the phenomena users look for when rating a conversation with the TaskBot, and more analysis is necessary in future work to improve these features.

References

[1] E. Agichtein, M. Johnston, A. Gottardi, C. Flagg, L. Vaz, H. Shi, D. Zhang, L. Ball, S. Liu, L. Dai, D. Pressel, P. Goyal, L. Hu, O. Ipek, S. Sahai, Y. Lu, Y. Liu, D. Hakkani-Tür, S. Hu, H. Rocker, J. Jeun, A. Iyengar, A. Mandal, S. Kuzi, N. Vedula, O. Rokhlenko, G. Castellucci, J. I. Choi, K. Bland, , Y. Maarek, and R. Ghanadan. Alexa, let's work together: Introducing the second alexa prize taskbot challenge. In *Alexa Prize TaskBot Challenge 2 Proceedings*, 2023.

[2] C. Gao, W. Lei, X. He, M. de Rijke, and T.-S. Chua. Advances and challenges in conversational recommender systems: A survey. *AI Open*, 2:100–126, 2021.

[3] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

[4] V. Sanh, A. Webson, C. Raffel, S. Bach, L. Sutawika, Z. Alyafeai, A. Chaffin, A. Stiegler, A. Raja, M. Dey, M. S. Bari, C. Xu, U. Thakker, S. S. Sharma, E. Szczechla, T. Kim, G. Chhablani, N. Nayak, D. Datta, J. Chang, M. T.-J. Jiang, H. Wang, M. Manica, S. Shen, Z. X. Yong, H. Pandey, R. Bawden, T. Wang, T. Neeraj, J. Rozen, A. Sharma, A. Santilli, T. Fevry, J. A. Fries, R. Teehan, T. L. Scao, S. Biderman, L. Gao, T. Wolf, and A. M. Rush. Multitask prompted training enables zero-shot task generalization. In *International Conference on Learning Representations*, 2022.

[5] C. Siro, M. Aliannejadi, and M. de Rijke. Understanding user satisfaction with task-oriented dialogue systems. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, jul 2022.

[6] S. Soltan, S. Ananthakrishnan, J. FitzGerald, R. Gupta, W. Hamza, H. Khan, C. Peris, S. Rawls, A. Rosenbaum, A. Rumshisky, et al. Alexatm 20b: Few-shot learning using a large-scale multilingual seq2seq model. *arXiv preprint arXiv:2208.01448*, 2022.

[7] S. Soltan, S. Ananthakrishnan, J. FitzGerald, R. Gupta, W. Hamza, H. Khan, C. Peris, S. Rawls, A. Rosenbaum, A. Rumshisky, C. S. Prakash, M. Sridhar, F. Triefenbach, A. Verma, G. Tur, and P. Natarajan. Alexatm 20b: Few-shot learning using a large-scale multilingual seq2seq model, 2022.

[8] E. A. Stuart. Matching methods for causal inference: A review and a look forward. *Statistical science: a review journal of the Institute of Mathematical Statistics*, 25(1):1, 2010.

[9] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.