Zero-Shot Bot

Amogh Mannekote, Oluwapemisin Bandy-Toyo, Vyom Aasit Pathak

Project Final Report to Spoken Dialogue Systems, CSE 4930/6930

Fall 2021

Table of Contents

1. Introduction

1.1 Motivation

Designing task-oriented dialogue systems manually has been at the forefront of developing very important systems in the research community. Moreover, developing such systems for specific tasks requires a lot of in-domain data, time, and manual effort. Businesses and organizations need dialogue systems to automate handling large numbers of queries from their customers, which can become cumbersome to develop and maintain.

Schema-based Dialogue

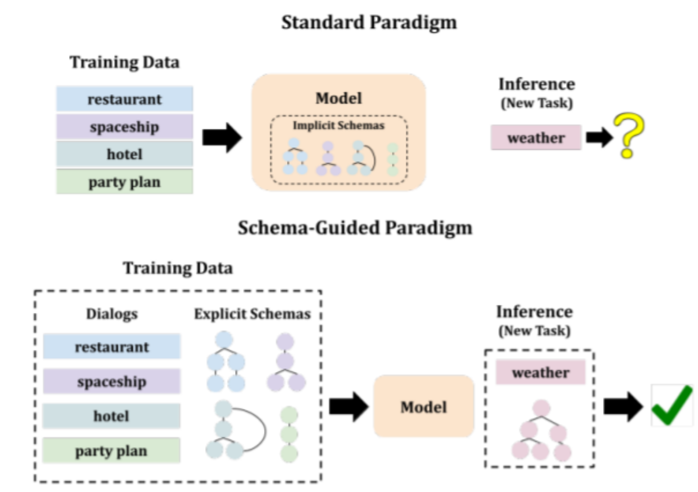


Figure 1: In the standard paradigm, data driven models implicitly learn the task-specific dialog policies (i.e., schemas). This precludes generalization to an unseen task at inference time. In contrast, in the schema-guided paradigm, dialog policy is explicitly provided to the model through a schema graph. At inference time, the model is given the schema for the new task and can therefore generalize in a zero-shot setting.

However, when learning NLU and dialogue management from corpora using machine learning algorithms, the tight coupling between the specific task and the general skills precludes the transferability of the learned models to similar, related domains and tasks.

Transferring among domains
For example, if we consider the general class of tasks of "reservations" or "bookings", there are several skills that are common across multiple domains. For instance, whether the domain is a hiking trip or a restaurant, the bot needs to be able to offer suggestions, ask for confirmation, look up information (and report it back to the user), etc. The only unique aspects of the dialogue would be the specifics of the items that are being spoken about (e.g., cuisine vs hiking distance), the order of the questions, and the language used.

Transferring among tasks
Similarly, there can be transfer among different tasks within the same domain as well. For example, if there is already training data available for booking a restaurant, a zero-shot transfer to a different task within the same domain could be canceling an existing reservation. Here, the

commonalities between the two tasks is the domain-specific vocabulary (such as “rooms”, “dates”, “length of stay”, etc.).

With zero-shot learning of dialogue management, the designer of the dialogue agent would not need to spend a lot of time collecting or writing examples for conversations. Instead, he/she can simply exploit the learned behaviors from the related tasks/domains, and would simply have to specify the schema for the new domain/task.

1.2 Description of System

Authentic need or desire

Our system is intended to examine the feasibility and user-friendliness of real-world interaction of users with schema-based zero-shot dialogue systems. Research papers mostly focus on quantitative evaluation of these systems using metrics such as accuracy, precision, recall, and F1-scores on a held-out test dataset of an unseen domain. We were interested in the question of what it would take to get it up and running as a practical system.

Task

Our system helps users to book cab rides from their source to their destination. It greets the user, asks them for their name, their start and end locations, and once they have confirmed, it makes an API call to a pre-populated database of available drivers, and informs them of the details of their ride (including the price, the name of the driver, the waiting time, the service provider used, and the license plate of the car).

Persona

The persona of our agent during the mid-term round robin was very no-nonsense and to-the-point. However, after our first round-robin, we included some personality in the utterances to clarify to the user that our system was still a novice and was still learning to do tasks in this unseen domain. For example, our agent now greets users as following in the beginning of the conversation:

Hi there, I am your zero-shot cab ride booking assistant. I have been trained to assist customers book flights, restaurant tables, meetings, etc., but this is my first time helping anyone book a cab, so I hope you will go easy on me :)

If at any point you think I lost you completely, please bear with me.

Just say "Hi" to begin talking to me.

1.3 Excerpt from Dialogue

USER: >> Can you give me a ride to TeaStory
SYS: >> May I have your name, please?

USER: >> Daniel
SYS: >> OK, and where are you departing from?

USER: >> University of Florida
SYS: >> Alright, where do you want to go?

USER: >> TeaStory
SYS: >> I found a Lyft ride for you from 'University of Florida' to 'teastory' for 33 credits that could pick you up in 20 minutes. Should I book that for you?

USER: >> Sure
SYS: >> Ok, I have booked you a ride with licence plate '901 FSA'. Your booking ID is '987'.

USER: >> Thanks
SYS: >> Ok, I have booked you a ride with licence plate '901 FSA'. Your booking ID is '987'.

USER: >> goodbye
SYS: >> Goodbye. Enjoy your ride!

2. Description of System Architecture

2.1. Initial Prototype and Results

Our initial prototype was a dialogue system with a Jupyter notebook front-end. This interacted with our zero-shot model in the backend in order to provide responses to a user's input. We tested out the model's ability to respond to utterances in domains that it was not explicitly trained on such as ride sharing. The result of this was a dialogue system whose responses were not always appropriate. It lacked the ability to extract entities in an utterance which made it difficult to respond to the user intelligently. We received a lot of feedback from users during the first round robin which informed our improvements in the final version.

Users told us that it felt as though the system wasn't actually understanding what they were saying because it didn't use grounding techniques. It also wasn't able to extract entities which it would use to make the responses more informative. Another sentiment that the users also expressed was the inability of the system to extract multiple bits of information in one slot as opposed to one by one. Taking these into consideration, we aimed to develop a bot that addressed the aforementioned concerns.

2.2. Functionality Expansion and Refinement

On the first iteration, we tested the feasibility of the zero-shot model i.e. whether it was even possible to consider building a system from. After that we built a bare-minimum bot that could talk about checking balance in zero-shot settings. This system could not firstly extract the entities and also couldn't make any calls to the API to fetch the information. Next, we worked on developing a simple slot extraction system where we used GPT-3 Neo to extract the slot value from the user utterance. Once the entity extraction was done, we built the backend API that could fetch the results based on the input entities. This was done around the first week of the mid-point evaluation. After that, we further refined it by improving the slot extraction by using GPT-2 instead of GPT-3 Neo and some more parameter changes for the same which increased the slot extraction significantly. After that, we developed scenario cards and a simple chat system through which the user could interact with the system. At the end we also included a persona of the system.

2.3. User Studies

User testing was done formally during the following phases:

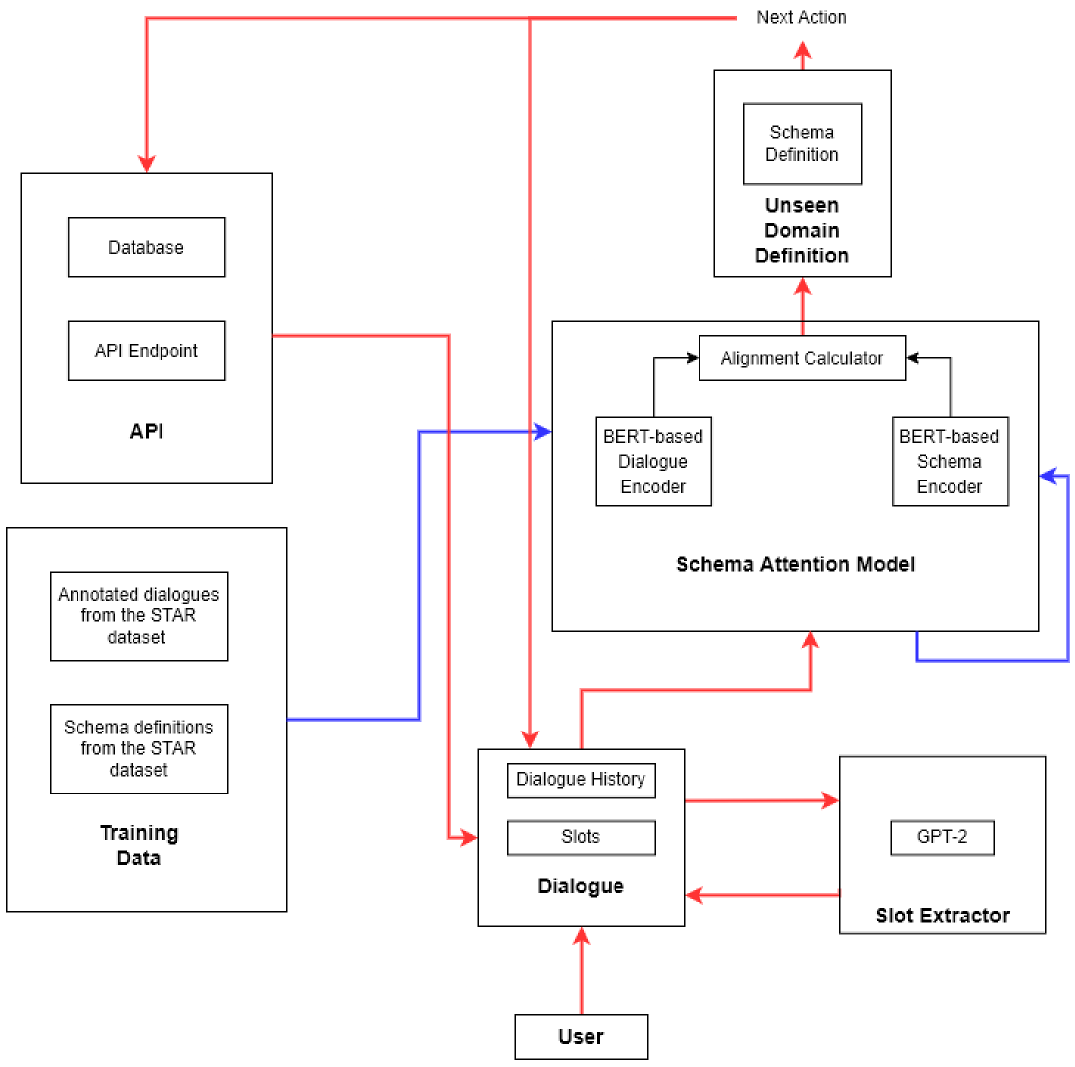
- Midpoint evaluation round robin - Here user's were part of our class Spoken Dialogue System. We conducted this in computer lab 231. It was conducted for 50 minutes with random users. Each User was given a scenario card to show them how to proceed with the system.
- Internally amongst the team for different dialogue schemas - We created different schemas individually and the other team members tested the system wth that schema. We conducted this session for around 30 minutes.
- Final evaluation round robin - Here user's were part of our class Spoken Dialogue System. We conducted this in computer lab 231. It was conducted for 50 minutes with random users. Each User was given a scenario card to show them how to proceed with the system.

2.4. Final Architecture.

The details of each module is described below:

1. Schema Attention Module - The core zero-shot model that calculates the alignment between the dialogue context and the schema definitions. It outputs a probability distribution for the next action,
2. Unseen Domain Definition - This is the schema for the new, unseen domain. It is represented as a graph, and each node corresponds to a different point in the dialogue. In code, it is represented as a JSON file.
3. Slot Extractor - This module uses GPT-2/3 to extract the slot values from user utterances. It hasn't been trained on any data. Instead, we prime the mode l with a few examples in the following form:

   System: What is your name?
   User: I'm Alan Turing.
   Question: The user's name is
   Result: Alan Turing (the model has to predict this)
4. API - Returns dummy responses for queries (e.g., returns the details of a cab randomly)
5. Dialogue - The dialogue history up until the current point. It also keeps track of the slots.
6. Training Data - The annotated training data (annotated with user intent, slot values, etc.). In our case, this comes from the STAR dataset.



Packages, Libraries, Frameworks
Our code base is primarily based on https://github.com/shikib/schema_attention_model (which is the accompanying code for the paper). The repository uses PyTorch for the machine learning parts of the code. On top of that, we are also using HuggingFace's Transformers library to invoke GPT-2 for the slot-extraction task. We used UF's HiPerGator to train and run our models. Most of our implementation for this project was done using Jupyter Notebooks and VSCode.

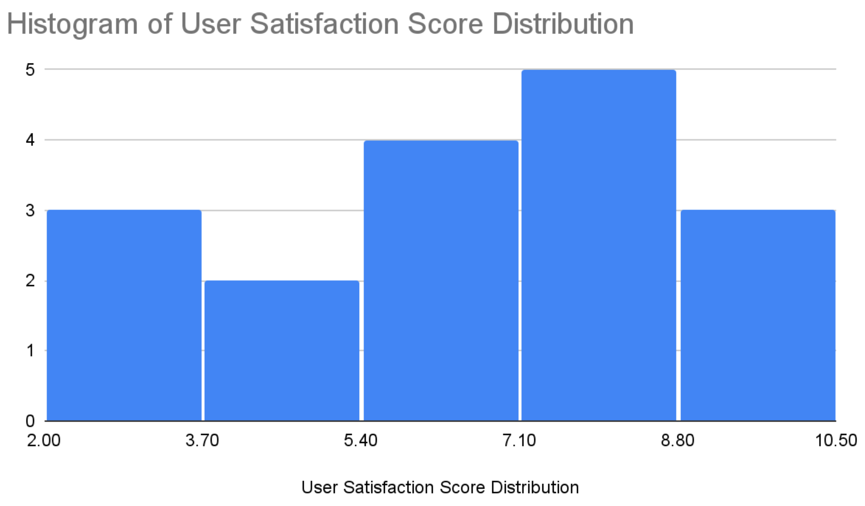
The schema definitions are made in YAML and all the code is written in Python.

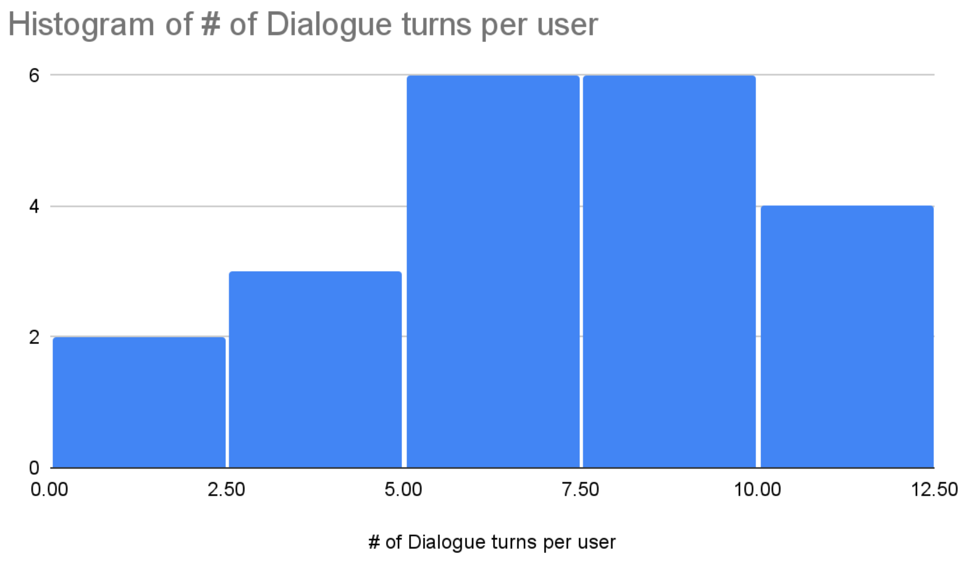
3. Final Evaluation Round Robin

3.1. User Interactions

Firstly, the user's name was taken for a matching. Then consent to record their audio was taken. Total of 17 users with an average of 5 minutes of interaction. Each user was then given a scenario card based on which they have to interact with the system. Then they interacted with the system through text-based chatting. Then, after the system interaction they were asked to fill out a post-survey.

3.2. Descriptive Statistics





3.3. Qualitative Reflections

- As users interacted with our system, we realized that we needed a way to explicitly communicate its limitations to them. Otherwise, they usually had unrealistic expectations of the system's capabilities.
- We realized that when users were informed that the system was still in its early stages of learning and had had no in-domain training data shown, they were less frustrated with the system's shortcomings.
- Slot extraction was a key capability that was lacking in our model during the midterm study. When we introduced a much more robust slot-tracking module for our second round-robin, we noticed that users had a better experience with the system.

Goals Review

Since our system (both for the midterm as well as the final round robins) was slightly different from our initial proposal, we will only be referring to the goals that are relevant to our revised system below. The completed goals are highlighted in yellow.
Minimal Goals: (1) Run the evaluation script for the Schema attention model (2) A chat interface to talk to the ZS-bot.
Basic Goals: (1) Run the ZS-bot on a custom user-domain (2) Bug fixes and improvements.
Stretch Goals: (1) Voice input (2) Answer the following research questions

- ○ Are there other aspects of the conversation logs that we can "factor out" (e.g., agent persona, casualness, etc.) into the schema?
- ○ Are there more fields we can add to each slot/domain to introduce additional helpful inductive biases, and as a consequence, improve accuracy (e.g., explicitly indicate whether a system action is a request or a confirmation or an inform, **or brief text descriptions, as in SGD**)?
- ○ How can we iteratively incorporate training data into a domain/task?
- ○ Fix a custom domain/task. Can we perform an analysis of what is the best way of specifying the schema for it (there are variations possible in the intent and slot names, order of slots, etc.)?
- ○ Do the slots have to be categorical? What about unseen slot values? Can we somehow handle them (say, inspired by TRADE)?

4. PARADISE Model

4.1 List of Predictors

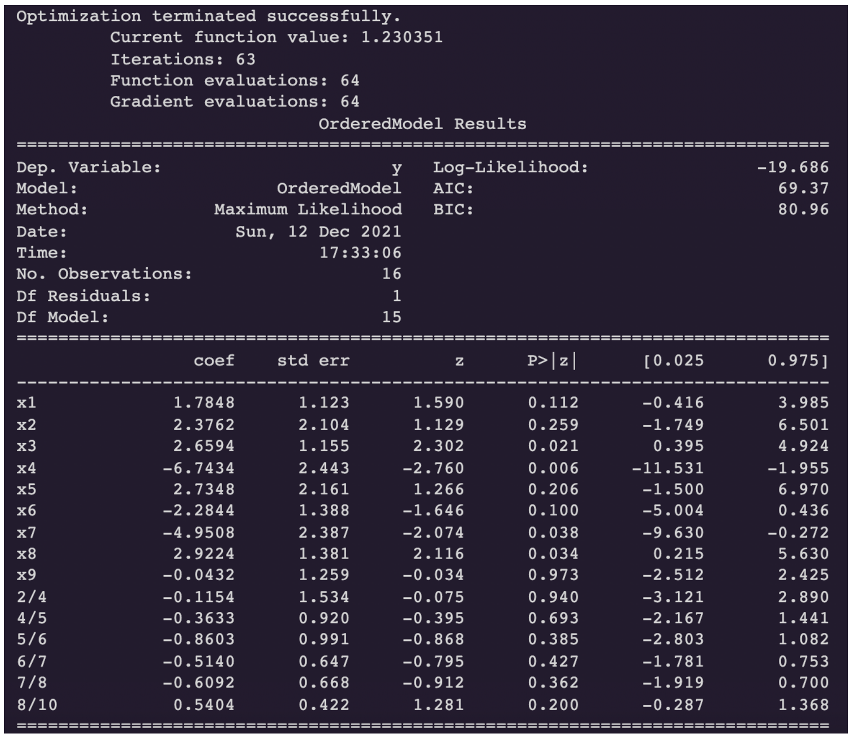
1. sys_ver - Version of the system developed.
2. num_turns - Number of turns between USER and SYS.
3. is_happy_path - Was the given dialogue a happy path?
4. num_repeat - Number of times the system says something.
5. recog_errors - Intent recognition for the dialogue for that user.
6. card_num - the index of the persona/scenario card given to the user
7. query_check_bug - While calling the API, there was an error.
8. code_error - Code threw an error while executing.

9. short_mem - Number of times the system asks the user to re-mention the slot (forgetting that they had just mentioned it)

4.2 User Satisfaction Metric

- User task Completion - On a scale of 1 to 10, was the user able to complete the task?

4.3 Regression Analysis



```
Optimization terminated successfully.
         Current function value: 1.230351
         Iterations: 63
         Function evaluations: 64
         Gradient evaluations: 64
                             OrderedModel Results
==============================================================================
Dep. Variable:                      y   Log-Likelihood:                -19.686
Model:                   OrderedModel   AIC:                             69.37
Method:            Maximum Likelihood   BIC:                             80.96
Date:                Sun, 12 Dec 2021
Time:                        17:33:06
No. Observations:                  16
Df Residuals:                       1
Df Model:                          15
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
x1             1.7848      1.123      1.590      0.112      -0.416       3.985
x2             2.3762      2.104      1.129      0.259      -1.749       6.501
x3             2.6594      1.155      2.302      0.021       0.395       4.924
x4            -6.7434      2.443     -2.760      0.006     -11.531      -1.955
x5             2.7348      2.161      1.266      0.206      -1.500       6.970
x6            -2.2844      1.388     -1.646      0.100      -5.004       0.436
x7            -4.9508      2.387     -2.074      0.038      -9.630      -0.272
x8             2.9224      1.381      2.116      0.034       0.215       5.630
x9            -0.0432      1.259     -0.034      0.973      -2.512       2.425
2/4           -0.1154      1.534     -0.075      0.940      -3.121       2.890
4/5           -0.3633      0.920     -0.395      0.693      -2.167       1.441
5/6           -0.8603      0.991     -0.868      0.385      -2.803       1.082
6/7           -0.5140      0.647     -0.795      0.427      -1.781       0.753
7/8           -0.6092      0.668     -0.912      0.362      -1.919       0.700
8/10           0.5404      0.422      1.281      0.200      -0.287       1.368
==============================================================================
```

- Here the order of the variable is given in the model image. The model used is an Ordered Regression Model as our output variable is an ordinal variable, where the BFGS method is used for optimization.
- The least p-value is 0.006 which is for Number of repetitions. This suggests that it is the most significant value. Other than that, Happy Path, API Errors, and Code Errors are significant features as well.
- After a quick glance I can see that the lower the value of the number of repetitions and Query Check Bug, the higher the User Satisfiability score.
- With more developer's hours, we could improve the system by working on decreasing the number of repetitions by the user i.e. make the system understand the user utterance in one dialogue by using different schemas.

5. Collaboration Report

How often did we meet and how much time did we invest in this project?

We met about once in two weeks for 5 hours each. Additionally, we also worked asynchronously on the modules assigned to ourselves. In the meetings, we discussed which tasks each of us had to do. If we had discussed the task in the last meeting, we would talk about each other's progress and see if any of us needed any help.

Individually, we invested around 5 hours a week for this project and during deadlines, around 10-12 hours a week.

How did we manage source code?
We did this using a shared GitHub repository as well as shared HiPerGator instances.

How did we track tasks/bugs?
We did this through a combination of Google Docs, GitHub issues, and Discord messages.

Who did what

Amogh Mannekote

- Came up with the project idea. This involved identifying the research paper in question, understanding it, and brainstorming with Vyom Aasit Pathak and Oluwapemisin Bandy-toyo about ways to turn it into a usable system.
- Set up the source code and bug tracking management systems.
- Implemented a wrapper layer over the existing model to support chatting with the model.
- Added support to the model for handling previously unseen schemas.
- Co-worked on the entity extraction module based on GPT3-Neo for the mid-term round robin with Vyom Aasit Pathak
- Co-developed the study protocol with Vyom Aasit Pathak for the mid-term round robin.
- Helped identify areas of improvement for the system from the mid-term round-robin interactions.
- Was actively involved in creating the reports and presentations.
- Worked on developing the PARADISE model

Oluwapemisin Bandy-toyo

- Creation of a React UI. This is something that we plan to incorporate in future iterations.
- Creation of persona/scenario cards which allowed the user to personify someone attempting to complete specific tasks such as booking a ride somewhere.
- Transcribing user interactions that occurred during the round robins.
- Formulating different portions of the project reports i.e a portion of system architecture description.
- Creating alternative schemas for user interactions. This allows for different variations in how the system responds to users .
- Taking users through our bot's testing and survey process during the round robins.
- Worked on developing the PARADISE model

Vyom Aasit Pathak

- Brainstormed with Amogh Mannekote about the idea on how to use the paper as a viable commercial system.
- Co-worked on the entity extraction module based on GPT3-Neo for mid-term round robin with Amogh Mannekote .
- Co-developed the protocols and the process to follow for the mid-term round robin with Amogh Mannekote .

- Built a more robust GPT-2 entity extraction module with decent priming for final round robin which was a significant improvement.
- Was actively partaking in developing the reports, and presentations.
- Worked on developing the PARADISE for testing the usability of the dialogue system.

What went well?

- The motivation for our idea was well-founded. We got validation for the idea of using schema-guided dialogue systems with zero-shot learning to quickly prototype systems from our classmates who also built dialogue systems for this class.
- We were able to pivot and recover well from our initial idea of creating two bots (one for creating a bot and the other for actually chatting with) as we realized it was infeasible.
- This was a complex project that required understanding many moving parts, and the whole team came together exceptionally well to make sense of the whole project and execute their individual tasks.
- Everybody was actively involved in the brainstorming sessions, which led to a lively sharing of ideas.

What did not go so well?

- We did not meet as frequently as we would have liked to. The number of physical co-working sessions could have been higher.
- Our preparation for the midterm round-robin was sub-par. We were not able to collect as many dialogues as we would have liked.
- We made some assumptions about the capabilities of our model (assuming that the model could perform slot extraction), which turned out to be wrong. This could have been avoided by more careful reading of the paper and the code.
- We were not able to include a voice module which could have been better in-terms of user experience.