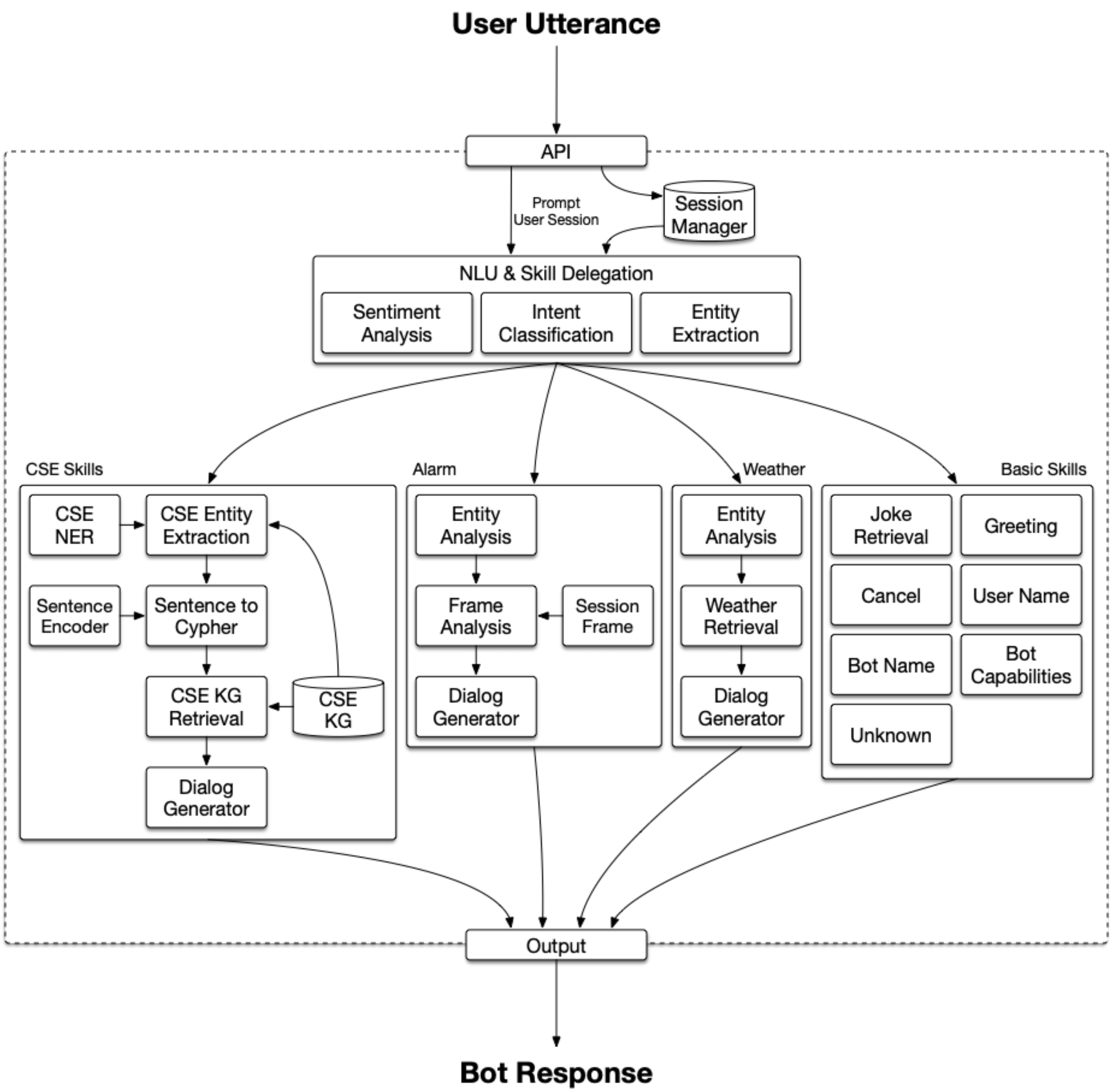
Final Project

Jeff Sheldon jeffsh@cs.washington.edu

For my final project, I have produced a chatbot that is capable of inferring user intent and delegating the handling of that intent to a series of specialized skill implementations. The chatbot can provide CSE course catalog data, provide the weather, set a virtual alarm, and execute several other skills. Additionally, the chatbot can support multiturn conversations, user personalization, and emotion.

Design Overview



The chatbot architecture relies on three primary components: user session data to track user information and multi-turn conversations, natural language understanding & skill delegation, and skill implementations.

Initial API Call & User Sessions

Chatot initiation is accomplished by passing an utterance to the chatbot API. New users are assigned a unique UUID and user session. The UUID is used to retrieve existing sessions. When a user calls the API with an existing UUID, the session manager retrieves the session for the UUID. The utterance and User Session are then passed to the NLU & Delegation module for processing.

Natural Language Understanding & Skill Delegation

Given the user utterance, the NLU unit runs three models against the utterance:

1. Intent Classification: this is accomplished via the fine-tuned DistilBert transformer network trained for prior assignments.
2. Named Entity Extraction: Spacy is used to identify named entities within an utterance. A custom wrapper around Spacy provides convenience functionality, combining B-I tags into single units. For example, B-LOC San & I-LOC Francisco are combined into a single LOC entity.
3. Sentiment Analysis: a pretrained transformer is applied to determine positive or negative sentiment within an utterance, with a confidence score. The NLU uses the confidence score as a measure of the intensity of the sentiment, which is recorded in the user session.

The intent is used to delegate control of the chatbot response to a specific skill handler. The delegator first examines the user session for the state of the prior recorded intent. If the prior intent is marked as incomplete, the new intent is ignored for the purpose of skill delegation and the prior intent is honored instead. This allows the bot to support multi-turn operations for the purpose of frame filling. However, if the new intent is recorded as "*cancel*", the prior intent is marked complete in the session and the bot informs the user that they may initiate a new action.

Once the delegator has determined the appropriate skill handler to invoke, the utterance, intent, named entities, and user session are passed to the skill handler. A reference to the Spacy model is also passed to the handler, should the handler have further need of performing sentence analysis.

The following intents are honored by the chatbot: *cancel, weather, tell joke, greeting, alarm, cse course content, cse course prerequisite, cse course id, change user name, user name, what is your name, what can I ask you*.

CSE Skill Handler

A single handler is used to respond to all three CSE-related intents. At the core of the handler are four primary components. Given a user utterance, these components will determine a query to execute and build a natural language response for the user.

The utterance is initially passed through CSE NER, a custom trained Spacy model capable of identifying CSE course numbers, titles, and topics. CSE NER extracts the entities within the utterance. Additionally, the CSE knowledge graph is also used to identify known entities within the utterance.

Google's Universal Sentence Encoder encodes the concatenated CSE intent and utterance to produce a vector that captures the semantic meaning. A CSV provides known statements, their associated intent, and corresponding Cypher query. A semantic vector is also produced for each known statement and intent. The inclusion of the CS intent in the embedding is intended to help guide the process to the correct known statement. The utterance vector is then compared to all known statement vectors to determine the most similar, using:

$$similar(V_u, V_{known}) = 1 - \arccos\left(matmul(V_u, V_{known})\right)$$

The Cypher selected according to vector similarity and the CSE entities extracted are passed to information retrieval. The entities are married to the Cypher, which is run against the CSE knowledge graph.

The results set is passed to the dialog generator. Each known statement is also matched to two response templates: one to be used if results are returned, one to be used if no results are returned. The dialog generator determines the appropriate statement to use and returns it to the user. If results are returned, the dialog generator will decorate the response template with data from the result set.

Alarm

The alarm skill will pretend to set an alarm for the user. If the user does not provide all necessary data, the skill will engage in a multi-turn conversation with the user to fill in the alarm frame with the required data.

Initially, the skill inspects the named entities passed by the delegator for relevant information. Any relevant entities are used to fill in a frame.

Next, an analysis of the frame is done. This will determine if further information is required from the user before the alarm can be set.

Finally, a response is generated and passed back to the user, based on the results of the frame analysis.

Note: this multi-turn functionality will be further explained in the Custom Improvements section.

Weather

The weather skill initially inspects the given named entities for location data. This operation, while relying on an intelligent model to gather the entities, is crude in the assumptions it makes. Specifically, it assumes that all locations are given in order of small to large. For example, given *Seattle, USA, WA*, the entity inspector would assume Seattle is in the state of USA, in the country of WA.

The gathered locations are used to populate an API call to OpenWeatherAPI for information retrieval. If no location data is given, the location is set to Seattle by default.

The API response is used to decorate a templated natural language response for the user. The template remains static, meaning no elements (other than those values in the API response) will ever vary. If the user did not provide a location, the response will let the user know that a location is required, and Seattle was used by default. If the API call fails, a canned response is given to the user, explaining the failure.

Basic Skills

Each of the following skills only perform vary basic internal operations, before generating a templated natural language response to pass back to the user.

- (Change) User Name: extracted entities are examined for a user name. If a name is found, it is recorded in the user session. Otherwise, the bot will generate a name for the user and record it. The chatbot then informs the user that the name has been set. Similarly, the user may query the chatbot for the user's name, which will be echoed to them according to what exists within the user session.
- Bot Name: The user may ask the chatbot what the bot's name is. An API call is made to https://api.namefake.com/ to generate a bot name, which is recorded in the user session (meaning the bot may have multiple personalities). The bot then informs the user of its name.
- Tell Joke: An API call is made to https://icanhazdadjoke.com/ to retrieve a random joke. If the API call fails, a pre-written joke is selected. The joke is echoed to the user.
- Greeting: One of several pre-written greetings is randomly selected and returned to the user. Greetings are sorted according to sentiment. If the user's mood is positive, a positive greeting is used. Otherwise, a negative greeting is used. If the user has previously specified their name, the name stored in the user session will be appended to the greeting.
- Cancel: The current intent stored in the user session is marked as complete, regardless of its current state. This frees up the chatbot to begin handling other intents. A natural language message informing the user of the cancelation is produced and returned. The message produced is chosen according to the user mood stored in the user session.
- What Can I Ask You: The chatbot replies with a natural language response, detailing all supported intents.
- Unknown Intent: The chatbot replies with a natural language response, informing the user that the current intent is unsupported and detailing all supported intents.

Error Handling

Knowledge Level

In the CSE Handler (which handles all three CSE intents), sentence embeddings are used to compare a user utterance to a library of known statements and the Cypher query that satisfies the statement. Additionally, two natural language response templates are also provided, per known statement: a templated response to be issued if the Cypher query returns results, and a static response to be issued if the Cypher query returns no results. After running the query, the result set is analyzed and if it is empty, the static no-result response is sent back to the user.

The no-result responses are written in a way that tries to anticipate why no results were returned. For example, given a Cypher query where a single filter of *course.number = <COURSE_ID>* is used to filter the result set, the response may ask if the user got the course number right.

Because the no-result responses are static, they are incapable of informing users of other issues. For example, it is possible that the CSE NER model could fail to extract "CSE P 590" from an utterance, instead extracting "CSE P" as a course id. In such a scenario, the Cypher statement will execute the search on "CSE P" and no results will be returned. The user would be informed that they had not provided a valid course number, when they had.

A more informative approach could attempt to catch such errors, prior to running the Cypher query. In the above example, a regular expression could catch that case and inform the user what had happened.

Additionally, the static response could be made dynamic, showing the user what had been run against the knowledge graph. Such a response would allow the user to see that their stated course number may have been correct, but the chatbot's understanding of the utterance was flawed.

Understanding Level

The alarm handler supports multi-turn conversations, to collect all the alarm date and time. On every turn, the handler checks the extracted entities for dates and times. If they are found, they are set on the frame. If they are not, the chatbot will ask the user for them and note what it had asked for. This allows the chatbot to attempt to retrieve data it either has not yet received or potentially did not understand previously in the selected entities.

If the user has asked for a specific piece of information and the user does not provide it, the chatbot will assume a default value and inform the user. This allows the handler to always make it to frame confirmation, regardless of data provided. Because the chatbot informs the user when it assumes and it grounds the final alarm with the user when asking for confirmation, the user can override the chatbot's assumptions.

This system allows the chatbot to always proceed to alarm confirmation, but the system will never set the alarm until the user has confirmed that the chatbot's assumptions about missing data are correct. What it does not do is account for misunderstood entities. A user may legitimately provide "noon" as a time, but the NER model may fail to identify it as a time. Because this failure is never explicitly stated to the user, the user must catch on to the failure, given the chatbot's assumptions, and attempt to correct it by restating "noon" as "12 pm".

Intent Level

That chatbot features two levels of intent handling. The first level is intent classification and confidence in the classified intent. Every utterance will be classified for intent. The confidence of the classification will then be compared to the intent confidence threshold of the chatbot, which is set to 0.6. If the intent confidence is below this threshold, the NLU module will delegate to an intent clarification handler. The handler will record the user utterance in a frame and engage in a multi-turn conversation, asking the user if the intent classification is correct. If the user answers affirmatively, the original prompt and intent classification will be forced through to the appropriate handler.

The second level of intent error handling is unsupported intents. If the classified intent is not supported, it will be delegated to an unknown skill handler. The chatbot maintains a DialoGPT[1] model and chat history specific to the user session. The user's prompt will be passed to DialoGPT and the model's response issued back to the user. Both the user prompt and the model response are maintained in a session chat history. This allows the user to carry on a conversation with the model, unique to that user. The model output is generated using top_p=0.5 word selection.

If intent classification falls below the confidence threshold, the user affirms that the intent is correct, and the intent is unsupported, the original utterance will be passed to the unknown intent handler.

The result of this implementation is two-fold. It allows the user to correct the chatbot if an intent is mis-classified. It also allows the user to engage in an entertaining conversation with the chatbot, within the

[1] https://huggingface.co/microsoft/DialoGPT-medium

context of the user's utterance. However, if the intent is unsupported, no clarification is passed to the user. This may fool users into believing that the chatbot is engaging with them for the purpose of fulfilling some skill, when in fact the chatbot is merely engaging in conversation that will end with no skill being performed.

If a user is confused about what the chatbot supports, the user may ask the chatbot what it can do and the chatbot will provide a list of all supported skills.

Custom Improvements

Multi-Turn Conversation + Alarm[2]

The alarm skill implementation features multi-turn conversation for frame filling, grounding, and data confirmation. The skill relies on the following data, provided by the NLU module:

- Intent: The intent classified by the NLU intent model. In the case of multi-turn conversations, this may not always be "alarm". For example, if the user provides a date to the chatbot, that could be classified as a "calendar". Alarm relies on "yes" and "no" intents for grounding confirmation.
- NER Entities: Alarm relies on entities extracted to fill its frame. Specifically, the date to set the alarm on and the time to set the alarm for.
- User Session: The session tracks data across turns. That includes:
    - Current Intent: If Alarm has not yet completed, this will be set to "alarm" to maintain skill delegation.
    - Is Intent Complete: This will be set to false until the Alarm frame has been filled and confirmed by the user. While false, the session current intent will control delegation.
    - Frame: Stores alarm data across turns. Includes:
        - Date: The date specified by the user.
        - Time: The time specified by the user.
        - Asked: The last request made to the user by Alarm.
        - Confirmed: True if the user has confirmed the alarm, else false.

Alarm begins by querying the session to determine if it is in the middle of a multi-turn conversation, according to Is Intent Complete. If true, Alarm will retrieve the session frame. Otherwise, Alarm will store a new frame on the session.

Alarm will then examine the named entities that have been extracted by the NER model in NLU. If date and/or time entities are found, it will take the first instance of the entity and set it in the frame. This allows a user to interact with Alarm and provide varying amounts of data to Alarm. A user may initiate Alarm with no day or time information, with only one or the other, or with both pieces of information. All scenarios will be handled gracefully by Alarm.

If Alarm is seeking data confirmation, intent will be examined for "yes" or "no". The frame confirmation property will be set appropriately.

Alarm will then examine the current state of the frame, to determine what information is still required. Alarm will initially seek date and time information if it does not yet have it. If data is missing, it will be

[2] Git: chatbot/ir_nlg_alarm.py

requested from the user. Alarm also records what it has requested. For example, if a date is missing, Alarm will request the date. The user may reply with a date, and it will be set on the frame. Otherwise, Alarm will fill the frame with a default value and inform the user. This allows the user to ignore certain requests while allowing Alarm to move on with the process.

When the frame is filled, Alarm will ground the information with the user and seek confirmation. At this stage, Alarm is seeking a “yes” or “no” intent to set to a confirmation field in the frame. If the user replies affirmatively, Alarm will mark the intent as complete in the session and confirm to the user that the alarm information is set. Otherwise, Alarm will request clarification, allowing the user to set a new data.

Alarm Intent Cancellation

At any point during a multi-turn Alarm conversation, a user may indicate to the chatbot that they wish to cancel the operation. This is the only way for the chatbot to halt the Alarm intent and move on to another intent. If the intent model classifies intent as “cancel”, the cancel handler will be invoked, and the session current intent marked as complete.

Shortcomings & Issues

When Alarm encounters date and time entities, it will always take the first instance of the entity type, ignoring all subsequent instances. For example, given the following utterance:

“Set my alarm Monday to Friday at 7am”

The NER model would extract entities: Date: Monday, Friday; Time: 7am

The user’s goal is to set an alarm every weekday. However, Alarm will only honor the first date encountered, Monday. Thus, the alarm will only be set for Monday at 7am.

Alarm relies solely on NER entity extraction to identify dates and times. A failure by the NER model to correctly extract entities will directly hamper Alarm’s ability. A more robust approach would provide some regular expression matching as a backstop to NER.

Alarm makes no assumptions about the relationship between dates and times. Everything provided by the user is taken at face value and set to the frame. This allows users to make absurd requests, such as “today at 7am” when 7am is already in the past.

Alarm is quick to allow a user to ignore a request, filling the requested data with a default value. A more engaging implementation may interact with the user more dynamically in such a scenario. Additionally, the user session tracks a user’s mood. This could be used to determine how impatient a user is, which could determine how quickly Alarm gives up on a request.

Quantitative Samples

<u>Sample 1</u>
U: Set an alarm
B: What day do you want the alarm for?
U: Monday
B: What time should I set the alarm for?
U: 7am

B: Ok, I'll set an alarm for you Monday at 7 am. Sound good?
U: yes
B: Your alarm is all set for Monday at 7 am!

Sample 2
U: Set my alarm at 6am
B: What day do you want the alarm for?
U: I don't care
B: Ok, I'll set an alarm for you every single day at 6 am. Sound good?
U: no
B: Did I get something wrong?
U: Just set it for Wednesday
B: Ok, I'll set an alarm for you Wednesday at 6 am. Sound good?
U: sure
B: Your alarm is all set for Wednesday at 6 am!

Sample 3
U: Set an alarm this weekend at 10am
B: Ok, I'll set an alarm for you this weekend at 10 am. Sound good?
U: nevermind
B: Got it. What else can I help you with?

Other Multi-Turn Skills

In addition to Alarm, the chatbot also utilizes multi-turn conversations to confirm intents that fall below the confidence threshold. The chatbot will ask the user to confirm that the intent classification is complete. The original user utterance is stored on the session and session marked as incomplete.

If the user response is classified as intent "yes", the original intent will be honored and forced through delegation. Otherwise, the chatbot will request that the user restate their query more clearly.

Personalization / Multiple Users[3]

User sessions allow data to be stored, specific to a single user. The API integrates with a Session Manager, which stores unique User Sessions, per an assigned UUID. On calling the API, the supplied UUID allows the Session Manager to retrieve the User Session. Data stored in the session includes:

- Name: If the user defines a name for themselves, it will be stored here.
- Bot Name: If the user asks the chatbot what its name is, that name will be stored here.
- Current Intent: Used to control skill delegation when a multi-turn conversation is in progress.
- Is Intent Complete: Works directly with Current Intent to identify multi-turn conversations.
- Frame: Supports the tracking of information for a multi-turn conversation.
- Mood: Stores an array of sentiment analyses performed on user utterances.
- Chat History: Stores conversation for DialoGPT, for conversation context.

[3] Git: chatbot/usersession.py; chatbot/ir_nlg_names.py

Among the intents supported by the chatbot are *change_user_name*, *what_is_your_name*, and *user_name*. When an utterance is classified as one of these intents, a user may change their own name, ask the bot its name, or ask the bot to repeat their own name. In the later, if the user has not yet defined a name, the chatbot will pick a name randomly for the user and set that on the session.

The user's name is also used when the chatbot greets the user if the name has been set. Otherwise, the chatbot will greet the user without a name. The chatbot's name is only used when the user asks that chatbot what its name is.

The User Session tracks conversational history when the user interacts with DialoGPT. When an intent is unsupported, DialoGPT is passed the user utterance, and the model's response is passed back to the user. The utterance and response are stored in the session chat history, to maintain conversation context.

Overall session tracking performs very well. Every user of the chatbot is allowed a unique session. Rare errors have occurred, where the client fails to pass the UUID back to the API, resulting in no session being recovered for the turn. While very rare, this bug still exists. In this scenario, repeating the utterance resolves the issue.

Usage of personalized user information is light in the chatbot implementation. The chatbot clearly demonstrates its ability to track unique session information and makes heavy use of that information in multi-turn conversations. But it rarely applies personalized user data to enrich conversations.

Incorporate Emotion

A sentiment analysis model is applied to every user utterance. Sentiment is classified as positive or negative and given a confidence score. This information is stored within the user session, providing a user-unique history of their sentiment across all conversation turns. The sentiment analysis model used is the Huggingface "sentiment-analysis" transformer pipeline[4].

Sentiment analysis is passed to the User Session as:

$$sentiment = confidence\ if\ sentiment\ is\ positive, else - 1 * confidence$$

Each sentiment passed to the session is appended to the back of an array. Therefore, the array maintains a history of all user sentiments, from oldest to newest. The session provides a method to retrieve the current user mood, derived from the entire history of sentiments. The user mood is defined as:

$$mood = \sum_{i=0}^{i=t} \left(1 - \left(\frac{1}{i+1}\right)\right) * sentiment_i$$

This allows the user's mood to accommodate nearly all recorded sentiments, progressively valuing sentiment more greatly, according to the freshness of the sentiment. The oldest sentiment is ignored.

User Session mood is applied in two scenarios:

1. When a user attempts to cancel an operation, the chatbot's response to the user is selected according to the user mood.

[4] https://huggingface.co/transformers/main_classes/pipelines.html

2. When the greeting intent is applied, the chatbot will select a greeting from a positive greeting list or a negative greeting list, according to the user's mood.

Quantitative Samples

Sample 1, mood: 4.8368
U: hello
B: Hi there.

Sample 2, mood: 2.9016
U: stop that
B: Got it. What else can I help you with?

Sample 3, mood: -0.9944
U: Hi chatbot
B: Are you doing ok

Sample 4, mood: -1.9544
U: cancel
B: I'm really sorry, can I do anything else for you?

Sentiment Model Performance

The Sentiment140[5] test dataset was used to evaluate the Huggingface sentiment-analysis pipeline. The dataset defines sentiment as 0 for negative, 2 for neutral, and 4 for positive. The Huggingface model only differentiates negative and positive. Neutral test data was discarded. 359 test records remain. Scikit-Learn Classification Report was used to analyze the model performance.

	PRECISION	RECALL	F1	SUPPORT
NEGATIVE	0.79	0.89	0.84	177
POSITIVE	0.88	0.76	0.82	182
ACCURACY			0.83	359
MACRO AVERAGE	0.83	0.83	0.83	359
WEIGHTED AVG	0.83	0.83	0.83	359

Demonstrative Examples

CSE Course Content

Utterance: *Tell me about CSE 517*

Walkthrough

1. In the NLU module, the intent model classifies the intent as cse_course_content with .9999 confidence.
2. The intent and prompt are passed to the CSE handler.
3. The CSE NER model extracts the following entities: 'CSE 517', 'CSE'
4. The sentence encoder creates an embedding of the concatenated intent and prompt. This vector is compared to all known statement vectors.

[5] http://help.sentiment140.com/for-students

5. With .8497 confidence, the prompt is matched to "*cse_course_content What is <C_ID> about?*"
6. The corresponding Cypher is filled in with the appropriate entities and run against Neo4j: *MATCH (c:Course) WHERE c.number=~ '(?i)CSE 517' RETURN c*
7. The natural language template for the Cypher query is populated with the data returned from the result set and returned to the user: *I can show you the course description.[\n<description>].*

Response: *I can show you the course description.*
Overview of modern approaches for natural language processing. Topics include language models, text classification, tagging, parsing, machine translation, semantics, and discourse analysis.

Utterance: *what courses are about natural language processing?*
Response: *Here are some courses similar to that:*
CSE 447: Natural Language Processing
CSEP 517: Natural Language Processing
CSE 517: Natural Language Processing.

Utterance: *Are any courses offered jointly with E E?*
Response: *Here are some jointly taught courses:*
CSE 470: Computer Architecture II
CSE 587: Advanced Systems And Synthetic Biology
CSE 488: Laboratory Methods In Synthetic Biology
CSE 371: Design of Digital Circuits and Systems
CSE 577: Special Topics In Computer Vision
CSE 578: Convex Optimization
CSE 586: Introduction To Synthetic Biology
CSE 576: Computer Vision
CSE 486: Introduction To Synthetic Biology
CSE 475: Embedded Systems Capstone
CSE 469: Computer Architecture I
CSE 474: Introduction to Embedded Systems
CSE 487: Advanced Systems And Synthetic Biology.

CSE Course Prerequisites

Utterance: *What can I take after taking CSE 143?*

Walkthrough

1. The intent model classifies the intent as cse_course_prerequisites with .9996 confidence.
2. The intent and prompt are passed to the CSE handler.
3. The CSE NER model extracts the following entities: 'CSE 143', 'CSE'
4. The sentence encoder creates an embedding of the concatenated intent and prompt. This vector is compared to all known statement vectors.
5. With .7623 confidence, the prompt is matched to "*cse_course_prerequisites Which course can I take after taking <C_ID>?*"
6. The corresponding Cypher is filled in with the appropriate entities and run against Neo4j: *MATCH (c:Course)-[r:PREREQ]->(cp:Course) WHERE cp.number =~ '(?i)CSE 143 ' RETURN c*
7. The natural language template for the Cypher query is populated with the data returned from the result set and returned to the user: *You can take the following courses:[\n<display>]*

Response: *You can take the following courses:*
CSE 154: Web Programming
CSE 311: Foundations Of Computing I
CSE 331: Software Design And Implementation
CSE 351: The Hardware/software Interface
CSE 340: Interaction Programming
CSE 341: Programming Languages
CSE 373: Data Structures And Algorithms
CSE 374: Intermediate Programming Concepts And Tools
CSE 391: System and Software Tools
CSE 412: Introduction to Data Visualization
CSE 414: Introduction to Database Systems
CSE 416: Introduction to Machine Learning
CSE 469: Computer Architecture I
CSE 474: Introduction to Embedded Systems

Utterance: *what's required for cse 486*
Response: *You'll need to take the following courses:*
MATH 136
MATH 308
MATH 307
AMATH 351
AMATH 352

Utterance: *What class should I take if I want to take cse 475*
Response: *You'll need to take:*
CSE 369: Introduction to Digital Design
E E 271
CSE 466: Software For Embedded Systems
CSE 474: Introduction to Embedded Systems
E E 474
E E 472

CSE Course ID

Utterance: *What's the course number for Natural Language Processing?*

Walkthrough

1. The intent model classifies the intent as cse_course_id with .9994 confidence.
2. The intent and prompt are passed to the CSE handler.
3. The CSE NER model extracts the following entities: 'Natural Language Processing'
4. The sentence encoder creates an embedding of the concatenated intent and prompt. This vector is compared to all known statement vectors.
5. With .9758 confidence, the prompt is matched to "*cse_course_id What is the course number for <C_TITLE>?*"
6. The corresponding Cypher is filled in with the appropriate entities and run against Neo4j: *MATCH (c:Course) WHERE c.title =~ '(?i)*Natural Language Processing*' RETURN c*

7. The natural language template for the Cypher query is populated with the data returned from the result set and returned to the user: *I found the following course numbers for that title:[\n<number>]*

Response: *I found the following course numbers for that title:*
CSE 447
CSEP 517
CSE 517

Utterance: *What classes are offered jointly with e e 371?*
Response: *These courses are taught jointly with that:*
CSE 371: Design of Digital Circuits and Systems

Utterance: *What's the title for cse 163*
Response: *The course title is Intermediate Data Programming.*

Alarm

Alarm uses multi-turn conversations to fill a frame of required information. The first example below will walk through a multi-turn conversation and will note the chatbot state for each turn.

TURN 1
Utterance: *Set an alarm for me*
Walkthrough

1. NLU identifies intent as 'alarm' with .923 confidence and delegates to the alarm handler.
2. Alarm handler initializes a new frame.
3. No entities are found.
4. Frame analysis notes that no date is provided.
    a. User Session intent complete is set to false.
    b. Frame > last asked is set to 'DATE'.
    c. A response is returned to the user, asking for a date to be provided.

Response: *What day do you want the alarm for?*

TURN 2
Utterance: *Friday*
Walkthrough

5. NLU identifies intent as 'calendar' with .5776 confidence.
6. NLU notes that the prior alarm intent is incomplete and delegates to the alarm handler.
7. Alarm handler retrieves the current frame from the user session.
8. DATE entity 'Friday' is extracted and set to the frame.
9. Frame analysis notes that no time is provided.
    a. User Session intent complete is set to false.
    b. Frame > last asked is set to 'TIME'.
    c. A response is returned to the user, asking for a time to be provided.

Response: *What time should I set the alarm for?*

TURN 3
Utterance: *Set it for 7am*
Walkthrough

10. NLU identifies intent as 'alarm' with .4546 confidence.

11. NLU notes that the prior alarm intent is incomplete and delegates to the alarm handler.
12. Alarm handler retrieves the current frame from the user session.
13. TIME entity ‘7 am’ is extracted and set to the frame.
14. Frame analysis notes that the frame has not been confirmed yet.
    a. User Session intent complete is set to false.
    b. Frame > last asked is set to ‘CONFIRMED’.
    c. A response is returned, grounding the frame and asking for confirmation of frame info.

Response: *Ok, I'll set an alarm for you Friday at 7 am. Sound good?*

<u>TURN 4</u>

Utterance: *No*

Walkthrough

15. NLU identifies intent as ‘no’ with .9919 confidence.
16. NLU notes that the prior alarm intent is incomplete and delegates to the alarm handler.
17. Alarm handler retrieves the current frame from the user session.
18. No entities are extracted.
19. Frame > last asked is ‘CONFIRMED’ and intent is ‘no’, so frame > CONFIRMED is set to false.
20. Frame analysis notes that the frame > CONFIRMED is false.
    a. User Session intent complete is set to false.
    b. Frame > ‘CONFIRMED’ is deleted.
    c. A response is returned to the user, asking for clarification.

Response: *Did I get something wrong?*

<u>TURN 5</u>

Utterance: *I meant Saturday*

Walkthrough

21. NLU identifies intent as ‘definition’ with .7378 confidence.
22. NLU notes that the prior alarm intent is incomplete and delegates to the alarm handler.
23. Alarm handler retrieves the current frame from the user session.
24. DATE entity ‘Saturday’ is extracted.
25. Frame analysis notes that the frame has not been confirmed yet.
    a. User Session intent complete is set to false.
    b. Frame > last asked is set to ‘CONFIRMED’.
    c. A response is returned, grounding the frame and asking for confirmation of frame info.

Response: *Ok, I'll set an alarm for you Saturday at 7 am. Sound good?*

<u>TURN 6</u>

Utterance: *yes*

Walkthrough

26. NLU identifies intent as ‘yes’ with .9968 confidence.
27. NLU notes that the prior alarm intent is incomplete and delegates to the alarm handler.
28. Alarm handler retrieves the current frame from the user session.
29. No entities are extracted.
30. Frame > last asked is ‘CONFIRMED’ and intent is ‘yes’, so frame > CONFIRMED is set to true.
31. Frame analysis notes that frame > CONFIRMED is true.
    a. User Session intent complete is set to true.
    b. A response is returned, grounding the frame and confirming the alarm is set.

Response: *Your alarm is all set for Saturday at 7 am!*

Utterance: *set an alarm Monday morning*
Response: *Ok, I'll set an alarm for you Monday at morning. Sound good?*

Utterance: *set my alarm next week every morning*
Response: *Are you talking about alarm?*

Note: NLU identified the intent as alarm, but the confidence of .5973 was below the threshold of .6. The chatbot asks the user to confirm that the intent classification is correct. If the user answers yes, alarm will be set as the session current intent and the call forwarded to the Alarm handler.

Weather

Utterance: *Show me the weather in San Francisco, California*

Walkthrough

1. In the NLU module, the intent model classified the intent as weather with .9951 confidence.
2. Entities LOC 'San Francisco' (B, I) and LOC 'California' were extracted from the utterance using the NER model.
3. The process was delegated to the weather handler.
4. The weather handler formed a call to OpenWeatherAPI. San Francisco and California were passed to the API, in order of most specific to least specific.
5. The JSON response is used to fill a natural language templated response.
6. The populated natural language response was returned to the user.

Response: *Here's the weather for San Francisco,California.*
The current weather is 64 degrees and scattered clouds. Humidity is at 76%.
The high temperature today is 90, low is 51.
Sunrise is at 05:48, sunset at 20:27.

Utterance: *Is it cloudy in Seattle?*
Response: *Here's the weather for Seattle.*
The current weather is 68 degrees and clear sky. Humidity is at 55%.
The high temperature today is 73, low is 63.
Sunrise is at 05:13, sunset at 21:01.

Utterance: *tell me the weather*
Response: *Querying for weather without specifying a city is currently unsupported. Here's the weather for Seattle.*
The current weather is 68 degrees and clear sky. Humidity is at 55%.
The high temperature today is 73, low is 63.
Sunrise is at 05:13, sunset at 21:01.

Tell Joke

Utterance: *Tell me a joke*

Walkthrough

1. The NLU intent model classified the intent as tell_joke with .9982 confidence.
2. The task was delegated to the joke handler.
3. The joke handler formed an API call to and JSON was returned.

4. The JSON payload was parsed, and the joke was returned to the user.

Response: *Two muffins were sitting in an oven, and the first looks over to the second, and says, "man, it's really hot in here". The second looks over at the first with a surprised look, and answers, "WHOA, a talking muffin!"*

Utterance: *say something funny*
Response: *Why did the girl smear peanut butter on the road? To go with the traffic jam.*

Utterance: *Do you have a sense of humor?*
Response: *If you want a job in the moisturizer industry, the best advice I can give is to apply daily.*

Change User Name

Change user name supports basic multi-turn conversations. If the user states they would like to change their name, but they don't provide a name, the chatbot will ask the user for a name.

<u>TURN 1</u>
Utterance: *I'd like to change my name*
Walkthrough

1. The NLU intent model classified the intent as change_user_name with .9718 confidence.
2. The NER model finds no named entities to extract in the statement.
3. The task is delegated to the change user name handler.
4. The handler notes the lack of a name given and marks the intent as incomplete in the session.
5. The chatbot returns a response, requesting a name.

Response: *Ok, what should I call you?*

<u>TURN 2</u>
Utterance: *You can call me Jeff*
Walkthrough

1. The NLU intent model classified the intent as change_user_name with .9718 confidence. (note that this is irrelevant, as the prior intent in the session is incomplete and so it will continue to drive delegation)
2. The NER model extracts entity PERSON "Jeff".
3. The task is delegated to the change user name handler.
4. The handler identifies that a PERSON entity is given.
5. "Jeff" is set in the user session as the user name.
6. The chatbot returns a response, grounding the user name with the user.

Response: *Nice to meet you, Jeff!*

Utterance: *My name is Mike*
Response: *Sounds good Jeff, er, I mean, Mike.*

Note
As the user had previously set their name to Jeff, the chatbot knows that a prior name already was defined in the session and provides a cheeky response while noting this fact to the user.

Utterance: *My name is John*
Response: *Ok, what should I call you?*

Note: The NER model failed to identify a PERSON tag for "John", leading the handler to engage in a multi-turn conversation to define a name.

User Name

Utterance: *tell me my name*

Walkthrough

1. The intent model classified the intent as user_name with .9962 confidence.
2. The task was delegated to the user name handler.
3. The handler identified the existence of a user name in the user session.
4. A natural language response was formed, populated with the session user name.

Response: *Your name is Mike.*

Utterance: *what is my name*
Response: *You never told me what your name is. I'll call you Han.*

Note: I restarted the session for this example, so no user name was defined. The name handler recognized the lack of a user name, randomly selected one from a list of pre-defined names, and set it as the user name, before returning a response informing the user of this action.

Utterance: *Who am I*
Response: *Your name is Han.*

What Is Your Name

Utterance: *Do you have a name?*

Walkthrough

1. The intent model classified the intent as what_is_your_name with .9905 confidence.
2. The task was delegated to the bot name handler.
3. The handler noted the lack of a previously defined bot name in the session.
4. The handler made an API call to **https://api.namefake.com/** and received a JSON response.
5. The JSON was parsed for the name, which was set in the user session as the bot name.
6. A natural language response was formed, informing the user of the bot's name.

Response: *I go by Concepcion Hilpert.*

Utterance: *What is your name*
Response: *Officially, Concepcion Hilpert, but you can call me whatever you want.*

Utterance: *Tell me your name*
Response: *I go by Concepcion Hilpert.*

Greeting

Utterance: *Hello*

Walkthrough

1. The intent model classified the intent as greeting with .9885 confidence.

2. The task was delegated to the greeting handler.
3. The session provided a was previously defined name as “Darth Vader”.
4. The user’s mood was requested from the user session, which was returned as 2.6855 (*>= 0 is considered positive, < 0 is negative*).
5. Because the user’s mood is positive, a greeting was randomly selected from a predefined list of positive greetings.
6. “Darth Vader” was populated on the response and the response returned to the user.

Response: *Good day Darth Vader.*

Utterance: *hi*
Response: *...hi... Darth Vader.*

Note
I purposely drove the user mood below 0 before testing this example. Because the user’s mood is negative, the chatbot selected a greeting from the negative set of greetings.

Utterance: *sup*
Response: *Are you talking about greeting?*

Note
The intent model’s confidence in assigning greeting as the intent was .4289, below the threshold of .6. As a result, the model is seeking clarification (in the form of a yes/no response) on the intent.

Cancel

Utterance: *cancel*

Walkthrough
1. The intent model classified the intent as cancel with .9911 confidence.
2. Because the intent is cancel, delegation ignored the current state of the prior intent and forwarded the call directly to the cancel handler. This allows the chatbot to break off multi-turn engagements when the user wishes to cancel.
3. The handler set the prior intent to complete.
4. The handler identified the user’s mood as negative (-1.8179).
5. A natural language response, informing the user of the cancelation was selected, was selected according to the user’s negative mood.

Response: *I'm really sorry, can I do anything else for you?*

Utterance: *stop that*
Response: *I'm really sorry, can I do anything else for you?*

Utterance: *nevermind*
Response: *I'm really sorry, can I do anything else for you?*

What Can I Ask You

Utterance: *What can you do?*

Walkthrough

1. The intent model classified the intent as what_can_i_ask_you with .9969 confidence.
2. The task was delegated to the bot capabilities handler.
3. The list of supported intents was collected. For each intent, underscores were replaced by spaces, to make the intent more human readable.
4. The bot responds to the user with the list of capabilities in natural language.

Response: *I can perform the following tasks:*
cancel
weather
tell joke
greeting
alarm
cse course content
cse course prerequisites
cse course id
change user name
what is your name
user name
what can i ask you
get intent clarity

Utterance: *what are you capable of*
Response: *I can perform the following tasks:…*

Utterance: *Show me your skills.*
Response: *I can perform the following tasks:…*