Human-Machine Dialogue Project
AI Cooking Assistant

Rigo Andrea

I. Introduction

This project builds a small demo conversational agent with the aim of helping the user cook a recipe, following it step by step. The user can directly select a recipe (assuming that it is present in the assistant database) or ask the system to propose some recipes. After a recipe is selected, the system will read the ingredient list for the user, and will help to follow the steps by telling him/her what the current step is, then switching to next step when the user finished the current one and tells the system to proceed. Additionally, the system can manage a shopping list for the user and handle some basic chitchat. The system is task based, designed to help the user in some predefined tasks such as following a recipe, adding and removing items from a shopping list and handle chitchat on a basic level. The recipe database is hand-written and the assistant is build on top of the Rasa framework [1] version 2.8.17. This report describes the tasks the system is designed to carry out in section II, the dialogues that the system can handle in section III, the model used in section IV, the data in section V and the evaluations techniques and results in section VI. The code is available on the GitHub repository.

II. Domain

As mentioned in the introduction, the system can handle three main tasks:

- Cooking assistance
- Shopping list management
- Chitchat and FAQs

A. Cooking assistance

The user can ask the model to suggest a recipe, for example by saying *"What can I cook today?"*. The system will then display a list of recipes for the user to choose from. Since the database only contains four recipes, the code will currently display all of them. If the assistant would be subject of further development it would have to be adjusted to display a random selection of recipes. The assistant will then guide the user to fill the required slots to start cooking a recipe: the desired recipe itself and the number of people the user would like to cook for. A more experienced user that already knows which recipes the system knows can directly select a recipe and number of people, for example Carbonara, by saying *"Let's cook Carbonara"* or *"I would like to cook Carbonara for two"*. Recipes are treated as sequences of steps, where the user can move as he/her sees fit. By default the system will move to the next step when the user communicates that the current step is done; but the user can ask the system to repeat the current step, go back any number of steps, for example to check for mistakes, while a more experienced user can skip any number of steps if, for example, he/she already knows how to prepare most of the recipe. Since in some cases the ingredient list can be relatively long and hard to memorize, the user may request the system to repeat the recipe's ingredients. This can either be done with *"Tell me the ingredients"* to get the entire ingredient list, or by saying *"How much guanciale?"* to request the amount of a single ingredient. The user can also ask for the amount of an ingredient used in the current step without specifying which one, for example using *"How much of it?"* as input. In this case the model will look in the database for which ingredient is used in the step and provide the amount. If more than one ingredient is used it will list them and ask the user to disambiguate.

B. Shopping list management

The system maintains a shopping list for the user that can survive across conversations. For the scope of the project the list is not permanently saved to disk, but it will stay in memory when issuing the `\restart` command as long as Rasa's action server is running. The shopping list is a simple list of ingredients. The user can ask to see it by saying *"Read my shopping list"*. Since the assistant is supposed to be used with a voice interface and having a voice assistant read a very long shopping list to you can quickly become annoying as it will take a long time, the list is organized in pages, each one will contain five entries. The user can ask the system to read a specific page, for example by saying *"Read me page three of my list"*. Based on the user input the system can add to the shopping list a single ingredient, the ongoing recipe (i.e. the user could plan to cook the same thing again, and therefore he/she would need the same ingredients) or any other recipe in the database. For the latter the model will guide the user to fill the required slots to select a recipe to add. The last action in this task is removing items from the list. When reading the grocery list the system will prefix each item with a progressive identification number, i.e. *"1 500 grams of pasta"*, then *"2 250 grams of butter"* and so on. The user can then specify the item to be removed by its ID, for example by saying *"Remove item two from my list"* to remove item two.

C. Chitchat and FAQs

This task is implemented in order to make the assistant able to handle some simple out of domain requests, such as *"How are you?"*, *"Tell me a joke"* or *"Are you a bot?"*. Chitchat intents are handled by grouping them under a single Retrieval Intent [2] which is then linked to the proper responses using a single rule [3]. The chitchat intents that the model can handle are just a few. However, adding more intents and responses is very easy and doesn't require to add more rules. This task also handles a few FAQs, questions the user can ask about the system own features. For example *"What can you do for me?"* or *"How do I use the shopping list?"*.

D. User Group

The target users are home cooks who like to cook for passion or just for necessity. The audience therefore are people of any gender in the age range of 16-80.

III. Conversation Design

The system can handle requests to start a new task, either selecting a recipe to cook or a recipe to add to the shopping list. In this case it will activate a form and guide the user to retrieve the necessary information: the desired recipe and the number of people the user is cooking for. Once the user has requested a recipe, the assistant will compare it to the recipes in the database using a simple string similarity metric to find the closest match. If no match is found, i.e. the user requested a recipe that is not present in the database, the system will try to recover by suggesting some recipes and ask the user to select another one. All the other requests are directives in which the users asks for an action to be performed, for example adding an ingredient to the list or to repeat a step, and the machine will perform the action and reply. This type interaction doesn't need a form as the required information is assumed to be in the directive. The user takes the initiative by directly selecting a recipe or requesting a recipe suggestion. Then the machine takes the initiative by querying the user to gather the information necessary to start a recipe. Once a recipe is selected, the system guides the conversation by describing a step and the user only has to notify the system when the current step is completed. However, even if a recipe is running the user can take the initiative at any time by starting another task, or asking the system to perform another action. For example while the cooking task is ongoing the user may ask to add another recipe's ingredients to the shopping list, starting a different task. The initiative is therefore mixed. The number of people to cook for, the number of steps to skip or backtrack, the shopping list page number, item IDs and ingredient amounts are all handled by a single entity `number`. This is because from a data point of view they are identical and this could confuse the model if they were handled as distinct entities, especially in a project like this where NLU data is scarce. A simple custom entity extractor allows the model to understand that sentences like *"Just for me"* and *"Skip this step"* that don't explicitly provide a number refer to the `number` entity of value 1. While selecting a recipe the system will guide the user to collect the necessary information. It will then ask for an explicit confirmation, for example: *"I understand you'd like to cook Carbonara for two people, is it right?"*. While an implicit confirmation is preferable for longer forms, an explicit confirmation is applicable in a case like this, where the required slots are just two. If the user replies that it's not correct, the system will make him/her fill the slots again. Alternatively the user can directly change her mind at any point. For example, assume that user selected Carbonara for two people and the system is asking for the explicit confirmation as discussed previously. She/he can change mind by saying *"Actually I'm going to cook Amatriciana"* to switch to Amatriciana from Carbonara as indicated previously. In this example, the system will remember the number of people to cook for, and switch to Carbonara without asking the number again. Adding a recipe's ingredients to the shopping list also activates a form to select a recipe to add but since it requires the same slots as the form to select a recipe to cook, it works exactly in the same way. The user may also require to stop the current recipe or even opt out of an ongoing form by saying *"Stop this recipe"* or just *"Stop this"*. The conversation is composed of user directives and machine guidance when the system requires information. Turn taking is therefore fairly simple, with the user and system replying to each other depending on the current task. To help novice users the assistant will suggest how to initiate some task, action or a question that the user can ask (a FAQ) to get information on how to use some functionality. For example, when replying to a greeting such as *"Hi!* the agent will reply with *"Hey! I'm your cooking assistant, you can ask me what I can do at any time!"* directing the user to ask what the system can do, discovering how to use the system. If the user asks to see his/her shopping list and it turns out to be empty the system will suggest how to add an item to it. When a recipe preparation starts and the system lists the ingredients, it will also tell the user that it is possible to ask it to repeat the ingredient list or the amount of a single ingredient and that the user can also skip steps or backtrack as desired. Similarly, when describing a recipe's step, the model will also ask the user to tell it when to proceed. Because in longer recipes the corresponding sentence *"Let me know when you are ready for the next step"* would be repeated several times, this might annoy the user. Therefore the assistant asks it only for the first three steps, then it will output just the step description as by now the user should know how to tell the machine to go on to the next step. The assistant also uses some conversation markers when running an action requested by the user, like *"Great"*, *"Ok"* and *"Sure"* followed by the output. In case the model is not able to understand the user's input it will trigger a fallback rule that will ask the user to rephrase. Similarly if the model is not able to predict the proper action to execute, i.e. the prediction confidence is too low, it will utter `utter_default` asking the user to rephrase. The model is also able to understand a limited set of out-of-scope questions that are not part of the domain the assistant operates in, for example *"What's the capital of Italy?"* and will reply by telling the user that it can't help with that question since it's only designed to help in cooking. Finally, the system is capable to handle two simple cases of ambiguous input. If the user activates the recipe selection form and tries to select two recipes at once or provides two different numbers of people to cook for at the same time, the system will tell the user that it can only handle one recipe at a time, or that only one number of people should be selected.

IV. Conversation Model

The assistant is built on top of the Rasa framework and uses its default suggested pipeline [4]. It employs Rasa's Memoization Policy [5], Rule Policy [6] and TED Policy [7]. The Memoization Policy remembers the stories from the training data and checks if the current conversation matches them. If so, it will predict the next action from the matching stories. The TED (Transformer Embedding Dialogue) Policy is a multi-task transformer [8] that recognizes entities and predicts the next action based on the conversation history. The Rule Policy handles conversation parts that follow a fixed behavior and makes predictions based on any rules in the training data. As described before, here rules are used to implement the fallback mechanisms and chitchat. The Fallback Classifier [9] classifies a message as `nlu_fallback` when the NLU confidence is too low, in this case the threshold is the default 0.3. A rule is then used to activate `utter_fallback` whenever `nlu_fallback` is predicted. Similarly, when a message is classified as out-of-scope, a rule activates `utter_out_of_scope`, telling the user that the assistant can't help with the query. As for chitchat, when a message is classified a such a rule activates the Response Selector [10] component, that predicts the appropriate response from a set of candidates. The assistant can handle 13 chitchat intents. The suggested NLU pipeline is composed of:

- Whitespace Tokenizer: Creates a token for every whitespace separated character sequence.
- Regex Featurizer: Creates a vector representation of user messages using regular expressions. It creates a list of regular expressions, then for each of them a feature will be set marking whether this expression was found in the user message or not.
- Count Vector Featurizer: Creates bag-of-words representation of user messages, intents, and responses.
- Dual Intent Entity Transformer (DIET) Classifier [11]: A multi-task architecture for intent classification and entity recognition.

The assistant can only run in a terminal and be used via text. Ideally it would be used via voice and it would run through a custom Alexa skill in an Alexa-enabled device, or any other voice-enabled device such as Google Home, but I didn't have any such device at my disposal. If it were to be further developed to be integrated with such devices, some minor changes might have to be made. For example in Alexa systems the `stop` intent would be unnecessary as all Alexa skills already have a similar intent, `AMAZON.StopIntent` [12], by default.

V. Data Description and Analysis

Collecting data for this project was fairly easy thanks to Rasa Interactive [13] which allows to interact with assistant while annotating inputs, then saves the conversation as a story and its NLU data. Rasa also allows to set checkpoints in stories which connect any story ending in a checkpoint to any story starting from the same checkpoint. This allows to easily fork and combine different story blocks, making writing longer stories much easier. Using this technique 135 stories were collected. The NLU data is composed of 20 distinct intents with an average of 33.7 examples each, for a total of 674 examples. Synonyms are employed to map spoken numbers like *"two hundreds"* to their numerical values. The system uses eight slots and four entities: the recipe, the number of people to cook for, ingredients and ingredient's unit, for example grams, kilos and so on. The recipes database was hand-written just for this project and it only contains four recipes, with an average of 4.75 ingredients and 10.75 steps per recipe. Since the project only serves as a proof of concept, the database is encoded directly in a Python file, in a format similar to JSON composed of nested dictionaries: the database is a dictionary where each key is a recipe name, and each entry is a recipe. Each recipe is itself a dictionary that contains a list of steps and a list of ingredients. Each ingredient is a dictionary containing the name, amount and unit of the represented ingredient. Each step is a dictionary containing the step description and a list of numbers pointing to the ingredient list entries corresponding to the ingredients used in the step.

VI. Evaluation

The system was evaluated using both intrinsic evaluation and extrinsic evaluation. The intrinsic evaluation is integrated in Rasa, while for the extrinsic one a human evaluation was performed.

A. Intrinsic Evaluation

NLU intrinsic evaluations were carried out using the K-fold cross validation technique for more accurate results, with k set to 5. For this system's NLU pipeline Rasa's evaluation procedure [14] computes accuracy, F1 score and precision for the intent classification, entity extraction and response selection tasks. Tables I, II and III show the results of intent classification, entity extraction and response selection (chitchat) respectively. Figures 1, 2, and 3 show the intent, entities and response selector confusion matrices respectively. Additionally, Rasa tests the assistant with test stories and computes the accuracy at story level (i.e. how many stories are completed correctly) and at action level (i.e. how many times the TED policy predicted the correct action). Both accuracies are 1, as all test stories are completed correctly. As can be seen from the evaluation results, the assistant performs fairly well. However it's worth noting that the assistant was both trained and evaluated on conversations written by me, not on real conversations collected from human subjects. Since I knew how the assistant was supposed to work while I was writing training and test stories, the evaluation is inherently biased. Additionally, although the model achieves good scores, from the confidence distributions of the three NLU tasks, shown in figures 4, 5, and 6, can be observed that the system does some mistakes with high confidence. This means that the NLU data should be improved and expanded to reduce the mistakes the system does.

	Accuracy	**F1 Score**	**Precision**
Train	1	1	1
Test	0.960	0.957	0.959

Table I
INTENT EVALUATION RESULTS

	Accuracy	**F1 Score**	**Precision**
Train	1	0.999	0.999
Test	0.996	0.983	0.992

Table II
ENTITY EXTRACTOR EVALUATION RESULTS

	Accuracy	**F1 Score**	**Precision**
Train	0.984	0.984	0.985
Test	0.804	0.790	0.815

Table III
RESPONSE SELECTION (CHITCHAT) EVALUATION RESULTS

B. Extrinsic Evaluation

Human evaluation is very important in the evaluation of AI assistants since it allows to observe how the system performs when used by the end user in a real world scenario. For this project the assistant was tested by three people for as long as they'd liked. They were then asked to answer seven questions, rating from 0 to 10 their experience while using the assistant. The results are shown in table IV. The results are relatively good, but some of the subjects used sentences that were never seen by the assistant during training and it wasn't able to generalize to them. This is also reflected on the relatively low rate on the ability of the system to understand the user as shown on the table. It also doesn't perform well in the chitchat task but that's to be expected since it is handled with simple rules and the chitchat queries it is designed to understand are just a few.

VII. Conclusion

The assistant developed in this work is simple but can successfully manage a shopping list and help the user

Question	**Rate (0 to 10)**
Rate the system's ability to help you follow a recipe	8
Rate the system's ability to help you manage a shopping list	8,34
Rate the system's ability to do chitchat conversation	5,84
Rate how close is this conversation to a conversation with a human	6,84
Rate the speed of the system in solving the task in comparison to a human	8
Rate the system's ability to understand you	6,5
Were the system responses clear and helpful?	7,5

Table IV
HUMAN EVALUATION RESULTS

follow a recipe step by step. It performs quite well as long as the user sticks to sentences and conversations that appear in the training set, but it fails to generalize to unseen data. It could be improved by letting many more users use it and collecting the conversations in order to make both NLU data and training stories much larger and more diverse. It could also be possible to employ pre-trained components in the NLU pipeline, such as spaCy's [15] tokenizer, featurizer and entity extractor, which could lead to better performance since they were pre-trained on a much larger volume of data than what was used in this project.

REFERENCES

[1] "Rasa framework." https://rasa.com/.
[2] "Rasa chitchat and faqs." https://rasa.com/docs/rasa/2.x/chitchat-faqs/.
[3] "Rasa rules." https://rasa.com/docs/rasa/2.x/rules.
[4] "Suggested config." https://rasa.com/docs/rasa/model-configuration/#suggested-config.
[5] "Rasa memoization policy." https://rasa.com/docs/rasa/policies/#memoization-policy.
[6] "Rasa rule policy." https://rasa.com/docs/rasa/policies/#rule-policy.
[7] "Rasa ted policy." https://rasa.com/docs/rasa/policies/#ted-policy.
[8] V. Vlasov, J. E. M. Mosig, and A. Nichol, "Dialogue transformers," 2019.
[9] "Fallback classifier." https://rasa.com/docs/rasa/components/#fallbackclassifier.
[10] "Response selector." https://rasa.com/docs/rasa/components#responseselector.
[11] T. Bunk, D. Varshneya, V. Vlasov, and A. Nichol, "Diet: Lightweight language understanding for dialogue systems," 2020.
[12] "Connecting rasa to alexa." https://rasa.com/blog/connect-your-rasa-ai-assistant-to-amazon-alexa/.
[13] "Rasa interactive." https://rasa.com/docs/rasa/2.x/command-line-interface/#rasa-interactive.
[14] "Testing rasa assistants." https://rasa.com/docs/rasa/2.x/testing-your-assistant/.
[15] "Spacy nlp." https://spacy.io/.

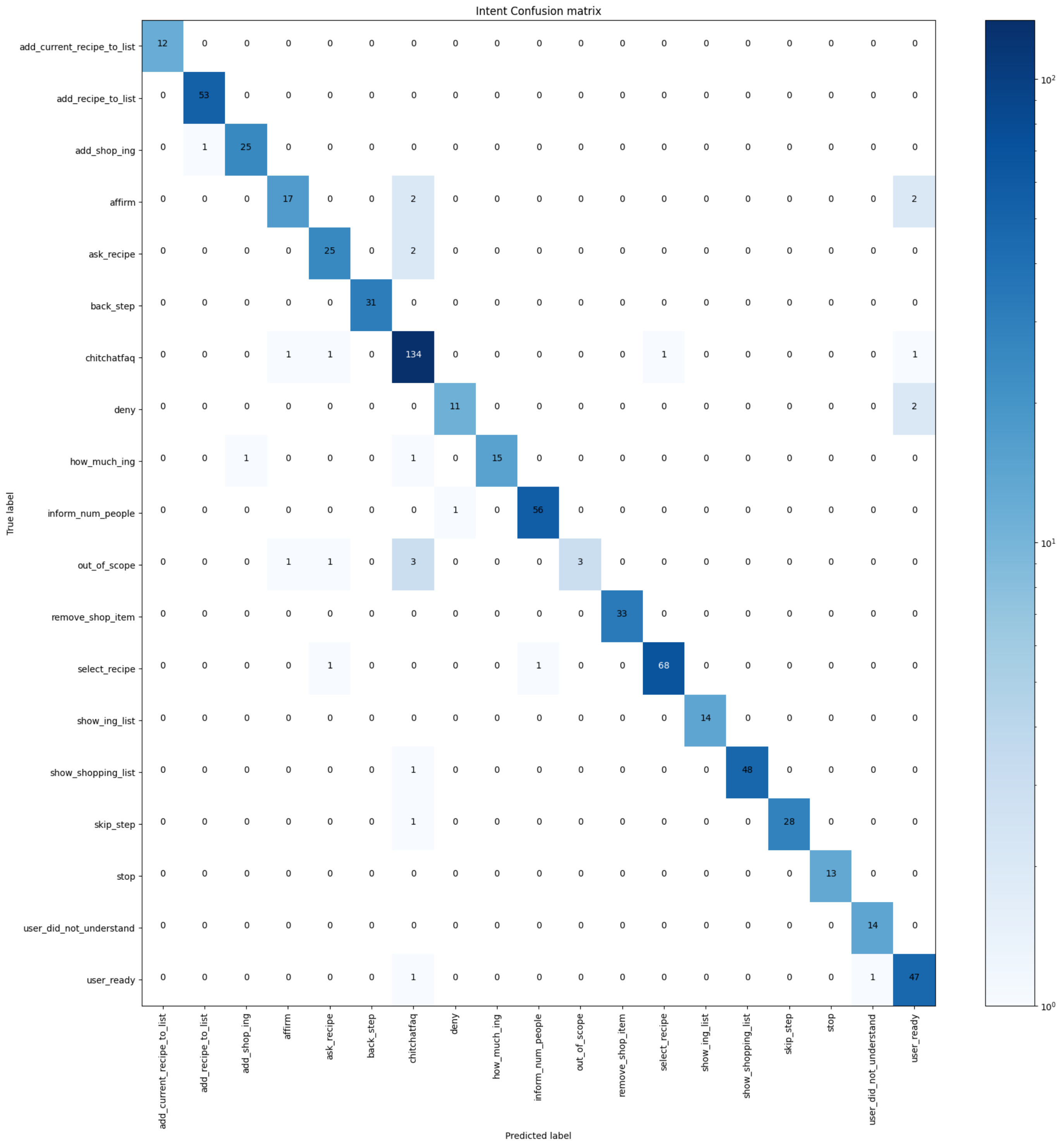

Figure 1. Intent confusion matrix

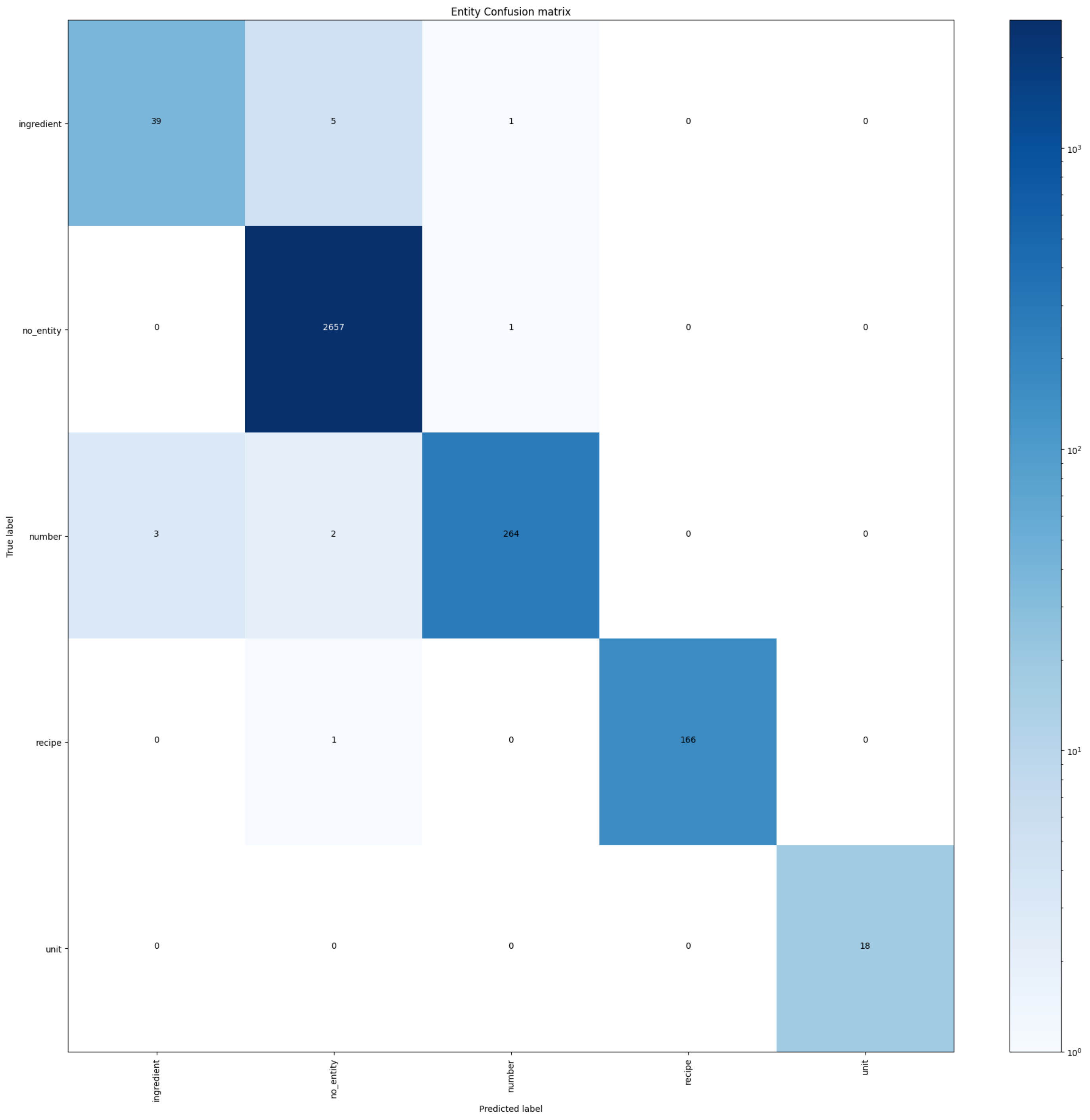

Figure 2. Entity confusion matrix

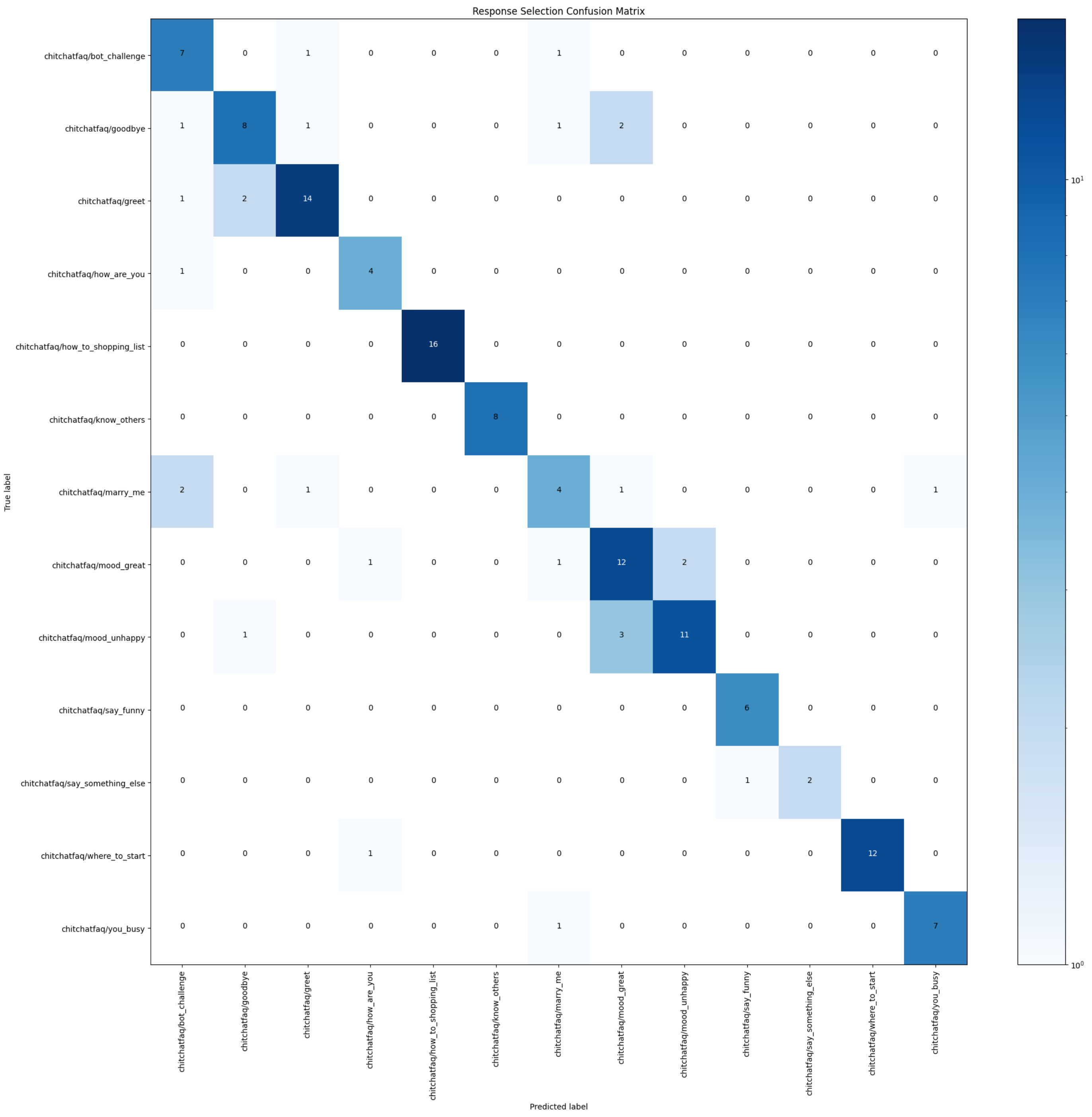

Figure 3. Response selection (chitchat) confusion matrix

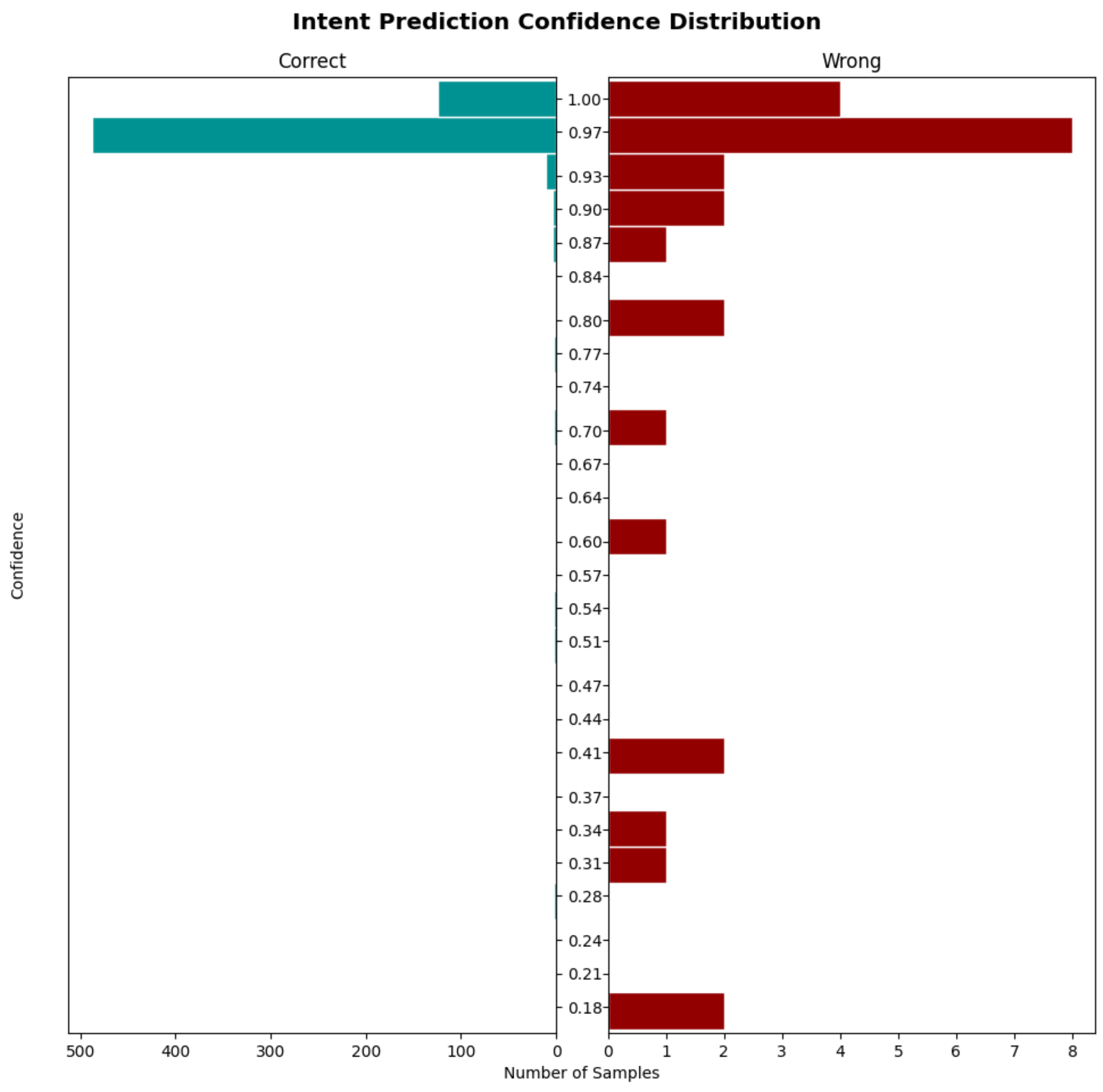

Figure 4. Intent prediction confidence histogram

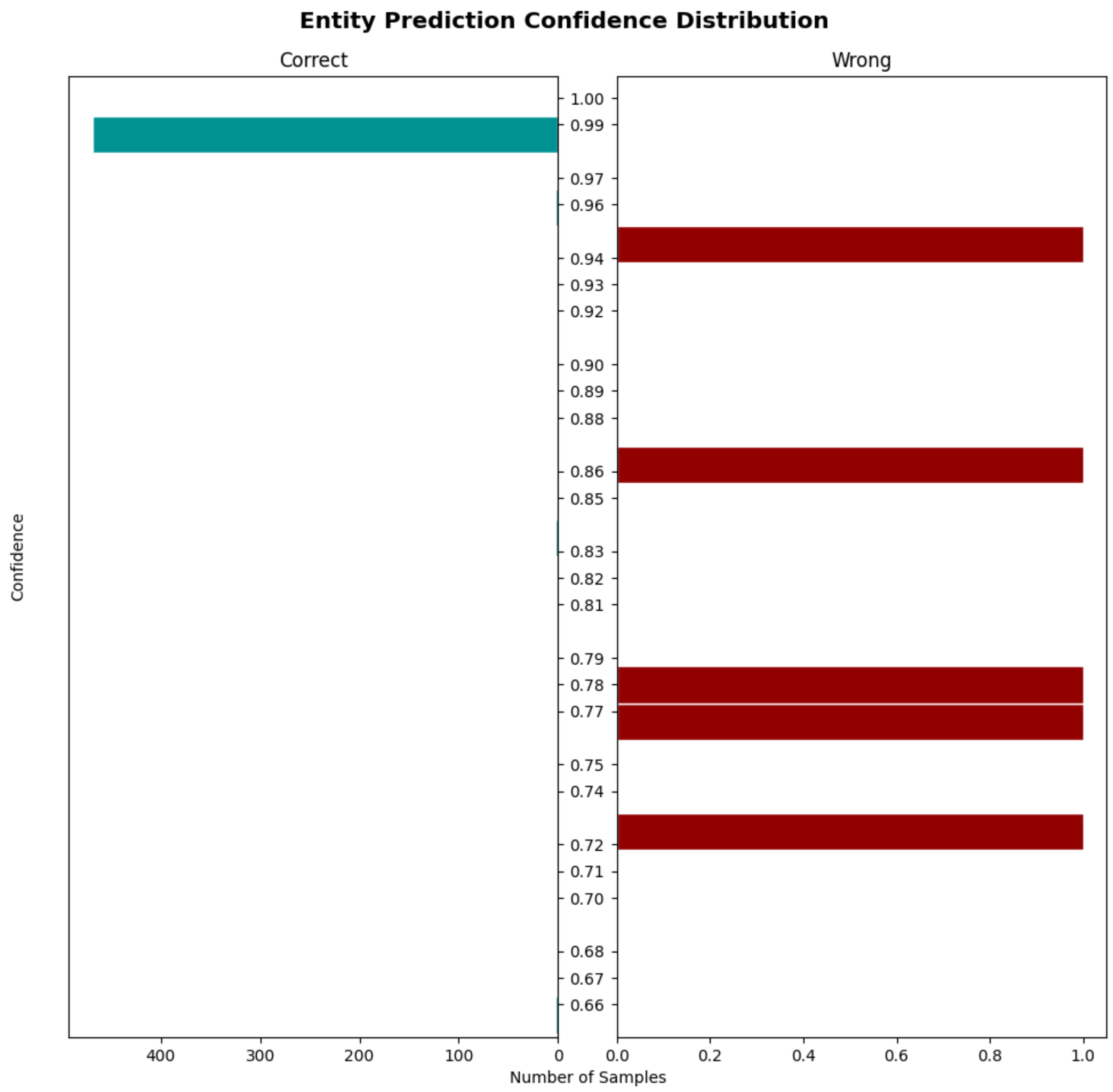

Figure 5. Entity prediction confidence histogram

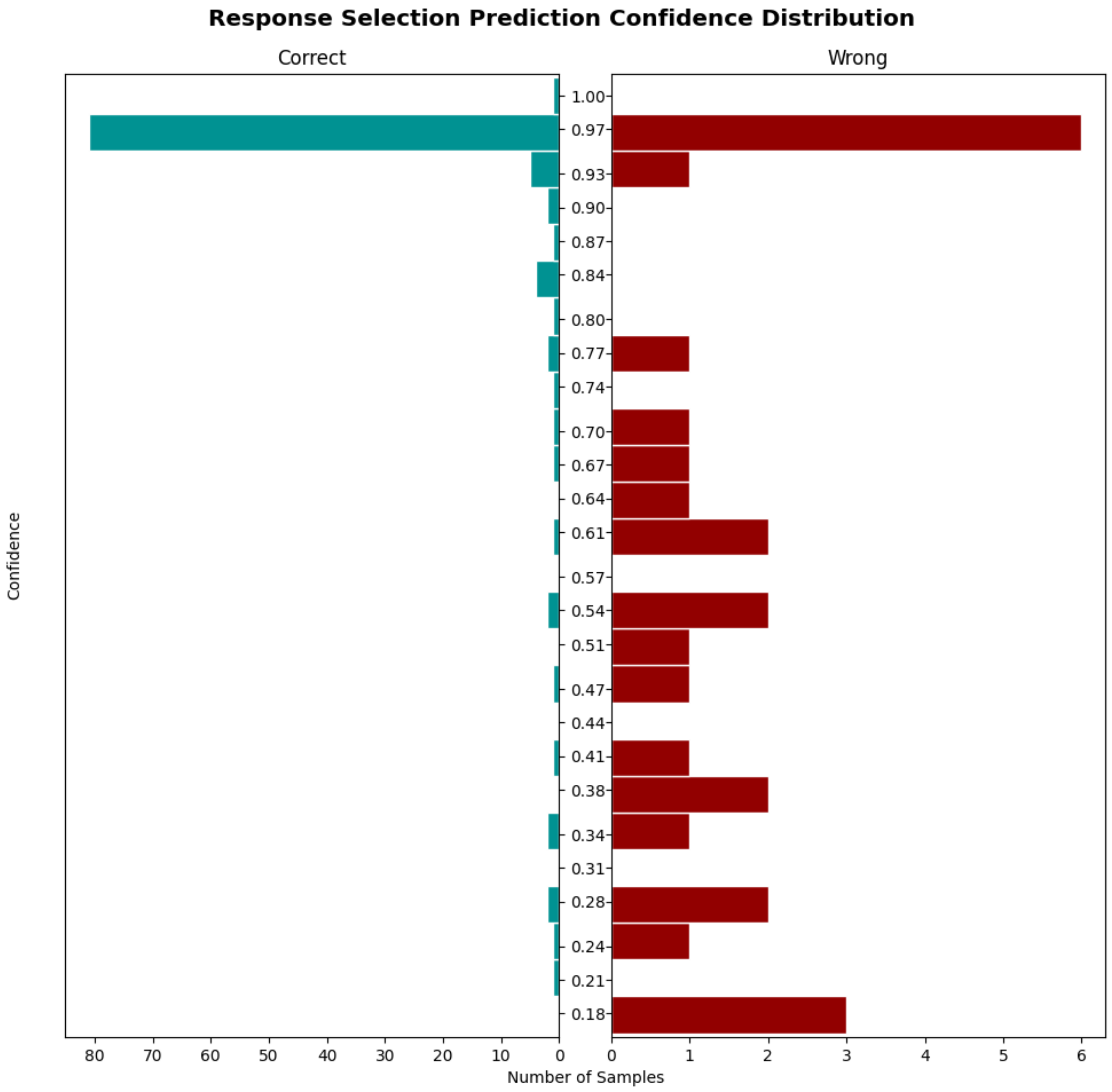

Figure 6. Response selection (chitchat) confidence histogram

Appendix

The slots that the systems makes use of are:

- *recipe_key, num_people, step_idx*: These represent the current ongoing recipe, they are the current recipe, the number of people to cook for, and the current step
- *recipe, number*: These represent the recipe and the number of people that the user selects in a form. They will then be saved in *recipe_key* and *num_people* if the user confirms. The user confirmation is saved in *confirm_recipe_form*. This means that the system can fill the slots with the user selection, but if the user doesn't confirm, the previously running recipe can continue as the new recipe didn't yet overwrite the old one.
- *number*: this represents the number of people to cook for, the amount of an ingredient to add to the shopping list, the page of the shopping list to visualize, and number of steps to skip or backtrack. *ingredient, unit*: The first represents an ingredient name when the user asks about an ingredient amount or wants to add it to the list. The second represents the measure unit of the amount, for example grams or kilos.

The entities are *recipe, number, ingredient, unit* and they have the same meaning of the corresponding slots of the same name.