Human-Machine Dialogue Project: The Champ's Pizza Hut Assistant

Adnan Irshad **(241763)**
University of Trento
Trento, Italy
adnan.irshad@studenti.unitn.it

I. Introduction

This work presents a RASA-based voice assistant to improve human-machine interaction that streamlines the pizza ordering and pickup process, providing users with a seamless and engaging experience. The dialogue system referred to as "Champ's Pizza Hut Assistant" is designed to handle various tasks related to pizza ordering, especially with a focus on pickup orders including presenting menu information, offering recommendations, assisting with topping customization, and managing the order pickup. By leveraging natural language understanding (NLU) and AI technologies, the dialogue system enables users to interact with the system in a manner that closely mimics human conversation, enhancing the overall customer experience.

The Champ's Pizza Hut Assistant is a task-based system, primarily focusing on three major areas:

First task of Menu Information involves providing detailed descriptions of available pizzas, toppings, and sizes, as well as offering recommendations based on popular choices from the customers. This feature allows customers to make informed decisions and discover new pizza combinations that align with their preferences.

Second topping Customization task, offers users the flexibility to modify their pizza according to their tastes. It facilitates the addition and removal of toppings in the standard toppings enabling customers to create a pizza that is entirely personalized.

Third task of Order Placement, is managed through the `pizza_order_form` and `order_pickup_form` together, where it assists users in selecting pizza type, toppings either standard or customized ones, and quantity. It confirms the user's choices, allows for additional orders or modifications. Once the order is confirmed, the assistant transitions to handling the pickup process which is handled by `order_pickup_form`. Lastly pickup form, calculates the preparation time for the order and helps the customer set a suitable pickup time and ensures time falls within the restaurant's operating hours and meets all other criteria, such as being after the preparation time. The system also handles changes to the pickup time and generates an order ID upon final confirmation.

II. Conversation Design

A. *Conversations*

Assistant handles a variety of conversation types, each tailored to different stages of the pizza ordering process. These conversations are meticulously structured in the stories.yml file, which defines the expected flow for each type of interaction. The primary conversation types include:

- **Menu Inquiry and Recommendations:** Users can request detailed information about the available pizzas, toppings, sizes, and prices. The assistant can also offer recommendations based on popular or frequently ordered items.
- **Pizza Selection, toppings, size, quantity:** Once the user has decided on a pizza, the assistant helps them select the type, size, and toppings. Once the customer selects the `pizza_type` with intent `inform_pizza_type`, the pizza order form is activated and it fetches the selected pizza's standard toppings and asks customers for their preferences either to go with these standard toppings or would like to customize. This process is interactive, allowing users to decide if a customer says he wants to customize then `pizza_order_form` is deactivated the `pizza_custom_topping_form` is activated which fetches the available custom toppings from the defined dictionary and shows to the customer. This conversation allows customers to make custom preferences by adding or removing toppings as their own. Once the customer is happy with their customization, the form is deactivated and again `pizza_order_form` becomes activated which takes the customer's custom toppings and updates them with selected pizza toppings and goes to the next slots which are `pizza_size and pizza_quantity`. This form-driven approach ensures that all necessary information is collected before proceeding to the next step.
- **Order Confirmation:** After the selection process, the assistant confirms the order details with the customer, allowing for any necessary modifications. Like customer can request to add another pizza with `add_another_pizza_general` intent or a customer can directly request that

"wants to add another Margarita pizza" with `add_another_pizza_specific` intent. In this case, custom action adds the currently selected order into `order_list` and reactivates the pizza order form for the second order. Also, the customer can remove an item/pizza from his with `remove_pizza_from_order` intent or he can also cancel the entire order to start the order process or say goodbye.

- **Pickup Arrangement:** After the order is confirmed, the assistant assists in scheduling the pickup time. This is managed through the `order_pickup_form`, ensuring that the user's preferred pickup time is validated against the restaurant's operating hours and order preparation time.

These conversation types are designed to transition smoothly between each other, maintaining context and continuity throughout the interaction.

B. Terminating loops and Conversations

The assistant is programmed to terminate conversations or loops when the user's task is complete, or when the user explicitly indicates that they wish to end the interaction. This is managed using the stories, rules, and custom actions defined in stories.yml, rules.yml, and actions.py.

To handle the termination of conversations, the assistant incorporates mechanisms for recognizing user intents to end the interaction, such as when a user decides to cancel an order or simply says goodbye. The possible ways to end the conversation are as follows:

- The user can end the conversation at any time by saying 'goodbye' or 'stop' with intent end_greet and `no_longer_hungry_stop_conversation`.
- If the user decides to cancel the order and goodbye interactions together, intents like `end_greet` triggered, which gracefully terminates the ongoing loop and ends the conversation.

C. Interaction Properties

The system is mixed-initiative, as both the user and the system drive the conversation. The assistant is designed to balance between taking the initiative and allowing the user to drive the conversation. For instance, after presenting the menu, the assistant might prompt the user with, "Which pizza would you like to order?". Also, if the assistant asks "Would you like to go with Standard toppings or Custom Toppings". Customers can take the initiative from the assistant and instead of answering the assistant's question, customers can ask about the toppings information like "What toppings do you offer in customization" or "When do you open" by taking control over the assistant.

One of the key properties is context maintenance, where the assistant remembers previous user inputs and uses this information to guide the conversation when a customer takes control. For example, during the pickup form, when the assistant asked for the customer's pickup time, and in response to this question, the customer made a counter question "What is your closing time?". The assistant handles this type of question by providing information and also repeating the same question which the assistant asked previously e.g. "We are open from 12 PM to 11:00 PM. When do you want to pick up your order". This allows users to switch topics or revisit previous points in the conversation without confusion. The assistant is also designed to be polite and engaging, using phrases that make the interaction feel friendly and personalized.

D. Acknowledgement and Ground-taking

Acknowledgement is a key aspect of effective communication, and the assistant is programmed to acknowledge user inputs by confirming actions and repeating key information. For instance, when a user selects a pizza, the assistant responds with, "You've chosen a Seafood pizza with Shrimp, Mussels, and Clams." Even during the forms, when a customer selects a pizza or size or toppings, before asking next slot, the assistant responds with acknowledgement sentences or words e.g. "Nice choice, we have added Seafood" or "Got it"

E. Errors Recovery and Ballback Responses

The assistant includes fallback responses for situations where the user's intent is unclear or when the assistant fails to recognize an input. For instance, if the user says something ambiguous like "I'll pick at midnight" the assistant responds with, "I'm sorry, I couldn't understand the time you provided. Could you please specify the exact pickup time again?". In cases where the assistant encounters an unrecognized or out-of-scope query, it employs a fallback mechanism. The assistant is equipped with fallback mechanisms, as defined in the rules.yml file and a custom action `action_default_fallback` in action.py, which triggers when the user's intent is unclear or when the input does not match any predefined intents.

An "out-of-scope" intent has been created to manage inquiries about topics beyond the bot's capabilities. Additionally, a fallback classifier is included in the NLU pipeline, which triggers the `nlu_fallback` intent when the confidence level of all other intent predictions falls below a predefined threshold, empirically set at 0.5.

III. Data description and Analysis

A. Dialogue Data

Dialogue Data is encapsulated in the stories.yml and rules.yml files, which define various interaction scenarios that the assistant might encounter during user interactions. The dialogue data is structured to handle a wide range of user interactions, from basic

greetings and inquiries to complex order modifications and customizations. The stories.yml file captures the sequence of intents and actions that represent different conversation flows. There were a total of 48 stories. The rules.yml file is used to enforce specific conversational 54 rules that need to be adhered to, such as mandatory questions or responses in certain scenarios

B. Training data

The training data is crucial for developing the Natural Language Understanding (NLU) model, which allows the assistant to classify user intents and extract relevant entities from conversations. The data is organized in the domain.yml and nlu.yml file. The domain.yml file encompasses the specific entities, intents, slots, and actions that are relevant to the pizza ordering process. The nlu.yml takes the intents and entities and provides the annotated examples of user inputs are provided for each intent and entity. The NLU model is trained on this data to accurately recognize user intents and extract relevant entities from their inputs. The assistant trained to recognize 38 different intents with 6 entities. To improve the flexibility of the assistant, synonyms for various entities are defined, allowing the assistant to recognize different user phrasings. A total 14 Slots such as `pizza_type` used to capture and store user preferences throughout the conversation. A total of 27 predefined responses that the assistant uses to communicate with customers. The domain.yml file also lists 48 actions that are used to train the NLU model including both standard responses and custom actions defined in the `actions.py` file. Three forms are used to collect multiple pieces of information sequentially.

C. Domain data

Assistant's domain data is managed primarily through the use of Python dictionaries. These dictionaries represent various elements of the system, such as pizza types, sizes, toppings, and their respective prices. They play a crucial role in validating user inputs during interactions, ensuring that customers can only select from available options and that all selections are accurate and consistent. The dictionaries also facilitate the dynamic generation of menus, allowing the system to present customers with up-to-date information about the available pizzas, including details like standard toppings and prices for different sizes. By leveraging Python dictionaries, the system can quickly respond to user requests, making the ordering process smooth and efficient.

D. Database

The database component of the assistant is implemented using a relational database, managed through SQLAlchemy and SQLite. This database plays a crucial role in the pizza recommendation feature of the pizza assistant by serving as the central repository for all customer orders. Every time an order is placed, detailed information about the order—such as the type of pizza, toppings selected, quantity, and size—is stored in the database. This historical data forms the basis for the recommendation system.

When generating recommendations, the system queries the database to retrieve past orders. It processes this data to identify trends, such as which pizzas and toppings are most popular among customers. By analyzing the frequency and combination of orders, the system can determine the most favoured vegetarian and non-vegetarian pizzas. This allows the pizza assistant to recommend the top choices to customers, tailoring suggestions based on collective customer preferences.

IV. Conversation Model

A. Model

The Champ's Pizza Hut Assistant was developed using the RASA framework, which integrates with the Alexa Voice Assistant to provide a voice-enabled, AI-driven experience. This framework is structured around two core components: Natural Language Understanding (NLU) and Dialogue Management (DM). The NLU component plays a critical role in interpreting user inputs by identifying the user's intent, extracting relevant details, and determining the most suitable responses. On the other hand, the DM component orchestrates the flow of the conversation. RASA includes an action server that enables the execution of custom actions, such as processing orders or generating recommendations. In RASA, trackers allows the assistant to consider previous user interactions when crafting responses, thereby offering a more contextually relevant

B. Pipeline

Pipeline is built using several configurations to optimize intent classification and entity recognition. Each pipeline designed to effectively process user inputs, extract the necessary information, and generate appropriate responses. Four configurations were tested, each containing a combination of tokenizers, featurizers, and classifiers, tailored to maximize the assistant's ability to understand and respond to user inputs.

Major pipelines components that were used in configurations are: **WhitespaceTokenizer:** splits the user's text into individual words or tokens based on whitespace. **CountVectorsFeaturizer:** creates a numerical representation of the customer's text by generating character-level n-grams (1 to 4 grams). **LogisticRegressionClassifier:** machine-learning model used to classify the customer's intent based on the features generated by the CountVectorsFeaturizer. **CRFEntityExtractor:** identifies and extracts named entities from the customer's text, such as pizza size, toppings, or type. It is crucial to understand the specific details of the customer's request. **ResponseSelector:** selects the most appropriate response from a set of candidates based on the customer's input and context. **SpacyNLP:** with SpacyTokenizer and SpacyFeaturizer leverages

pre-trained word vectors enhancing the assistant's ability to handle diverse linguistic inputs. **DIETClassifier:** serves a dual purpose by both classifying intents and extracting entities, thus simplifying the pipeline architecture while maintaining high performance.

C. Policies

The policies in helps the decision-making process of assistant, determining how it should respond at each step of the conversation. Each configuration tested during the development process includes a mix of policies that balance memorization of training data with the ability to apply rules and predict actions based on the context of the conversation.

TEDPolicy: Leverages transformer-based embeddings to retain conversational context and predict the next action. This policy is particularly effective in handling complex dialogues where maintaining the flow of the conversation is critical. The `max_history` parameter is set to 20 from allowing the policy to consider up to 20 previous actions or inputs and this changes leads best performance in **config.yml** and **config-2.yml**. **AugmentedMemoizationPolicy:** improves upon standard memorization techniques by considering a broader context of (`max_history: 4`), ensuring the assistant's responses are consistent with previous interactions. **MemoizationPolicy:** Similar to the AugmentedMemoizationPolicy, this policy memorizes actions but for a shorter history length. It provides a fallback mechanism for situations where the bot needs to recall recent interactions quickly. In **config-4.yml**, followed it and it slightly improves the results. **RulePolicy:** employed to enforce specific conversational patterns, such as handling fallback scenarios when the assistant encounters uncertain or ambiguous user input.

D. Configuration Alexa

Integration with Alexa was achieved through a custom connector, alexa_connector.py with **ngrok** which facilitates communication between the RASA framework and the Alexa Voice Assistant. This connector handles incoming voice commands by converting them into text, which is then processed by RASA. The assistant's responses are sent back to Alexa, which converts them into speech for the user.

V. Evaluation

A. Intrinsic

The evaluation of assistant involved a series of experiments (approx. 7) designed to test and optimize the performance of the assistant across four configurations. These experiments focused on intrinsic metrics, such as accuracy and F1-score on test stories. There were total 30 test stories separate from training stories. The evaluation process was structured around four different configurations, each tailored to optimize specific aspects of the assistant's NLU and DM capabilities. Below is the brief detail of each config and experiment and best results are reported in table I and table II.

- **Experiment 1 to 3: Config.yml**: This configuration is a combination of the `WhitespaceTokenizer`, `CountVectorsFeaturizer`, LogisticRegressionClassifier, and CRFEntityExtractor. The TEDPolicy with a `max_history` of 20 was employed to maintain conversational context, alongside the `AugmentedMemoizationPolicy` and RulePolicy for enhanced dialogue management. Initially `max_history=8` and `TDD opochs=20` leading results around 70% then refining the `max_history=20` and `TED epochs=100` leads to best performance of conversation Level to **93.3%**.
- **Experiment 4:** adjustments were made in **config-2.yml** to improve the model's performance, particularly in intent classification and entity extraction. `LogisticRegressionClassifier` was fine-tuned with an increased `max_iter` of 2000, and `ResponseSelector` was configured with a higher max_iterations of 100,000 to enhance response selection accuracy. Conversation Level performance remains same but there was a slight increase in action Level performance.
- **Experiments 4 and 5:** utilized config-3.yml, which introduced the `SpacyNLP`, `SpacyTokenizer`, `SpacyFeaturizer`, and `DIETClassifier`. This configuration aimed to leverage pre-trained word vectors and a more advanced intent and entity recognition model to enhance the assistant's understanding of user inputs.
- **Experiment 6 and 7:** final set of experiments was conducted with config-4.yml, which continued to utilize SpacyNLP but reverted to a slightly simpler entity recognition model with the CRFEntityExtractor. The DIETClassifier was retained, but its entity recognition capabilities were disabled to reduce complexity. As compared to previous experiments 4 and 5, its accuracy improved to **90%**.

TABLE I: Action Level Results

Experiment No.	Correct	F1-Score	Precision	Accuracy
0 to 2	169/174	0.983	0.993	0.977
3	170/174	0.984	0.990	0.981
4 to 5	159/174	0.932	0.950	0.919
6 to 7	163/174	0.953	0.954	0.936

B. Extrinsic

The extrinsic evaluation involved deploying the assistant in a live environment, where users interacted with it over a designated period, primarily through command-line mode. User feedback was gathered to

TABLE II: Conversation Level Results

Experiment No.	Correct	Accuracy
0 to 2	28/30	0.933
3	28/30	0.933
4 to 5	26/30	0.86
6 to 7	27/30	0.90

assess satisfaction levels and to rate the assistant's performance on specific tasks. The feedback, captured through a set of six questions (detailed below in table III), provided insights into the assistant's real-world effectiveness. The results indicated that while the assistant performed well in certain areas, there was a noticeable gap between its intrinsic performance metrics and its actual performance in live interactions. This discrepancy suggests a need for further refinement, particularly through the collection of more diverse data from real users.

TABLE III: Human Evaluation Results

Question	Average Rating
Ability to accurately understand and complete your requested task.	6.8
Effectiveness of the assistant in handling casual, non-task-related conversation.	5.5
Efficiency and clarity in helping you to customize your toppings.	8.5
Ease and accuracy of placing an order through the assistant.	7.2
How closely does this interaction resemble a conversation with a human?	6.0
Speed at which the assistant completes tasks compared to a human.	8.5

CONCLUSION

In this project, I have developed and evaluated a voice-enabled AI-driven assistant for The Champ's Pizza Hut using the RASA framework integrated with Alexa Voice Assistant. The assistant was designed to manage various tasks, including pizza ordering and pickup, customization, and providing recommendations. Through extensive experimentation with multiple configurations, the assistant's ability to classify intents, extract entities, and manage dialogues was measured.

The intrinsic evaluation demonstrated that the assistant performs well in controlled environments, with high accuracy, precision, and F1-scores across multiple test cases. However, the extrinsic evaluation highlighted areas for improvement, particularly in real-world interactions where user feedback revealed the need for better customization options and enhanced conversational abilities.

APPENDIX

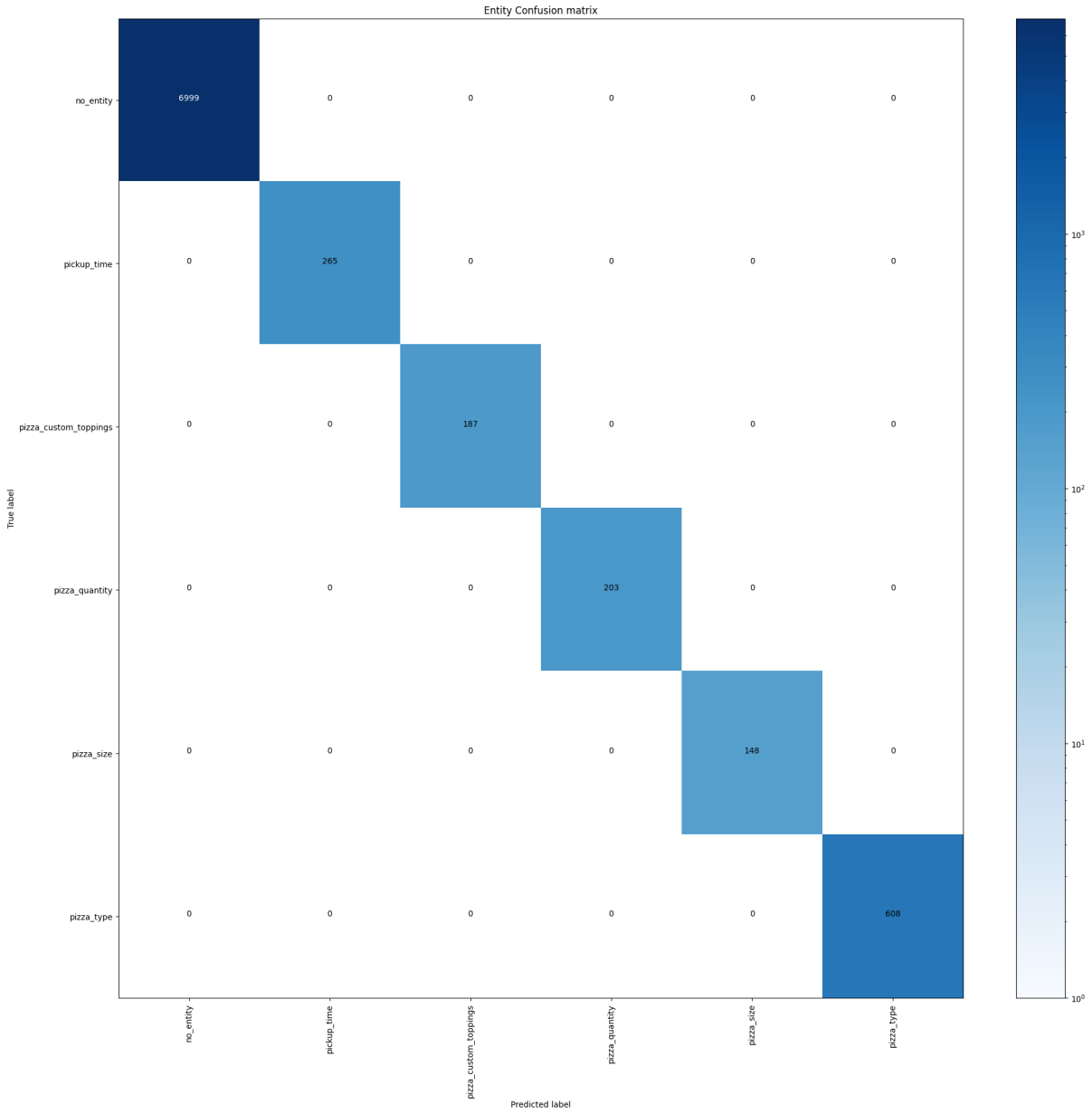


Fig. 1: Config-1: CRF-Entity Confusion Matrix

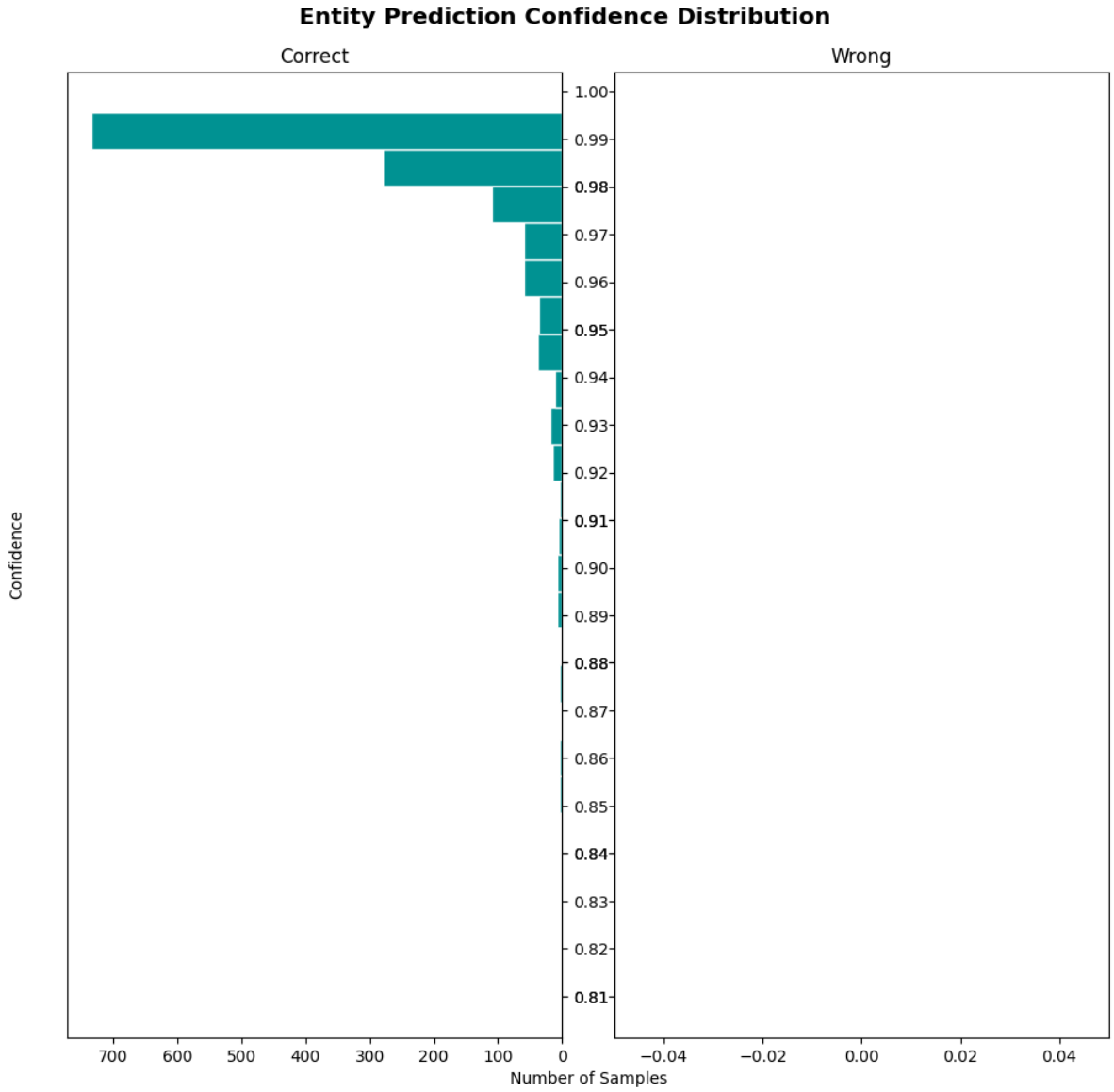


Fig. 2: Config-1: CRF-EntityExtractor Histogram

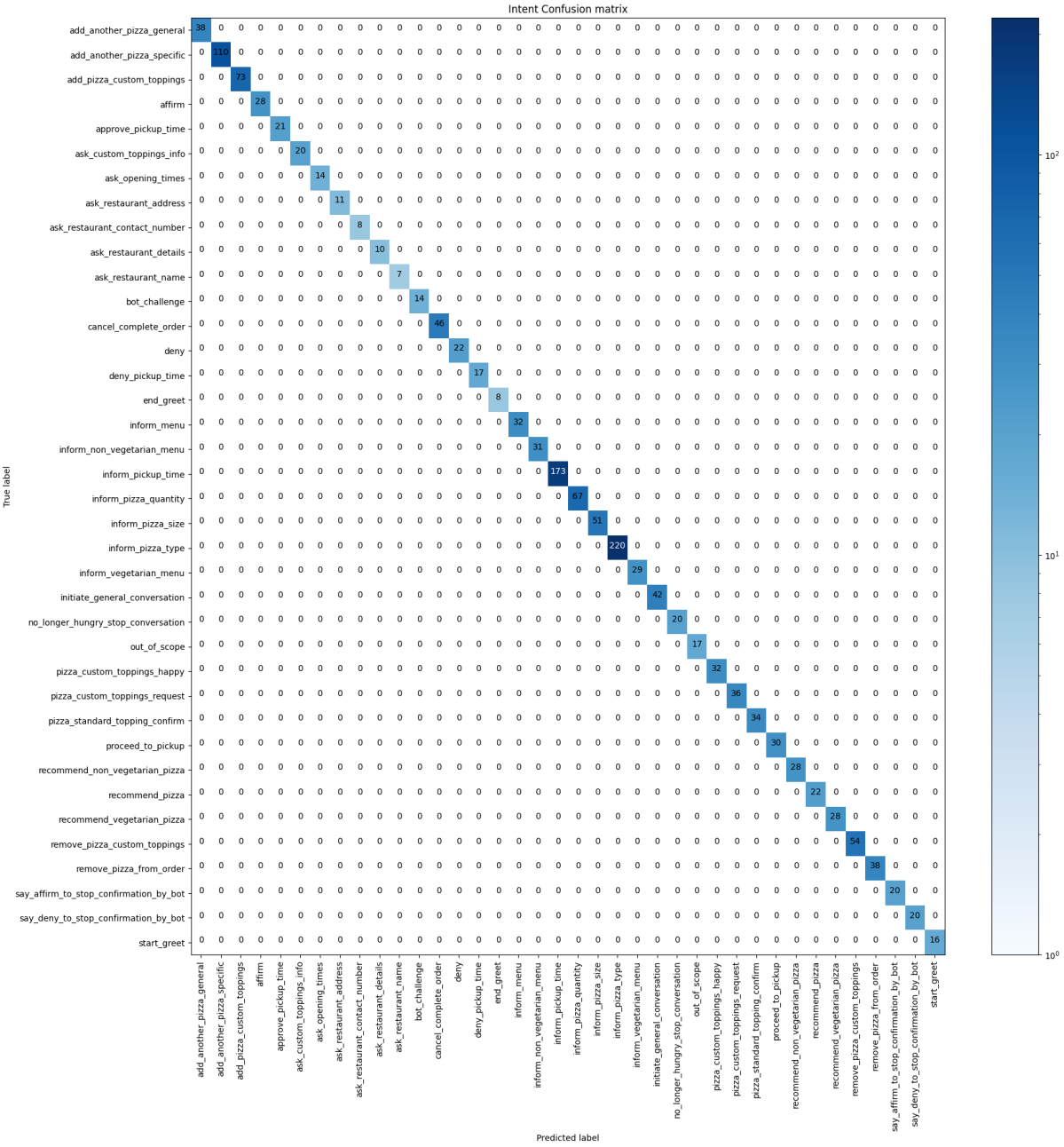


Fig. 3: config-1 Intent Confusion Matrix

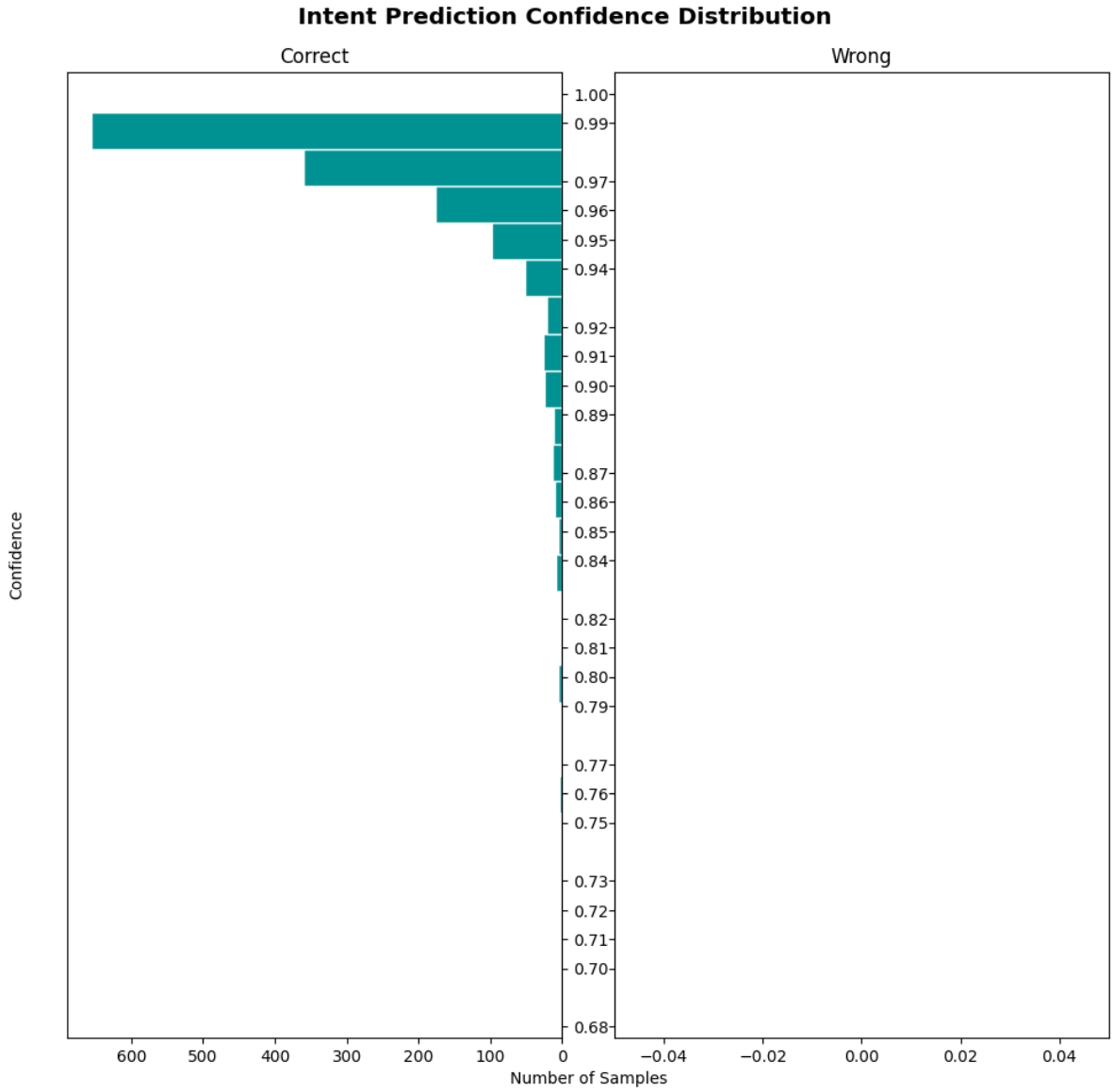


Fig. 4: Config-1: Intent histogram

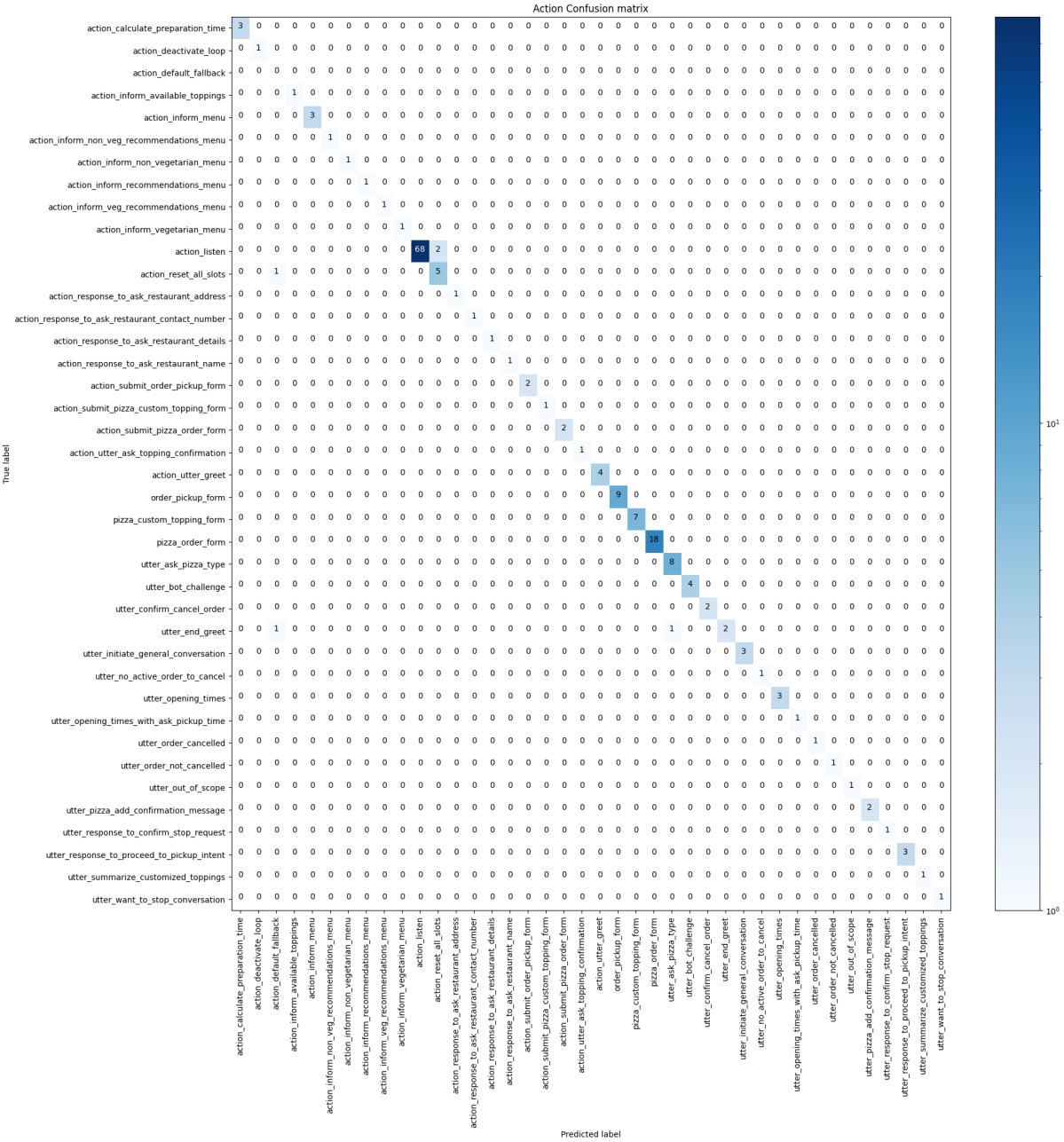
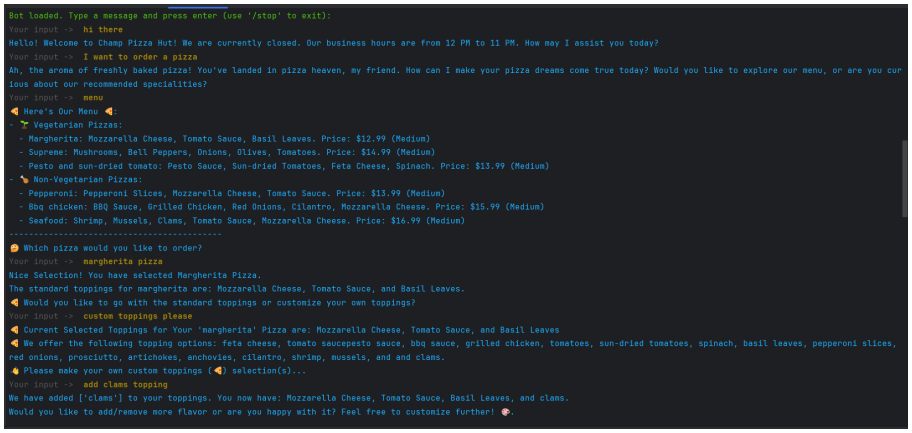
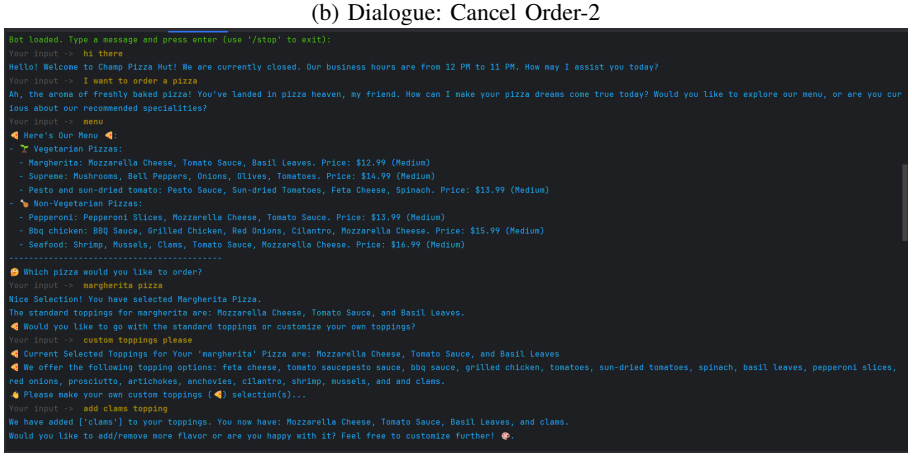


Fig. 5: Config-1: Story Confusion Matrix



(a) Dialogue: Cancel Order-1

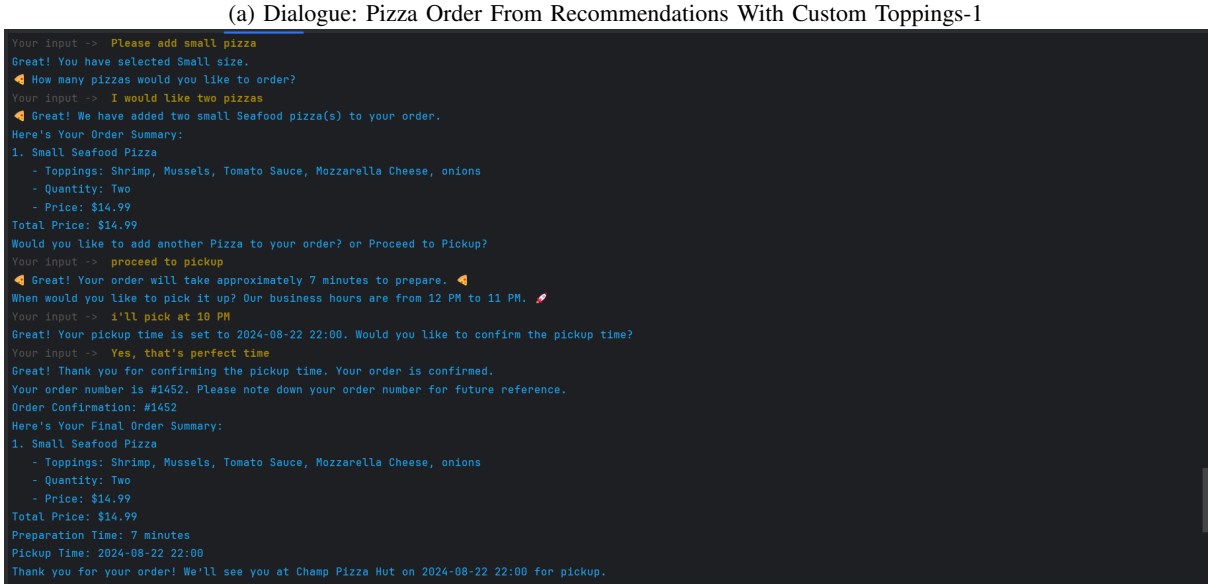
(b) Dialogue: Cancel Order-2



(c) Dialogue: Cancel Order-3

Fig. 6: Dialogue: Cancel Order

(a) Dialogue: Pizza Order From Recommendations With Custom Toppings-1



(b) Dialogue: Pizza Order From Recommendations With Custom Toppings-2

Fig. 7: Dialogue: Pizza Order From Recommendations With Custom Toppings