Human-Machine Dialogue Project - Trento's Pizza Bot

Edoardo Maines (232226)

University of Trento

edoardo.maines@studenti.unitn.it

1. Introduction

The "Trento's Pizza Bot" project involves creating a voice assistant capable of managing pizza orders from customers and, at the same time, allowing pizzeria employees to check the status of ingredients in stock and, if necessary, place orders to replenish supplies. Based on these assumptions, the voice assistant is a multitask-based model because it helps users complete the aforementioned tasks. Thus, the assistant is designed to understand user requests, ask for any missing information, and complete the tasks in the most effective and efficient manner possible. As mentioned earlier, the system can handle two tasks: pizza ordering and stock management.
The first task involves ordering a pizza by specifying the type of pizza, its size, and, if desired, adding a maximum of one topping. Additionally, the user can request the addition of a beverage from the menu. If the user is not satisfied with the choices made previously, they can ask the assistant to make changes to the order or cancel the order entirely. Once the order is complete, the user can choose the type of delivery: take-away at the pizzeria or delivery to the selected address.
Regarding the stock management task, the user can request information from the assistant about the quantities of ingredients in stock and place orders to replenish supplies. To ensure this task is used only by authorized personnel, the user must pass a login phase where a username and password are required before requesting any information or placing an order.
Additionally, the system can answer various questions about the menu, including available pizzas, available sizes, beverages offered by the pizzeria, available toppings to add to the pizza, the price of a specific pizza, and the total of the order up to that point.
The system has been developed to handle both direct user questions (user-driven) and generic questions that require more involvement from the assistant in the conversation (bot-driven). The voice assistant was developed without a specific user target (USA or EU) because some information required by the assistant is needed in a specific format for the USA (address), while, for example, the phone number is requested in the Italian format (ten digits). This decision was made to facilitate system development but is easily adaptable by modifying the examples used for training.

2. Conversational Design

The conversation between the user and the system always begins with a greeting and an introduction to the pizzeria. Following this, the conversation can split into two different paths based on the user's requests. Clearly, the two paths depend on the goal the user wants to achieve (pizza ordering or stock management). As mentioned earlier, a feature of the system is its flexibility, which allows it to offer alternatives to the user.
Unfortunately, the system has some limitations: among these are the inability to order more than one pizza per order and to add more than one topping. Additionally, the virtual assistant is not able to handle over-informative users. For example, when changing information during the delivery form, the user is asked to provide all the information again.

2.1. Initiative

The system is mixed-initiative, as both the user and the system drive the conversation. As mentioned earlier, to order a pizza, the user can either directly provide the requested information, skipping some steps, or make a general request, in which case the system will ask for the necessary information.

2.2. Acknowledgement and Ground-taking

Grounding, in addition to being present in dialogues between humans, plays a crucial role in the development of conversational agents. It has been demonstrated that a user might find some of the system's responses confusing, especially when the system does not provide an explicit acknowledgment signal. Often, in a conversation, it is necessary to reach a common ground, so the listener must ground or acknowledge the speaker's utterance. For this reason, grounding by acknowledgment techniques have been introduced, using words like "Perfect," "Nice," and "Got it" at the beginning of a sentence when the system needs to ask the user several questions. The use of these expressions indicates to the user that the system has acknowledged the information previously provided.
Additionally, grounding by demonstration has also been employed, where the system uses the information provided by the user to formulate the next question. An example of this can be seen in the confirmation questions the system asks after receiving a request from the user.
Finally, in some utterances, such as during the login phase, adverbs like "Last thing" have been introduced at the beginning to achieve common ground. For example: "Last thing, can you enter your password?"

2.3. Error Recovery

Error handling is a key part of a conversational agent. The system must be able to manage ambiguous or incorrect inputs so that the conversation can proceed without impacting the flow. User inputs are validated, and if errors are detected, the user is asked to provide the information again. Below are some of the inputs that the system validates: pizza type, pizza size, pizza topping, drink, username, and password.
Additionally, it often happens that the user wants to change some information already stored by the system. For this reason, various utterances have been developed that end with a confirmation question, to which the user can respond affirmatively, negatively, or directly request a modification from the system.

2.4. Fall-back Responses

Occasionally, users may make requests or statements that have not been previously handled by the system. To address such scenarios, fallback techniques have been implemented to mitigate potential issues. Specifically, an out-of-scope intent has been introduced to manage cases where users inquire about topics beyond the bot's capabilities. Additionally, a fallback classifier has been incorporated into the NLU pipeline, which triggers the "nlu fallback" intent when all other intent predictions fall below a predefined confidence threshold, set empirically at 0.5. For instance, if a user asks, "What is 2 + 2?" and the system cannot determine an intent with sufficient confidence, the response will be, "I'm sorry, I didn't quite understand that. Could you rephrase?"

3. Data Description and Analysis

3.1. Dialogue Data

Conversation data are used to train the dialogue management model and were generated by creating dialogue flows based on what the user might say. All these dialogues are saved in the files stories and rules, which represent possible dialogues between the user and the system. The actions (responses) that the system can perform are divided into two categories: utterances (domain file) and custom actions (Python file actions.py).
The total number of stories generated is 58, which consist of various examples of how the conversation can proceed and examples of interruptions. The action space consists of 59 actions (utterances + custom actions).

3.2. NLU data

As for the NLU data used to train the model, some data were provided directly by Rasa, such as the greet and goodbye parts. Subsequently, for the development of intents, data generated during classroom exercises were used. All these examples are used to teach the model to extract useful information from the user's input messages, including user intents and possible contained entities. As mentioned earlier, except for some examples generated in class, all example sentences were manually created following patterns of ordering conversations. To help the model correctly identify them and enhance its robustness, additional information such as lookup tables and synonyms was added. Lookup tables are used to extract entities with a limited set of values, while synonyms are employed when there are multiple ways users might refer to the same concept. This strategy was employed only in Configuration 1, while in Configuration 2, a custom component was tested for correcting entities in cases of spelling errors or similar terms.
The NLU examples are divided into 36 intents, 16 entities, and 19 slots.

3.3. Domain data

To use the system, one must be aware of various pieces of information such as pizzas, toppings, drinks, and available ingredients. Two methods have been used to manage this information: Python dictionaries and a database. Python dictionaries are used to create a digital representation of the menu and are utilized during the user input validation phase. They are also used when the user requests the menu list or the individual prices of items. The database, on the other hand, is used to store the quantities of ingredients and toppings in stock. When a user orders a specific pizza, adds a certain topping, or orders a drink, the database is updated by modifying the quantities of the specific item ordered. The same process applies when an order is modified, requiring an increase in the quantity of an item that the user no longer wants. Additionally, the database contains the necessary information for employee login to access stock data: these details include the ID and password.
Thus, the database plays a central role in the stock management task. The data displayed to the user are extracted through SQL queries from the four tables in the database: toppings, drinks, ingredients, and login.

4. CONVERSATIONAL MODEL

4.1. Model

The Trento's Pizza Bot bot was developed using the RASA framework, which enables the creation of a voice-based AI system. The RASA framework offers two main components: Natural Language Understanding (NLU) and Dialogue Management (DM). The NLU component handles intent classification, entity extraction from user inputs, and response retrieval, while the DM component determines the next action based on the context.
Additionally, RASA provides an action server that can be used to run custom actions to manage the various functionalities offered by the assistant.
Finally, the assistant's conversations are stored within a tracker memory, allowing access to the assistant's memory from custom actions. This is particularly useful when managing the dialogue based on previous inputs.

4.2. NLU pipeline

The NLU pipeline consists of several components that work sequentially to process user inputs into structured outputs. For this project, two different NLU pipelines were developed to compare various entity extractors while keeping the initial components the same: we selected SpacyNLP as the language model, which allows for the use of pre-trained word vectors. This choice enabled us to incorporate the SpacyTokenizer and SpacyFeaturizer into the pipeline. Alongside the SpacyFeaturizer, we employed additional featurizers such as RegexFeaturizer, LexicalSyntacticFeaturizer, and CountVectorsFeaturizer to extract feature vectors from user utterances.
Now, let's examine the key differences between the two proposed NLU pipelines:

Configuration 1 utilizes the DIET Classifier (Dual Intent Entity Transformer) for both intent classification and entity extraction, utilizing a shared transformer architecture for both tasks. Additionally, it includes a supporting component, the RegexEntityExtractor, which uses a lookup table for entity extraction. The final components of the pipeline include the EntitySynonymMapper, which standardizes synonyms to a single slot value, and the ResponseSelector, which predicts the appropriate response from a set of candidate options.

For Configuration 2, the goal was to differentiate the components used: the DIET Classifier was employed solely for intent classification, while CRFEntityExtractor was selected for entity extraction. CRFEntityExtractor utilizes Conditional Random Fields (CRF) to identify and extract entities from text by modeling the relationships between tokens. It learns from annotated data to predict entities based on the context and features of each token. Additionally, in Configuration 2's NLU pipeline, a custom component was added to handle spelling

correction and correction of entity classification by CRFEntityExtractor. The functionality of this custom component will be discussed in more detail later. The final component of the NLU pipeline is the ResponseSelector.

4.3. Policies

Thanks to the policies, the assistant is able to decide which action to take at each step of the conversation. As for Trento's Pizza Bot, the policy structure utilized in the bot's configurations 1 and 2 comprises three key components, each serving a distinct role in enhancing dialogue management. The TEDPolicy is based on the Transformer Embedding Dialogue (TED) model, configured with a *max_history* of 8 to retain the context from the last eight turns of conversation, and is trained for 100 epochs. It also uses the *constrain_similarities* parameter to ensure consistent treatment of similar intents. The AugmentedMemoizationPolicy builds upon the basic memoization approach by adding an extra layer of context, with a *max_history* of 4 to remember the last four turns and improve response accuracy based on prior interactions. Finally, the RulePolicy addresses low confidence action predictions by setting a *core_fallback_threshold* of 0.4, which triggers a fallback action named *action_default_fallback* when confidence falls below this threshold.

4.4. Automatic Speech Recognition (ASR) & Text-To-Speech (TTS)

As mentioned earlier, Trento's Pizza Bot is a voice assistant, and Alexa was used for automatic speech recognition and text-to-speech. Thus, thanks to Alexa, it was possible to use the ready-to-use Automatic Speech Recognition model for handling the voice component, while the Rasa framework manages user intents and how to continue the conversation.

4.5. Custom Component

During conversations to test the integrated system, some issues were encountered with Alexa's recognition of certain inputs, such as the entities needed for the conversation flow. This led to spelling errors that caused the model to malfunction and, consequently, the inability to progress in the conversation. To address this issue, two strategies were tested, one for each configuration. In the first strategy, as mentioned earlier, the EntitySynonymMapper component was used. By using a list of all possible spelling errors observed while testing the system, it improves the classification of entities. However, this strategy has a limitation: it is necessary to know deterministically the spelling errors that Alexa makes when converting speech to text. The strategy used in configuration 2 aims to bypass this limitation by introducing a custom graph component, as previously mentioned. This component is designed to correct the entities extracted by the classifiers before they are used for intent prediction. Let's take a closer look at how this component works: this component is positioned between the entity extraction component CRFEntityExtractor and the intent classifier DIET Classifier. Once the entities are extracted from the message and classified, their values are checked against a list containing the correct values for all entities considered important for the system. The spelling check of the entities is performed using two Python libraries: spellchecker and fuzzywuzzy. Once the correct value is extracted, the entity is updated. However, it is not possible to correct only the value of the entity. Entities in Rasa consist of several features, including the *start* and *end* indices of the entity in the user's message. Therefore, since the length of the entity may change after correction, the component also updates these indices to ensure accurate classification and extraction of the entities. However, the value of the entity can influence the classification of the CRFEntityExtractor. Therefore, once the correct value is obtained, through a dictionary, the component updates the type of the extracted entity. In this way, the DIET Classifier will have the entities written and classified correctly, and will be able to classify the intent without errors.

5. Evaluation

5.1. Natural Language Understanding

To evaluate the chosen configuration concerning the NLU model and compare it with other potential NLU pipelines, the evaluation was conducted using the method recommended in the RASA documentation. Specifically, the training data was split globally into 80% for training and 20% for testing. The various models were then trained on the training split and evaluated on the test split. This procedure was repeated three times. We compared two different configurations (NLU pipelines):

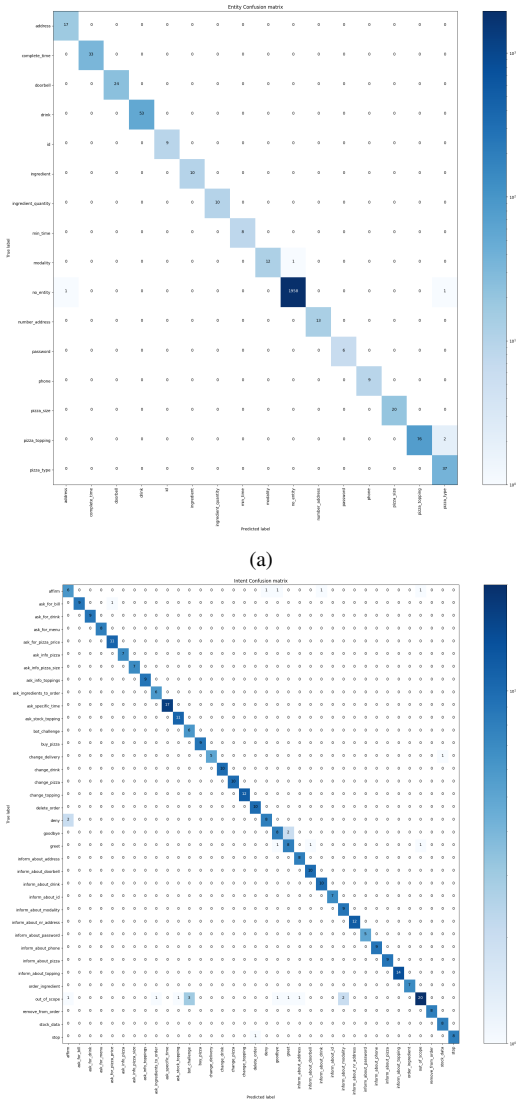
- **Config 1**: composed by *DietClassifier* for intent classification and the entity recognition using *RegexEntityExtractor* as support in entity recognition
- **Config 2**: the one using *CRFEntityExtractor*, one *custom GraphComponent* (new Rasa Component) for spelling correction end entity recognition correction and *DietClassifier* for intent classification

Table 1: *Performance of the two configurations*

		Config 1				*Config 2*			
	Run	1	2	3	avg	1	2	3	avg
Intent	*Acc.*	92	94	92	**93**	92	94	93	**93**
	Prec.	93	94	93	**93**	93	95	94	**94**
	Rec.	92	94	92	**93**	92	96	93	**94**
	f1	92	94	93	**93**	92	95	93	**93**
Entity	*Acc.*	99	99	99	**88**	99	99	99	**99**
	Prec.	98	96	98	**97**	99	97	98	**98**
	Rec.	99	97	100	**99**	100	99	99	**99**
	f1	98	97	99	**98**	99	98	98	**98**

In Table 1, we present the performance results for the two different configurations. As anticipated, the two Configurations exhibit very similar outcomes, as the only distinction between them is the entity extractor used. In the first configuration, DIET is employed for both intent classification and entity recognition, while in the second, DIET handles only intent classification, with CRFEntityExtractor responsible for entity recognition.

Looking at Figure 1, the most significant observation is the dominance of the diagonal values, which indicates that the classifier is correctly predicting the majority of the entities. For instance, entities like *complete_time*, *drink*, *doorbell*, and *pizza_size* show high numbers on the diagonal (e.g., 33, 53, 24, and 76 respectively), signifying that these entities are correctly identified with minimal confusion. The *no_entity* label, which represents cases where no specific entity should be extracted, has the highest count of correctly predicted instances with a value of 1058. This indicates that the classifier is very proficient in recognizing when there are no entities present, which is crucial for reducing false positives.



(a)

(b)

Figure 1: *Configuration 1 (DIET + REGEX)*
(a): Entity Extraction (b): Intent Classification

The *no_entity* row shows a small misclassification where one instance was wrongly predicted as *number_address*. While this error is minimal, it suggests that there might be a subtle overlap in the model's ability to distinguish between these two labels in certain contexts. Another slight misclassification is observed for the *password* entity, where 6 instances are correctly classified, but no misclassification errors are evident in the surrounding cells. This suggests that while the model is accurate, more data or refined training may be needed to enhance its precision further.

Some entities like *id*, *ingredient*, *min_time*, and *min_time* have relatively low occurrences in the dataset (with counts ranging from 8 to 10), but they are correctly classified without significant confusion. This suggests that even with limited data, the DIET classifier is capable of maintaining good accuracy, which is promising for entities with fewer examples.

As for classification of intents, DIET makes few missclassifications errors especially with *out_of_scope*, *bot_challenge* and *inform_about_modality*. This is likely due to the fact that some examples used for the *out_of_scope* intent are very similar to those used for other intents, leading the bot to confuse them

Regarding the results of the Configuration 2, the CRFEntityExtractor for entity extraction shows very similar but improved results: in the confusion matrix, it can be observed that only a few errors concern the entity *doorbell*.

Regarding intent classification in Configuration 2, the results are very similar to the other configuration, but some intents are still misclassified: mostly *out_of_scope* with *goodbye*.

Unlike before, this configuration includes the custom component used to adjust entity extraction. From the results, it is possible to observe that this component improves classification for some intents that were previously confused in the first configuration but introduces new errors with an intent that was previously classified correctly: *goodbye*.

See Figure 2 and 3 in the Appendix for comparisons between the different configurations in terms of confusion matrices.

5.2. Dialogue Management

With RASA, the dialogue model can be evaluated using a set of stories designed to assess the bot's performance in various situations. Test stories were created to address two tasks, taking into account potential ambiguous scenarios that may occur when the user is uncooperative and deviates from the expected path. The dialogue model was tested and demonstrated strong performance at the action level, accurately predicting 195 out of 195 actions with an accuracy of 1.00, as indicated in the first row of Table 2 (see Figure 4 in the Appendix for the corresponding confusion matrix). This test assesses the model's ability to respond to user intents by predicting the correct actions. As a result, the dialogue-level accuracy is also 1.00 since all the test stories are correctly predicted. It's important to note that a story fails if at least one action is predicted incorrectly.

For the final analysis of the RASA policy, an ablation study was conducted to determine the contribution of each of the three components to overall performance, with results presented in the table below. It is evident that RulePolicy and TEDPolicy are the two most significant policies. A more detailed table is available in the appendix (Table 4).

	Accuracy	Precision	Recall	F1-score
All	1,00	1,00	1,00	1,00
Rule + Memoization	0,89	0,93	0,89	0,91
Rule + TED	1,00	1,00	1,00	1,00

Table 2: *Action level performance - Ablation study*

A second methodology for testing the entire system involved having four different people try out the bot in order to obtain feedback directly from the end user. After having a conversation with the system, the users were asked to rate the conversation and the system overall on a scale from 1 (not satisfied) to 5 (very satisfied). Based on the observations, it was noted that users were always able to complete the assigned tasks, however, as can be seen in the Table 3, the overall rating is 3,8. Several areas for improvement were highlighted: the system requires a high level of pronunciation accuracy, which at times leads to difficulties in word and entity recognition. Another observed issue is the recurrence of certain phrases, which made the conversation feel somewhat repetitive for some end

users. Lastly, in certain situations, the system struggles to recognize entities such as phone numbers and specific drinks. A problem encountered by everyone is related to the system's response speed during the conversation. Unfortunately, this is due to the computer used, which does not offer high performance. On the other hand, some strengths were observed, including the system's ability to recognize a good variety of terms, effective communication, and ease and intuitiveness in both usage and conversation structure.

	User 1	User 2	User 3	User 4	User 5	Mean
Score	3	4	4	3	5	3,8

Table 3: *Human evaluation results*

6. Conclusion

This report presents the project called "Trento's Pizza Bot," a conversational agent that allows customers to order a pizza and enables employees to check stock levels and, if necessary, order new ingredients. During development, two NLU pipelines were tested, and it was observed that for intent classification, the DIET Classifier supported by lookup tables and Regex synonyms performs almost the same as the DIET Classifier supported by the custom component, which operates at the entity level to correct spelling errors and classification errors. Regarding entity extraction, the DIET Classifier behaves very similarly to the CRFEntityExtractor.

A. INTENTS & ENTITIES

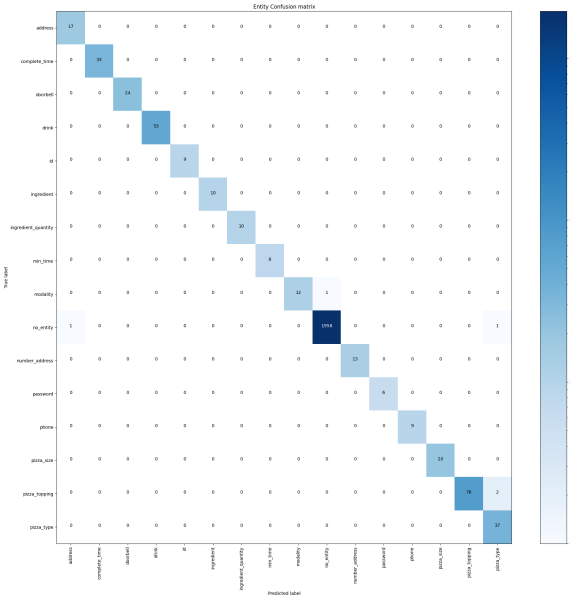
Intents

- affirm
- ask_for_bill
- ask_for_drink
- ask_for_menu
- ask_for_pizza_price
- ask_info_pizza
- ask_info_pizza_size
- ask_info_toppings
- ask_ingredients_to_order
- ask_specific_time:
- ask_stock_topping
- bot_challenge
- buy_pizza
- change_delivery
- change_drink
- change_pizza
- change_topping
- delete_order
- deny
- goodbye
- greet
- inform_about_address
- inform_about_doorbell
- inform_about_drink
- inform_about_id
- inform_about_modality
- inform_about_nr_address
- inform_about_password
- inform_about_phone
- inform_about_pizza
- inform_about_topping
- order_ingredient
- out_of_scope
- remove_from_order
- stock_data
- stop

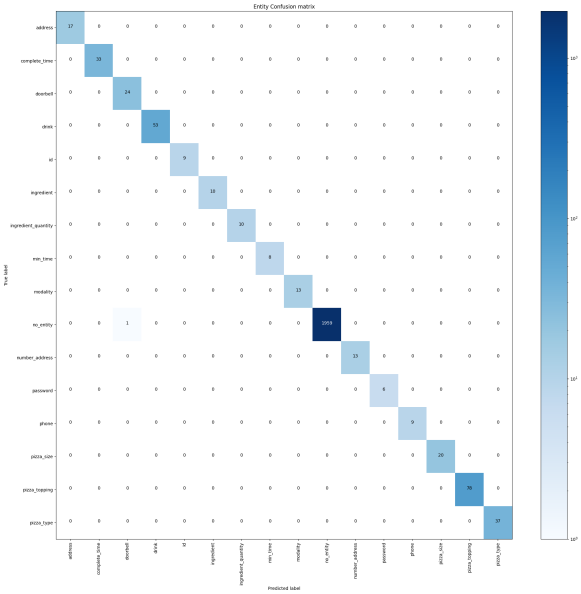
Entities

- pizza_type
- pizza_size
- pizza_topping
- id
- password
- ingredient
- drink
- ingredient_quantity
- ingredient_under_threshold
- modality
- address
- number_address
- phone
- doorbell
- complete_time
- min_time

B. EVALUATION

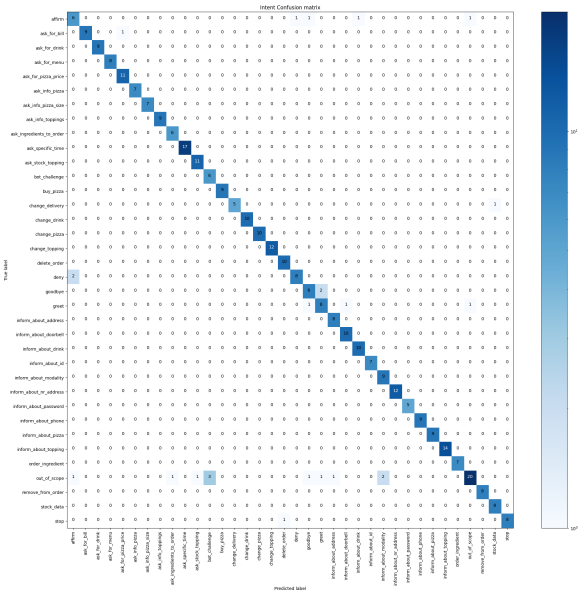


(a) *Config. 1 - DIET + REGEX*

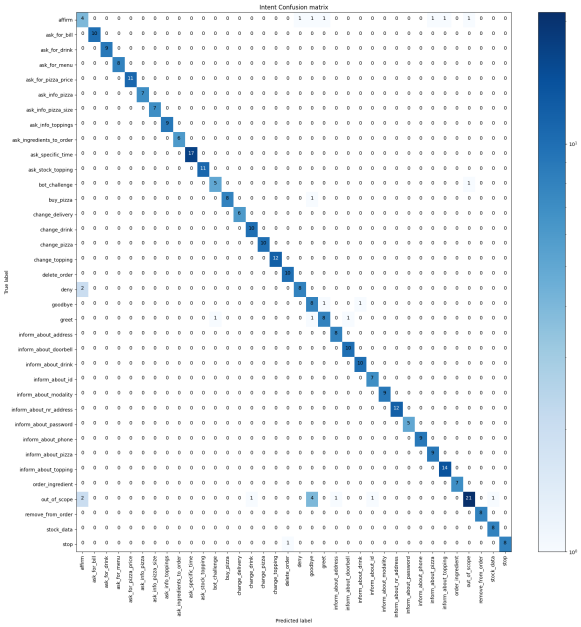


(b) *Config. 2 - CRFEntityExtractor*

Figure 2: *Confusion Matrices - Entity Extraction*



(a) *Config. 1 - DIET*



(b) *Config. 2 - Custom Component + DIET*

Figure 3: *Confusion Matrices - Intent Classification*

	Accuracy	Precision	Recall	F1-score	Action Rate	Dialog Acc.
All	1,00	1,00	1,00	1,00	195/195	1.00
Rule + Memoization	0,89	0,93	0,89	0,91	174/195	0.33
Rule + TED	1,00	1,00	1,00	1,00	195/195	1.00

Table 4: *Action level performance - Ablation study complete*

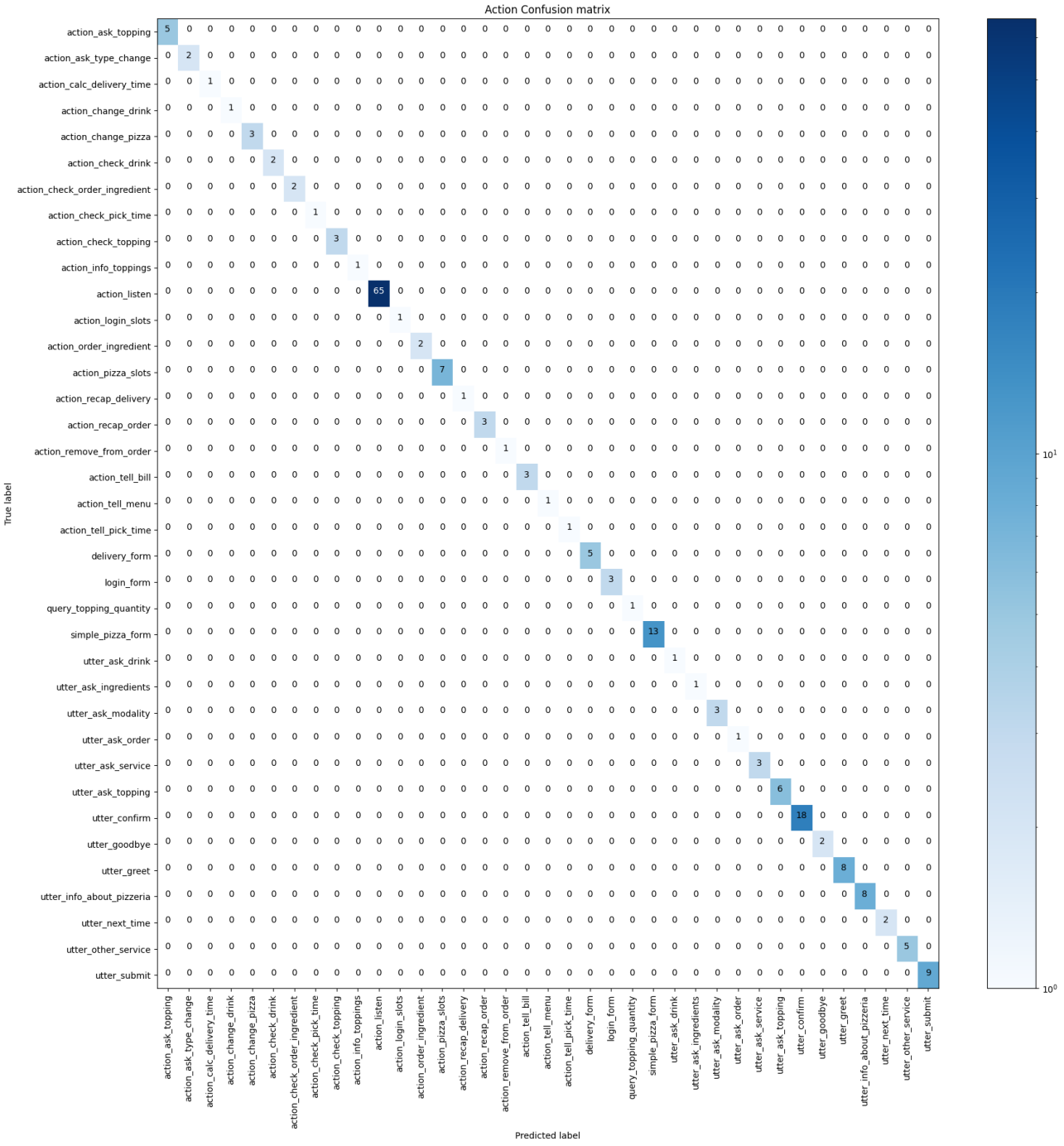


Figure 4: *Confusion matrix dialogue model at action level*