HMD Report

Francesco Vaccari
francesco.vaccari@studenti.unitn.it
[239927]

Mattia Tognato
mattia.tognato@studenti.unitn.it
[238857]

I. Introduction

The project we developed is a task-based model that interacts with the user through natural language. The abilities of the chatbot can be grouped into three main tasks: pizza ordering, order cancellation and general question answering on topics relevant to the context. In a real-world scenario, we imagine our system to be the one answering calls coming from clients of a pizzeria. The natural language understanding and generation abilities of the model were implemented using Rasa, while the text to speech and automatic speech recognition are put in place through Alexa skills. The Rasa project was built from scratch, taking from [1] only the necessary for the integration with Alexa.

II. Conversation Design

We identify two main conversation flows in our model: pizza ordering and order cancellation. In the first path the user lists the items he would like to order, then confirms the order, and finally goes through the checkout procedure where he can choose to do take-away, have the order delivered, or make a table reservation. This checkout step requires the user to provide information like preferred time, address and number of people, depending on the choice made. Once the user completes these steps, the system considers the conversation terminated and responds accordingly to any other request. In the second path, the user has to express the wish to cancel an order placed in another call and, in the procedure initiated by the chatbot, the user will be prompted to provide the number associated to the order and, after confirmation that the order was indeed cancelled, the user can proceed in making a new ordination, as the system does not consider the completion of this task an end of conversation.

At any point in the dialogue, the user can ask questions regarding:

- Address, opening hours, menu of the pizzeria
- Total cost of the order placed
- Affluence, with the possibility of asking about a particular day
- Intolerances or preferences, with the possibility of asking for specific menu options
- Functions the bot can perform

We initially found that being able to make questions at any point of the conversation was distracting since a response from the bot would steer the dialogue away from the goal of completing the task; we solved this problem by making the bot repeat the sentence uttered prior to the question from the user in order to get the conversation back on track.

The user asking questions does not really influence the conversation outside of the immediate response, with the exception of when the user specifies a particular intolerance or preference, in that case the information is stored and the system will check if there are conflicts when placing the order and will inform the user about their presence.

Regarding initiative, the user is the one to choose at the start of the conversation which task he wants to perform. Additionally, the user can ask questions about multiple topics at any point in the conversation, although the chatbot regains the initiative immediately after, with the objective of completing the procedure. For this reason the model can be defined as mixed-initiative, even if the flow of conversation is strongly guided by the chatbot. This design choice allows for a more straigh-forward, and consequently with less mistakes, interaction between user and machine.

After the system obtains some information, it usually produces a responses that starts with *"Ok, so..."*, *"You have chosen to..."* or *"You said ..."*, to make the conversation feel more natural, in some cases repeating the choice made by the user. For example, in the pizza ordering procedure, the chatbot will repeat the entire ordination made by the user to make sure that it was understood correctly. This is especially important because of the complex system of ordination we were able to implement: the user can specify multiple items with different amounts or sizes in the same utterance.

However, since the system is not perfect, the chatbot asks for confirmation by listing back to the user the ordered items. And trying to prevent future mistakes, we ask the use to either rephrase the order or specify one item at a time. This mechanism is fundamental because it tries to make the user produce sentences that are more likely for the system to recognize correctly. Another case in which the bot will ask for confirmation is for cancelling an order made in a previous conversation.

Another mechanism to try and recover from mistakes of the intent recognition module, is implemented with the Fallback Classifier: when an intent is recognized with a confidence under a certain threshold, the bot will simply repeat the last utterance.

III. Data Description & Analysis

Regarding intent classification, to obtain a reliable model we needed to generate many examples for the most complex

intents:

- **tell_number**: This intent should be predicted when the user specifies a number in its response, which in our case represents either the order number or the number of people for making a reservation. To create these examples we combined appropriate sentence prefixes and combined them with numbers.
- **tell_address**: For this intent we used two datasets coming from [5] and [6] to teach the model how an address should look like. The value of the address is combined with random prefixes to create the final example sentence.
- **tell_time**: This intent is used to recognize when the user expresses a preference for time of delivery, take-away or reservation during the checkout procedure. To generate the examples we combined random sentence prefixes with times and dates in different formats.
- **order**: Arguably the most important intent of this project, we combined appropriate prefixes with up to 4 randomly chosen items with amounts and sizes not always specified. In these examples we used groups for tagging, this is the feature that allows us to match the amount and size to the item ordered, even when multiples items are specified.

All other intents are fairly simple and contain considerably less examples for the classifier to train on. Since many intents are fit to multiple steps of the conversation we use many slots to implement the logic and allow to the custom actions to know at which point of the conversation the system is.

We use lookup tables and synonyms for a more reliable entity recognition. In particular, we made sure that the entity values used in examples would match the transcription performed by the speech recognition of Alexa.

All the utterances of the chatbot are defined in the actions, which are all custom to implement correctly the logic necessary to complete the tasks.

The vocabulary contains 7117 items and we have in total 22 intents, 8 entities, 27 slots and 23 custom actions. In total there 40107 examples, where 27.084 of them belong to the *order* intent while the intent *thank_you* consists of only 4 examples. Even though the dataset is strongly imbalanced, we don't find it to influence the performance of the model.

A. Database

Regarding static data, we decided not to employ a database in our project because there is no need of a complex solution when a simple text file holds all the information that is necessary to us. We use a function defined alongside the custom actions to write and read from this file. The menu is also stored statically as lists inside the *actions.py* file.

IV. Conversation Model

We defined two pipelines, main and alternative, to test and compare different components and see how they perform in our project.

The main pipeline consists of:

- **WhiteSpaceTokenizer** with the *case_sensitive* option set as False to better deal with the Alexa transcription of speech
- **RegexFeaturizer** that allows us to lookup tables for entities
- **CountVerctorsFeaturizer** which is a simple yet effective featurizer
- **DIETClassifier**, used only for intent classification and trained for only one epoch
- **CRFEntityExtractor** that, when compared with DIETClassifier for entity recognition this component, proved to be superior in performance
- **DucklingEntityExtractor** which is used to recognize number and dates in many different complex formats.
- **EntitySynonymMapper** which lets us use synonyms for entity recognition
- **ResponseSelector** which is not used since we do not have responses defined
- **FallbackClassifier** that intervenes when the predicted intent confidence is under the 0.3 threshold.

Since our project is entirely based on rules, the only policy to decide which actions to perform is the **RulePolicy**.

As for the alternative pipeline to compare against, we decided to use a Spacy-based pipeline with the **SpacyTokenizer** and **SpacyFeaturizer** components. Furthermore, we wanted to test the **DIETClassifier** for both entity extraction and intent classification. This pipeline consists of more complex components and should in theory perform better, but as we will see this did not prove to be true.

V. Evaluation

The evaluation can be split into two parts: intrinsic and extrinsic evaluation. For the intrinsic evaluation we used rasa built-in commands to evaluate the performance of the two trained models with the proposed main and alterantive pipelines. For the extrinsic evaluation we asked to a group of people to use our system to carry out three tasks, and then asked them to rate the chatbot according to 4 criteria.

A. Intrinsic Evaluation

For the nlu evaluation, rasa provides a command that compares two different pipelines against each other. This command creates an 80/20 split of the dataset, then trains the two models and finally evaluates them on the test split. In table I we are mostly interested in the *macro* performance values since they represent better the results from minority classes. The performance of the main pipeline is superior than the performance of the more complex alternative pipeline. The most important result of the evaluation is the entity recognition aspect of the two models, with the main pipeline being much better than the alternative; this difference is especially felt when testing manually the chatbots.

Since we implemented the chatbot using only rules, to perform evaluation on test stories we needed to write them from scratch. We trained fully both models with *rasa train* and then we tested with *rasa test core* once for each pipeline.

	Pipeline	
Metric	**main**	**alternative**
accuracy	0.9982	0.9947
micro avg precision	0.9983	0.9949
micro avg recall	0.9982	0.9947
micro avg f1-score	0.9982	0.9948
macro avg precision	0.8671	0.8150
macro avg recall	0.8831	0.7746
macro avg f1-score	0.8581	0.7733
weighted precision	0.9985	0.9955
weighted recall	0.9982	0.9947
weighted f1-score	0.9982	0.9949

TABLE I
NLU EVALUATION ON MAIN AND ALTERANTIVE PIPELINE

As expected, for both models all actions are predicted correctly by the model, with expections of *action_listen*. However, this does not cause unexpected behaviour during the normal use the chatbot.

The plots here discussed can be found in the Appendix. In figures 1 and 2 we can see that both models are able to identify correctly most of the intents, this is as expected since the component for intent classification is the DIETClassifier in both pipelines. We are mostly interested in the entity recognition and classification confusion matrices, in figures 3 and 4, where we can see how the CRFEntityExtractor performs much better than its competitor. This difference can be attributed to the fact that the DIETClassifier is a transformer which is a more powerful architecture, but with the disadvantage of requiring much more training data than a non-transformer based machine learning model like the CRFEntityExtractor. In fact, we can clearly see a difference in prediction confidences in figures 5 and 6. We also include the plots in figures 7 and 8 that show that the action *action_listen* is involved in the mislassification errors.

B. Extrinsic Evaluation

To perform an extrinsic evaluation of the model trained on the main pipeline we asked 7 people to use our chatbot. Each candidate receives a list of three tasks to complete, after completing each one the bot is reset and a new conversation is started. After the three objectives are completed we ask them to judge the model by giving a score from 1 (worst) to 5 (best) for each of the four questions we give them. These questions are:

- Did you find the system easy to use? Did it take a small effort to complete the objective?
- Was the chatbot able to understand what you meant and act accordingly? If it made any mistake, was it able to recover successfully?
- Did the conversation feel natural as if you were having a conversation with a human?
- Was it easy to understand what the bot was asking and what information was trying to obtain from you?

and correspond to the following aspects:

- Ease of Use, User Friendliness.
- Understanding, Accuracy, Error Recovery
- Naturality
- Clarity, Intuitiveness

	Candidates							Average
Question 1	4	4	4	4	4	4	5	4.14
Question 2	3	4	4	4	4	3	3	3.57
Question 3	4	3	5	3	5	2	2	3.42
Question 4	3	5	5	5	4	4	4	4.28

TABLE II
EXTRINSIC EVALUATION SCORES

In table II we can see the results of the extrinsic evaluation. Looking at the results the bot is especially lacking in the *Naturality* aspect, but performs well in the *Clarity* aspect. This is due to how the model responses were written, with repetitions and always forming a full sentence. Altough this not really *human-like*, this design decision helped making the bot more clear.

While observing users interact with our system we noticed 3 issues:

- **Ambiguity** A particular response was misinterpreted during the extrinsic evaluation; the sentence in question is *"Ok, do you want anything else?"*. The objective of the bot in the context is to close the items ordering step and move on to the checkout procedure. However, the question was interpreted multiple times as a general request about the whole procedure, and not only regarding menu items. In fact, some users tried to respond by asking to make a reservation or choose the delivery option including details about the process. This problem could be easily solved by changing the bot question to *"Ok, do you want anything else to order?"*
- **Framework** The second issue is about our implementation of the ordering procedures, with each step aiming to obtain a necessary piece of information. Some candidates were including in their requests too many details, for example the sentence *"I want to make a reservation for tomorrow at 8 p.m. for 2 people."* is not recognized by our bot because of the strict steps we designed to complete a specific task. This problem does not have a simple solution as it would require changing the whole structure of our project and designing it to be more flexible.
- **Additional information** When an user included some information not relevant to the context of pizza ordering, for example in the request *"I want 10 pizzas for a party."*, the bot was not able to classify correctly the intent, sometimes triggering the fallback intent. This problem is solvable either by generating a much bigger number of intents, or by employing a more powerful natural language understanding model.

VI. CONCLUSION

In conclusion, we were able to design and implement a somewhat realistic chatbot to handle a conversation with a client of a pizzeria. We started from scratch, using Rasa documentation [2] and tutorials [4] as reference, and built a good system according to the evaluation carried out. We took into consideration how the Alexa Skill ASR transcription works,

and adapted examples and entities accordingly. Additionally, we experimented with two pipelines to try and understand what would happen with different components, and, after choosing the best ones, we evaluated our model against a third party to obtain an external and neutral view of our work. Regarding this last aspect of evaluation, we learned which are the flaws and poor design decision of our chatbot, and we propose possible solutions and improvements.

REFERENCES

[1] Giuliano Tortoreto. PizzaBot. https://github.com/giTorto/HMD-PizzaBot/tree/2023.(2023).

[2] Rasa documentation, https://rasa.com/docs/rasa/

[3] Rasa testing documentation, https://rasa.com/docs/rasa/testing-your-assistant/

[4] Rasa video tutorials, https://www.youtube.com/@RasaHQ

[5] Source of USA addresses, https://www.kaggle.com/datasets/ahmedshahriarsakib/list-of-real-usa-addresses

[6] Source of italian addresses, https://www.kaggle.com/datasets/openaddresses/openaddresses-europe?resource=download&select=italy.csv

APPENDIX A
TASKS IMPLEMENTED

We include a list of the task we implemented in our project:

- More complex ordering system that can handle multiple items with different sizes and amounts, and is able to match each entity to the correct menu item ordered. Items can also be added to the ordination in multiple separate requests, this feature has the purpose of also acting as a fallback.
- Error recovery during ordering procedure in which the user is asked to confirm whether the order taken is correct.
- More menu items, including desserts and drinks.
- Retrieve and cancel an order made in a previous conversation.
- Asking questions to obtain information about the pizzeria, like address, opening hours, menu.
- Asking about affluence, with the possibility of specifying a particular day of the week in the request.
- Asking about the total cost of the order placed in the current conversation.
- Asking about what the chatbot can do.
- Asking about menu options for dietary requirements, with the possibility of specifying a preference. If specified, the system will check whether the items ordered satisfy this constraint and will inform the user in case of conflicts.
- More complex checkout procedure in which the user can choose between three options: delivery, take-away and making a reservation.

APPENDIX B
INTENTS

In this section, we illustrate the different intents used in the project, dividing them into similarity groups.

- *bot_challenge*: recognizes when the user is asking the chatbot whether it is human
- *confirm,deny,nothing_else*: used for general positive or negative user responses
- *greet,thank_you*
- *order*: used to recognize when the user expresses the will to make an order, or when he already includes specific items in the request
- *out_of_scope*
- *ask_address,ask_affluence,ask_allergies,ask_cost,ask_functions, ask_opening_hours,ask_menu*: recognize general information request from the user
- *change_order*: used to initiate the procedure for canceling an order made in a previous conversation
- *takeaway,delivery,table*: recognizes the option chosen during checkout
- *tell_time,tell_address,tell_number*: recognizes when the user is providing an address, a date and time, or is responding with a order number or number of people, both represented by the same intent

APPENDIX C
ENTITIES

In this section, we show the different entities in our project.

- *allergies*: recognizes if the user asks or informs the system with a particular intolerance or preference such as vegetarian, vegan, lactose-free or gluten-free
- *time*: recognizes the date and time the user specifies during checkout
- *address*: recognizes the address for delivery/takeaway
- *number*: recognizes a number, set by the DucklingExtractor
- *amounts*: recognizes the amount of each item the user is ordering
- *pizza_sizes*: recognizes the size of the items during item ordering
- *pizza_types*: recognizes which pizzas the user is ordering
- *other_item_types*: recognizes whether the user is ordering desserts or drinks

APPENDIX D
CUSTOM ACTIONS

- *action_utter_imabot,action_utter_functions*: give information about the bot
- *action_confirm_order,action_order_correct,*
  action_order_incorrect,action_order: complete order procedure
- *action_utter_greet,thank_you,action_you_are_welcome*: actions for politeness
- *action_utter_out_of_scope*: response to the out of scope intent
- *action_repeat_last_message*: repeats the last message sent by the chatbot, used as a fallback mechanism to regain initiative and keep the conversation headed in the right direction

- *action_utter_cost*: informs the user about the total cost of the order
- *action_utter_address,action_utter_affluence,*
  action_utter_allergies,action_utter_opening_hours,
  action_utter_menu: responds to the user about general information requests
- *action_change_order,action_change_order_confirmed,*
  action_change_order_not_confirmed: used to initiate and complete the order cancellation procedure
- *action_takeaway,action_delivery,action_table*: used during checkout in parallel to handle the different paths of conversation

APPENDIX E
INTRINSIC EVALUATION RESULTS

In this section we show the plots generated by the rasa built-in test commands. All results can be reproduced by running the *evaluation.py* file in the project directory. The script will perform the nlu comparison between models, training of both, and then testing with the 7 test stories included.

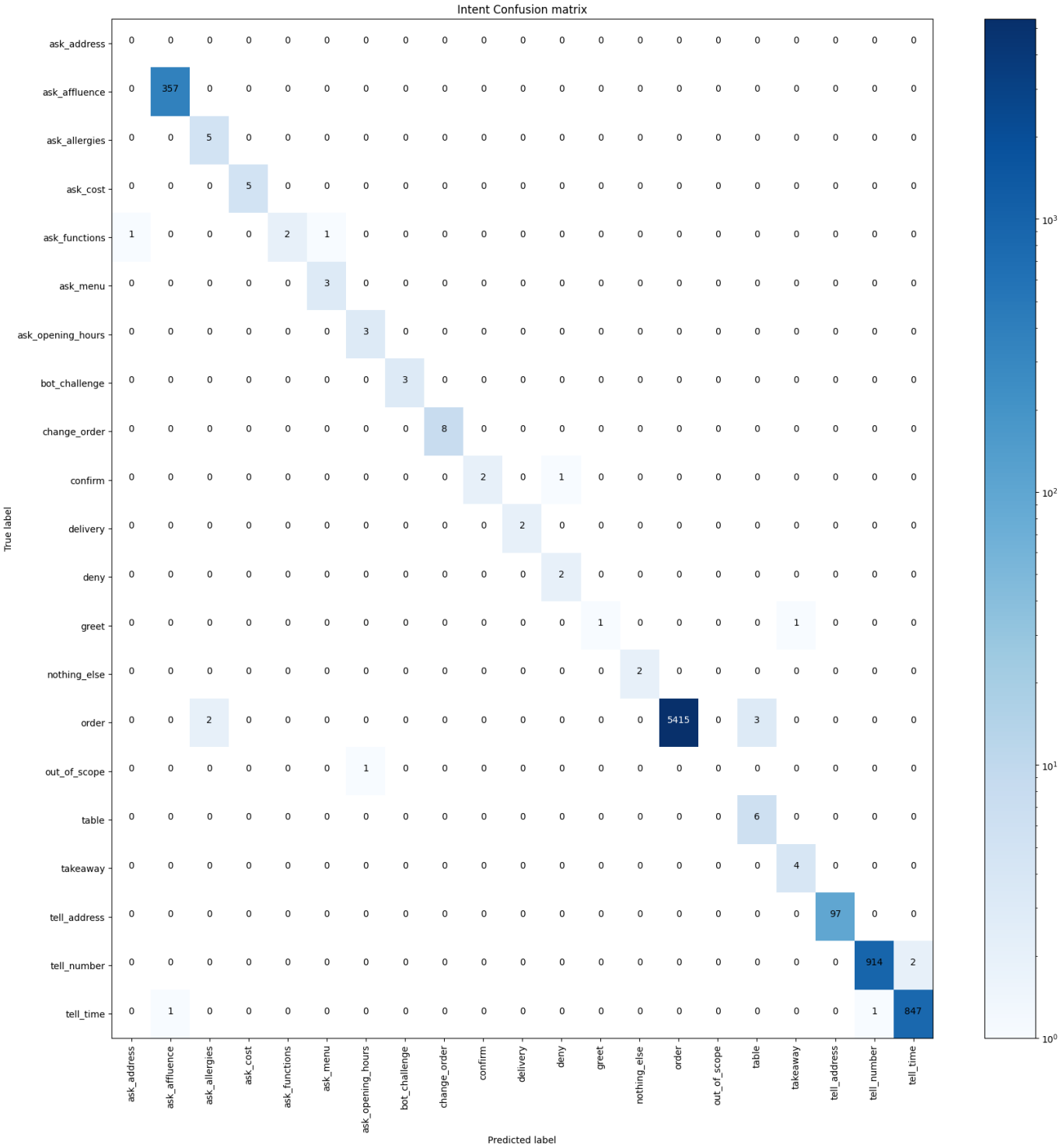

Fig. 1. Intent confusion matrix main pipeline

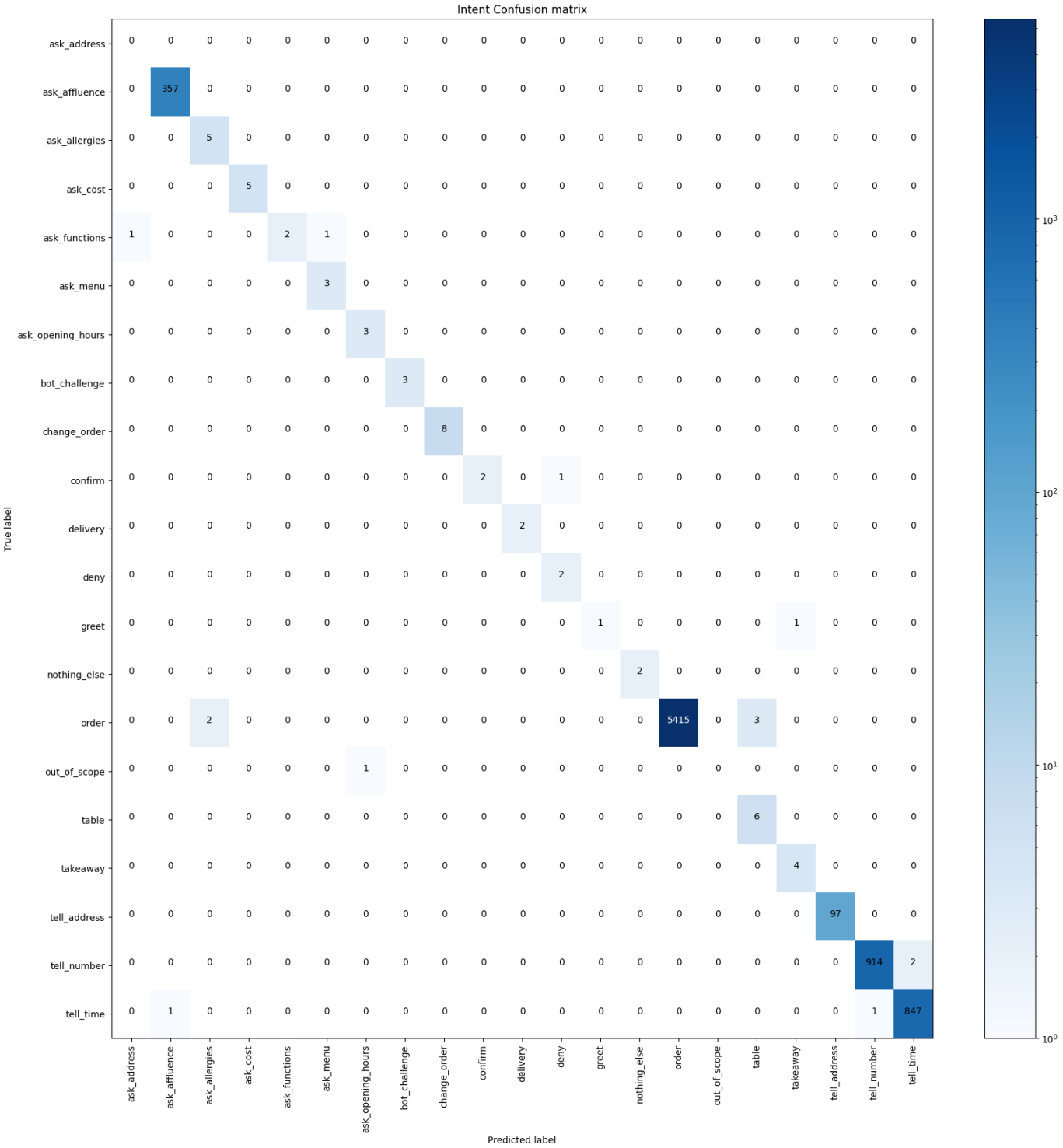

Fig. 2. Intent confusion matrix alternative pipeline

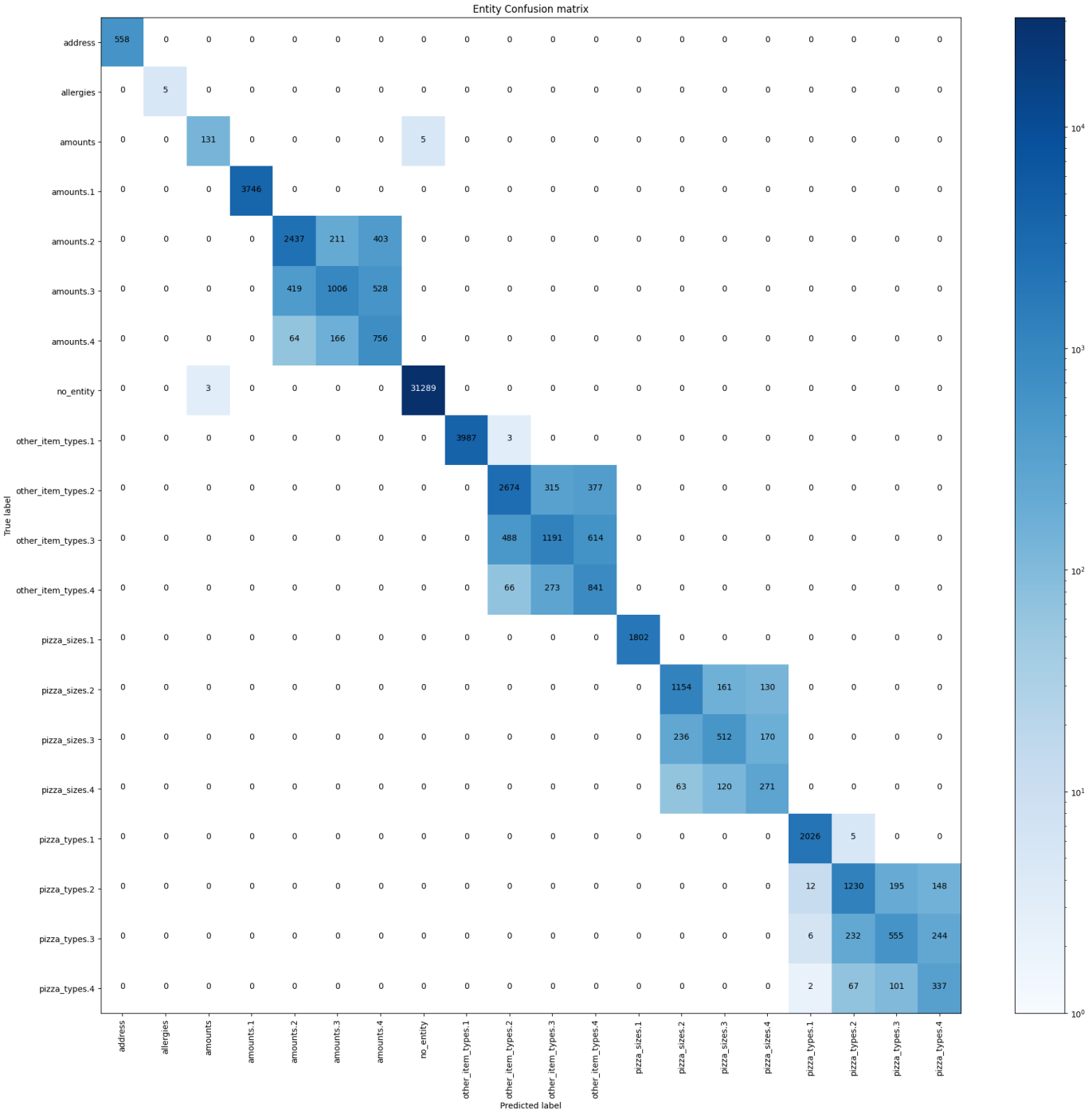

Fig. 3. CRFEntityExtractor confusion matrix main pipeline

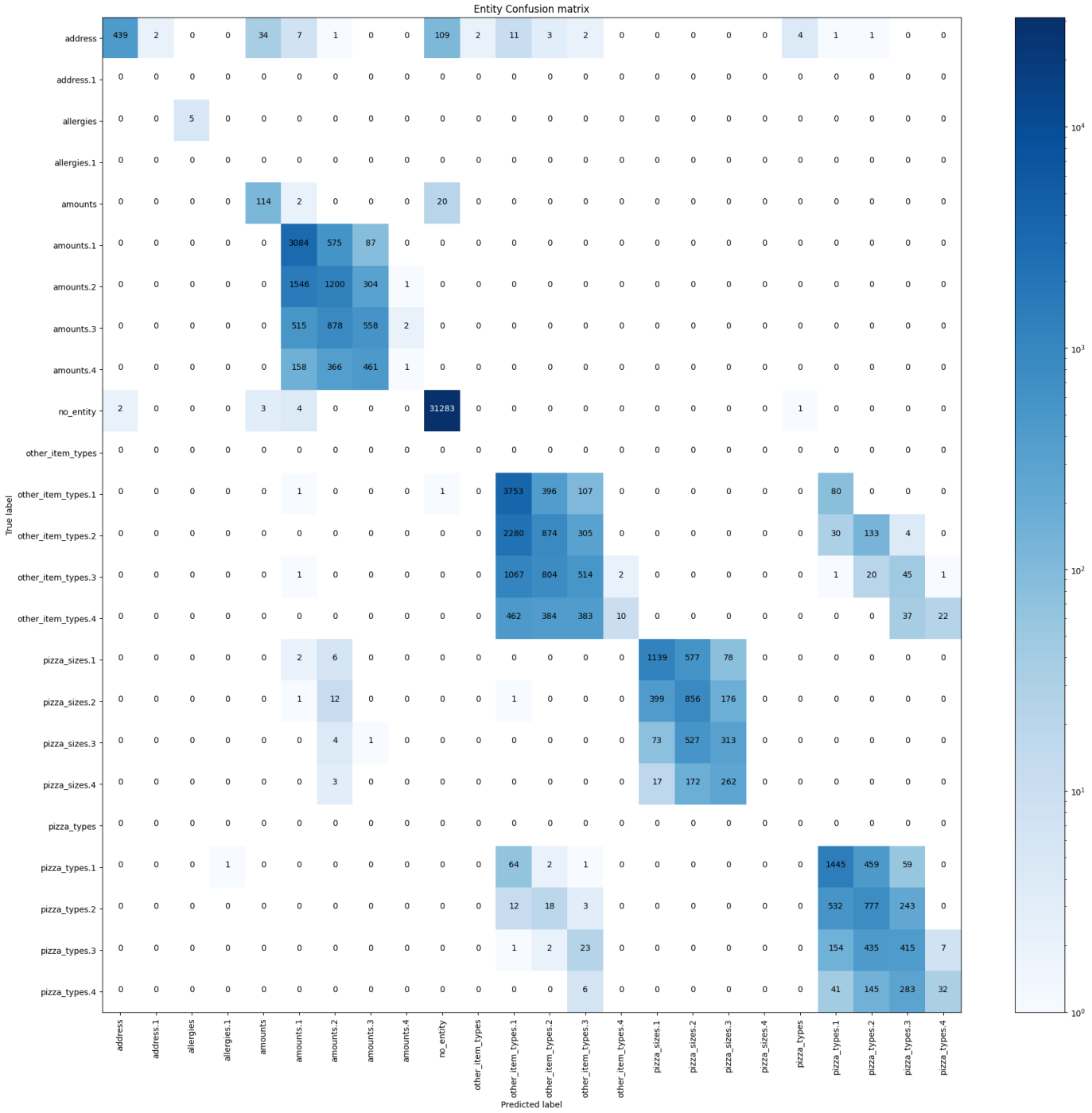

Fig. 4. DIETClassifier confusion matrix alternative pipeline

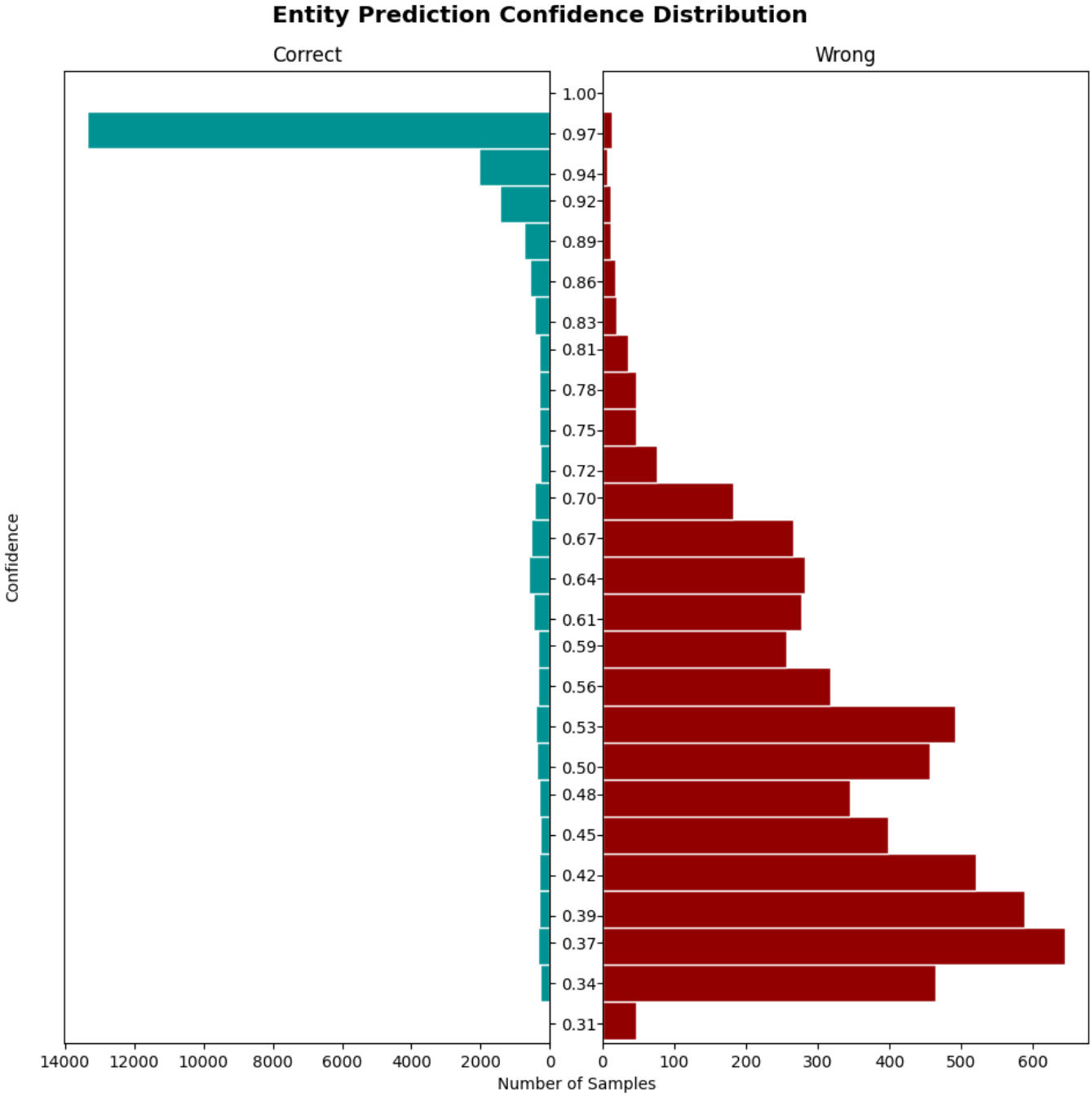

Fig. 5. CRFEntityExtractor entity predictions confidence histogram

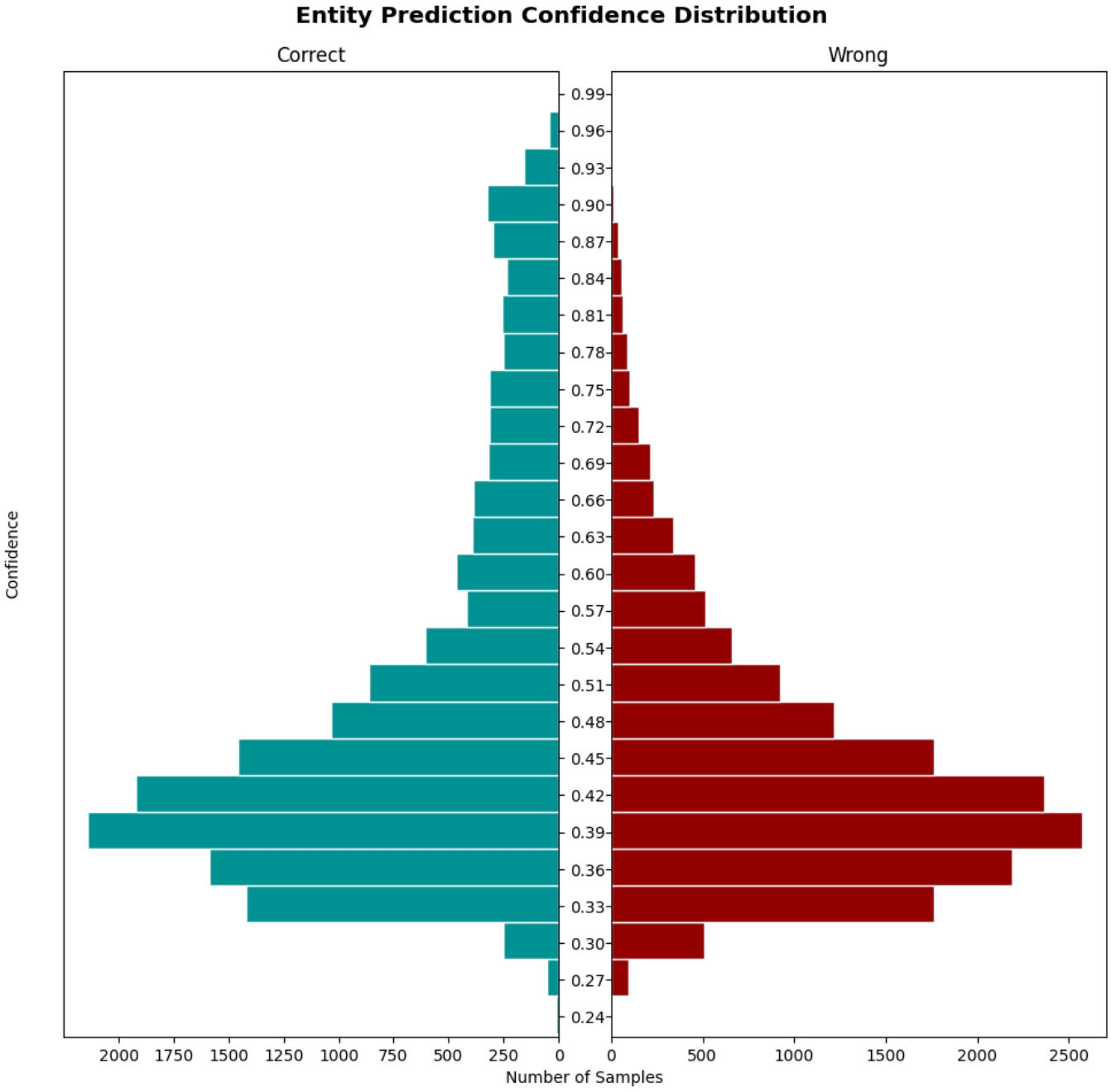

Fig. 6. DIETClassifier entity predictions confidence histogram

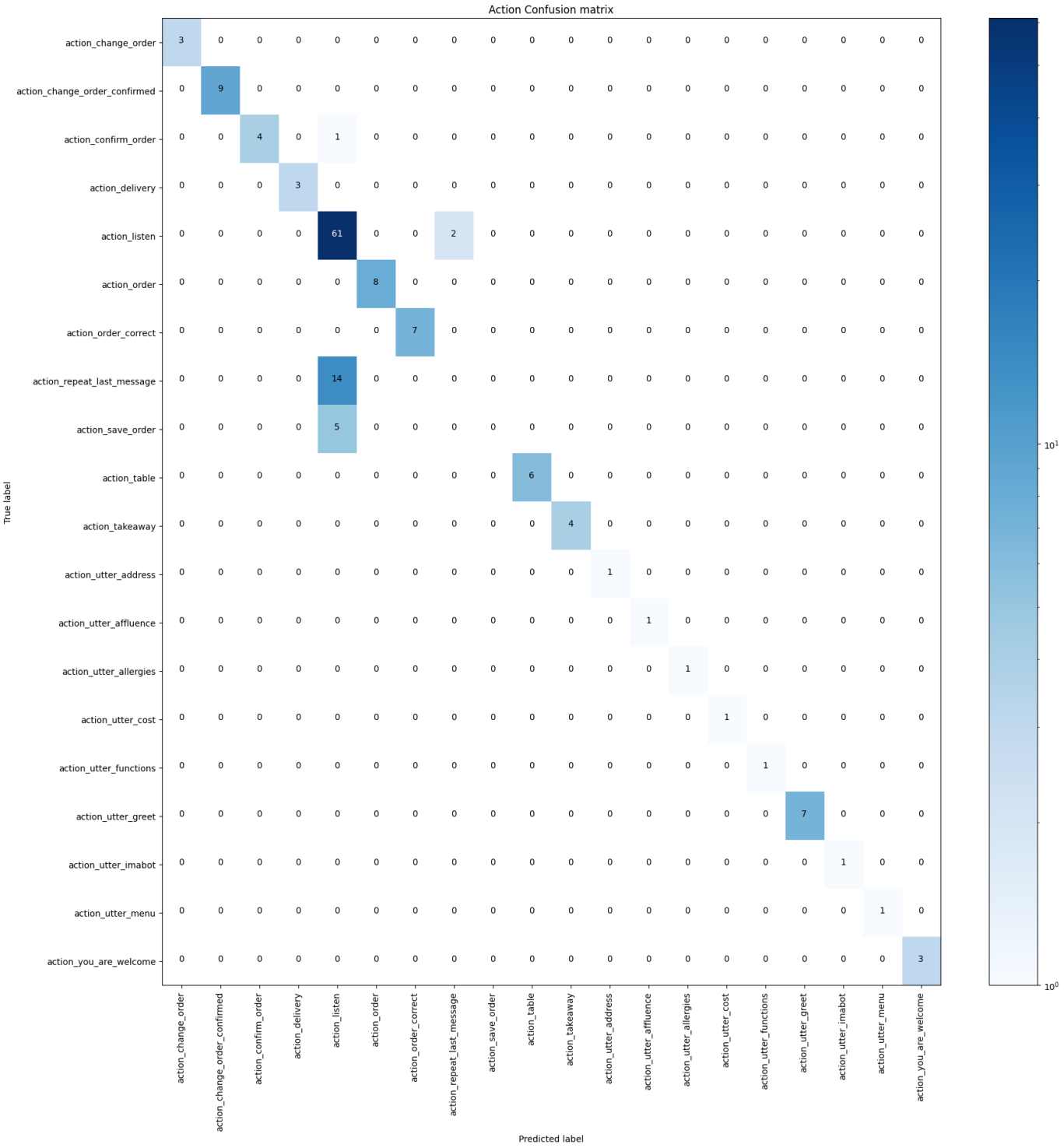

Fig. 7. Story confusion matrix main pipeline

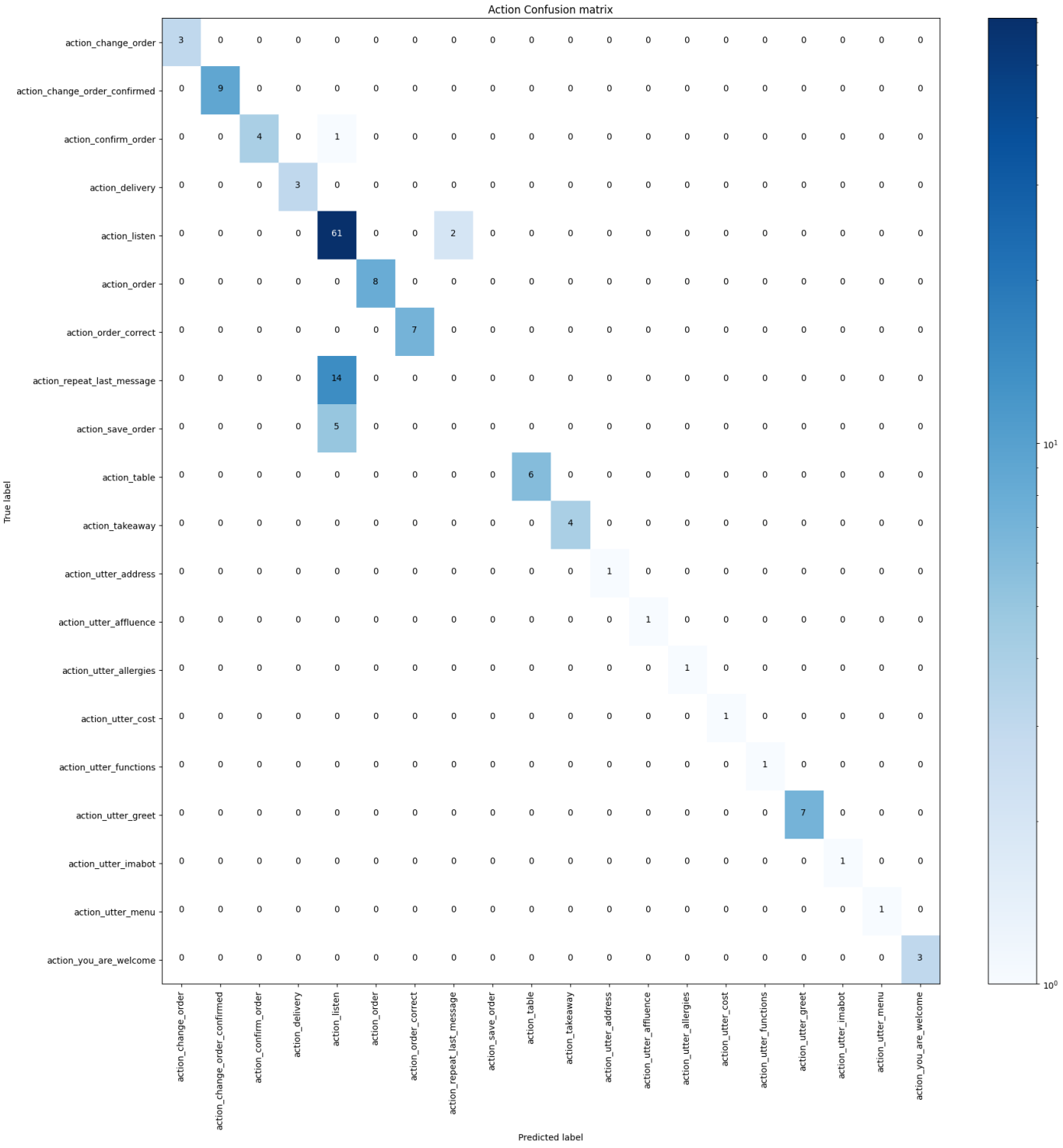

Fig. 8. Story confusion matrix alternative pipeline