ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ϯ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
Εθνικόν και Καποδιστριακόν Πανεπιστήμιον Αθηνών
ΙΔΡΥΘΕΝ ΤΟ 1837

ATHENA
Research & Innovation Information Technologies

M913 Dialogue Systems

Final Project: UniPal

University **Pal** – Academic Assistant

Kylafi Christina-Theano (Theatina) – LT1200012

Table of Contents

Introduction 2
Project Description 2
System Configuration 3
System Specifications 3
Features – Supported Scenarios 3
NLU Pipeline 4
Tokenization 4
Feature Extraction 4
Intent Recognition 4
Entity Extraction 4
DIET Classifier 4
Dialogue Policy 5
MemoizationPolicy 5
TEDPolicy 5
RulePolicy 5
Evaluation 6
Test Stories 6
Human Evaluation 6
Results 7
Configuration Comparison 7
Human Evaluation 11
Future Work 12
References 13

Introduction

Project Description

The task of the project at hand was to build a dialogue system (goal oriented chatbot) focused on a specific domain, able to support functions (scenarios) with the use of domain-specific intents, slots and custom actions (web crawling, information extraction, text processing, etc.).

In our case, **UniPal** [1] – the chatbot that was built is an **academic assistant**, able to fetch various types of information from the University (National and Kapodistrian University of Athens) Department's (Department of Informatics & Telecommunications) website (web crawling using custom actions), perform processing functions on the acquired data and then present them in a user-friendly, simple and comprehensible way. Analysis on the system's specifications is performed in detail in the sections that follow.

[1] GitHub Repository

System Configuration

System Specifications

Features – Supported Scenarios

Our bot has been trained to provide a number of university-related functionalities:

1. **Announcements**
   The user asks for the N latest announcements (default value defined in respective action – N: 10) posted in the department's website and then they are fetched and presented to the user in a "Title – URL – Details" format.

2. **Exams Schedule**
   Students can request the exam schedule for a specific type of studies programme (Postgraduate & Undergraduate), of a specific period (January, June or September) of a certain year. Then, the bot is web crawling the university's site, gets the proper schedule and gathers all the information on the exams (class name, date of examination & time of examination) which finally presents in a simple and comprehensible way (alphabetically sorted class names for easier class detection).

3. **Class Timetable**
   Same as above, where the class timetable for a specific studies type, semester and year is requested this time. The output displays information about each class (alphabetically sorted), including the class type, professors and weekdays/time/hall of the respective lectures.

4. **University Contact / Location / Access Information**
   When asked, UniPal collects and displays the department's contact, location and access information all together to the user.

5. **University Staff Contact Details**
   Additionally, users can get all the staff's contact information, (names sorted alphabetically), presented as "Name – Staff type – Email Address - Number" .

6. **Psychological Support Information**
   A small extra feature is the psychological support scenario, where based on the users' answer, UniPal tries to cheer them up using gifs (or cat photos) and if that does not help, the user is provided a link of the psychological support page of the university's website, displaying all the information needed to seek help.

In order to provide the aforementioned features, **custom actions** were created to perform the proper tasks and subtasks such as web crawling, data fetching, text cleaning, text processing, slot validation, slot reset, etc. included in the **actions.py** file.

NLU Pipeline

NLU components sequentially process the user input through the application of multiple functions (previous component's output is next component's input), aiming to extract useful information about user input intents and entities. Each component has a certain functionality and task to perform on the input text, explained in the following section.

Tokenization

To begin with, the input text is split into tokens. Depending on the input structure and possible special data formats, multi-intent prediction, etc., a variety of tokenizers could be used such as WhitespaceTokenizer, JiebaTokenizer (only Chinese), MitieTokenizer and SpacyTokenizer.

Feature Extraction

The next step of the NLU pipeline is to create feature representations for the model training phase. Feature extractors can return sparse or dense feature vectors, while both types return sequence and sentence feature representations. Then the classifier (next component) can make use of those features (whichever type suits them) to perform the intent classification task.

Intent Recognition

In this component the task of intent classification is performed, where an intent from the chatbot's domain is assigned to an incoming user message. Some classifiers make use of feature vectors (e.g. LogisticRegressionClassifier, SklearnIntentClassifier) produced in the preceding steps (sparse or/and dense types), while others extract their own features (e.g. MitieIntentClassifier) or search for keywords (e.g. KeywordIntentClassifer).

Entity Extraction

Entity Extractors are assigned the task of extracting entities such as name, location, date, etc. from the user input text. They are implemented using various architectures and models such as SVM (Support Vector Machine), CRF (Conditional Random Fields), statistical BILOU transition model, regex, etc. Some of them can extract specific types of entities such as synonyms, dates/distances/amounts of money. Use of multiple extractors can lead to duplicates, therefore each one should focus on different types of entities.

DIET Classifier

Dual **I**ntent and **E**ntity **T**ransformer (DIET) is a multi-task architecture, performing both intent recognition and entity extraction. It provides the ability to simply use various pre-trained embeddings like BERT, GloVe, ConveRT, etc. and try different model configurations through parameter value experiments.

In our case, DIET classifier was used both for multi-tasking (intent classification & entity recognition) and for entity extraction only in the different NLU pipeline experiments, trained for 200 epochs with the application of sigmoid cross-entropy loss over the similarity terms (constrain_similarities: True), which results in better model generalization to real world test sets as mentioned in the documentation. As far as the other parameters, the default values were kept.

Dialogue Policy

The decision of the conversation steps that the chatbot will take, is a task performed using dialogue policies. Multiple or a single policy can be used, predicting the bot's next action with a confidence value and depending on that value and the number of policies in the configuration file that next action is then decided. The policy with the highest confidence "wins", while there is a priority if they come to a tie (RulePolicy > MemoizationPolicy | AugmentedMemoizationPolicy > UnexpecTEDIntentPolicy > TEDPolicy).

The policies that are used in our configuration files are the following:

MemoizationPolicy

A machine learning based policy, keeping a history of N previous turns to match any of the stories in the training data. It can predict an action with either 1.0 (from matching story) or 0.0 (None) confidence.

TEDPolicy

The **T**ransformer **E**mbedding **D**ialogue (TED) policy is a multi-task architecture for next action prediction and entity recognition, consisting of transformer encoders, shared for both tasks. Feature vectors of useful information (e.g. processed user input text, bot's system actions history, etc.), extracted in the previous steps or the proper transformer encoders are concatenated and fed to the dialogue transformer encoder. Then, embeddings of the dialogue and system actions are produced by the dense layers applied, whose similarity is calculated. Finally, the user sequence and the dialogue transformer encoder embeddings are concatenated and CRF (Conditional Random Field) is applied for contextual entities prediction.

In our experiments, we have tested this policy with 3 different max history parameter values, trained for 100 epochs, using constrain_similarities: True (mentioned in DIET classifier above).

RulePolicy

This is a rule-based policy, responsible for handling fixed-path parts of the conversation and its predictions are based on the manually defined rules in the training data.

The policy experiments that were performed, used 3 different core fallback threshold values (the confidence threshold under which the core fallback action is applied).

Evaluation

The final system versions (parameter/configuration experiments) were evaluated based on both system tests (using test stories) and human evaluation (questionnaire after user-bot interaction/scenarios). There is no standardized way of measuring the quality of such virtual agents, therefore we have established our own standards specifically for this project, focusing on the system's task completion efficiency domain-specifically.

Test Stories

Test stories were created (**9** test cases – written in /test/test_UniPal_stories.yml) for both the configuration experiments and the system evaluation process as follows:

1. Latest University Announcements (**1** test case)
2. Classes Timetable (**2** test cases)
3. Exams Timetable (**2** test cases)
4. Contact / Location / Access information (**1** test case)
5. University's Staff information (**2** test cases)
6. Psychological Support information (**1** test case)

Configurations

NLU pipeline and **Dialogue Policy experiments** were carried out (3 different cases each), with the respective configurations written in separate files (NLU experiments: config1_pipeline.yml, config2_pipeline.yml, config3_pipeline.yml | Policy experiments: config1_policy.yml, config2_policy.yml, config3_policy.yml)

After the NLU pipeline comparison, it was decided to proceed using the 2nd data processing pipeline (best f1-score on test set as shown in the results presented in section **Results**) to perform the policy comparison.

After the test stories evaluation process, the final 3 models (3 different policy combinations) will be given to human testers for further evaluation, the results of which are presented in the following section.

Human Evaluation

After the test stories experiments & testing, human evaluation is performed. This type of testing usually focuses on some aspects not able to be properly checked through system tests, such as the chatbot's effectiveness, efficiency and satisfaction (user satisfaction – turn/session scope, functional simplicity, matters of ambiguity, flow, etc.). In addition, this is the core of the usability definition, resulting in the confinement of discrepancies between evaluation methods in the past.

For this matter, a questionnaire was developed for the users after their interaction with the different versions/configurations of the chatbot, presented in the Results section below.

Results

Configuration Comparison

NLU Pipeline Testing

The results of the NLU pipeline experiments are saved in the respective results directories (results_config1_NLU, results_config2_NLU & results_config3_NLU), where the entity & intent confusion matrices, histograms, reports and errors are logged. The comparison results (2 runs, 2 story exclusion percentages) are stored in the “results” directory. Some of the aforementioned experiments results are shown in the following tables and graphs.

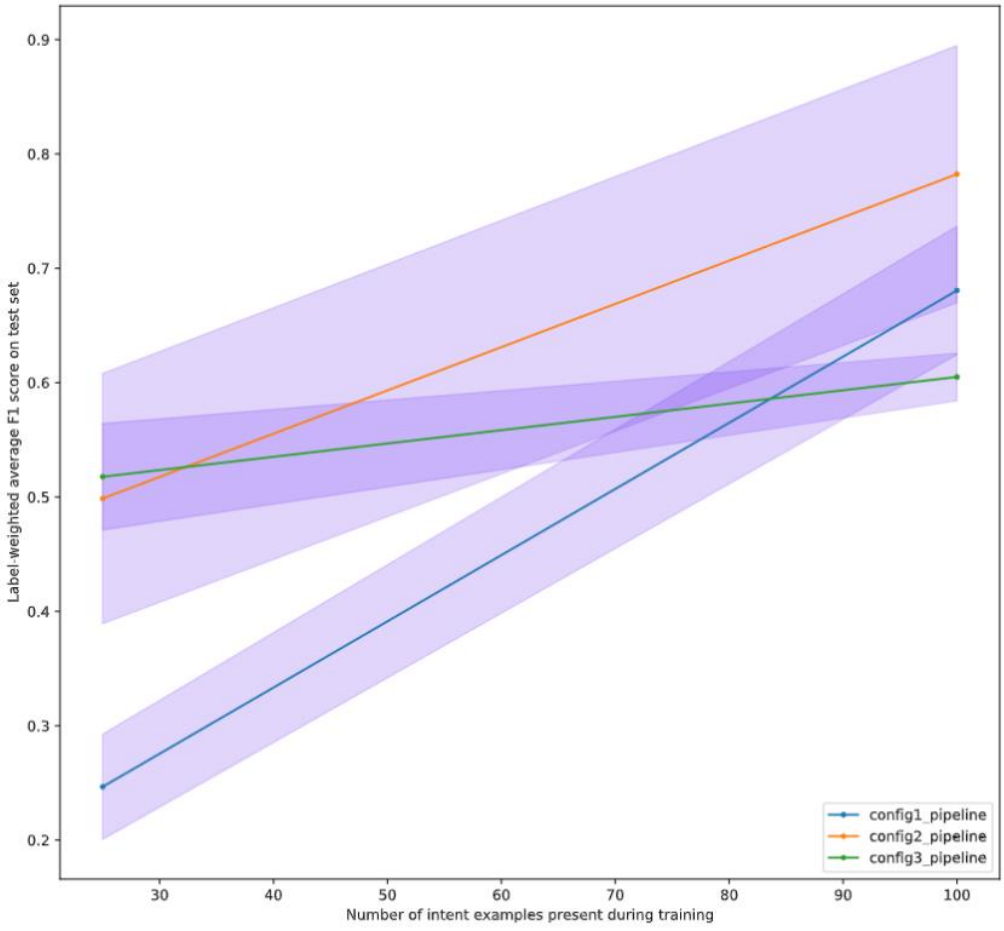

Figure 1. NLU model comparison

As supported by Figure 1, it seems that more intent examples would optimize the models’ performance measured in F1 score (especially of the first 2 configurations due to their gradient).

Based on the **entity/intent** confusion matrices, histograms and reports, the following observations were made:

- No **entity** confusion was detected (all the three configurations had an accuracy of 1.0)
- On **action** level, the 1st configuration had an accuracy of 99.2% (131/132), while the other 2 predicted all actions correctly (100% accuracy – 132/132)
- After **entity** histogram comparison (Figure 2), it is derived that in the first 2 configurations the average confidence is high, while in configuration 3 the average is lower (with a greater value range)

- The **intent** confusion matrices (Figure 3) show great differences between the 3rd and the other 2 configurations.

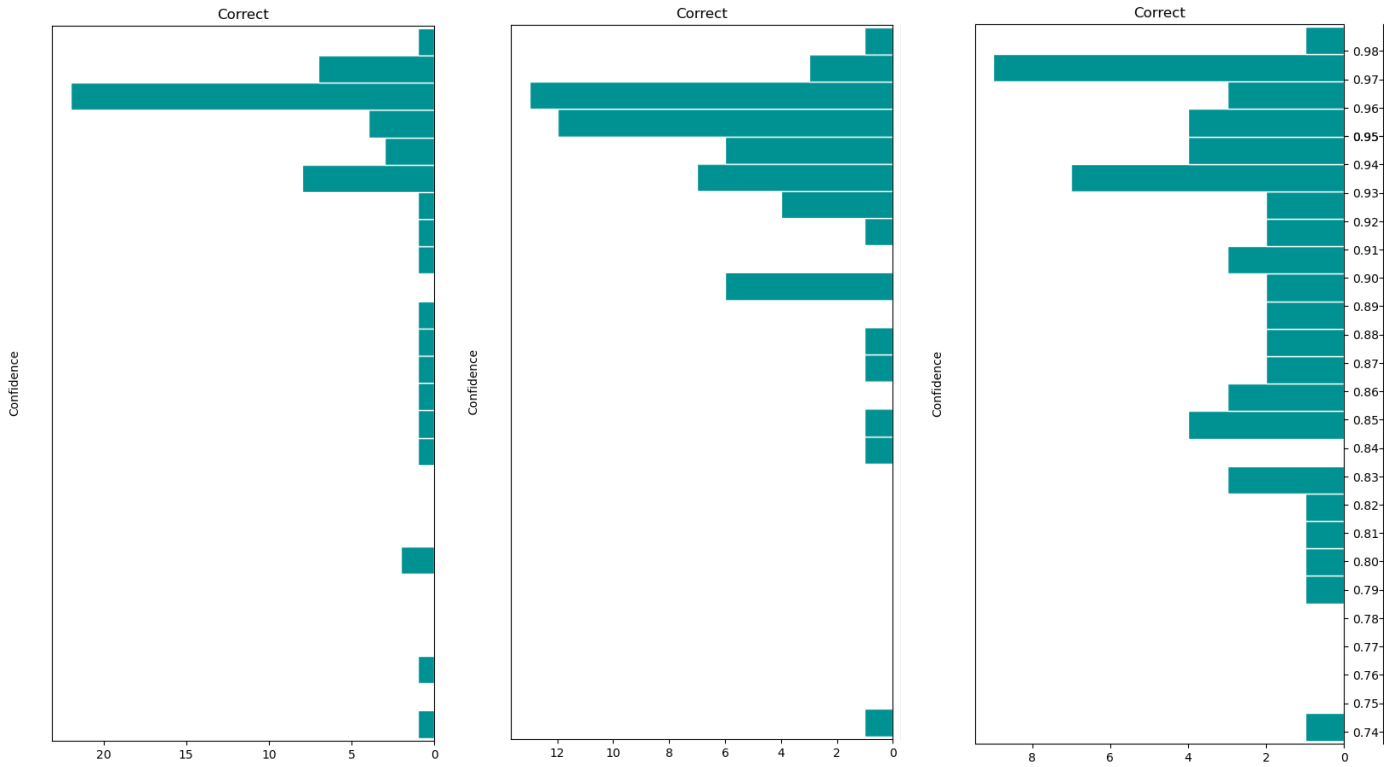

Figure 2. Entity histograms of configurations 1, 2 & 3

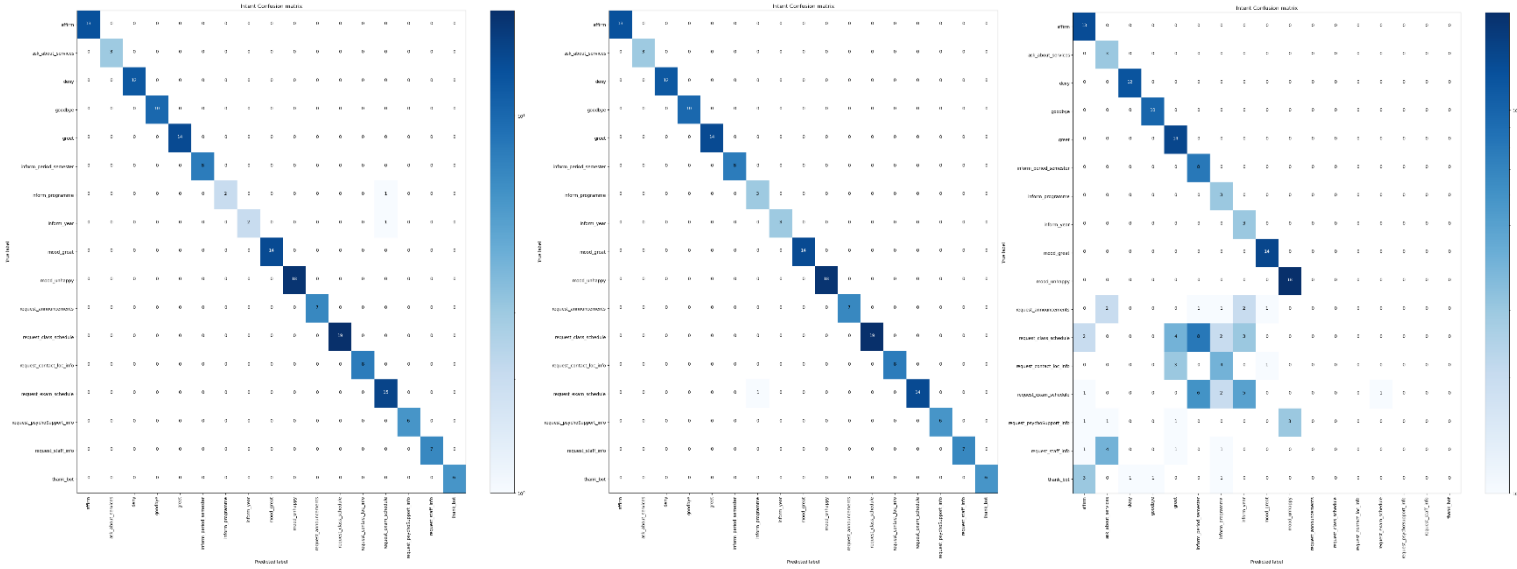

Figure 3. Intent confusion matrices of configurations 1, 2 & 3

Policy Testing

The results of the Dialogue Policy experiments are saved in the respective results directories (results_config1, results_config2 & results_config3), where the policy & story confusion matrices, histograms, reports and errors are logged. The results of the comparisons carried out (3 runs, 5 story exclusion percentages) are stored in the “comparison_results” directory. Some of those results are presented below.

Conversation level evaluation :

Run -- Story exclusion %	Configuration 1	Configuration 2	Configuration 3
1 – 0	7	9	9
1 – 5	9	9	9
1 – 20	5	6	8
1 – 50	6	4	5
1 – 95	0	0	0
Run 1 mean	**5.4**	**5.6**	**6.2**
2 – 0	9	9	9
2 – 5	9	8	8
2 – 20	7	8	9
2 – 50	5	5	6
2 – 95	0	0	0
Run 2 mean	**6**	**6**	**6.4**
3 – 0	9	9	9
3 – 5	7	5	8
3 – 20	7	6	8
3 – 50	8	6	6
3 – 95	0	0	0
Run 3 mean	**6.2**	**5.2**	**6.2**
All Runs MEAN	**5.87**	**5.6**	**6.27**

Table 1. Stories predicted correctly (out of 9) by the three policy configurations

Action level evaluation :

Metric	Configuration 1	Configuration 2	Configuration 3
Accuracy	0.99	1.00	1.00
$\text{Precision}_{\text{macro}}$	0.93	1.00	1.00
$\text{Recall}_{\text{macro}}$	0.94	1.00	1.00
F1_{macro}	0.93	1.00	1.00
$\text{Precision}_{\text{weighted}}$	0.99	1.00	1.00
$\text{Recall}_{\text{weighted}}$	0.99	1.00	1.00
$\text{F1}_{\text{weighted}}$	0.99	1.00	1.00
$\text{Precision}_{\text{micro}}$	0.99	1.00	1.00
$\text{Recall}_{\text{micro}}$	0.99	1.00	1.00
F1_{micro}	0.99	1.00	1.00

Table 2. Metric values of the three policy configurations results on action level

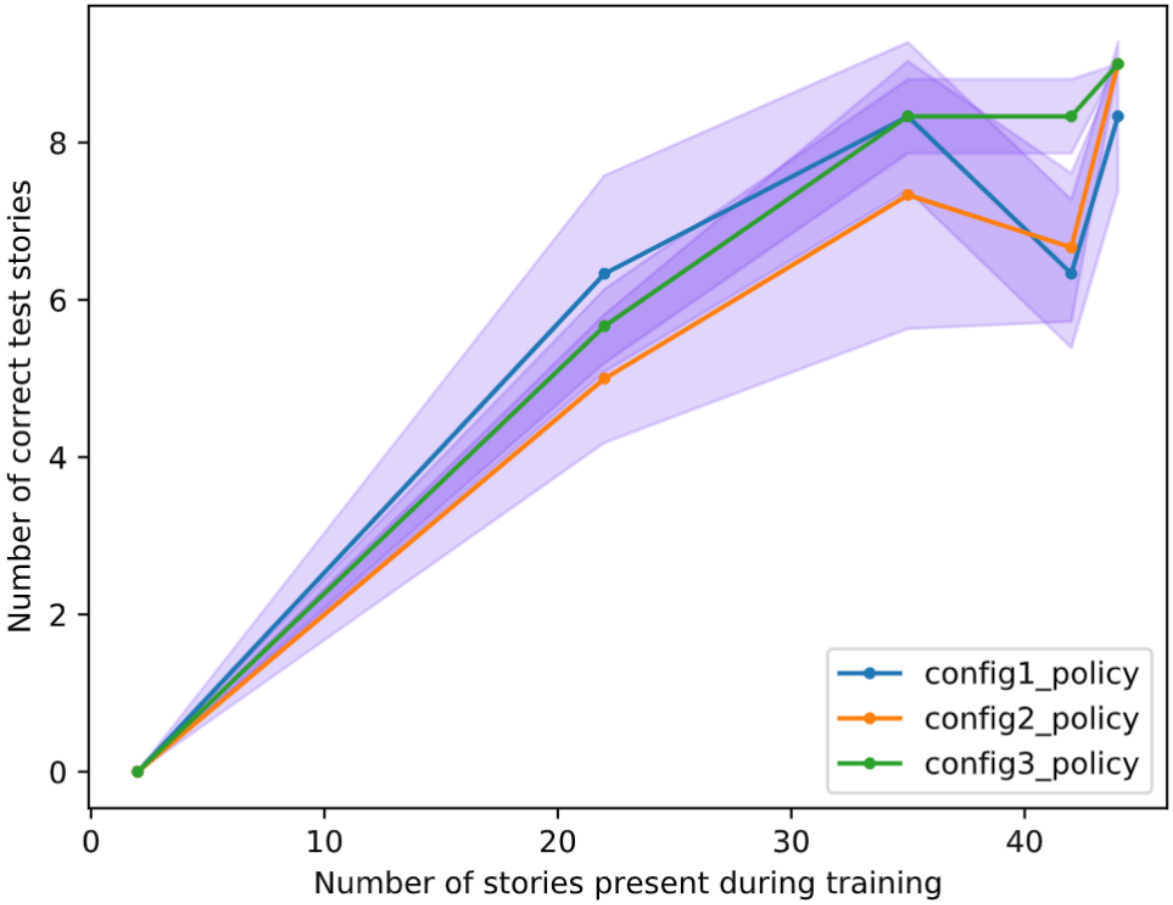

Figure 4. Core model comparison graph

Same as in the case of NLU pipeline comparison graph, it is derived that more stories would optimize the models' performance (ascending gradient in all configurations) .

Finally, based on the **story** reports and logfiles, it was observed that in the 1st configuration there was one failed story, while in the other two no failed stories were detected.

Human Evaluation

The final models that were selected after the previous evaluation processes, were tested through human evaluation as analysed in section Human Evaluation.

Tasks

The users were asked to perform short tasks via chatting with the bot, such as requests on timetables, university information or/and location, N latest announcements, etc. , covering all possible scenarios supported by UniPal chatbot.

Questions

To assess the task-oriented interaction between the user and the chatbot, the user evaluated multiple aspects on turn or conversation level, such as goal completion, user friendly approach, overall user satisfaction, etc.

The post interaction assessment questions on (overall) **conversation** or/and **turn** level were the following :

1. Was the final/sub task(s) completed?
2. Was the final/sub task(s) completed promptly?
3. Was the final/sub task(s) completed successfully?
4. Was the conversational flow smooth/natural?
5. Have you encountered any flow blocking obstacles resulting in a non-functional interaction?
6. If you made any typos, has the bot detected the correct words?
7. Could you take the chatbot for a real person?
8. Would you use UniPal's services personally (daily/weekly/monthly basis)?
9. Would you recommend UniPal to a friend?

Future Work

Although much work has been done for the development of the task-oriented chatbot at hand, there is always some room for fixes, optimization and further research, experimentation and analysis, such as the following :

1. In actions action_uni_class_schedule & action_uni_exam_schedule, as mentioned in the respective logfile (Timetables_accepted.pdf/.xlsx), some file formats are not working well with the data structures created for the actions, therefore it could be fixed by expanding the file format cases in the actions code (actions.py file)

2. Google calendar reminders could be added for exams, classes, project deadlines, etc.

3. More stories especially via Rasa's interactive mode would be able to boost the bot's confidence and remove ambiguity in complex dialogue cases

4. Multi-intent classification could be added to the system, for example request_exam_schedule & inform_programme could possibly be combined to create better rules and stories (conversation flows)

5. Finally, intentional typographical errors could be added to messages, intents, entities, etc. in test stories, aiming to crash test the system's robustness

References

(Articles & Papers the project at hand was based on)

1. Trends & Methods in Chatbot Evaluation
2. Slack API: Applications | UniPal Slack
3. How to call custom action from another custom action - Rasa Open Source - Rasa Community Forum
4. Medicare Locator
5. Connect your chatbot with google calendar | The Rasa Blog | Rasa
6. Forms
7. NLU Training Data
8. SimGus/Chatette: A powerful dataset generator for Rasa NLU, inspired by Chatito
9. Chatbots Using Python and Rasa - GeeksforGeeks
10. How to create a FAQ Chatbot with Rasa?
11. Introducing DIET: state-of-the-art architecture that outperforms fine-tuning BERT and is 6X faster to train | The Rasa Blog | Rasa
12. Model Configuration
13. Testing Your Assistant