Designing Automatic Conversational Testing for Task-Oriented Voice Bots

Company Project - NPRG071
Kirill Semenov
Project Supervisor: Mgr. et Mgr. Ondřej Dušek, Ph.D., ÚFAL
Company Consultant: Jan Cuřín, VP, NLP & Analytics at The MAMA AI

CS - Language Technologies and Computational Linguistics (ÚFAL),
Faculty of Mathematics and Physics, Charles University, Prague
June 10, 2023

Introduction

My company project at MAMA AI[1] aimed to create a prototype for automated end-to-end conversational testing of the company's bots. The company is developing task-oriented voice bots that cover a wide range of topics, from business and customer service to onboarding in companies and help for refugees; the main mode of accessing the voice bots is through telephony.

As the company is young, dynamic and intensively gaining customers, there is a need for robust testing infrastructure. With newly implemented features it is crucial to do automated regression tests of the existing production systems and discover potential incompatibilities introduced by the updates. The unit tests are in some situations insufficient as they do not cover the whole chain of components of the bots; manual testing is time-consuming. Because of that, the company is interested in creating a user simulator - a specific bot that pretends to be a user and is supposed to communicate with the bot we want to test instead of a human. This user simulator would be supposed to do "smoke testing" to see if the bot works and collect statistics about every call to provide the company with possible deviations. Thus, I was assigned this task for my internship.

[1] https://themama.ai/

The initial plan of the project was to create a bot that can react to the "basic" types of utterances (i.e., those which are met in any dialogue - greeting, saying goodbye, yes/no, thanking, etc.). Throughout the project, it became clear that it is technically easier to develop testing bots for a few specific bots and then generalize them to others. Therefore, in the present report, I will show the results of my work on creating a user simulator for two company products. The report is organized in the following way. In Section 1, I present the theoretical overview of the current state of the art in the field of user simulators and voice bot testing, including various approaches to manual and automated testing and the metrics used for that. Section 2 describes the building process of the testing bot, including the improvement of the whole architecture of MAMA AI libraries, as well as the resulting form of the simulator. Section 3 explains the plan for the statistical analysis of the results of the project. In Section 4, I speculate on the perspectives of the user simulator project for the company. The Appendix presents the outputs that the user simulator generates for the MAMA AI infrastructure.

1. Theoretical Analysis

1.1. Overview of the User Simulators

A user simulator (US) in NLP is a dialogue system (DS) that pretends to be a user in order to evaluate or train a bot we are interested in (Deriu et al., 2021). As we are primarily speaking about the industrial applications of bots, from now on, the bot that is tested will be called the production bot (or PB). This testing approach was created to fulfill several purposes, mainly to automate the routine checking of the production bot functions, especially in the case of continuous integration, as well as to discover the weaknesses of the PB that a human developer cannot predict. Recently, with the development of the reinforcement learning-based DS, the US also serves as a training environment[2] for the PB (Deriu et al., 2021). The ultimate goal of a US is to run an external, end-to-end testing of a bot, so a "proper" US should take a textual input and release a natural language output. However, sometimes it is not feasible, and in this situation, the US works on the level of intents (getting them both as input and output), which makes the testing procedure easier but excludes ASR and NLU modules from the checks. By this parameter, we can classify the US into language-level (or surface-level) and intent-level (or semantic) ones.

The first USs were created in the early 2000'ies, e.g. (López-Cózar et al., 2006), and were completely rule-based. Then, retrieval-based US were used, see (Keizer et al., 2012): in this case, they used template-based grammars and a corpus that allowed them to fill in the gaps with appropriate expressions. With the increasing popularity of neural networks, various neural approaches were used in the field, starting with (Kreyssig et al., 2018). As reinforcement learning (RL) started being used in NLP and DS, the next wave of the US popularity is related to the use of USs as environments for training the bots, not only evaluating them: see (Liu et al., 2022), (Chi et al., 2022) or (Dockes, 2021) - the latter is based on the generative adversarial networks training architecture, but the aim is the same - to use US for training, not for evaluation. It is worth noting, however, that even the rule-based USs were used for training earlier PB architectures. Concerning the internal structure, the experiments are run mainly with BERT-based (Devlin et al., 2019) user simulators, which take as input the tuples of the natural text, semantics (intents and values), and other meta-information, and return either only

[2] In RL, the environment is a notion of a component of the training system that takes the output of the system, possibly changes and returns the reward based on the previous system output. Based on the environment changes and the reward, the RL system decides which next step to make.

the natural texts or semantic representations. Examples of a task-oriented user simulator of that kind are presented in (Lin et al., 2022), and the generalized version of that is presented in (Lin et al., 2021).

Throughout the last two decades, not many open frameworks for US construction have been suggested. The agenda-based user simulator approach (ABUS) (Schatzmann et al., 2007), according to the literature, although 15 years old, is still the most widespread one. Its main principle is the following: the simulator randomly generates a goal (i.e., a set of constraints and requested pieces of information that it wants to provide to and get from a PB), and based on that goal, a stack of the intents is generated. This stack is called agenda and is dynamically adjusted depending on the inputs from the PB (for instance, if a PB provides the wrong information, an intent "negation" will be added to the top of the agenda). The initial paper also contained a statistical implementation of the model; however, there are newer neural implementations of the same approach, for instance, a Deep Q Learning-based one (Li et al., 2016). Also, the initial paper suggests the intent-level operation of the system, but the later implementations within this framework also include the end-to-end language level US. In recent years, alternative frameworks (together with implementations) were introduced, namely, the aforementioned Neural user simulator, or NUS (Kreyssig et al., 2018), and the task-oriented user simulator, or TUS (Lin et al., 2021). NUS approach differs from ABUS by, firstly, operating on the language level by default, and secondly, by being based on an annotated in-domain corpus; these features make this approach better in terms of variability of the outputs and amount of manual work required, but on the other hand, they make the US less interpretable. The TUS approach tries to retain the advantages of NUS by using the neural model with a pretrained language model; but to minimize the hallucinations and increase the interpretability, the inputs and outputs to the model are the semantic representations of the domain, intents, and slots with values (optionally, the output can also include the natural language utterance). So far, TUS seems to be the most developed approach to the US architecture, but according to the experiments provided in the paper, it does not outperform ABUS (although requiring fewer data for fine-tuning); it is also more computationally demanding than statistical and neural implementations of ABUS.

The works presented above mainly tackled the academic analysis of the US; however, when it comes to industry, there are only few industrial products that provide user simulators. The full-fledged packages for user simulation are represented by Selenium[3], Klearcom[4], PyDial[5], and Cyara[6] (former Botium), as well as the testing functionality in the widespread dialogue managers like Watson Assistant and DialogFlow CX (Wang et al., 2022). However, in most cases, the range of available features is relatively narrow: usually, they allow developers to use only handcrafted examples for "hardcoded" cases of regression testing; thus, there is no way of testing "blind spots" based on randomization and scenarios unexpected by the PB developers. Because of the handcrafted nature of the simulators, the products are expected to be used only for testing, not for RL training. On the other hand, a significant advantage of the infrastructures mentioned above is their comprehensive toolkits of statistics and visualization of the testing outcomes. The latest product we know, BotSIM[7] (Wang et al., 2022), is aimed at combining all advantages of the existing bot as well as adding new functionalities to them: it consists of three components: generator (ABUS-based automatic model that generates scenarios based on training corpus and, thanks to T5 paraphrasing module, can produce varying phrases for the same intent); simulator (an end-to-end, i.e., from ASR to NLG, a module of the

[3] https://ghostinspector.com/landing/selenium-testing/
[4] https://www.klearcom.com/
[5] https://pydial.cs.hhu.de/
[6] https://cyara.com/
[7] https://github.com/salesforce/botsim

dialogue system); and remediator (an interactive environment with visualization of statistics of the testing sessions and suggestions on improving the PB).

In a real-world situation, not only the set of informational constraints and requests can differ between the users, but also their behavior given the same goal. To model that, an ideal US should be able to generate various "personalities" for testing that. For instance, the users can differ by cooperativity (one user will tend to provide as much information as possible without prompts, while the other will respond laconically), as well as rudeness and other dimensions. However, there seems to be no substantial research in that field. The only work we could find that investigates the cooperativity level in detail is (López-Cózar et al., 2006), where they created three types of US: very cooperative (which answers all questions of the PB in the way expected by the PB developers), cooperative (which answers the questions appropriately but sometimes does not provide all the information on the first request), and not very cooperative (which sometimes answers inappropriately). The authors compared the performance of two PBs on the three types of user simulators and showed that one of production bots was performing significantly worse with the not-very-cooperative US, while another one had similar metric values; this means that some PB models can be more robust to a wider variety of users even if those do not behave as expected by the PB developers. Apart from this work, little research focuses on this problem; even the theoretical formalizations of "cooperativity" or "rudeness" are not operationalized well, and there are no full-fledged frameworks for analyzing these parameters. In recent works, the experiments with personality are mentioned, but usually as a future perspective, for instance, in (Lin et al., 2022).

1.2. Overview of the Metrics Applied by User Simulators

The range of evaluation techniques for testing dialogue systems is broad, but only its subset can be (or usually is) applied by the US. Moreover, manual and US testing approaches have diverged over the last few years. For instance, the Chatbottest.com[8] project, which aims at generalizing the testing techniques and making them consistent for all possible task-oriented PBs, suggests using its guide to assess a PB performance manually; in most cases, it consists of the instructions (for instance, "Repeat a message to the chatbot four times in a row, try with something easy like "hi", "thank you", or "goodbye"), and questions drawing tester's attention at describing a US's behavior (for instance, "Does it have different answers for the same sentence?"). The spectrum of the features covered within the questionnaire is broad, which makes it a good starting point for testing; however, most of the questions are formulated in the yes/no form or expect a detailed response. The first type of questions thus resembles the primitive unit tests that do not require a full-fledged US, while the second one is too hard to be formalized by an automated bot and requires human assessment. Thus, the fields of human and automated testing are formalized and applied in different ways, so to understand what can be used as a metric for a user simulator, we need to classify the multitude of metrics and subset the most appropriate ones.

In a number of works, for instance (Li et al., 2021), the set of evaluation methods is divided into three groups: automatic evaluation, human evaluation, and US-based evaluation. The automatic evaluation there is subdivided into language evaluation (measuring the fluency and entropy of the words, or the sentence embeddings of the utterances), and task-oriented metrics such as task-success rate and dialogue efficiency, which will be discussed in detail later. The US-based evaluation refers to the intent-level (for instance, ABUS) and surface-level (for instance, NUS) evaluation frameworks we

[8] https://github.com/chatbottest-com/guide/wiki

have already discussed. The human evaluation tackles more “deep” features of the dialogue, for instance, consistency or engagement, that is by now hard to formalize and thus is assigned to annotators’ judgments. However, the tripartite differentiation seems inconsistent because the user simulator frameworks are actually based on the automated metrics enumerated in the paper; the US rather represents a level of organizing the testing routine of the dialogues with PB’s, and only then the metrics (mainly the ones described here as the automated ones, like task-success rate) will be applied to it.

A more profound and consistent classification approach is presented by (Deriu et al., 2021), where several dimensions of the automatic evaluation are shown. Firstly, the metrics can be subdivided into extrinsic (evaluation of an end-to-end system with respect to a goal of the dialogue) and intrinsic (evaluation of the system components, e.g., automatic speech recognition (ASR) or dialogue state tracking (DST), within a system). The authors (and us) are more interested in the extrinsic ones.
Secondly, when it comes to automating the evaluation, we can choose what to approximate: the formalized (language- or intent-level) and easily-calculable features, or we try to predict the “subjective” human estimates of the dialogue quality. The former approach is called user simulation, and we will cover it in more detail later; the latter is called “user satisfaction” and is arranged as a supervised machine learning task where the algorithm trains on the dialogues as inputs and their human estimates as outputs.

When it comes to automated extrinsic metrics for user simulation, the two main types of evaluation are applied: on the one hand, we measure how well the task was accomplished (how much necessary information was accepted/provided, was the end of dialogue reached) - this is called task success rate. The task success rate tends to be the primary metric type for the bot evaluation. Alternatively, we measure the “cost” of a dialogue run (measured in time, number of turns, and so forth) - this is called dialogue efficiency. The task success rate is usually formalized similarly to the goal in ABUS - it consists of a set of requests and constraints needed to be fulfilled. We can create numerous custom functions to measure task success rate - the number of correctly identified intents/entities, the ratio of correctly provided pieces of information, etc.

1.3. Overview of Intent Classification Approaches

The project's initial idea included creating a US for detecting the “basic”, bot-independent phrases such as greeting, apologies, thanking, saying yes and no, etc. However, we needed to formalize the definition of a “basic” intent. To do that, we analyzed several widespread intent taxonomies.
The first one and one of the most widespread in the industry is CUED (Young, 2009). It suggests approximately 40 types of intents depending on their semantics and the attributes (slots) each intent can have. There is a one-level hierarchy of the intents: all intents fall into four types - information providing; query; confirmation; and housekeeping. The closest type to what we need is housekeeping, as it contains intents like “hello()”, “thankyou()”, “bye()”, “repeat()” or “help()”. However, the confirmation type (which includes not only confirmation but also negation) is also partially related to what we need: some of the intents in this group can take on attributes (like “negate(a=x, b=y)”), while others can be used without valencies (like “confirm()” and “negate()”), thus making them “simple enough” for us to be interested in them. However, this emphasizes the fact that yes-no utterances have an inherently different structure compared to other “housekeeping” utterances because, by default, the yes-no utterance assumes arguments they agree or deny; when omitted, these arguments are just assumed to be contained in the context.

The second taxonomy we analyzed (Stolcke et al., 2000), is based on the DAMSL dialogue act annotation system, with some adjustments. This system also represents the "flat" hierarchy of 42 intents, categorized into several classes: statements, questions, backchannels, turn exits, answers, and agreements. Several intents (the ones we are primarily interested in), like GREETING and APOLOGY, were not included in any of the groups. In this classification, we can see that the yes/no answers (which are assigned to the answers and agreements type) are again separated from the non-categorized "general" utterances. Moreover, there is a separate group of backchannels (utterances like "okay," "I see," that demonstrate listener's engagement in the dialogue) which is considered to be significant enough to be assigned a different type. Most probably, the reason for that is the frequency distribution of various intents: according to the statistics provided in the paper, the number of utterances containing backchannel type intents amounts to 19% of the whole Switchboard corpus (corpus of human telephone conversations), while the intents CONVENTIONAL-OPENING (greetings), APOLOGY or THANKING are found in less than 0.1% utterances each.

The last approach we would like to cover here is called DIT++[9]. It is a project by Tilburg University, first presented in (Bunt, 2009) and then developed into the ISO 24617-2 standard (Bunt et al., 2020), which tries to introduce a deeper linguistic approach towards dialogue analysis. The taxonomy of intents (which are called "communicative functions" there) is based on Dynamic interpretation theory (DIT), with some borrowings from DAMSL. Contrary to those mentioned above, this taxonomy is deeply hierarchical. Because of a more functional perspective (based on linguistic semantics and pragmatics), rather than a "surface" one (based only on the utterances), the grouping of the intents is quite different; in particular, the "general" functions are scattered around various types of intents at different depths. For instance, greeting and apologizing phrases belong to the "social obligation management" class, pausing (phrases like "please wait a minute") belongs to turn management, and "yes-no" relate to the "information providing" class. Thus, although being more profound from the theoretical perspective, this grouping was not very well applicable to the tasks of our project.

2. Building the Testing Bot

2.1. Defining the Task

Before the beginning of the work, we expected the user simulator to have the following features:

1. recognize and generate "basic" intents and utterances; all PB's within the company will be tested with these "basic" utterances;
2. test the English-speaking bots;
3. test the PB's in an end-to-end manner (it should make phone calls to the telephone numbers of the bots and talk to them in natural language).

Considering all the above-mentioned theoretical research of the frameworks that define the user simulators, the metrics used there, and the company's tasks, we have readjusted the goals of the project. The US developed within my project should meet the following requirements:

[9] https://dit.uvt.nl/

1. **Bots and tasks covered**: as there are different levels of urgency and maintenance of different PB's in the company, the better tactic is to create a US that should test two specific bots for their bot-specific tasks and then generalize it to other company products. The range of intents is also changed to those critical to the two PB's in question;
2. **Natural languages**: as the more urgent tasks are related to the Czech-speaking bots, we switch our focus from English to Czech;
3. **Level of testing**: although intent-level testing is quite popular in US, we stick to the end-to-end testing procedure of the bots and simulate phone calls between the US and PB;

Furthermore, we made the following general design decisions:

4. **US framework**: the bots I was assigned to test had a small variety of intents or entities to test; thus, a full-fledged agenda was unnecessary for them. However, in the latter perspective, it would be necessary, so we chose a simplified version of language-level ABUS for our tasks for two reasons: firstly, we want our US framework to be completely interpretable (thus, we would not prefer NUS), and secondly, we do not need it for training purposes (thus we do not need a costly TUS);
5. **Metrics**: we are interested in the extrinsic metrics because the company already has unit tests for the intrinsic modules, such as dialogue state tracking. Regarding language fluency, we are not interested in these metrics either because NLG modules are arranged as template-based ones; thus, they do not assume high variability that needs to be continuously controlled. The extrinsic metrics we used were both the task-success rate and the dialogue efficiency metrics.

2.2. Architecture of the MAMA AI Bots

Before presenting the structure of the user simulator that I developed, we would need a short description of the general architecture of the PBs in MAMA AI, as my user simulator is also created within this framework. As with any other full-fledged module-based DS, these bots have five modules: automatic speech recognition (ASR), natural language understanding - detecting the user intents and entities from the input string (NLU), dialogue management - decision system of leading the dialogue (DM), natural language generation (NLG) and text-to-speech synthesis (TTS). Apart from that, there is a telephony module for making and handling calls, as the bots are run through telephony. My work was mostly tackling DM and partially NLU and telephony, so these modules will be explained in detail below.

The DM module is organized as follows: the whole bot consists of a set of nodes connected with each other. Each node, an "atomic unit" of a bot, consists of two parts - a body and a jump table. The former can contain actions related to the input or output (sending textual output, creating statistics, analyzing input structure, etc.), while the latter is responsible for action selection (AS) - the process of redirecting from the current node to the next one given the current input. The AS module is a sorted list of conditions, which redirect to a specified node once it meets the first true condition. The conditions can take both the semantic level of the input (intents, entities) and the surface level (ASR outputs as natural language strings). The AS module is also the only part of the node that can take on the input - the node body can only work with the last input that the AS captured. Thus, the combination of the nodes interconnected by AS modules with actions performed in the bodies of the nodes makes the whole bot. We must also mention that we can pass the parameters to the bot while initiating the call. The example of the bot is demonstrated below in the form of a Python code.

```
class Bot(BasicBot):
    def __init__(self):
        ...
        self.param_a = get_node_argument('a') # passing parameters while calling

    def _node_root(self): # typical node of the dialogue
        self.send_output('hello world') # node body: sending output (optional)
        return {Intent_X: self._node_x,                # AS module: format:
                self.custom_condition(): self._node_y} # {condition: node if
                                                       #  condition is True}

    def _node_x(self):
        self.custom_action(self.param_a, self.input) # node body: doing custom
                                                     # operations (optional)
        return {Intent_Y: self._node_y,
                self.any_input(): self._node_root}  # AS module: order matters -
                                                    # firstly input is
                                                    # compared to Intent_Y, then
                                                    # to other conditions

    def _node_y(self):
        self.send_output(...)
        return self._node_root  # AS module: can deterministically redirect to
                                # another node, but does not take input from a
                                # user in this case
```

The NLU module is based on a pretrained encoder language model (LM), namely, Seznam small-e-czech[10], which is trained on monolingual Czech data. For each bot, there is a separate instantiation of this module, which is finetuned on the custom in-domain data provided by the bot developers. The module is trained for predicting the intents and named entities based on natural language input; thus, for each bot, a training set of the natural language utterances and the corresponding intents and/or entities should be provided.

The telephony module is responsible for running the DS for telephony. It sets the variety of the inputs and outputs (not only voice data can be processed, but also Dual-tone multi-frequency signaling, or DTMF), as well as the infrastructure for the automatic calls (scheduling the bot calls for a particular user with particular parameters on some regular basis).

2.3. Work Progress

In this chapter, I will provide an overview of the implemented US with the primary attention on the modules introduced above - DM, NLU, and telephony - and the motivation for the particular architectural solutions.

[10] https://huggingface.co/Seznam/small-e-czech

DM Module

As the US is ultimately aimed at testing all company's PBs, with various use cases for each bot, it is necessary to transmit information on the details of a particular simulation run to the US. To arrange this variety of actions, we have clustered the US behavior into three parameters: the PB under question, scenario, and sub-scenario. By "scenario" we mean some general group of goals that the PB offers to perform and the US simulates; the "sub-scenario" is created to model either the the particular variants of a goal within a scenario, or the variability of the user's personality given a particular task (if the user is cooperative, if she uses synonyms for the expected phrases, etc.).

Let us show an example of ISP-bot - one of the PBs that MAMA AI provides to the local internet service provider. It is necessary to test how the bot answers several questions regarding the subscription details (how to interrupt the subscription, how to terminate it, etc.). In this case, the PB under question is "ISP", the scenario is "service" (including all questions about subscription), and sub-scenarios are "interrupt_service," "terminate_contract," "interrupt+terminate" (when we model a user that has two questions at the same time), etc.

To perform the goals and estimate the PB reaction, I created a combination of the "saying" and the "listening" nodes: the former ones aim at producing the phrases according to the agenda and taking the input (in the AS part), while the latter ones thoroughly analyze the inputs, compare them with previous outputs and update the agenda. It is necessary to mention that in the US, the AS parts of nodes do not directly depend on the agenda; they instead depend on the surface form of the PB phrases: this allows the bot not to get stuck if a PB does not behave as expected. For instance, when the US has to ask for repetition, it will get from a default node "x" into the node "check_repeat" of checking whether the PB reacted appropriately; but even if the PB did not react properly, the bot would still return to the default node "x" and repeat the attempt, without being stuck in the node "check_repeat" and waiting for the expected PB reaction.

In addition to agenda evaluation, we include several other quantitative dialogue parameters. Similarly to the classifications provided in Section 1, they can be classified into task-success and efficiency metrics. The former can be formalized as a binary classification of "successful" or "failed" task metric, but we are interested in a more fine-grained evaluation of the task success, which will be described below.

The task-success statistics include the counter of the appropriate reactions: it is based on the NLU module of the US, which analyzes the intents of the PB and, based on a handcrafted correspondence table in the form of {US intent: [adequate PB intents]}, determines if the intent is an adequate reaction to the intent used by the US at the previous stage (the PB intents were taken from the NLU module of MAMA AI, described above). As a result, we count the adequate reaction ratio as a fraction of the adequate reactions to the total number of the reactions.

As an experiment, we also used a semantic metric, which uses custom fine-tuned Transformer LMs but with a different objective function from the one used in the NLU module. We took several BERT models (and other pretrained encoder LMs) that allow for fine-tuning the next sentence prediction objective function (NSP). We hypothesize that we can approximate the appropriateness of the PB reaction to the US prompts by reducing it to the NSP task. To do that, we consider the US prompts the first sentences and the PB reactions the second sentences. To train the model, we collected all examples of the utterances that the US and the PB can generate, created the positive and negative

training examples, and fine-tuned the pretrained LMs on them. For the positive training pairs, we took the US prompts and the expected PB utterances. The negative examples were combined from two sources: randomly shuffled pairs of the US prompts and PB utterances, and random Czech sentences from the CzEng corpus instead of the US prompts, with random PB utterances. We compared four models for this task (the main criterion was the ability for the NSP fine-tuning) - Czert by the University of West Bohemia[11] (one of the largest monolingual Czech models), the DeepPavlov model for 4 Slavic languages, including Czech[12], the BERT base multilingual model[13] and the small-e-czech model by Seznam[14].
Unfortunately, we could not find an NSP-trainable Czech (or multilingual) model pre-trained on conversational data, as that would have been the closest data to what we have. NSP is a binary classification; thus, we can compute the ratio of the "correct" sentence pairs divided by the total of the sentence pairs for a particular dialogue.

We also included metrics focused on dialogue efficiency, namely: the time of the dialogue in seconds, the length of the dialogue in turns, and the number of PB fallbacks during the dialogues (i.e., the reactions when a PB says it does not understand US's utterance or when PB keeps silence). The lengths of the dialogue in turns and seconds can be compared to the expected length (based on the talk with a human); thus, we pass the information on the ground-truth length of the dialogue to the bot when we start the call. The US also uses the ground truth length as a constraint for the unexpected cycles in the dialogue: if the talk exceeds the double ground truth length of the dialogue, it is automatically interrupted.

At the end of each call, the bot sends the information about the dialogue to the company's Slack channel. The information consists of a short Slack message with the main snippets (if the agenda was fulfilled and if the time exceeded the limits) and a link to a JSON file with all metainformation about the call (the PB under test - scenario - sub-scenario triplet, time of the call, etc.), and all the above-mentioned statistics. The examples of the Slack message and the JSON file are provided in the Appendix.

NLU Module

As mentioned above, each bot's NLU models are trained separately. Thus, a fine-tuning set of intents should be created for the US. However, grouping the phrases into intents may sometimes be tricky, as the same phrases in different bots can be used for different purposes, which then causes misclassification during inference. Selecting the appropriate set of training examples, as well as the whole taxonomy of the training intents, is thus becoming a tricky task.

On the other hand, using the full-fledged intent-level NLU for action selection is not necessary: we can create custom functions for the dialogue policy. Given the fact that the PBs I worked with usually had a small set of "trigger phrases" that the US needed to catch, I could rely on the surface forms of the PB phrases. I used the Levenshtein distance between expected trigger phrases for each intent, and the ASR results to check if the normalized similarity between the input and the expected line is above the threshold (usually approximately 0.8, but it depended on the length of the trigger phrases).

[11] https://huggingface.co/UWB-AIR/Czert-A-base-uncased
[12] https://huggingface.co/DeepPavlov/bert-base-bg-cs-pl-ru-cased
[13] https://huggingface.co/bert-base-multilingual-uncased
[14] https://huggingface.co/Seznam/small-e-czech

With the help of the Levenshtein distance function, I made the whole process iterative. We can show it on the example of a particular node that test one feature of a PB:

1. At the first step, the whole NLU function in the node (the AS block) is substituted with the Levenshtein distance-based function, i.e. it only reacts to the substring match. This allows the US to run through the dialogue given the "ideal" inputs.
2. Then, we start generalizing the AS block by coming up with a new intent (or a set of intents) for this node and fine-tuning the NLU model for performing in this node. We add the newly fine-tuned intents to the AS block and start testing them together with the Levenshtein distance-based function as a backup function (the bot interface can show whether the dialogue manager reacted to the intent or to the Levenshtein function, thus we can see if the new intent works well in a particular node).
3. If the intents work well on a node they were designed for, we run the calls with other scenarios and sub-scenarios where this newly created intent is not supposed to be, to see if there are false positive reactions on this function. Given these observations, we try to adjust the intent phrases so that they maximize the positive response in the expected situations (in a specific node they were created for), and at the same time minimize the positive response in other cases. The steps 2 and 3 are then being iterated several times.

In the best case scenario of such training for a particular node, we can stop using the Levenshtein distance-based function, if it performs well besides this backup support. However, we will probably only partially get rid of the substring similarity, as the PBs are continuously developed, and we will need to update our NLU models again; in this case, having the Levenshtein distance backup functions would come in handy.

Other Modules

To automatize the calls, it was also necessary to update the telephony module. Firstly, I added the DTMF function of the bot's outputs, as one of the PBs needed to take it as input. Secondly, I helped update the scheduler of the bot calls so that they are run by default regularly until further notice.

The other modules, such as ASR and TTS, were taken off-the-shelf from the MAMA AI infrastructure. As for the NLG module, currently, I used a pre-collected list of phrases for each intent; thus it essentially represents the basic rule-based NLU module depending on the appropriate intent in each turn.

3. Results and Performance

The agenda system we created showed good robustness and reliability when we manually checked the logs of the US-PB recordings: we made 30 runs of the testing calls and checked them manually, and we could not agree with the agenda outputs only 3 times. But how informative are the other metrics that we collected? Do some of them correlate better with the agenda outputs than others?

We formalized this question the following way. Let the agenda outputs (whether the agenda is fulfilled or not) be the silver labels of the dialogue classification. Thus, the distribution of the dialogue outputs with respect to agenda fulfillment is a categorical (binary) distribution. The other metrics, on the contrary, are continuous distributions: the ratio of correctly recognized intents, or the LM-based

semantic similarity, the ratio of the real time to expected time, or other metrics described in Section 2.3, can mostly take values within [0, 1] interval; but some of them can take on values in the whole range of rationals. Thus, we can use statistical methods to measure the correlation between the categorical and continuous variables.

There are few ways of comparing categorical and continuous distributions. The most widespread is logistic regression, in which we fit one subset of the data and test on another. Another one is the point-biserial correlation coefficient. We will try both methods to compare the correlation between our statistics and agenda fulfillment. For the logistic regression, we will first fit the data with each statistic separately and then fit it with the multidimensional inputs (with tuples of all statistics we have as input).

As the US was integrated into the MAMA AI infrastructure only recently, we are currently collecting the data for this statistical analysis. Thus, we will show the results of this statistical testing at the project presentation.

4. Conclusions and Perspectives

We have presented the user simulator (US) for the production bots of MAMA AI company. It is a phone-based bot that calls the production bots (PB) to test their ability to fulfill a set of tasks they are expected to perform. Before the call, the bot takes on several parameters, such as which bot to call and which scenario and sub-scenario to test, as well as the expected values for the length of the dialogue. Then, the user simulator imitates the human user and, according to the in-built agenda, tests the subset of features of the production bot, also collecting the statistics of the dialogue (such as the length of the talk, number of correctly recognized intents, etc.). In the end, the user simulator sends a message to the MAMA AI slack channel with brief information on whether the testing run was successful and a detailed JSON report on the statistics of each call.

Currently, the user simulator is applied to two production bots - the bot used for the interns onboarding in the automotive company and the information point of the ISP customer information. Five scenario – sub-scenario pairs are tested within the automotive bot, four scenario – sub-scenario pairs for the ISP bot, as well as the the “random” sub-scenario for both PBs: how each bot will behave given random inputs. We scheduled the series of calls that comprises all the functions daily; thus, by the day of the project presentation, we will have at least one hundred examples of the talks between the user simulator and the production bots so that we can provide statistical speculations on these results.

The range of the PBs under question and the US functionality presented here is quite narrow. Thus, there are broad perspectives for enhancing the user simulator. Firstly, both bots tested so far did not have any specific named entities to process (such as the person’s name, address, numbers, etc.); other production bots in MAMA AI are working with this type of information. Thus, a necessary update for the current agenda mechanism would make it more flexible to pass and recognize the named entities throughout the talk.

The second dimension of expansion is the variability of the US personality. Currently, the scenarios and sub-scenarios (explained in Section 2.3) present only different tasks of the bot; however, as was

discussed in Section 1.1, the human users differ not only by their tasks but also by how they communicate their needs. Thus, it is necessary to provide various degrees of surface variation within the same use case - for instance, using paraphrases, providing more or less information per turn, being willing to repeat the same information, or even being consistent with the same pieces of information throughout the talk. This expansion of the "user personalities" will also be an expansion perspective of the US.

Finally, the architecture of the US (the relation between the nodes, the NLU intents, etc.) for testing the production bots was manually handcrafted by me; however, if we want to automate the process of testing, it would make sense to generate the structure of the user simulator for a new bot automatically. As all bots within the company are task-oriented, and their dialogue policies are predefined, it seems possible to create a "mirroring" structure of a user simulator for a particular bot. Let us explain the idea of "mirroring" with an example of the ISP bot: from the perspective of the PB designers, they had to create a system of nodes that would lead a user to asking what she wants to know, and then to provide the answers; in order to do that, they had to collect the possible user inputs (to train the NLU module) and a set of prompts for the NLG module. But if we look at that from the US perspective, we need to create a set of nodes that would let the US get to the point when it asks the questions, and then to listen to the instructions; with respect to the NLU and NLG, the training phrases just switch the places compared to the situation in PB. Thus, it sounds reasonable to have an algorithm that would, given the PB set of nodes for doing the task X, sample user inputs and PB prompts, to automatically generate a US set of nodes for testing the task X, sample US outputs (same as user inputs for PB) and inputs from PB (same as PB prompts). Most probably, it would still need post-editing; however, at least the semi-automatic creation of the "scaffolding" structure of a new user simulator might optimize the US development to a great extent.

References

Harry Bunt. 2009. The DIT++ taxonomy for functional dialogue markup. In pages 13–25, Budapest, Hungary. Decker, Sichman, Sierra and Castelfranchi (eds.).

Harry Bunt, Volha Petukhova, Emer Gilmartin, Catherine Pelachaud, Alex Fang, Simon Keizer, and Laurent Prévot. 2020. The ISO Standard for Dialogue Act Annotation, Second Edition. In pages 549–558, Marseille, France. European Language Resources Association.

Dafeng Chi, Yuzheng Zhuang, Yao Mu, Bin Wang, Jianzhu Bao, Yasheng Wang, Yuhan Dong, Xin Jiang, Qun Liu, and Jianye Hao. 2022. Offline-to-Online Co-Evolutional User Simulator and Dialogue System. In pages 98–113, Abu Dhabi, Beijing (Hybrid). Association for Computational Linguistics.

Jan Deriu, Alvaro Rodrigo, Arantxa Otegi, Guillermo Echegoyen, Sophie Rosset, Eneko Agirre, and Mark Cieliebak. 2021. Survey on evaluation methods for dialogue systems. *Artificial Intelligence Review*, 54(1):755–810.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Caroline Dockes. 2021. *Building a Conversational User Simulator Using Generative Adversarial Networks*. MA thesis, University of Cambridge, Trinity College.

Simon Keizer, Stéphane Rossignol, Senthilkumar Chandramohan, and Olivier Pietquin. 2012. User Simulation in the Development of Statistical Spoken Dialogue Systems. In Oliver Lemon and Olivier Pietquin, editors, *Data-Driven Methods for Adaptive Spoken Dialogue Systems*, pages 39–73. Springer New York, New York, NY.

Florian Kreyssig, Iñigo Casanueva, Paweł Budzianowski, and Milica Gašić. 2018. Neural User Simulation for Corpus-based Policy Optimisation of Spoken Dialogue Systems. In *Proceedings of the 19th Annual SIGdial Meeting on Discourse and Dialogue*, pages 60–69, Melbourne, Australia.

Association for Computational Linguistics.
Xinmeng Li, Wansen Wu, Long Qin, and Quanjun Yin. 2021. How to Evaluate Your Dialogue Models: A Review of Approaches.
Xiujun Li, Zachary C. Lipton, Bhuwan Dhingra, Lihong Li, Jianfeng Gao, and Yun-Nung Chen. 2016. A User Simulator for Task-Completion Dialogues.
Hsien-Chin Lin, Christian Geishauser, Shutong Feng, Nurul Lubis, Carel van Niekerk, Michael Heck, and Milica Gašić. 2022. GenTUS: Simulating User Behaviour and Language in Task-oriented Dialogues with Generative Transformers. In pages 270–282, Edinburgh, UK. Association for Computational Linguistics. arXiv:2208.10817 [cs].
Hsien-chin Lin, Nurul Lubis, Songbo Hu, Carel van Niekerk, Christian Geishauser, Michael Heck, Shutong Feng, and Milica Gašić. 2021. Domain-independent User Simulation with Transformers for Task-oriented Dialogue Systems. In volumes 445–456, Singapore and Online. Association for Computational Linguistics. arXiv:2106.08838 [cs].
Hong Liu, Yucheng Cai, Zhijian Ou, Yi Huang, and Junlan Feng. 2022. A Generative User Simulator with GPT-based Architecture and Goal State Tracking for Reinforced Multi-Domain Dialog Systems. In pages 85–97, Abu Dhabi, Beijing (Hybrid). Association for Computational Linguistics. arXiv:2210.08692 [cs].
Ramón López-Cózar, Zoraida Callejas, and Michael McTear. 2006. Testing the performance of spoken dialogue systems by means of an artificially simulated user. *Artificial Intelligence Review*, 26(4):291–323.
J. Schatzmann, B. Thomson, K. Weilhammer, H. Ye, and S. Young. 2007. Agenda-Based User Simulation for Bootstrapping a POMDP Dialogue System. In volume Companion Volume, Short Papers, pages 149–152, Rochester, New York. Association for Computational Linguistics.
Andreas Stolcke, Klaus Ries, Noah Coccaro, Elizabeth Shriberg, Rebecca Bates, Daniel Jurafsky, Paul Taylor, Rachel Martin, Carol Van Ess-Dykema, and Marie Meteer. 2000. Dialogue Act Modeling for Automatic Tagging and Recognition of Conversational Speech. *Computational Linguistics*, 26(3):339–373.
Guangsen Wang, Samson Tan, Shafiq Joty, Gang Wu, Jimmy Au, and Steven Hoi. 2022. BotSIM: An End-to-End Bot Simulation Framework for Commercial Task-Oriented Dialog Systems. In pages 178–190, EMNLP. Association for Computational Linguistics. arXiv:2211.11982 [cs].
S. Young. 2009. CUED Standard Dialogue Acts.

Appendix. Json and Message Examples

Example of the Slack Message

The Slack message is a snippet of the whole analysis made by the US within a talk. It shows crucial meta-information about the call (what were the production bot, the scenario, and the sub-scenario tested; date and time of the call), as well as two ultimate outcomes of the analysis - whether the agenda was fulfilled or not; and whether the time exceeded the permissible limits. The agenda fulfillment is, of course, the main piece of information that we want to get from the US. However, the information about the time (whether the dialogue was too short or too long) is one of the easiest ways to check the possible technical problems with both US and PB (as the bots would either consistently crash earlier than expected or enter the loops).

The last line leads to the JSON file containing all statistics collected by the US within the call.

MBotAPP 11:37 AM
prod bot: ISP-dev

scenario; sub-scenario: service; transfer

date; start time: 2023-06-07; 11-36-17

Agenda Fulfilled: ❌

time: 🟩 length within permissible limits

[**length: exp-d:** 100s, **real:** 89s; **deviation: exp-d:** 20%, **real:** 10%; **maxed out:** NONE]

full stats: link

Example of the Full Statistics

The complete statistics file consists of the following blocks:

1. The meta-information - the parameters of the production bot, scenario, and sub-scenario, as well as testing date and time;
2. The statistics block (all statistics are described in detail in Section 2.3):
    a. The agenda block comprises the binary class of whether the agenda was fulfilled and, if not, which tasks were left unfulfilled;
    b. The time block, consisting of the overall time of the dialogue and its comparison with the ground truth time;
    c. The statistics of the turns (the total number, the number of adequate reactions, the number of "fallbacks," the expected number of turns, and the ratios based on that);
    d. The semantic similarity based on the NSP metric of the BERT model.

Below you can see the example of a call that has not fulfilled the agenda, but its time statistics are reasonable.

```
{
  "metainfo": { // meta-information about the call
    "production_bot": "ISP-dev",
    "scenario": "service",
    "sub_scenario": "transfer",
    "date": "2023-06-07",
    "start_time": "11-36-17"
  },
  "info": { // main statistics block
    "agenda": { // a. agenda block
      "is fulfilled?": false, // bool
      "not fulfilled tasks": "get_transfer_info" // specification of
                                                 // unsuccessful tasks
    },
    "time": { // b. time block
      "total": {
        "real": 89.221535, // real time (seconds)
        "expected": 100 // expected time (passed to the US)
      },
      "deviation": {
        "real": 0.10778464999999997, // difference between the
```

```
                                    // real and expected time
        "permitted": 0.2 // the permissible limits of difference
      }
    },
    "turns": { // c. turns statistics
      "total #": 5, // number of turns (US + PB utterance pairs)
      "correctly recognized": 5, // # correctly recognized turns
      "expected #": 4, // expected number of turns (passed to the US)
      "correct/total": 1.0, // “correctly recognized/total #” turn ratio
      "total/expected": 1.25, // “total #/expected #” turn ratio
      "wrong reactions": 0 // number of “fallback” reactions
    },
    "semantic_similarity": { // d. NSP-based semantic similarity
      "czert": 0.4, // czert model
      "deeppavlov": 0.4, // deeppavlov model
      "seznam": 0.8 // seznam model
    }
  }
}
```