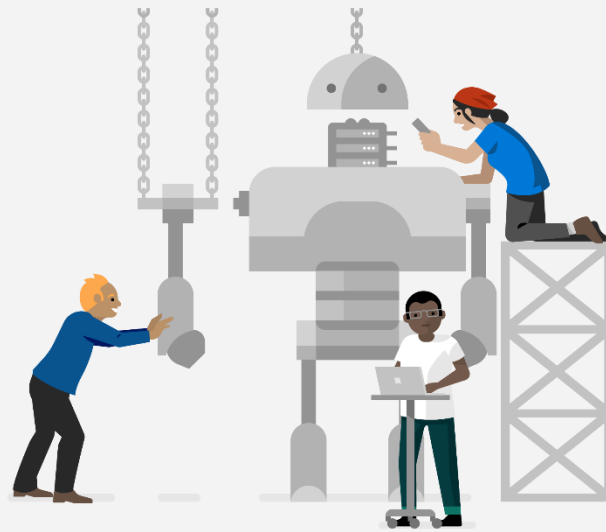


MLOps for chatbots

Applying context free grammar to test the performance of intent recognition systems



Zen van Riel (2008508)

Department of Cognitive Science & Artificial Intelligence

z.m.n.vanriel@tilburguniversity.edu

Supervisor: Dr. Emmanuel Keuleers

Second reader: Dr. Juan Sebastian Olier Jauregui

External supervisor: Dennis Mulder

Word count (main content): 8084 words

Preface

During the bachelor Cognitive Science & Artificial Intelligence, AI has been a core topic which has been discussed from both a humanities perspective as from a technical perspective. Looking to apply profound AI within software, I spent a significant amount of time developing the core of my software development passion. Having worked as a software tester and more recently a Cloud Solutions Architect at Microsoft, I sought to find a thesis project which could bridge my experiences gained in the past few years. The presented paper is an opportunity to cross these domains while remaining perceptive of trends in modern cloud-native software engineering.

The completion of this project is not without thanking individuals which have helped tremendously. From a personal perspective, I thank my friends and my mother for continuous support throughout. From a professional perspective, my internal supervisor Emmanuel Keuleers has been very helpful in his role as a supervisor. Accurate and useful weekly feedback was provided, showing his great interest in ensuring that a complete thesis can be written. Additionally, I thank all Microsoft colleagues for inspiring me and others to put forth their best efforts on any project. A special thanks is obligatory for several colleagues. These include Andreea Mares and Harmke Alkemade for involving me in the Brainy bot project, which was pivotal in inspiring me to write a thesis on the topic of chatbots. Finally, as has been the case with every project during my internship, I thank Dennis Mulder for his supportive attitude. This is especially the case in giving freedom while trusting me to do a good job.

MLOps for chatbots: Applying context free grammar to test the performance of intent recognition systems

Abstract

Modern chatbots often rely on an intent recognition system (IRS) to understand user input. Automated testing is an integral part of the software development lifecycle, yet it is not covered for such systems. This study leverages previous work to propose a preliminary test framework. The framework relies on a context free grammar to generate utterances of varying complexity, optionally adding spelling errors as noise. It can use such utterances to test an IRS and assert its performance using several test oracles. The framework can be valuable to support various MLOps scenarios. To show the framework's value in practice, an IRS relying on a machine learning service (Microsoft LUIS) is trained using the Frames dataset. The effect of complexity and noise on intent recognition performance is tested. In the case of utterances with noise, comparisons are made between the IRS with and without spell check preprocessing. Adding spell check preprocessing at lower complexities decreases or negates the negative impact of noise on performance. Overall intent recognition performance is the IRS's strength while all-or-nothing entity recognition performance generally degrades when complexity and noise is added. When entities are considered separately, performance is less prone to degrade with complexity. Future work can improve on the framework in a multitude of ways. The context free grammar may be replaced by a more general-purpose utterance generation mechanism, and different kinds of noise can be implemented. Future iterations may also consider multi-intent utterances and other IRS services. Regardless of what future work will focus on, the current framework is functional in asserting the quality of IRSs, and by extent conversational chatbots.

1. Introduction

Chatbots are a software interface used for many purposes, such as assisting in customer service. Improved artificial intelligence, especially in the domain of natural language processing, make it plausible to develop chatbots which can reason and react to users, even in the case of complicated dialogs (Dale, 2016). Natural language processing for such chatbots is often implemented in a separate natural language understanding model, referred to as an intent recognition system (IRS). When a new version of this system is developed, it should be thoroughly tested before being released just as any other software component. While several existing frameworks to automatically test software exist, they do not suffice to test an IRS component in all test domains (section 1.1).

The main aim of this paper is to propose a rudimentary test framework for an IRS which can support various MLOps scenarios. Assuring quality before releasing a machine learning (ML) model is one of the aims of MLOps, short for machine learning operations (a derivative of DevOps; section 1.1).

MLOps can be considered as a set of practices to allow for automatic validation of ML models. This validation can take place before promoting new models into operation, referred to as a production environment, or to verify models which are already in a production state (Microsoft, 2020). By validating an ML model before release, software components depending on such a model remain functional. To allow MLOps scenarios, the proposed framework should be designed as an automatic, non-intrusive process. The framework includes two components to fulfill these requirements. Component A is an utterance generator based on a context free grammar (CFG) (Chomsky, 1956). A supplementary subcomponent can introduce noise in the form of spelling errors. The second component, Component B, can query the IRS with utterances from component A and assess whether the IRS performance is satisfactory based on a given testing condition.

Component B is used to conduct analyses comparing the expected IRS output under various test conditions (section 2.3.1) to the true output. The aim of these analyses is twofold. First, it is an exercise to show that the framework is capable of automatically testing the IRS. Secondly, it reveals that adding complexity and noise to utterances decreases IRS performance in a multitude of cases. For utterances which contain noise, such influence can be remedied by applying spell check preprocessing at lower complexity levels. Overall, results indicate that the IRS can be assumed to recognize the overall intent of an utterance, although some details within the intent may not be recognized.

1.1 Background

Creating machines with the capability of displaying intelligent behavior is a pivotal goal in artificial intelligence. One form of such behavior could be considered the ability of machines to converse with individuals in a way that is indistinguishable from other humans (Turing, 1950). ELIZA is a primitive example of a program allowing a computer to communicate with humans using natural language. It can be referred to as a conversational agent or chatbot. To understand the user's intention, the chatbot relies on matching user input to pre-existing rules (Weizenbaum, 1966). While early chatbots such as ELIZA can only mimic simple conversations, more sophisticated chatbots extend conversational abilities to more domains. The ALICE chatbot introduced a common markup language (AIML), used in the creation of new chatbot prototypes (Abushawar & Atwell, 2015) and new chatbots altogether such as Mitsuku (Worswick, n.d.). Many chatbots of this variety are predominantly created as a means of entertainment or simply to create a conversational partner. Yet, as adoption of chatbots increased over time, Brandtzæg and Følstad (2017) found that productivity is the most important motivational factor for people to use chatbots. This productivity is expressed in obtaining information or assistance efficiently. To facilitate this, a modern chatbot can be developed using a variety of software user interfaces (UI). Traditional UI such as buttons and forms may suffice if the chatbot's goal is constricted and clear (Alkemade, Mares, & Riel, 2020). A less restrictive approach is to always allow the user to type in free form during chatbot interaction. Chatbots may be designed to allow a user to freely communicate using natural language interfaces based on text or speech (Glass et al., 1995). Using this approach, user input (referred to as an

utterance) may be matched with static commands. Such a chatbot can still be restrictive if the chatbot only understands utterances which match with a small set of pre-determined rules (Veldsink, n.d.). To eradicate these restrictions, the chatbot can employ a rigid natural language understanding (NLU) model. The NLU model can be considered an intent recognition system (IRS) as its main purpose is to interpret the intent behind utterances. Such an IRS can allow for nonrestrictive UI as many utterances can be understood by the chatbot, which can allow for more sophisticated and customized human-computer interaction. A chatbot might require such an intelligent IRS to lead a user closer to a goal if other UI does not suffice (Zadrozny et al., 2000). Going forward, *chatbot* refers to goal-oriented conversational agents relying on a text-based natural language interface and an IRS component.

Chatbots using natural language UI are in stark contrast to traditional software UI. However, traditional UI has been a large focus in the field of human-computer interaction due to its precedence in most software. To accommodate for the magnitude of difference between traditional UI and natural language UI, a paradigm shift in this domain may be necessary. Conversations which lead users to intended goals could become a central part of software design (Følstad & Brandtzæg, 2017). Software design unfolds from the software project's development methodology. The simplest development methodology is to analyze a problem and then implement it in code. While this approach is suitable for small scale projects, larger software systems require more granular and segmented stages. A primitive model is one in which a software's development proceeds through stages in order, limiting interaction between the steps only if necessary (Royce, 1970). As software projects have grown, several software development lifecycle (SDLC) models have been explored, eventually with more focus on feedback loops between stages. These models cover all steps in delivering and maintaining software (Ruparelia & Nayan, 2010). The unity of different teams and processes within an organization to successfully implement such models is a crucial element of DevOps (a contraction of development and operations). From the perspective of the software as a product, DevOps can be considered a set of practices to allow software to be delivered continuously using SDLC models (Zhu, Bass, & Champlin-Scharff, 2016).

Testing is an integral component of widely adopted SDLC models, with a general goal of ensuring the quality of software by validating its requirements and functionality. While test cases can be executed manually, automated testing may limit the cost and time investment required for manual testing. Many software applications, such as a large majority of web applications, rely on existing tools for test automation (Angmo & Sharma, 2014). These mature frameworks are excellent for covering many test domains. Existing frameworks can be used to automatically test chatbots within some test domains. One such domain is security testing for common security vulnerabilities, including XSS and SQL injection (Bozic & Wotawa, 2018). For chatbots, an important contender for regular testing is its IRS component. Testing such a system is regarded as integration testing, which concerns testing between software components (Linnenkugel & Mullerburg, 1990). It is important to test this component of chatbots for two reasons. First, without a functional IRS a chatbot is likely unable to provide meaningful answers to queries as it is designed to rely on a correct interpretation of utterances. Secondly, a modern IRS often

relies on ML algorithms. For such a system, testing its performance before and even during live deployment can be crucial in preventing feedback loops radically influencing the model's behavior (Sculley et al., 2015). This testing practice is a typical MLOps scenario, as MLOps aims to apply DevOps practices to facilitate continuous delivery of ML models while maintaining quality assurance during production (Microsoft, 2020).

When considering testing an IRS as an MLOps process, the set of valid inputs is a concern due to the extensive nature of natural language. In contrast to software using more traditional UI, a novel software testing methodology is required that takes this into account. Baresi and Young (2001) define that a software testing method requires a test oracle, regarded as a measurement of any kind that can determine whether software adheres to expected behavior under a certain test case. As an IRS can accept natural language as input, there is difficulty in determining the set of test cases and test inputs to validate the system under a given test oracle. Previous work has contributed to minimizing these difficulties. Bozic, Tazl and Wotawa (2019) propose a test case generator which depends on an AI planning system. This reduces manual work as the system can dynamically generate test cases. Vasconcelos, Candello, Pinhanez and Santos (2017) provide a testing tool intended for automatically testing a batch of predefined input and output where the expected output acts as the test oracle. The tool can introduce spelling errors which further assist in testing the robustness of the IRS. The main limitation of this tool is that such a batch of questions does not inherently cover the wide range of possible user inputs. The sets must be manually created and considered by a tester. Another approach is proposed by Ruane et al. (2018) in which a testing framework can add noise to input by using various techniques, such as manipulating the grammar of the input. While the divergent input is automatically generated, the original input set is once more determined beforehand. Bozic and Wotawa (2019) have attempted to ease IRS testing by applying a metamorphic testing approach, which includes a method of generating user input. In this approach, the output of the IRS is checked against a relationship between input and output. Such a metaphoric relation may be the mapping of a user's input to a set of intents. A limited CFG was applied to generate utterances serving as input for an IRS, and it was useful in finding defects. The CFG employed a large set of rules and was capable of omitting words to generate many different utterances. Despite this, the CFG was not supplemented to replicate human errors such as spelling mistakes.

1.2 Proposal of the test framework

Previous literature has tackled the problem of automatically testing IRSs in various ways. Often progress is made in one aspect, such as automatic test case generation, yet a different aspect is not considered, for instance the addition of noise to utterances. In the current study, the learnings of previous work are considered to propose a new framework to test an IRS. The main differentiator of this framework is that it is an attempt to combine previous work to facilitate a more coherent approach such that the framework can be used in practice. The framework should not just be able to generate utterances by using a CFG (Bozic & Wotawa, 2019) but also add noise to said utterances automatically (Vasconcelos, et al., 2017; Ruane et al., 2018). In addition, the framework's components are considered with compatibility for modern MLOps scenarios in mind. The upcoming section describes the specifics behind the creation of an IRS, which relies on an open source dataset provided by Microsoft. Following this, the methodology behind the framework's components are described. Proceeding this is the description of several experimental conditions, which are implemented as test oracles in the framework. In the results section, the outcomes of these analyses utilizing the framework are presented. They serve to display the practical value of the framework, and to test the effect of complexity and noise on the performance of the IRS. Lastly, the results from these analyses are discussed, determining strong and weak points of the IRS under testing. Limitations of the current implementation are elaborated, as the framework is still preliminary. As a concluding remark, possible future improvements for the framework are proposed.

2. Method

2.1 Training the IRS

Braun, Hernandez-Mendez, Matthes and Langen (2017) express that development efforts in the field of intent recognition have led to the availability of several services based on machine learning to create IRSs. Such services may not require the end user to explicitly program them. A cloud-based service of this nature was used to train an IRS. In this limited context, cloud refers to the on-demand usage of an online service relying on both software and hardware managed by a certain party (Foster, Zhao, Raicu, & Lu, 2008). As a service within the Microsoft Azure cloud offerings, Language Understanding Intelligent Service allows the creation of NLU models by requiring only labelled examples of utterances (Microsoft, n.d.). Considering there is insufficient control over the ML methods used by LUIS due to its proprietary nature, it can be considered an automatic machine learning service.

To provide the necessary labeled utterances, an open source dataset provided by Microsoft was used. This dataset comprises 1,369 chat conversations between 12 participants stored in a JSON document. The conversations were collected over a period of 20 days. In each conversation, one user was paired up with another who was assigned the role of a chatbot. The user mimicking a chatbot had access to a database containing data such as flights relevant for booking a vacation trip. The regular users' aim

was to find a suitable trip by chatting with the presumed conversational agent (Asri et al., 2017). The dataset contains 19,986 turns, defined as a message sent by either user (utterance). Each utterance in the dataset is labeled by domain experts with account for interrater reliability. The labels concern several aspects, such as the overall intent of an utterance.

2.1.1 Data preprocessing

Several preprocessing steps were necessary to ensure that the utterances could be consumed by LUIS. Since the intent recognition is only concerned about user input, utterances sent by the presumed conversational agent were removed. The utterances were filtered further to only contain labels necessary for training the IRS. The remaining label is the utterance’s dialogue act, which conveys two important concepts. First, the dialogue act name represents the overall intent of the utterance. Secondly, the dialogue act contains arguments which represent details within the intent. These concepts are implemented in LUIS as intents and entities, respectively, and will be referred to this naming convention going forward. For an utterance such as “I want to fly to Berlin”, the intent in Frames would be labeled as *inform* with the entity *destination city* which has the value *Berlin*. Frames contains 12 possible intent types sent by the regular users. Some utterances do not have a concrete intent in Frames, such as “cool!”. Since LUIS requires an intent to be specified for every utterance, such utterances were labeled with the intent *none*.

Importantly, one utterance can contain multiple intents. Rychalska, Glabska and Wroblewska (2018) propose an NLU model which can extract multiple intents from a single utterance. However, such models are not generally supported by services such as LUIS at the time of writing. All utterances containing multiple intents were removed from the training set to avoid conflict.

Table 1
Intents trained in LUIS

Intent	Example	Labeled Utterances
Affirm	Yeah do that please	164
Confirm	Is this my only option?	57
Goodbye	Ok bye	74
Greeting	Hey!	55
Inform	We're leaving from Tofino.	3,884
More info	What else can you tell me?	32
Negate	Not good enough	78
None	Cool !	106
Request	Great, how much is it?	394
Request alternatives	What else?	143
Request comparison	What is the cheapest place to leave?	172
Switch frame	Let's upgrade to business class!	460
Thank you	Fantastic. Thanks a bunch!	357

Frames utterances may contain none, one, or multiple of 42 possible entity types. The six most common entities were chosen while the labels for the other entities were filtered out. These six entities were chosen to create a balanced set of entities with different data types (strings of text, numbers, and dates). The entities were each assigned a pre-trained entity type in LUIS according to their data types. Each entity was labeled as an entity role in LUIS, which can help in separating different entities using the same pre-built entity type. As an example, the entity *origin city* is paired with the pre-built entity *geographyV2*.

Table 2
Entities trained in LUIS

Entity Role	Prebuilt Entity Type	Example	Labeled Utterances
Origin city	geographyV2	from Berlin	1,023
Destination city	geographyV2	to Paris	1,941
Number of adults	number	6 adults	640
Number of children	number	two kids	152
Budget	money	500 euro	883
Start date	datetimeV2	I want to leave next Friday	916

From the number of labeled utterances for both intents and entities a concern may be raised about class imbalance. For intents, the number of *inform* utterances far exceeds that of other intents. To counter this concern, it should be noted that this distribution reflects realistic chatbot conversations as conducted in the Frames experiment. The *inform* intent is far more common than an intent such as *thank you*, as the *intent* inform is a necessity to reach the desired end state of the conversation (booking a trip). Additionally, previous work using LUIS indicates that a difference in utterance counts does not impact overall intent recognition performance (Ruane et al., 2018). For entities, the class imbalance is also negligible as the labeled entities in Frames only serve to specify the roles of the pre-built entities in LUIS.

This methodology relies on several pre-trained NLU model(s) which support LUIS. First, the six entities rely on default entities pre-trained in LUIS (table 2). Both the data and models used to train these pre-trained entities is proprietary. Secondly, the machine learning model(s) used by LUIS to train the intent recognition on the Frames utterances and how these intents are extracted from utterances remains concealed. Considering this, the IRS trained using LUIS can be considered as a black box ML model.

2.1.2 Creating the model in LUIS

After these preprocessing steps, 5,976 utterances remained with labels for the intent and respective entities, if any. To easily extract the required data, the TypeScript programming language was used to create a partial interface which describes the Frames dataset structure. In addition, TypeScript was used to add the utterances to LUIS by communicating with the LUIS application programming interface (API) (Microsoft, n.d.). The publication of the model's public prediction endpoint followed the model creation. The publication of the model used the default LUIS settings with the exception of turning on Bing spell

check v7, which enables the option of preprocessing utterances by a spelling check engine before being analyzed by the IRS (Microsoft, n.d.). The spell check's implementation is obscure as it is proprietary in the same fashion as LUIS.

2.2 Implementing the test framework

2.2.1 Component A

The role of component A is to generate input utterances to test the IRS by conducting several parameters set at the start of a test run. These parameters include desired utterance complexity (amount of entities per utterance), the relative amount of noise, and the sample size of the input. The utterances are generated by expanding a CFG which is constrained by said parameters.

2.2.1.1 Implementing the CFG

The CFG's role is solely to generate user utterances automatically in the form of English sentences. These utterances are limited to the domain of querying a chatbot intended for booking a trip. To reasonably limit the scope of the current project, it can only produce utterances with the *inform* intent, alike those from the Frames dataset. In addition, it produces utterances which are intended to be similar to queries occurring at the beginning of a chatbot conversation.

The RiTa.js library was used to define the grammar rules defining the start, non-terminal, and terminal symbols of the context free grammar (Howe, n.d.). Note that the grammar rules are not meant to reflect the utterances from the Frames dataset exactly, as this is the training data for the IRS. The utterances consist of one or more entities, up to utterances of the highest complexity which contain six entities. Notably, an utterance can have multiple entities, but only a maximum of one entity per type. Grammar rules were written for the entities using the predefined functions made available by RiTa.js. First, the entities *origin city* and *destination city* were implemented by adding terminal symbols using a filtered list of cities with a population above 100,000. This results in a partial utterance such as "I want to go from Amsterdam" and "My destination is Paris". Secondly, the entities *number of children* and *number of adults* rely on synonyms for the respective age groups using both numeral and verbose representations for the number of children or adults. This can result in a partial utterance like "in addition I have 2 kids" or "we are with five adults". The *budget* entity combines both symbolic representations of various currencies (\$) with other representations and acronyms where applicable (dollar, USD). A random number is chosen, in line with reasonable budgets for a given currency, resulting in a partial utterance alike "my budget is 500 euro". Lastly, the *start date* entity is implemented as verbose expressions for dates, such as "next Friday" or "sometime next month".

Outside of the standard grammar rules, logic is implemented to ensure that entity values do not overlap with each other, such that the *origin city* always differs from *destination city*. The order of the entities in generated utterances is always shuffled randomly using the Fisher-Yates algorithm. Lastly,

there is a check in place to guarantee that each generated utterance in a set of utterances is unique. The result is a CFG which can be expanded to produce utterances with a pre-set number of entities. Such utterances serve as input for the IRS under testing.

2.2.1.2 Adding noise to the CFG output

To assess the performance of the IRS more thoroughly, component A includes a function to add spelling errors as a simple, consistent way to add noise to the CFG output. Spelling errors are added based on two rules. First, a spelling error is always a letter from the English alphabet. A random letter is chosen depending on letters located near the letter used after the inserted spelling error. The nearest letters assume the QWERTY keyboard layout. Second, the amount of spelling errors in an utterance depends on a parameter set at runtime, determining the relative amount of words in an utterance for which spelling errors can be added. This ensures that longer utterances do not contain relatively more spelling errors than shorter utterances. Note that words of a single character and strings of text including non-alphabetical characters are ignored in the process of adding noise. Numerical values will never contain a spelling error to guarantee that such values retain a basic comprehensiveness.

Figure 1

Example of an utterance generated by the CFG with all 6 entities underlined. Its divergent form includes spelling mistakes. The overall intent is *inform*.

*Is there a flight that leaves Manhattan we want to fly up to Amsterdam my budget is 2028 dollars.
I want to go Monday, in addition I have 6 children, the trip needs to be accommodated for 3 adults.*

Ids there a flihgtht that leaves Manbhattan we want tpo flty up to Amksterdam mty budget is 2028 dollars, i want to glo Monday, in asddition i hsave 6 chilteren, the trip needs to bwe afccomodated for 3 adults.

2.2.2 Component B

Component B has three roles. First, it retrieves a set of utterances from component A, and queries the IRS using this set. Querying the IRS relies on the LUIS API. An important notion is that LUIS has a batch test functionality which can be used to test a large set of test input (Microsoft, 2020). However, a limitation is that this feature is not implemented in the API. As a solution, the input is tested one-by-one by using the default LUIS query endpoint. Such an endpoint is naturally available for other services as well, as chatbots relying on an IRS must be able to reach it via a network request. By programing the component to make use of a generic endpoint, the component may be easier to adopt for other services as it does not rely on a LUIS-specific testing feature.

Secondly, component B must compare whether the IRS output recognized the utterance sufficiently according to a test oracle. One of the three test oracles described in section 2.3.1 can be

chosen at the start of a test run. This also includes an option to enable spell check pre-processing for the IRS.

Third, based on the overall performance assessment across all utterances in a set, component B either approves the test run or rejects it. A parameter set at the start of a test run is a threshold ranging from 0 to 100%, indicating the percentage of utterances which must be recognized under a given test oracle to approve the IRS. Depending on the exit status, a certain action could be taken to support various MLOps scenarios. For instance, if the test run is approved, an IRS version can be promoted to a production environment.

2.3 Testing the intent recognition system using the framework

2.3.1 Experimental Conditions / Test Oracles

Three tests were conducted to test the performance of the IRS. Subsequently, these were implemented as test oracles for the test framework. For each test, there are three conditions. Condition A relies on samples from the CFG, without any spelling errors. Considering there is no noise in this condition, the IRS does not make use of spell check preprocessing. The sample from condition A is reused for condition B, ensuring that the base utterances between conditions are identical. However, condition B diverges the utterances from condition A by applying spelling errors to the utterances. The framework's noise parameter was set to 0.5, indicating that half of all eligible words in a sentence will contain a single mistake. Condition B consists of the sub-conditions B1 and B2, which both make use of the same divergent utterance samples. Within condition B1, the IRS does not make use of spell check preprocessing. In condition B2, every query is preprocessed by spell check.

These conditions are applied across three tests. In test 1, the main intent recognition is analyzed. The CFG outputs utterances with the *inform* intent. When querying the IRS, the top-scoring intent is returned. The return value is either *inform* or a different, incorrect intent, meaning that this is a binary recognition metric. The justification for this measurement as a test oracle is that if an IRS is good at intent recognition, at the very least a chatbot can understand the overall goal of a user's utterance. Note that this oracle does not test whether the IRS is biased towards recognizing this intent. Discerning intent recognition bias is not the focus of the current work. Moreover, such a bias could even be beneficial to the overall user experience as the *inform* intent is more common in real conversations (indicated by its precedence in the Frames dataset). Recognizing such a crucial intent is a prerequisite for a good user experience in which the chatbot is useful in leading the user to the end goal of booking a trip. However, this test does not guarantee that the IRS can extract the details within an intent, thus another test which considers entity recognition is required.

In test 2, the recognition of all entities within an intent is analyzed. This test compares all expected entities of input to the output from the IRS. If any entity is not recognized, the IRS output is considered a miss. As a result, IRS output is only considered correct if and only if all entities are

recognized (all-or-nothing). An entity can be recognized at two levels. Taking *origin city* as an example, the entity type is *geographyV2* with the role *origin city*. In this test an entity is correctly recognized if the IRS can match the text corresponding to the entity and its entity type, even if its role was not. The entity role is mainly supportive and does not have to be matched as most entropy for the entity is captured by the recognizing the entity text and its type. However, to be scrupulous it should be noted that a mismatch for the entity role does display a certain level of uncertainty. In practice, if the role of an entity within an utterance is unknown, one solution is to simply proactively query the user to confirm the entity's role, if necessary (McTear, 2018). Overall, this test oracle represents a more optimal user experience, assuming that the overall intent is recognized. The chatbot does not have to query the user for unknown entities if all entities within an utterance can be recognized. In the worst case a simple confirmation may be necessary if the IRS does not recognize some entity roles. While this test oracle is sufficient to test overall entity recognition performance, yet another test is necessary to understand more about the performance of each individual entity type.

In test 3, the recognition of specific entity types within an intent is analyzed. Just as test 2 this test concerns utterance entity recognition, but in contrast the output is a miss only if a single expected entity is missing from the output, disregarding whether other entities are recognized. To conduct this test for each entity type, test 3 consists of six different samples for each condition, where one entity was kept constant for each sample. The main aim of this test oracle is to allow for analyses regarding the IRS entity recognition performance on a per-entity basis.

2.3.2 Sampling the output from the CFG

There are six complexity levels to take into consideration when sampling utterances from the CFG. The first level involves input with only a single entity. The second level contains two entities, and so forth up to all six entities. For levels two through five, different combinations had to be made between the selection of entities. Simple combinatorics were applied to determine how many splits the sample required for each complexity level to attain an equal distribution of all possible entity combinations.

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

In this definition n represents the total number of entities, whereas k represents the number of entities in each input of a given sample, which ranges from 1 to 6 depending on the complexity level under consideration. For test 1 and 2 $n = 6$ as all entity combinations should be considered and there is no need to guarantee that a specific entity is in every utterance. For test 3 $n = 5$ as one entity is kept constant for every measurement, entailing that $n - k$ entities are left out while assuring that there is one specific entity in each utterance.

Considering that the CFG can be used to generate many unique utterances, the sample sizes for each test are substantial. Test 1 has a sample size $n = 300$ for each complexity level, resulting in a total

sample size $tN = 300 * 6 = 1,800$. This sample is shared for test 2. As test 3 required six measurements each with a different entity as a constant, six different samples each with a size $n = 300$ utterances were generated. For test 3, $tN = 1,800 * 6 = 10,800$.

3. Results

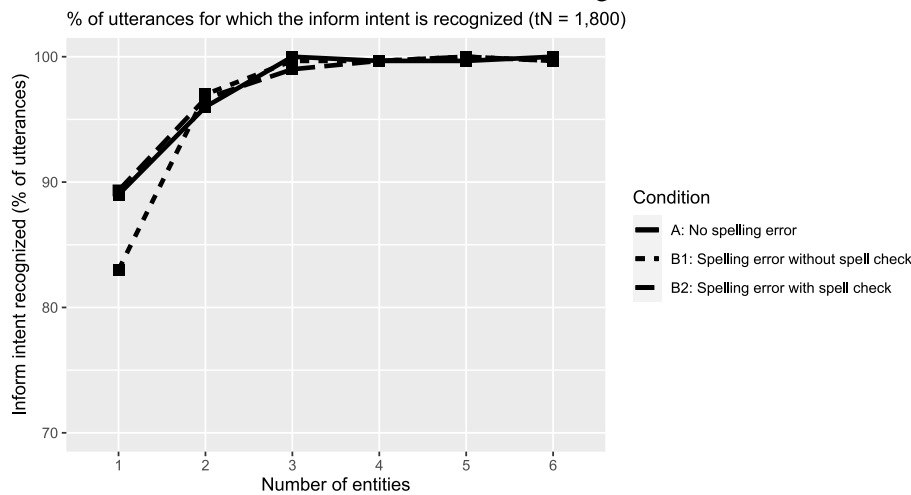
The tests presented in this section rely on binary metrics (section 2.3.1): either the IRS matches a certain expected output for a given condition, or it does not. Considering this and the ample sample sizes, several differences between the performance of experimental levels and conditions are inferred from the visuals alone and are not backed up by statistical tests¹. In addition, each test shares the same conditions.

Condition A denotes utterances without spelling errors, while condition B denotes utterances with spelling errors. The differentiator between conditions B1 and B2, is that spell check preprocessing is applied to B2.

3.1 Test 1

Figure 2

% of utterances for which the overall intent is recognized, $tN = 1,800$



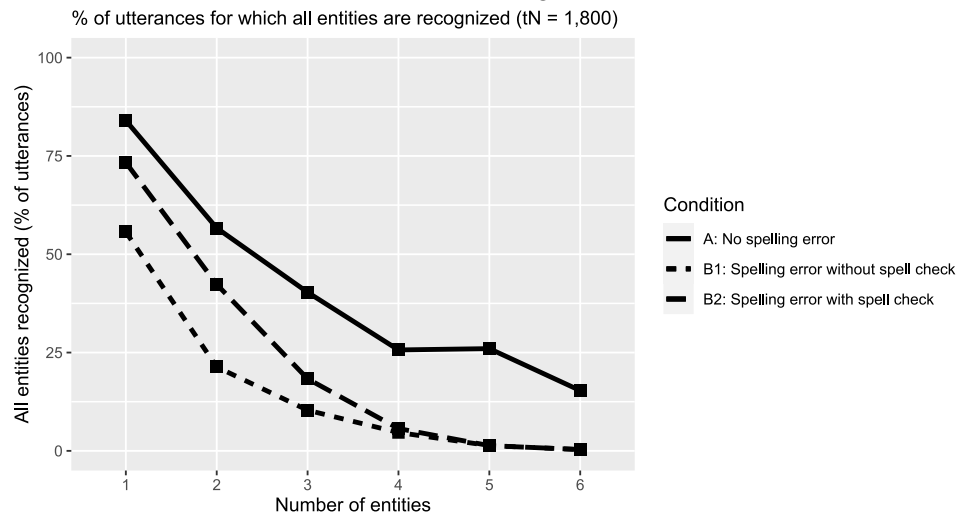
For the first test, accuracy is determined by the percentage of utterances correctly identified with the *inform* by the IRS. Chance-level performance is $\frac{1}{13}$ as there are 13 total intents which can be recognized, and the IRS must choose a single intent for which it is most confident. Figure 2 reveals that when utterances contain a single entity, the intent is recognized 89% for condition A and 89.33% for condition B1. While these conditions differ insignificantly, condition B2 performs worse at 83% accuracy. There is no significant difference between conditions A, B1, and B2 for utterances with two or more entities. From three entities onwards the accuracy is either 100% or very close to perfect accuracy.

¹ To provide the visualization and formatted data for the results in this section, a module was interposed within the framework to convert and export the experimental data for analysis

3.2 Test 2

Figure 3

% of utterances for which all entities are recognized, $tN = 1,800$

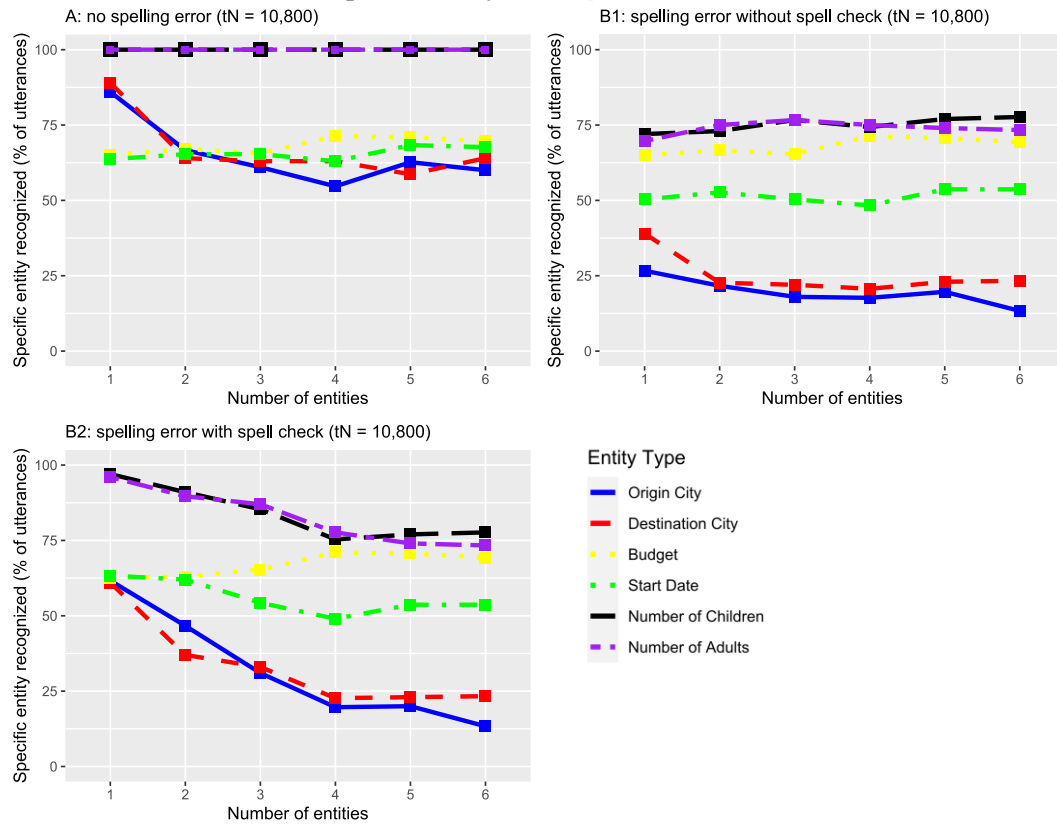


For the second test, the measurement for accuracy entails that all entities need to be recognized in an utterance. Here a chance-level performance is not as easy to determine. There are six different entities, so for utterances with a single entity the chance performance could be considered as $\frac{1}{6}$. However, the task is more complicated than determining which entities are in a sentence, as the text associated with the entities needs to be recognized as well. In addition, the IRS may also recognize no entities in an utterance which contains several entities. There is no simple baseline for this task, as any performance expectation depends on the real-world scenario in which the IRS is used (section 4.2). Regardless of which baseline is determined, figure 3 indicates that accuracy generally decreases for all conditions as the number of entities increases. For condition A, input with a single entity is recognized 84% of the time. Condition B1 is less accurate at 73.33%, and B2 accuracy is even lower at 55.66%. B1 and B2 performance converges at four or more entities per sentence. Finally, at six entities, the performance is at its lowest: 0.33% for both B conditions and 15.3% for A. There is always a clear performance difference between A and B conditions, where A performs better. The general decrease in accuracy follows a systematic curved slope, apart from recognition performance for A at five entities. This result does not differ significantly from condition A accuracy for utterances with four entities.

3.3 Test 3

Figure 4

% of utterances for which a specific entity is recognized, $tN = 10,800$



Within the third test, entity recognition accuracy is considered on a per entity basis, in which only the entity held constant needs to be recognized. In figure 4, each entity type is held as a constant in separate samples, such that the same conditions as test 1 and 2 can be represented on a per-entity basis. While this test does guarantee at least one entity is present in an utterance according to the sample's constant entity, an expected baseline can still only be set when considering the performance requirement of the IRS in a real-world scenario (section 4.2). Performance for the individual entities does not always decrease as the number of entities in an utterance increases. In some cases, performance stays the same as complexity increases and in rare cases performance even increases at higher complexity (such as budget between three and four entities per utterance). The most consistent performance is found for entities *number of children* and *number of adults*, which attain 100% performance for condition A for all complexity levels and are the best performing entities overall. Between conditions, accuracy for A is higher than B conditions for all entities except *budget*, for which performance remains the same. B2 performance is better or equal to B1 for input with entities up to and including three. However, the performance for B1 and B2 is identical for input with four entities and above. Both *city* entities are the lowest-performing types of entities, while generally better performance is found for the start date and budget entities.

4. Discussion

This paper aims to propose a test framework to automatically test an IRS, building upon a multitude of previous work (Vasconcelos, et al., 2017; Ruane et al., 2018; Bozic & Wotawa, 2019). The framework uses a CFG to generate utterances of varying complexity, optionally with spelling errors as noise. To show the framework's practical value, three tests were implemented as test oracles and were conducted using the framework. These tests reveal several strengths and weaknesses of the IRS and the benefits of using spell check preprocessing. This section discusses such revelations and contextualizes the performance of the IRS within real-world scenarios. Lastly, the shortcomings of the current test framework are discussed.

4.1 IRS strengths and weaknesses

First, inference from test 1 (figure 2) leads to the discovery that the IRS is very proficient at recognizing the overall intent of utterances. Performance can be considered near perfect if utterances contain three or more entities. If utterances contain fewer entities, especially if only a single entity is present, accuracy takes a hit. This is natural as utterances with a lower amount of entities are often short and may lack the rich context necessary to recognize the intent. If singular entity utterances contain noise, then performance is much lower if no spell check is applied. Apart from this scenario, spell check preprocessing has no effect, because accuracy does not decrease when spelling errors are present in the first place. This indicates that the IRS can often extract enough context from utterances to infer its intent, even with such noise.

Second, test 2 (figure 3) concerns IRS performance for all-or-nothing entity recognition. Accuracy is always highest when no spelling errors are introduced, and accuracy drops steeply when noise is added to utterances, indicating that noise does disturb IRS performance. Up to utterances with four entities spell check preprocessing diminishes this drop in performance, indicating that spell check preprocessing is effective. At higher complexity levels, the spell check preprocessing is insufficient in mitigating the noise. In general, recognition accuracy drops as the complexity of utterances increases. One exception to this can be seen for utterances without noise with between four and five entities per utterance, as no difference in accuracy can be inferred between these complexity levels. This is an unexpected result when considering the otherwise consistent downwards curve. Perhaps there is a peculiarity in the way that complexity is introduced using the current method, although this peculiarity could also be attributed to some oddity in how the IRS recognizes entities. With the current measurements and experimental conditions in mind, any reasoning behind this exception can only be considered as speculation. Regardless, all-or-nothing entity recognition is a weakness for the IRS for higher complexity utterances, irrespective if these utterances contain noise. This weakness is not necessarily a threat to the usability of the IRS in a real-world scenario (section 4.2).

Finally, test 3 (figure 4) reveals how the IRS recognizes specific entity types, where each entity type is held constant in separate samples. The most performant entities are *number of children* and *number of adults*, for which recognition of utterances without noise is perfect even at high complexity levels. For the IRS it is likely that such entities are rather easy to recognize as the IRS generally only needs to extract a numerical value and some limited context around this value. Note that the *budget* entity relies on a numerical value as well, but in addition a symbol representing a currency needs to be recognized for the built-in *money* entity type. It is expected that *budget* recognition accuracy is lower than that of other numerical entities. What may not be as obvious is why *budget* recognition performance is unaffected by noise. This may be a limitation with the way noise is introduced, as noise is not added to numerical values. A *budget* utterance may contain a word such as *euro* which can contain a spelling error, but this is not guaranteed. Other numeric entities such as *number of children* do always guarantee noise. As noise is not guaranteed for *budget*, this can be considered an oversight in the current CFG implementation (section 4.3). Regardless, even a small decrease in performance is not visible for *budget* with noise. Performance for *origin* and *destination city* entities are alike each other, which is natural as both entities require a city name and some context around it to be recognized properly (such as *from Berlin* and *to Paris*, respectively). The performance for the start date entity is relatively in the middle-ground. The CFG rules express the date entity as rather verbose phrases, such as *next Friday*, instead of clear date formats such as *dd-mm-yyyy*, indicating that this entity is not very trivial to recognize. Just as in test 2, performance for all entities except *budget* is lowered when spelling errors are introduced. Spell check always diminishes the effect of noise (if present) for utterances up to four entities per input, indicating that it is rather effective. At higher complexity levels, spell check has no effect, which also reflects what was found for test 2. Unlike test 2, there is less consistency in the increase of complexity decreasing performance. This indicates that different entity types are not recognized similarly: this is natural as they are expressed in language in dissimilar ways.

4.2 Real-world usage of the tested IRS

In a real-world scenario, the current IRS could be considered functional. It is very performant in the overall intent recognition of utterances, which is far above chance. Performant intent recognition can be considered a requirement as a chatbot is unable to reach a specified goal without it. To attain a solid user experience, the IRS should also be able to recognize entities within an intent. If a requirement is to recognize all entities within an intent without fail, then the IRS will not suffice. Across complexity levels and conditions, the IRS is likely to miss one or more entities. However, as test 3 indicated, individual entity recognition performance is very high for some entity types. In the absolute worst case, it is rather unlikely that the IRS will not recognize any entity within an utterance. It should be concluded that the current IRS will often miss some entities from utterances. By taking the entity recognition performance into account, a chatbot using such an IRS can implement logic to query the user for any missing details. This is often natural as a conversation is meant to consist of more than a single utterance. Whether or not

such a follow-up is acceptable or whether entities should be recognized in a single turn depends on the practical requirements and the context in which the chatbot is used.

4.3 Shortcomings of the current test framework

In general, the CFG is found to be functional in generating utterances, in line with previous work (Bozic & Wotawa, 2019). The introduction of noise also led to several findings indicating that noise influences IRS performance (Vasconcelos, et al., 2017; Ruane et al., 2018), especially for entity recognition. IRS performance was found to be strong for overall intent recognition while entity recognition was weaker, a finding supported by other work employing LUIS (Ruane et al., 2018). The similarities with previous work indicate a well-rounded implementation of previous work within the framework.

While the framework fits the problem domain well, it is limited in some regards. First, the CFG is used to generate very domain-specific utterances. The utterances can only be used to query a specific chatbot meant to book a trip. In addition, the current study only tests the *inform* intent, indicating that it does not cover all the chatbot scenarios even in this limited domain. Extending the current CFG within the discussed domain or even extending it to a different domain would be rather work-intensive, which is not ideal. A second limitation is that the noise introduced in the framework is rather simple and limited to spelling errors, even though an utterance can consist of a rich variety of noise. Third, the framework is designed to be compatible with the LUIS service. This does not indicate that the framework is completely incompatible with other services, as API requests within the framework can be modified to support other services. Rather, the current framework is mainly limited in that it is designed to test single intent utterances. End-users may pose utterances with multiple intents in chatbot conversations, which is currently not covered by the framework.

5. Conclusion

5.1 Recommendations for IRS implementation

The observations from this paper's experiments denote several effects of complexity and noise on IRS performance. From these observations, several recommendations can be made for implementing a chatbot and its IRS. First, it should not be assumed that all entities can be recognized in one utterance. The chatbot logic should be designed to take this into account and to query the user further when necessary. Such subsequent queries may be very specific in requesting a missing entity to ensure that all required entities are collected within a minimal number of turns. Secondly, enabling spell check preprocessing is a must. It can mitigate performance loss if utterances contain noise, at least in the form of spelling errors. At its worst, spell check does not impact the performance at all.

5.2 Current and future state of the framework

This paper presents a functional albeit preliminary framework and demonstrated its ability to test an IRS. Clear accuracy metrics can be generated from the current framework by using a variety of test oracles. Such metrics can be used to support various MLOps scenarios. The framework is automatic and can be useful in determining whether an IRS is ready for production, or in monitoring the performance of an IRS which is already in a production state. Crucially, the test oracles implemented thus far are varied and various parameters can be set to comply with performance requirements desired by a tester.

The most promising aspect of the proposed framework is that its shortcomings mainly come forth from the limited scope of the current paper. One critique of the current work is that other utterance generation techniques have not been explored. A more general-purpose utterance generator may be easier to extend and adapt for different chatbot domains compared to the CFG. For the next iteration of the framework, alternative techniques could be considered, perhaps based on natural language generation (Reiter & Dale, 1997). Within the general domain of sentence generation, ML-based approaches such as recurrent neural networks are used (Sutskever, Martens, & Hinton, 2011; Tang, Yang, Carton, Zhang, & Mei, 2016). Such existing methodologies may be useful in generating user utterances. Within the specific domain of chatbots, recurrent neural networks are applied to generate chatbot responses (Shang, Lu, & Li 2015). Kim, Oh, Kwon & Kim (2019) further iterate on this concept by employing a generative adversarial network approach for more dynamic context-based responses. It may be worthwhile to use such existing techniques to perform the opposite task of generating user utterances across domains and languages. In tandem with the development of a more general-purpose utterance generator, the framework could be expanded to support multi-intent utterances. Future work may also improve the framework from the perspective of introducing noise. The framework can be expanded to reflect other types of human errors, such as complex and context-dependent grammatical errors across interlingual and intralingual error categories (Richards & Schmidt, 2002). Lastly, the framework can be expanded to support other IRS services apart from LUIS. Compared to other shortcomings, this is a more trivial task as network requests can simply be extended to support other APIs.

While the framework is not yet mature, its current state is functional and allows future research and developments to iterate on it. As a result, the automatic testing of IRSs and by extent chatbots can be thoroughly improved.

Acknowledgments

Microsoft Corporation is not in direct involvement with the funding of this paper. During the writing of this paper, the author was employed as an intern. This thesis was written in tandem with daily involvement in other unrelated job responsibilities. Financial involvement is limited to the usage of LUIS, which was funded by an internal subscription intended for short-lived non-critical projects. The Microsoft logo and header image depicted in the front page are attributed to the Microsoft brand central.

References

- Abushawar, B., & Atwell, E. (2015). ALICE Chatbot: Trials and Outputs. *Computación y Sistemas*, 19(4). doi: 10.13053/cys-19-4-2326
- Alkemade, H., Mares, A., & Riel, Z. (2020). Brainy Chatbot. Retrieved May 11, 2020 from <https://github.com/microsoft/microsoft-teams-brainy-bot>
- Angmo, R., & Sharma, M. (2014). Performance evaluation of web based automation testing tools. *2014 5th International Conference - Confluence The Next Generation Information Technology Summit (Confluence)*. doi: 10.1109/confluence.2014.6949287
- Asri, L. E., Schulz, H., Sharma, S., Zumer, J., Harris, J., Fine, E., ... Suleman, K. (2017). Frames: a corpus for adding memory to goal-oriented dialogue systems. *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*. doi: 10.18653/v1/w17-5526
- Baresi, L., & Young, M. (2001). Test oracles. *Technical Report CIS-TR-01-02, University of Oregon, Dept. of Computer and Information Science, Eugene, Oregon, USA*.
- Bozic, J., Tazl, O. A., & Wotawa, F. (2019). Chatbot Testing Using AI Planning. *2019 IEEE International Conference On Artificial Intelligence Testing (AITest)*. doi: 10.1109/aitest.2019.00-10
- Bozic, J., & Wotawa, F. (2018). Security Testing for Chatbots. *Testing Software and Systems Lecture Notes in Computer Science*, 33–38. doi: 10.1007/978-3-319-99927-2_3
- Bozic, J., & Wotawa, F. (2019). Testing Chatbots Using Metamorphic Relations. *Testing Software and Systems Lecture Notes in Computer Science*, 41–55. doi: 10.1007/978-3-030-31280-0_3
- Brandtzæg, P. B., & Følstad, A. (2017). Why People Use Chatbots. *Internet Science Lecture Notes in Computer Science*, 377–392. doi: 10.1007/978-3-319-70284-1_30
- Braun, D., Hernandez-Mendez, A., Matthes, F., & Langen, M. (2017). Evaluating Natural Language Understanding Services for Conversational Question Answering Systems. *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*. doi: 10.18653/v1/w17-5522
- Chomsky, N. (1956). Three models for the description of language. *IEEE Transactions on Information Theory*, 2(3), 113–124. doi: 10.1109/tit.1956.1056813
- Dale, R. (2016). The return of the chatbots. *Natural Language Engineering*, 22(5), 811–817. doi: 10.1017/s1351324916000243
- Følstad, A., & Brandtzæg, P. B. (2017). Chatbots and the new world of HCI. *Interactions*, 24(4), 38–42. doi: 10.1145/3085558
- Foster, I., Zhao, Y., Raicu, I., & Lu, S. (2008). Cloud Computing and Grid Computing 360-Degree Compared. *2008 Grid Computing Environments Workshop*. doi:10.1109/gce.2008.4738445
- Howe, D. (n.d.). RiTa: the generative language toolkit. Retrieved March 28, 2020 from <https://github.com/dhowe/RiTaJS>
- Kim, J., Oh, S., Kwon, O.-W., & Kim, H. (2019). Multi-Turn Chatbot Based on Query-Context Attentions and Dual Wasserstein Generative Adversarial Networks. *Applied Sciences*, 9(18), 3908. doi: 10.3390/app9183908
- Linnenkugel, U., & Mullerburg, M. (1990). Test data selection criteria for (software) integration testing. *Systems Integration '90. Proceedings of the First International Conference on Systems Integration*, 709–717. doi:10.1109/icsi.1990.138737
- McTear, M. (2018). Conversation modelling for chatbots: current approaches and future directions. *Elektronische Sprachsignalverarbeitung*, 175–185.

- Microsoft (2019). *Frames* [Data file]. Available from <https://msropendata.com/datasets/1cc496ec-aaff-4576-b4bc-4a65798fa907>
- Microsoft (2020). MLOps: Model management, deployment, and monitoring with Azure Machine Learning. Retrieved May 2, 2020 from: <https://docs.microsoft.com/en-us/azure/machine-learning/concept-model-management-and-deployment>
- Microsoft (2020). Tutorial: Batch test data sets Retrieved May 2, 2020 from: <https://docs.microsoft.com/en-us/azure/cognitive-services/luis/luis-tutorial-batch-testing>
- Microsoft (n.d.). Bing Spell Check. Available from <https://azure.microsoft.com/en-us/services/cognitive-services/spell-check/>
- Microsoft (n.d.). Language Understanding Intelligent Service. Available from <https://eu.luis.ai/home>
- Microsoft (n.d.). LUIS Programmatic APIs v2.0 Retrieved May 2, 2020 from: <https://westus.dev.cognitive.microsoft.com/docs/services/5890b47c39e2bb17b84a55ff>
- Reiter, E., & Dale, R. (1997). Building applied natural language generation systems. *Natural Language Engineering*, 3(1), 57–87. doi: 10.1017/s1351324997001502
- Richards, J. C. & Schmidt, R. (2002). *Dictionary of language teaching and applied linguistics* (3rd ed., pp. 201-202). London: Longman.
- Royce, W. (1970). Managing the development of large software systems: Concepts and techniques. *Proceedings*, 1-9
- Ruane, E., Faure, T., Smith, R., Bean, D., Carson-Berndsen, J., & Ventresque, A. (2018). BoTest. *Proceedings of the 23rd International Conference on Intelligent User Interfaces Companion - IUI 18*. doi: 10.1145/3180308.3180373
- Ruparelia, N. B. (2010). Software development lifecycle models. *ACM SIGSOFT Software Engineering Notes*, 35(3), 8-13.
- Rychalska, B., Glabska, H., & Wroblewska, A. (2018). Multi-Intent Hierarchical Natural Language Understanding for Chatbots. *2018 Fifth International Conference on Social Networks Analysis, Management and Security (SNAMS)*. doi: 10.1109/snams.2018.8554770
- Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., ... & Dennison, D. (2015). Hidden technical debt in machine learning systems. *Advances in neural information processing systems*. 2503-2511.
- Shang, L., Lu, Z., & Li, H. (2015). Neural Responding Machine for Short-Text Conversation. *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. doi: 10.3115/v1/p15-1152
- Sutskever, I., Martens, J., & Hinton, G. E. (2011). Generating text with recurrent neural networks. In *Proceedings of the 28th international conference on machine learning (ICML-11)* (pp. 1017-1024).
- Tang, J., Yang, Y., Carton, S., Zhang, M., & Mei, Q. (2016). Context-aware natural language generation with recurrent neural networks. *arXiv preprint arXiv:1611.09900*.
- Turing, A. M. (1950). Computing Machinery And Intelligence. *Mind*, LIX(236), 433–460. doi: 10.1093/mind/lix.236.433
- Vasconcelos, M., Candello, H., Pinhanez, C., & Santos, T. D. (2017). Bottester. *Proceedings of the XVI Brazilian Symposium on Human Factors in Computing Systems - IHC 2017*. doi: 10.1145/3160504.3160584
- Veldsink, M. (n.d.). Miki Discord Bot. Retrieved May 2, 2020 from: <https://github.com/Mikibot/bot>
- Weizenbaum, J. (1966). ELIZA---a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1), 36–45. doi: 10.1145/365153.365168
- Worswick, S. (n.d.). Mitsuku. Retrieved May 2, 2020 from: <https://www.pandorabots.com/mitsuku/>
- Zadrozny, W., Budzikowska, M., Chai, J., Kambhatla, N., Levesque, S., & Nicolov, N. (2000). Natural language dialogue for personalized interaction. *Communications of the ACM*, 43(8), 116–120. doi: 10.1145/345124.345164
- Zhu, L., Bass, L., & Champlin-Scharff, G. (2016). DevOps and Its Practices. *IEEE Software*, 33(3), 32–34. doi: 10.1109/ms.2016.81

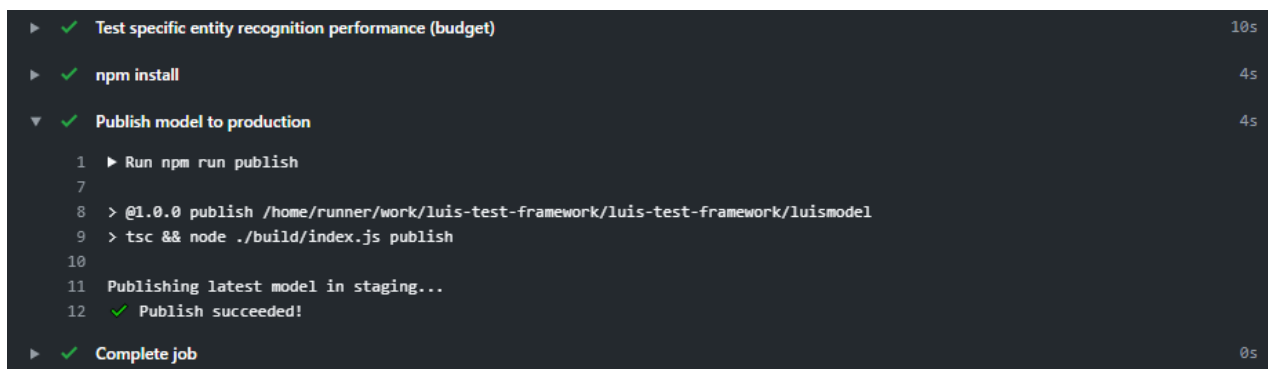
Supplementary Materials

The test framework is publicly available as an open source repository². It is a command-line interface (CLI) program written in TypeScript. Within this repository are help files which elaborate on the framework's usage. This section elaborates on several important preemptive remarks. First, the code used to preprocess the Frames dataset and send it to LUIS is included alongside the framework itself. Second, tests conducted in this paper are implemented as the framework's test oracles and can be used to recreate the tests using new CFG samples. For transparency, a data dump of the exact CFG samples and LUIS output used in this paper is included. It is also possible to only generate some example utterances without connecting to LUIS. Lastly, a proof of concept for a typical MLOps scenario is included. In this scenario, if the test run succeeds, the framework uses the LUIS API to promote the model's status from a test (staging) environment to a production environment.

Due to the framework's nature as a CLI program, it can be used within a variety of environments to execute tests automatically. As an example, the repository contains YAML files compatible with GitHub Actions as the provider for automated pipelines³. With minimal changes, such YAML files can be adopted to support a different scenario or even a different pipeline provider.

Figure 5

GitHub Actions approving a test run and publishing the LUIS model to production using the framework.



```
▶ ✓ Test specific entity recognition performance (budget) 10s
▶ ✓ npm install 4s
▼ ✓ Publish model to production 4s
  1 ▶ Run npm run publish
  7
  8 > @1.0.0 publish /home/runner/work/luis-test-framework/luis-test-framework/luismodel
  9 > tsc && node ./build/index.js publish
 10
 11 Publishing latest model in staging...
 12 ✓ Publish succeeded!
▶ ✓ Complete job 0s
```

² To consult the framework, refer to <https://github.com/Zenulous/luis-test-framework>

³ For more information on GitHub Actions, refer to <https://github.com/features/actions>

Afterword & Self-reflection

The motivation for this paper is twofold. I have previous experience in the field of software testing across test domains and found this to be extraordinarily important in assuring the quality of software. Software architecture is becoming increasingly complex, and to support such complicated and intricate systems proper testing needs to be in place. Secondly, I have been heavily involved in the development of a chatbot during my Microsoft internship. I have full confidence that a chatbot interface can provide a better user experience than traditional web applications. I feel strongly about this for chatbots with natural language interfaces, although there is a lot of work to be done to improve their reliability. This project is a way to express the progress in this domain, but also to indicate shortcomings. Apart from providing analytical results for this project, I desired the result to be more concrete. To this end, I decided on releasing a preliminary framework as open source software. Admittedly, I find it disappointing if a paper poses an intriguing concept, yet no open source references is included. If one of the goals of academia is to build upon the works of others, then I personally find it crucial to release practical examples of the work that has been done. Understandably yet regrettably, this is not always applicable and/or possible. Regardless, delivering an open source repository was one of the most motivating personal goals for this project.