Natural Language Processing for virtual assistants: what contribution synthetic data could bring to intents classification?

Sylvain Daronnat

Traitement automatique du langage pour les assistants virtuels :
Quel apport des jeux de données synthétiques pour la détection d'intentions ?

Natural Language Processing for virtual assistants:
What contribution synthetic data could bring to intents classification?

DARONNAT

Sylvain

Sous la direction d'Estelle Delpech (Airbus)

Tuteur : François Portet (LIG)

Entreprise : Airbus Group

UFR LLASIC

Département Informatique Intégrée en Langues, Lettres et Langage (I3L)

Mémoire de master 2 mention Sciences du Langage - 20 crédits

Parcours : Industries de la Langue

Année universitaire 2016-2017

AIRBUS

AIRBUS

Traitement automatique du langage pour les assistants virtuels :
Quel apport des jeux de données synthétiques pour la détection d'intentions ?

Natural Language Processing for virtual assistants:
What contribution synthetic data could bring to intents classification?

DARONNAT

Sylvain

Sous la direction d'Estelle Delpech (Airbus)

Tuteur : François Portet (LIG)

Entreprise : Airbus Group

UFR LLASIC

Département Informatique Intégrée en Langues, Lettres et Langage (I3L)

Mémoire de master 2 mention Sciences du Langage - 20 crédits

Parcours : Industries de la Langue

Année universitaire 2016-2017

Remerciements

Je tiens tout d'abord à remercier Estelle Delpech, ma tutrice à Airbus, qui m'a accompagné tout le long de ce projet et qui m'a permis de gagner en autonomie et en confiance en moi, tout en me poussant à toujours faire mieux. Je la remercie également pour m'avoir donné la chance de réaliser un stage sur un sujet passionnant dans une entreprise et un département qui valorisent la connaissance, la recherche, et l'esprit d'équipe.

Je remercie l'ensemble de l'équipe de développement du projet d'assistant virtuel, qui m'ont inclus dans leurs meetings et qui m'ont permis de me familiariser avec leurs problématiques.

Je remercie également l'entièreté du service des facteurs humains de l'entreprise, qui ont toujours su allier bonne humeur au travail tout en étant performants et toujours là pour m'aider en cas de besoin. Mes remerciements vont vers Kevin, Caroline, Thomas, Marielle, Christine, Laurent, Amine, Nataly, Luigi, Emmanuelle, Marie, Florence, Benoit, et sans oublier mon compagnon stagiaire, Franz.

Je remercie également messieurs François Portet, et George Antoniadis qui m'ont soutenu et accompagné dans ce projet de stage au sein d'Airbus.

Je tiens aussi à remercier mes camarades du Master Industries de La Langue, sans qui l'ambiance n'aurait pas été aussi bonne. Mes pensées vont donc tout particulièrement vers William, Renaud, Ieva, Louise, Pauline, Ali, Floriane, Elena et Maria.

Je tiens également à remercier mes parents, qui m'ont toujours soutenu quoique je fasse, et sans qui je n'aurai jamais pu m'épanouir et faire des études dans un domaine que j'affectionne.

Enfin, mention spéciale pour Lucas, qui me soutiens depuis l'école primaire et qui me fait relativiser la difficulté de mes études en étant en droit, Mathilde, chez qui j'ai réussi sans m'en rendre compte à lui faire découvrir et aimer les humanités numériques, et Laura, qui m'a soutenu sans faillir en tant que colocataire et, surtout, confidente pendant que j'étais sur Grenoble.

DÉCLARATION

1. Ce travail est le fruit d'un travail personnel et constitue un document original.
2. Je sais que prétendre être l'auteur d'un travail écrit par une autre personne est une pratique sévèrement sanctionnée par la loi.
3. Personne d'autre que moi n'a le droit de faire valoir ce travail, en totalité ou en partie, comme le sien.
4. Les propos repris mot à mot à d'autres auteurs figurent entre guillemets (citations).
5. Les écrits sur lesquels je m'appuie dans ce mémoire sont systématiquement référencés selon un système de renvoi bibliographique clair et précis.

NOM : **DARONNAT**

PRENOM : **Sylvain**

DATE : **07.09.2017**

SIGNATURE :

Table of contents

Introduction **8**

1. A virtual assistant technology at Airbus: 9
2. Internship goals: 10
3. Outline 11

Chapter 1 – Context and environment of the internship **12**

1. The Airbus Company 13
2. The Human Factor department: missions and purpose within Airbus 13

Chapter 2 – Report on paraphrasing techniques **15**

1. Introduction 16
2. Computer paraphrasing and generation 16
3. Conclusion 19

Chapter 3 - The Natural Questions Dataset **20**

1. Advices from experts on the methodology for collecting/creating questions 21
   - 1.1. Interview with a PhD student in cognitive science 21
   - 1.2. Interview with linguistics and human factor specialists 21
2. The methodologies for natural question collection 22
   - 2.1. Settings and general specificities of the interviews 23
3. Developer driven approach 25
   - 3.1. Validation of a set of queries with a pilot 25
4. User Driven Approach 26
   - 4.1. Use of specific scenarios during interviews with pilots 26
   - 4.2. Use of open type questions during interviews with pilots 28
   - 4.3. Problem Encountered 29
   - 4.1. Solution found to create a functional baseline dataset 30

Chapter 4 - The Synthetic Question Generator **33**

1. Specifications of the question generation program 34
2. Resources used by the program 35
   - 2.1. The Flight Operating Crew Manual (FCOM): 35
   - 2.2. Lexinet: 35
   - 2.3. Electronic Centralized Aircraft Monitoring (ECAM) Messages 36

3. Early design of the program² 37
4. Why we changed our approach 39
5. The final version of the synthetic question generator program 40
6. The final output of the question generation program 42

Chapter 5 – Using and testing the synthetic dataset 44

1. Why integrating machine learning into the project 45
2. RasaNLU: The Natural Language Toolkit 45
   2.1. How does the library work? 46
   2.2. Using RasaNLU 47
   2.3. The training 48
   2.4. Using the model 49
   2.5. Conclusion: 49
3. A change of program to experiment with the datasets: 50
   3.1. Why we had to stopped using RasaNLU 50
   3.2. The Model creation 51
   3.3. Predictions using the Models 51
   3.4. Conclusion on the baseline model creation and predictions 52

Chapter 6 – Evaluation 53

1. The evaluation metrics and tools 54
   1.1. Precision: ²54
   1.2. Recall: 54
   1.3. F1 Score 54
   1.4. (Standardized) Type Token Ratio: 55
   1.5. Confusion Rate: 55
   1.6. Confusion Matrix 55
2. The reference dataset 56
3. Evaluation results 56
   3.1. Results on the maximum dataset 56
   3.2. Results on balanced dataset excerpts 59
   3.3. Comparison between the human and machine classification 61
4. Conclusion on the evaluations 63

Conclusion 64

1. Reflection on the internship 65

2. Perspectives 65

Bibliography & Webography 67

Glossary and Acronyms 69

Figures - table of contents 71

Equations - table of contents 72

Tables - table of contents 73

Annexes – table of contents 74

Introduction

1. A virtual assistant technology at Airbus:

Airplanes are among the most technologically equipped means of transport in the world. Since its debut, Airbus always thrived to implement more and more technological upgrades in their airplanes, in order to both reduce the amount of the work for the pilots and to make airplanes more efficient and reliable. Following this logic, a project concerning the development of a virtual agent emerged within the company. The primary goal expressed during the early development stages of this virtual assistant is to assist pilots or maintenance personnel in the access to procedure, manuals, or any operational documentation in order to both reduce the cognitive load and to begin a transition into a more automatized environment.

The interaction between the virtual agent and the end-users must be easy and natural; this is why, in the end, the system will feature both voice recognition functionalities as well as a chat-box interface.

In the development of a project of such scale, which addresses not only human-computer interaction aspects but also data structuration and accessibility problems, we need first and foremost to precisely define what we want to do: what are the use-cases, what kind of information the system will be looking for and what kind of data we need.

The work detailed in this document was carried out within the development of the second version of the virtual assistant project. This allows us to have some insights about what was successful in the first version of the system and what was not, so we can decide on the problems that needs to be addressed first.
One of these problems was a lack of flexibility. During the final integrated tests, the end-users expressed their concerns about the system not being able to recognize most of their vocal queries due to a required wording being too restrictive and not allowing any freedom or variation during the interaction with the virtual assistant.

To address this issue, a decision has been taken to build the second version of the virtual agent toward a machine-learning oriented approach, in order to ensure a more flexible and dynamic program.
In a machine-learning context, data are crucial and the process of collecting them can be expensive in both time and money. This is the problem that this internship will address, by researching and creating a method to generate synthetic data, and experiment on them in order to see if using non-natural data is indeed effective to enhance the natural language understanding part of the system.

2. Internship goals:

This internship will be focused on the Natural Language Understanding (NLU) part of the virtual assistant project. The mains objectives are listed as follows:

- **To collect the largest possible corpus of « natural questions »:** What we want to achieve is to set up a methodology to collect the most natural possible queries an end-user would ask the system. This methodology should also allow us to have a large and diversified enough dataset in order to maximize the quality of our set of synthetic queries.
- **To create a program capable of generating "human-like" synthetic questions:** With the natural data and the information previously collected, we aim to develop a program capable of generating sentences carrying not only the correct semantic information, but also capable of reproducing the way the users would express them, with the same errors, vocabulary and sentences structures for instance. Therefore, the main goal of this synthetic dataset is to be qualitative before being large.
- **To understand how and why queries are produced by users:** By generating queries equivalent to the natural questions collected, we also want to understand the general intentions behind these questions, for instance why does the user have the need to ask about a specific subject in a specific situation? What are the different needs any user can have in a specific situation? How should the virtual agent react in a certain context? These questions will help us gain insights on the reason why the users would use the VA[1] in the first place, and why they would use it, allowing us to focus on certain specific aspects rather than some others.
- **To test the synthetic queries dataset with a classification program:** As this VA project is developed using machine-learning technologies, it is important to test the newly created synthetic dataset by using, preferably, the same technologies in order to evaluate the pertinence of the results produced by our generation program.
- **To analyse and understand the results of the experiments:** After having created a set of synthetic queries and having used them in a machine-learning program, we will study the yielded results and draw conclusions from them. These conclusions will allow us to see if our approach was pertinent and why. If our approach is not pertinent, we will be able to see what to do or change in order to improve the results.

[1] Virtual Assistant

A planning created at the beginning of the internship with milestones and estimated times of developments per steps is available in annex 1.

3. Outline

In the first chapter, we are going to give some context about the internship environment; where it took place, what are the missions of the Human Factor department, why and how this subject was addressed within Airbus.

In the second chapter, we are going to study an overview of computer paraphrasing and the generation of synthetic data in order to augment a system.

In the third chapter, we are going to study methodologies that would help us to create a baseline natural dataset; then we are going to detail the problems encountered after having created the methodologies and how we addressed them.

In the fourth chapter, we are going to present the generation program we created. We will detail the functioning of the earliest versions while explaining why we had to change our approach toward a more constrained, rule-based generation system.

In chapter five, we are going to detail the tools and scripts we used to experiment with the synthetic datasets: how we went from a machine learning toolkit to a simpler classification script based on lexical weights.

In the sixth chapter, we will present our evaluation metrics and the results yielded by our experiments.

In the conclusion, we will do a recap of our initial hypothesis. Then, we will reflect upon the internship in itself: how it went and what we gained from it. Finally, we will open new perspectives on the subject: what to change and what perspectives to address in the future version of the system.

Chapter 1

–

Context and environment of the internship

1. The Airbus Company

Founded in 1970 as "Airbus industry GIE", Airbus SE began as a European consortium of aircraft manufacturers. Over time its popularity and range of activities have evolved, and now Airbus Group is present on the helicopter scene with Airbus Helicopter as well as in the defence sector with Airbus Defence & Space.

Nowadays, the Airbus Group is one of the most important leaders in the market of commercial jetliners and military airlifters. In 2017, the airbus group employed a total 133,782 people over its sixteen sites in France, German, Spain, the USA, Spain, China, Japan and India. One of its main site is based in Blagnac, France. This location is one of the largest of all of the Airbus sites in the word; it is home to the Final Assembly Lines (FAL) of the A320, A330 A350 and A380 product family as well as the headquarters of the whole Airbus Group.

The company is most famous for having launched the first viable fly-by-wire airliner with the A320 in 1987 and for having created the world largest passenger airliner with the A380 in 2005.
In 2016, the company delivered its 10,000th aircraft, making the airbus fleet having performed more than 110 million flights carrying 12 billion passengers globally.[2]

2. The Human Factor department: missions and purpose within Airbus

This internship takes place in the Human Factor service of the company. The Human Factor department is part of AIRBUS "Engineering System and General System" division.

The term "Human Factor", in the industry, refers to the study of "how humans behave physically and psychologically in relation to particular environments, products, or services. […] A human factor specialist typically has an advanced academic degree in Psychology or has special training."[3]
At Airbus, the Human Factor department was created about 20 years ago to ensure that the tools and systems designed for ground and flying personnel are safe, adapted to the users and compliant with the guidelines of the aviation safety agencies, such as the European Aviation Safety Agency (EASA) or the Federal Aviation Administration (FAA) for instance.

[2] https://fr.wikipedia.org/wiki/Airbus (consulted on 08.31.2017)

[3] http://searchmicroservices.techtarget.com/definition/human-factors-ergonomics (consulted on 08.31.2017)

The range of services and abilities of the Human Factor department is large. Within the same team people with very different abilities and academic backgrounds are working together such as neuroscientists, ergonomists and linguists. The tasks they have at hands are equally diversified: they intervene on different levels from a research and development standpoint by working on new designs, interfaces and human-machine interface systems, to more operationally oriented subjects such as the certification of new systems, designs and functions of alarms.

The Human Factor department, over its lifetime, has grown to include more and more domains relevant to its activities. Recently, the service expressed its interest in more NLP-related technologies; such as speech to text applications for pilots and ATC[4]. In this context of automation, the relationship between the R&D[5] department and the Human Factor has strengthened throughout the year, which is why some problematics of the virtual assistant project are shared between the human factors and the R&D department.

In this context of collaboration, the internship was included in the weekly meetings of the VA project. It allowed discussing technical issues and deadlines while focusing on specifics subjects such as natural language understanding, intent classification or research on the machine-learning library to use with the system.

[4] Air Traffic Control

[5] Research and Development

Chapter 2

–

Report on paraphrasing techniques

1. Introduction

In order to have more insights on how to proceed with the development of the generation program, we decided to review a few documents on data generation and paraphrasing, as our goal is to produce quality data, as close as possible to the originals.

In the field of automatic text generation, the act of paraphrasing a sentence to output multiple other ones is not the most common approach. Few investigations were made towards this goal, especially toward paraphrasing questions, but we will detail some of these experiments in this chapter.

2. Computer paraphrasing and generation

When the need to augment the capacity of a corpus arises, multiple constraints also appear. Collecting data is both a long and expensive process and to reduce this burden, a program capable of paraphrasing an input data in order to have the maximum number of relevant equivalent synthetic data in return would be an interesting tool for both research and private experiments.

By studying the work of Schank, Goldman, Rieger III Riesbeck (1975), Nakov (2008), Marton, Callison-Burch and Resnik (2009) we will first discuss a way to generate variations of an input corpus by simply paraphrasing it. We are then going to study how this kind of transformation was done and what are some common approaches to this problem nowadays.

We can find a first occurrence of paraphrasing using computer generation in a thesis published at Yales University in 1975. Even if the technology mentioned in this article is ancient compared to what we can do today, we can still consider it the first attempt at addressing this kind of problem. It is interesting to note that, in their approach, the program is divided into 3 stages: “conceptual analyser”, “memory and inference program” and “generator” (Schank & all, 1975). Even if the systems architecture is different from what we would develop nowadays, we can still see similarities in the chosen approaches. For instance, the “conceptual analyser” is described as a mean to “takes a subset of English strings and maps them into a deep conceptual representation of their meaning” (Schank & all, 1975) which can be compared to the use of modern “Synsets” from the English lexical database WordNet. Synsets are described as “sets of cognitive synonyms […] each expressing a

distinct concept".In Wordnet, "Synsets are interlinked by means of conceptual-semantic and lexical relations."[6]

The name of the program developed by Schank & all is "MARGIE" for "Memory, Analysis, Response Generation, and Inference". In their work, a specific approach using conceptual dependency was detailed. This theory can be summarized as being centred around "a meaning structure […] underlying natural language that, when clearly defined, should serve as the basis for all processes that use natural language as input." Following this definition, the author of "MARGIE" designed the program to be capable of identifying key semantic term that carry most of the information of the question. Then, they associate each term with a specific tag (again, similar to the tagset available in WordNet). Each tag can be divided into specifics interchangeable structures that allow the program to output final paraphrases that keep the core meaning of the original input sentence.

As we mentioned in the last paragraph, representing and associating words by their semantic meaning seems crucial to the paraphrasing process. More recently, in 2008, Nakov and Hearst presented their work as part of the ECAI (European Conference on Artificial Intelligence) in which they detailed their approach on solving "relational similarity problems by using the web as a corpus". With their methods, the authors were able to achieve "state-of-the-art results on various relational similarity problems." (Nakov and Hearst, 2008).

In the described approach, they first "mine the Web for sentences containing the target nouns", then they extract "the connecting verbs, prepositions, and coordinating conjunctions" which they use in "a vector-space model to measure relational similarity" (Nakov, Hearst, 2008). After having extracted the sentences, they first analyze it with a part-of-speech parser (in this case, the Stanford parser) and then replace specific syntactic patterns found in the input query by other patterns according to predetermined association rules. For instance:

NP (proper noun) **NC** (common noun) => (refers to) **PP** (personal pronoun)

Constraining rules were also created in order to refine the possible transformations. For instance, the deletion of certain determiners was implemented in case of conflict between two proper nouns. The use of the Web as a corpus enabled Nakov and Hearst to process a large amount of data, allows the classification and correlation rates between target words and their predicted substitute to give "state

[6] https://wordnet.princeton.edu/ Definition of Synsets (consulted on 08.31.2017)

of the art performance". On the evaluation realised on the Levi-214 dataset, the system yielded up to "50% accuracy using verbs and prepositions" (Nakov and Hearst, 2008) to isolate potential substitutes. By comparison, the human subjects that were given the same substitution task achieved up to "78.4% accuracy" (Nakov, Hearst, 2008). The main advantage of this technique is that "it does not require knowledge about the semantics of the individual nouns", but at the same time "it might not work well for low-frequency words" (Nakov and Heast, 2008).

Another task that could greatly benefit from paraphrasing is statistical machine translation. In the approach of Marton, Callison-Burch and Resnik (2009) we can see a method for improving SMT Using Monolingually-Derived Paraphrases.

The methodology presented in the work of Marton, Callison-Burch and Resnik (2009) is different from the more common ways of improving SMT[7] which are often based on the use of parallel corpus with one being in a source language and the other in a target language.

In the approach of Martin, Callison-Burch and Resnik, they generate paraphrase candidates for out of vocabulary words in a monolingual corpus. To achieve so, the authors use "collocational profiles" (Martin & all, 2009) based on n-gram statistics in order to keep track of the most present co-occurrences throughout the text. This method is then put into use on a larger scale by generating "phrasal distributional profiles" that can contain up to 6 words before or after a sentence. Afterwards, each sentence is given a score that indicate if, within a text, one word or string is likely to appear after or before the input sentence. This approach allows generating the best possible candidates for an out of vocabulary phrase or word using previous occurrences, but it does have flaws. For instance, let's consider a short and non-specific sentence such as the following:

"*The __ is here*." (With the "_" symbolizing a placeholder for an OOV[8] word).

In such a sentence, the program would likely generate many probable (and thus, incorrect) candidates given the very common and ambiguous structure of the sentence. On the other hand, if the input is a very large sentence, the system may not be able to find a close occurrence to the input data and thus be unable to rank or even output possible candidates for the OOV sentence. In order to reduce the

[7] Statistical Machine Translation

[8] Out of Vocabulary

risk of having too many or few candidates, a confidence rate was created by "filtering out paraphrases" under a certain threshold (Marton, Callison-Burch and Resnik, 2009).

Overall, the method presented in this article was able to demonstrate that monolingually derived paraphrases can improve the performance of SMT systems. The metric used in this experiment is BLEU (Bilingual Evaluation Understudy) which gives a higher score the more an output from a statistical machine translation system "correlate to a human translation"[9]. In their experiments, the BLEU score increased from 15.21 points for the baseline system to 16.88 points for the paraphrases-augmented system on a dataset of 29,000 lines.

3. Conclusion

To conclude on the different paraphrasing methods and ways of finding and linking equivalent or related words, we can say that there is not one main method to achieve the creation of a reliable output, as the goals and definition of "equivalent" vary from one task to another. We ask the following questions: do we want to find interchangeable words carrying the same semantic meaning (Synsets)? Or do we want to find field-related words? In the end, how are we sure that the correlation between the input sentence or word and its paraphrases are maximized?

Even if there is not one generic way to address these issues, we were able to notice that in the three approaches that we presented, *the semantic* representation of the concept and terms we want to linked is crucial to the paraphrasing task. This leads us to take into account the context in which the words are used, and the intent with which they are employed.

[9] https://www.kantanmt.com/whatisbleuscore.php information on the BLEU score (consulted on 08/31/2017)

Chapter 3

-

The Natural Questions Dataset

In this chapter, we are going to details the process in which we created a methodology to collect natural queries from end-users. Although the VA project is open to all sorts of end-users (pilots, maintenance, training), we chose to focus on pilots as hypothetical end users for two reasons:

- they are the most common target users of the Human Factors department;
- we had access to people in the Human Factors department who had a pilot license.

Hence, we had better access to people with knowledge of our hypothetical end-users. First, we are going to summarize the advices and opinions we got from Human Factor and Aeronautic experts. Then we will explain how we structured our methodologies and what are their main specificities.

1. Advices from experts on the methodology for collecting/creating questions

1.1. Interview with a PhD student in cognitive science

Before creating a methodology, we confronted our ideas and points of views with various experts in the aeronautical field in order to have a better understanding of the interviewing process as well as specifics advices concerning our approach according to our needs.

One of the first person to be interviewed was Amine Laouar, PhD student in cognitive science at Airbus and ex fighter pilot for the French military. Even if our use-case was closer to an information retrieval system than a bot capable of actually interacting with the user, Amine told us that pilots would likely try to make the virtual agent do actions that they usually do themselves. These actions would not be strictly speaking critical to the flight security, but would be more related to the automation of lengthy processes. Instances of these “automatable” processes can be, for example, entering a flight plan in the system by taking into account weather updates, selecting different airstrips or changing the current flight level.

1.2. Interview with linguistics and human factor specialists

An interview was also had with Emmanuel Cannesson and Laurent Spaggiari, both linguists and human factor specialists in the Airbus Human Factor Department. They are also very familiar with certification processes and preliminary interviews with pilots and human factor experts, so their input was precious to us.

From this interview, we got a lot of information. The first inputs we got from them were about the previous problems of the first version of virtual assistant. For instance, we learnt that the main complaint the end-users made toward the system was that the virtual agent was not flexible enough concerning the wording recognizable by the system. Most of the time, the users had to say very specific queries in order to be correctly recognized and parsed by the system. This can be explained by the fact that most of the questions recognizable by the virtual assistant were hard-coded into it, the lack of variation was thus impossible to avoid.

The lack of flexibility of the previous version of the virtual agent led the development team of the virtual assistant to shift toward a more machine-learning base program in order to be able to allow a freer wording. This also highlighted the importance of the natural language understanding part within the virtual assistant project.

Concerning the natural language collecting methodologies, Laurent and Emma expressed some concerns about the way pilots would react to an interview that is too open and context-less. For them, it would likely get very quickly off-topic, lessening the benefit of the interviews. According to them, a better way to conduct interviews would be to ask broad-questions about our use-cases, such as “what kind of document do you usually refers to” or “what kind of need do you have in a specific situation”. So that we may know what are their needs and how to address them in our baseline dataset.

In order to know about what structures pilots would use, they simply advised us to use our linguistic knowledge as they do not think that the structures of the queries formulated by a person related to the aeronautical field would be much more different from someone who is not an expert.

2. The methodologies for natural question collection

The first step is to create a set of natural queries created by end-users. In this document, we are presenting three different approaches to collect what we call “natural questions” from pilots. These “natural questions” consist of questions and queries that a pilot would see himself asking to a search engine given a specific operational scenario. For now, we are not interested in collecting anything else but queries seeking information. Direct commands or orders awaiting interaction with the system (such as changing a destination or doing any action in general) are not our main focus during the corpora creation. We also want to know why test subjects would ask a specific question and what kind of answers they await, in order to gain knowledge about the needs they are expressing.

The approaches discussed in this document are divided into two paradigms: developer driven or user driven that can be subdivided into three distinct methodologies.

2.1. Settings and general specificities of the interviews

In order to collect the maximum amount of information, we defined a few important general specificities that we will apply to all of our interviews:

- **Recording of the interviews:** Each interview will be recorded using an orientable microphone. This will allow us to test the recorded queries with the Airbus speech-to-text program in order to isolate the most common oral disfluencies that can be found.
- **Using of a specific aircraft:** In order to collect questions relevant to our system, we decided to work on the most common aircraft found at the company, so all of our use-cases will virtually take place in a A320.
- **Use of English only queries:** During the interviews, English will be the mandatory and unique language in which queries would have to be expressed, as it is the default language for all airline companies around the world.
- **Not more than 2 hours of interviews:** In order to collect a sufficient amount of data while staying "natural", the individual interviews should not exceed a 2 hours duration. This would prevent frustration from the interviewee part, which could lead to low quality answers.

In order to get the most from these meetings, we also have to be clear and succinct in the explanation of what we want to collect and how we plan to do it. In order to ease these explanations, we plan to compare the virtual assistant to mainstream virtual agent, such as "Ok Google" from the Google Company or "Siri" from Apple.

As we expect to get many unrelated questions depending on the methodology used (see figure 1 on following page), we will sort the relevant questions to our use-cases afterward, allowing us to modify our methodologies if needed in order to optimize the amount of information we will get from the next interviews.

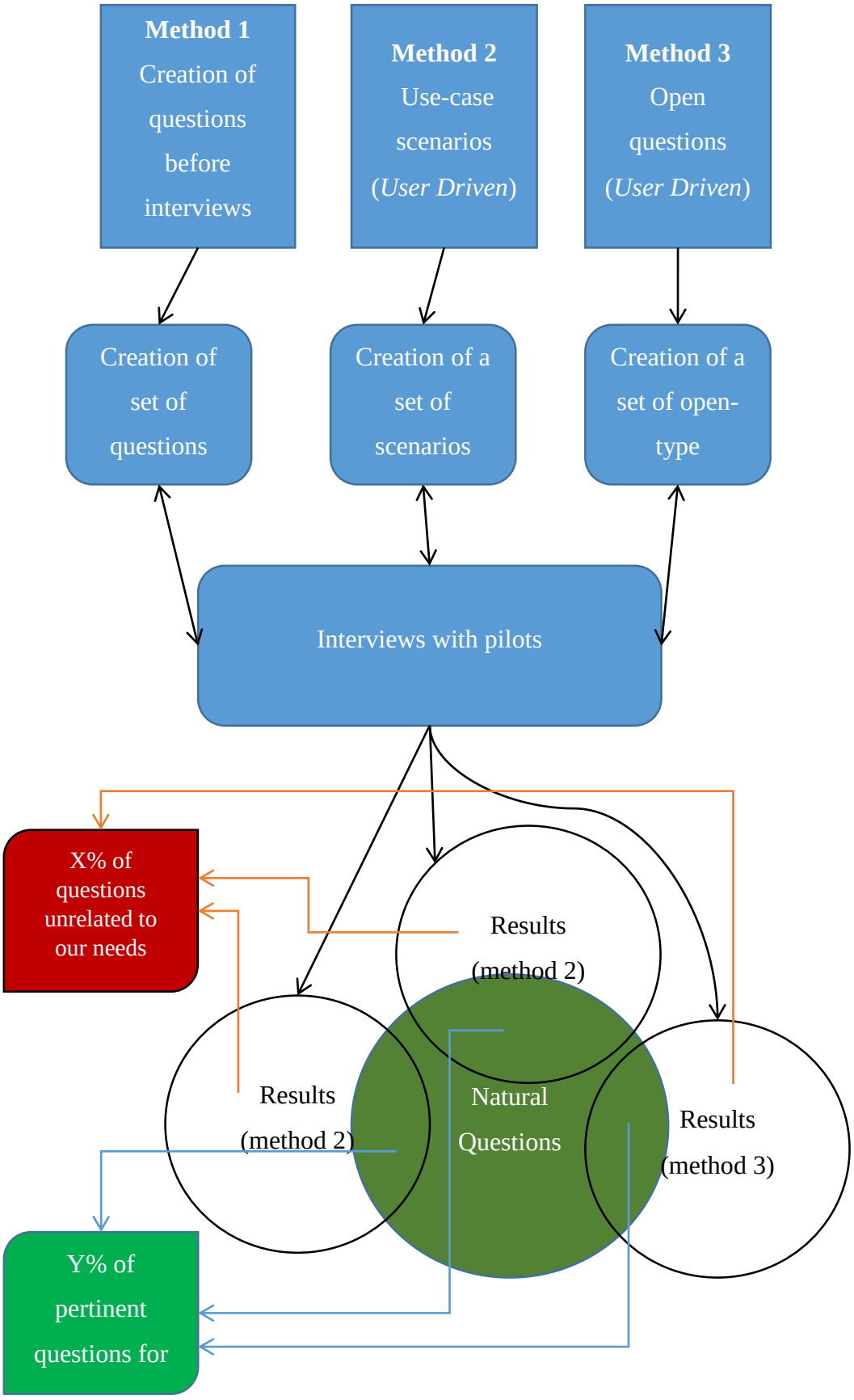


Figure 1: Schematic view of the three approaches

3. Developer driven approach

3.1. Validation of a set of queries with a pilot

3.1.1. Redacting a set of queries

In this approach, we first create a set of queries by using operational documentation such as the Flight Crew Operating Manual[10] (FCOM) for instance. In this approach, we assume that pilots would not be using many different queries structures than any other persons using another kind of virtual assistants. We choose to presume that our linguistic knowledge is enough to create queries using only operational documentation as reference and that we have enough knowledge of pilot operations.

We also rely a lot on previous works from the first version of the virtual assistant in order to have the largest possible scope of what subject a pilot would address and how he or she would address it (which words or structures were most commonly used for instance).
An example of the kind of question we can create:

- *"What is **APU** for?"*
- *"What does **cavalry alarm** mean?"*

All of these questions can be classified by patterns, structures and topics addressed. This will later allow us to understand what kinds of subjects we are often dealing with and what structures we should focus on in the question generation program.

3.1.2. Confronting the set of queries with pilots

In this second part of the methodology, we book interviews with pilots, we explain our methodology, and we present them the set of questions we created.

The goal of this experimentation is to see to what extent a question seems valid or not according to the pilot. We can then emphasize on specific topics and structures in order to make our set of natural questions closer to the operational reality.

[10] FCOM: Flight Crew Operations Manual, operational documentation used by pilots in order to look for procedures, good practice, or definitions of specifics systems.

After each interview with a pilot, we evaluate how we can improve the set of questions: what aspect we need to focus on, what topics are often dealt with, which topics are not. Then we use the newly improved set of questions in the following interview. We repeat the process on as many interviews as possible until we are sure that there is nothing major to improve or to add to our set of questions. Some indicators of this accomplishment can be:

- No more remark about a question that seems unlikely to be asked in an operational scenario;
- Very little improvement about possible queries to add to the set or topics/specific systems to address;
- More remarks focused on adding content rather than modifying the existing one, indicating that our baseline natural question set seems valid.

PROS	CONS
• Avoid wasting time on out of scope questions • Allows us to prepare for multiple scenarios, even the most unlikely ones • Allows us to create an important dataset • Guides the user, does not make him feel lost about what we want to get from him	• Can be biased, can influence the pilots on his or her questions production • Takes a lot of time, which can be used in vain if the pilots invalid most of the baseline questions during a single interview

Table 1: pros and cons of the validation of a set of question by pilots

4. User Driven Approach

4.1. Use of specific scenarios during interviews with pilots

4.1.1. Scenarios creation

In this methodology, we proceed by first creating a couple of specific use-case scenarios in which multiple questions are listed. This method should allow having the most comprehensive list of contexts and use-cases to work with. A use-case scenario is simply a description of a possible operational scenario, which should immerse the interviewee in a specific context, making the question production more accurate, focused, and easy. For every scenario, we also focus on the following elements:

- Do they have a specific need for information during this particular situation? If so why and what kind of information do they need?
- How do they picture themselves questioning an on-board search engine?
- What kind of questions would they ask?

For the creation of the operational scenarios, we plan to divide them according to the type of situation (nominal, abnormal, critical) and the flight phases (pre-flight, cruise and landing). The pre-flight scenarios will also include use-cases related to the "*climb*" phase, where the aircraft is increasing its altitude, and the landing scenario will include "*descent*" and "*approach*" phases, where the aircraft is about to land. Here are examples of scenarios classified according to these categories:

Nominal situation (nothing needs a particular attention):

- Pre-flight: *You're about to **take off** in 5mn.*
- Cruise: *You have to **change** your trajectory.*
- Landing: *You begin the **approach descent**.*

Abnormal situation (something is uncommon but not critical to the safety of the flight):

- Pre-flight*: **report on fuel** is missing.*
- Cruise: ***windshield alarm** goes off.*
- Landing: ***landing zone** polluted*. (Polluted means either "covered in snow, rain or toxic product" according to the FCOM.)

Emergency situation (an event that greatly compromise the flight safety):

- Pre-flight: ***fire** in avionic bay.*
- Cruise: ***engine 1 and 2** shut down.*
- Landing: ***landing gear** is failing on approach.*

Scenarios on Performance (optional for the pilot, but can be useful nonetheless):

- Pre-flight: *The weather is going to be **dry**.* (which would indicate different engine performances)
- Cruise: *The **fuel** is decreasing inconsistently.*
- Landing*: You are about to land on a **type A airport**.*

4.1.2. Re-designing of the scenarios

Once an interview is over, we proceed to focus on the questions we managed to get from it. We then sort the questions according to their relevance to our use-cases, and their structures in order to

see what topics are the most addressed and what kind of structures are the most commonly found. This information allows us to know which use-cases we designed must change, which ones must be modified and which ones are clear and "operational" enough to induce good queries production. Some of the following elements can indicate that our scenarios are relevant toward the natural question collection task:

- Clear instructions/settings: the pilot does not need to ask any or few specific details about the scenario and can instantly immerge himself in the operational situation
- A lot of clear and correct questions/queries are produced by the pilot
- The questions produced are clearly addressed to a virtual assistant

PROS	CONS
• Less biased than the first approach, because pilots are the only ones to produce queries • More realistic, due to the use of "operationally-valid" scenarios • Allows to explore different scenarios and situations while still controlling the environment (useful to constraint the question production on specific themes) • Well known methodology • Allows the pilot the visualize the situation, to avoid blank	• Can be unproductive if the questions we get are not information-centred or if the scenarios are not working • Can be slow to get a lot of questions from every session • The scenarios may be very limited in terms of quantity • Biased by the specificities of the scenarios

Table 2: pros and cons of using specific scenarios in user driven approach

4.2. Use of open type questions during interviews with pilots

4.2.1. Creation of a first set of open questions

This kind of approach is more flexible but less precise than the other. With this method, the emphasis is on the intents behind the questions a pilot would ask, rather than the core structures of the questions. This method is very common when gathering information about user habits when designing human-machine interfaces. According to Jenny Preece in "Human-Computer Interaction", 1994, open-questions should always answer the following problems:

- Why do you do this?

- How would you do it (or say it in this case)?
- Why don't you say <x> instead?
- Why don't you ask <x> before <y>?
- How would you correct an error <x>?

4.2.2. Adapting the questions

After each interview, we would analyse our open-type questions and see which ones are the most successful and what kind of answers we get by asking them. For this approach, it is not relevant to continue asking the same questions for each pilot. For instance, if we come to find that most of the pilots are expressing a need for an automation of the research of good practices within a specific kind of document, we would have to modify our questions in order to try and find other kind of usages

PROS	CONS
• Very flexible, can adapt to a lot of profiles/situations • Allows to explore very different use-case scenarios • Broaden the scope of what the virtual assistant would have to process	• Can be unproductive • Collected questions may only relate to one specific topic, lack of variation • Can lead to vague answers • Can disturb the pilot due to the openness of the questions • Does not really make users produce sample questions

Table 3: pros and cons of using open questions in an user driven approach

4.3. Problem Encountered

Having made the last adjustments to our methodologies we then proceeded to try and book interviews with multiples test or airlines pilots and pilot instructors. Unfortunately, we never got any answers from them, even after multiple contacts from both the linguists and human factor specialists who usually work closely with airline and test pilots. The only interview we were able to attend to was with two airlines pilots. But it happened at the very beginning of the internship, and was at the initiative of two engineers from the research & development department. In this interview the goal was simply to isolate preliminary use-cases to work with before going on with the design of the virtual

assistant system. As it was almost at half-way of the internship, we decided to move on and find others ways to create our own baseline dataset.

4.1. Solution found to create a functional baseline dataset

In order to move on in the design of the synthetic question generator program, we decided to make use of all of the resources we had in order to create the most accurate and "unbiased" dataset possible. In the next paragraph, we will present some of the use cases we used.

4.1.1. Use-case from an early interview we got with two airline pilots

These use-cases, as stated previously, were presented to pilots with the intent to isolate some preliminary use-cases for the R&D team to work with. Specific operational situations were designed and pilots were asked to give some example of the queries they would use for each one of them.
We can find an excerpt of these scenarios in the following table.

TITLE	"I WANT TO"	"IN ORDER TO"	PRIORITY	EXAMPLE QUESTIONS
See causes of symptoms	access the documents that describe the causes of a symptom	determine the limitations and take the appropriate actions	medium	"Hey we have-symptom- what do you think" "Tell me What's possible for -symptom- "
See problem related procedures	access the procedures to solve a particular problem	/	medium	"Give me procedure for -title of failure as per ECAM" "-symptom- for unsensed failure"
Describe a system	Describe my system in a natural and interactive way	/	Low	"Give me a user guide for -system-" "I want a full description of -system-" or " Give me a short view of -system-"
Describe the situation	describe my situation in a natural and interactive way	/	medium	"Explain context for -situation-"

Table 4: Excerpt from a preliminary document used to isolate relevant use-cases

With these data we managed to collect 20 queries formulated by an end-user, even if the quality of the queries was very inconsistent and a lot of them not being queries related to get information from operation documentation.

4.1.2. Set of questions validated by experts

We quickly needed to create more data in order to improve the synthetic question generator program. But the ones we got from the use-cases of the previous version of the virtual assistant project were not enough. This is why we created a set of questions based on the few structures we know end-users would use combined with ones we designed ourselves (which refers to the developer driven approach). On table 6, we can find an example of the template used to store these questions. It is also important to note that each query was associated with scenarios and the possible location of the answer.

SITUATIONS	Engine failure before V1
CONDITIONS/DETAILS	runway without taxi-way
ARTIFICIAL QUESTIONS (BASED ON FCOM)	What is the minimum runway width for a 180 degree turn?
POSSIBLE LOCATION OF ANSWER	PRO-NOR-SOP/Taxi

Table 5: Excerpt from the document listing potentially valid queries creates and their origins

Table 5 is an excerpt from the full list of the validated queries in annex 2. After having created a first draft, we booked short interviews with three persons within the Human Factor department that have an aeronautic background. Among these persons:

- One had with a private flying license.
- One was a former airline pilot.
- One had deep understanding of the airline ecosystems due to more than fifteen years within the company and working closely with engineers.

For every query we created, we asked them about their pertinence toward our goals. Most importantly, we also asked them if users would actually phrase the queries the same way as we did. Out of the 100 questions we created, only ten were judged pertinent and closed to the ones we would have in real-life scenarios. We then proceeded to add them in our baseline dataset.

4.1.3. Conclusion

By changing a few keywords and words order in the previously validated questions, we manage to create a few "safe" variation of our pre-existent question. These modifications were always quickly verified and validated by aeronautic experts from the Human Factor department.

In the end, we were able to increase the final baseline dataset to reach a total of 45 queries. The full natural dataset is available in annex 3. The amount of queries we collected is very low compared to what we expected to get from interviews with end-users (which was 200 to 500 questions). However, it allowed us to see that most of the structures used by pilots were very common and easy to create. We also got important information on the most common information they were searching for in operational documentation: procedures, references to systems, symptoms of problems or contexts for specific issues.

With the few inputs we got, we were able to regroup the needs expressed by pilots under the following 4 intents:

- "**Get full information Documentation**": for procedures, good practices and guides for instance (equivalent to information retrieval, the expected output is a list of document references).
- "**Get quick information**": for a quick definition or a reference to a system (the expected output is a link to a text snippet or figure inside a document).
- "**Get a structured information**": for a maximum, minimum or recommended value for instance (the expected output is a named entity, which is close to factoid question answering)
- "**Think about if**": far more advanced, this intent would imply that the virtual assistant would use external sources of information, dialog and reasoning mechanisms in order to solve a specific problem.

Chapter 4

-

The Synthetic Question Generator

1. Specifications of the question generation program

Before beginning to develop the generation program, we need first to be clear on what we want to implement in the program and how we plan to do it. The first approach we thought of can be summarized as follow: Firstly, the program would take a set of baseline queries as input, these queries would be the ones we collected using our previously explained methodologies. Secondly, the program would parse every query from the input baseline dataset and be able to divide the elements related to the core semantic meaning of the sentence (what we can call "keywords" for instance) from the ones carrying more structural information. What we want to achieve by isolating these elements is to be able to apply a different treatment for the syntagmatic and paradigmatic parts of the queries.

The generation program should, for each query, be able to associate the structural parts of the query to other equivalent ones according to list of predefined allowed structures. For instance, a query such as:

*"Give me the **procedure** for **fire** in cockpit"*

Would be developed into multiples other queries with an *equivalent* semantic meaning. An example of equivalent output would be for instance:

*"display the **fire procedure** in cockpit"*

Once we isolated the keywords within a query (in the last example it would be "fire", "cockpit" and "procedure") from the others, more structural and grammatical elements (in the same example it would be "give", "me"," the", "for" and "in"), we can then apply specific processings to these two differently categorized elements.

For the semantic variation, we would take the keywords and look for equivalent forms in the relevant resources. For instance, if we use the FCOM operational documentation, a "fire procedure" would be equivalent in term of meaning to a "fire and smoke procedure". To achieve a good matching between similar entities we would use a "fuzzy matching" system that "although not a 100 percent match, is above the threshold matching percentage[11]". In this use-case we would focus on specific operational

[11] https://www.techopedia.com/definition/24183/fuzzy-matching definition of Fuzzy Matching (last consulted on 08.31.2017)

documentation, such as emergency alerts (containing about 60 entries) or a list of pre-defined systems and symptoms that the R&D department extracted from maintenance documents.

For the lexical variation, which is included is the "paradigmatic" variations, we would employ a similar approach than the previous one, but with more freedom, given that these words are not restricted to the controlled language of aeronautics. Thus, we plan to make use of the WordNet functionalities incorporated in the Python Natural Language ToolKit (NLTK). From WordNet, we will use "Synsets". A Synset, is described as the following: "Synsets are interlinked by means of conceptual-semantic and lexical relations"[12]. Using this concept, for each token being identified as a keyword the program would find a matching Synset and get all corresponding elements that match the original term with its particular part of speech tag. For instance, if the term is "display" tagged as being a verb, possible outputs would be "show" or "exhibit".

The program is developed in Python 3. The main reason for it being that a lot of the existing components of the virtual assistant project are already developed in python, and this language is also a good way of creating quick prototypes, which is what we want to achieve here.

2. Resources used by the program

As regards to the resources, we want to use relevant operational documentation as the primary use-case of the project is to look for information in them. The documents we plan to use are as the following:

2.1. The Flight Operating Crew Manual (FCOM):

For more information, a page from the FCOM is available in annex 4. Given the large size of the document, we plan to first focus our attention on specific sections. To begin with, we will use a list of titles regarding specific emergency procedures (referred to by "level 3 procedures"), which are among the most critical information a pilot may need in a difficult situation, addressing engine failures, fire on-board and crew incapacitation among others.

2.2. Lexinet:

Lexinet is a dictionary built by Docland, the Airbus Documentation centre. It contains acronyms and definitions of numerous aeronautics terms. Even if a lot of the descriptions in the documents are not specifics to Airbus, it also provides a lot of hindsight concerning the acronyms and

[12] https://wordnet.princeton.edu/ definition of a Synset

systems exclusives to the company. This resource will be used mainly to get the full definition of acronyms included in our datasets, in order to normalize every abbreviation. For instance, "*I have APU fire*" would be automatically developed into *"I have Auxiliary Power Unit fire*".

2.3. Electronic Centralized Aircraft Monitoring (ECAM) Messages

In addition to the other documents, we will also use a file referencing every messages displayed in the cockpit sorted by their priorities. The ECAM is a monitor that "present information to the pilots. In the event of a malfunction, it will display the fault and may also display the appropriate steps of the remedial action."[13] The wording of the alert messages displayed on the ECAM will be useful for us, given that it is linked to the names of the procedures from the operational documentation.

Figure 2: Example of an ECAM display (picture from skybrary.aero)

[13] https://www.skybrary.aero/index.php/Electronic_Centralized_Aircraft_Monitor_(ECAM) explanations on the ECAM (last consulted on 08.31.2017)

3. Early design of the program[2]

Before developing a usable version of the synthetic question generator program, two versions were designed and tested. Here is a schematic of their architectures.

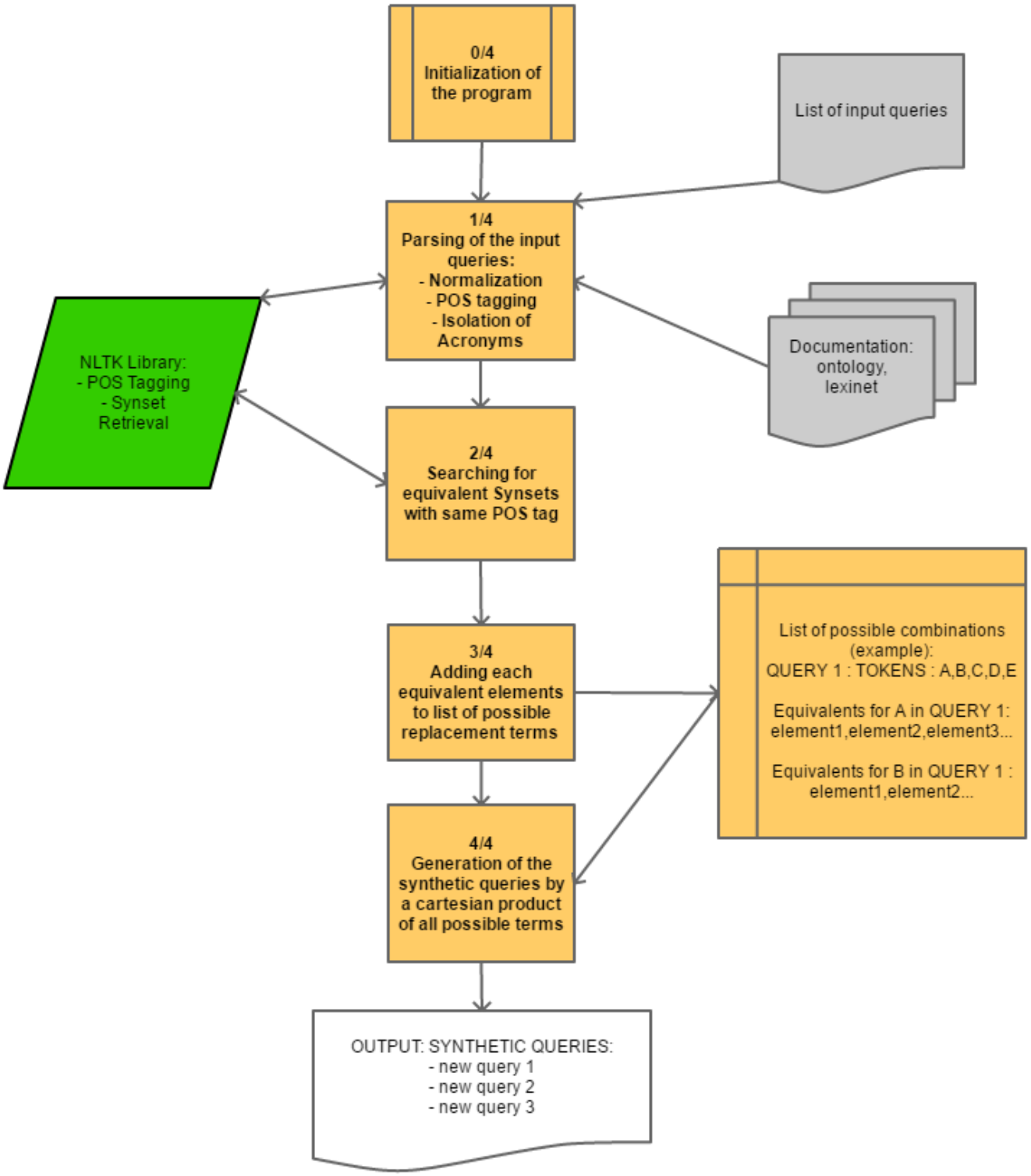


Figure 3: Schematic representation of the V1 and V2 versions of the generation program

The final generation was achieved by using Cartesian products. A Cartesian product is, in set theory, defined as " the set of all points (a,b) where $\mathbf{a} \in \mathbf{A}$ and $\mathbf{b} \in \mathbf{B}$ "[14]

ORIGINAL	A	B	C	D	E
EQUIVALENT TERMS				D1	E1
				D2	

Table 6: Example of a Cartesian product in the generation program

In table 6 we can see the representation of a sentence containing 5 words, were some have equivalent terms. In this example we would have 6 total possible output queries. For instance: "*A B C D E1*", "*A B C D1 E1*", "*A B C D2, E*" and so on. On the following table, we present a few input queries with their corresponding equivalents generated by the program.

INPUT	**OUTPUT**
show me the fire procedure	• *show me the fire procedure* • *show me the fire fuel man xfr procedure* • *show me the fire ecam abnormal procedure* • *show me the fire fuel manual transfer* • *show me the fire fuel manual transfer procedure* • *show me the fire overweight landing procedure* • *what is the procedure in case of hot engine*
What is the procedure in case of hot engine	• *what is the fuel manual transfer procedure in lpc front stator of hot pylon-to-engine forward attach fitting* • *what is the fuel manual transfer procedure in lpc front stator of hot pylon-to-engine fwd attach fitting* • *what is the fuel manual transfer procedure in lpc front stator of hot rib1* • *what is the fuel manual transfer procedure in lpc front stator of hot rib 1*

[14] http://mathworld.wolfram.com/CartesianProduct.html

	• *what is the fuel manual transfer procedure in lpc front stator of hot engine oil tank* • *what is the fuel manual transfer procedure in lpc front stator of hot oil storage* • *what is the fuel manual transfer procedure in lpc front stator of hot oil tank*

Table 7: Examples of output produced by the first versions of the generation program

4. Why we changed our approach

As we can see in figure 4, even if most of the queries produced are grammatically corrects, they are not valid as regards to our goals of having a "natural-like" output. As a matter of fact, even if the synsets allow the synthetic queries to convey a close semantic link between the input questions and the output ones, the results are often too broad which renders the dataset unusable in an operational context without manually selecting the valid synthetic queries, rendering the whole process of generation useless.

Moreover, as this system was functioning by performing Cartesian products, the final combinatorial of elements was way too important. For instance, with only 47 input queries, 1 billion and 800 million output questions (1810951799 in total) were created. The main reason for this kind of problem is that synsets are not restricted to our field of research: here, the aviation. Furthermore, we did not manage to parse differently the lexical and semantic tokens, rendering the search for equivalent synsets too broad and off-topic. In front of these results, we decided to change our approach in order to find a more controlled way to generate synthetic data.

5. The final version of the synthetic question generator program

In the final version of the generation program, we decided to use a more controlled way to generate data. Contrary to the previous versions of the program, we decided to base the generation on “rules”, thus constraining the generation instead of having to sort out the output. Our rules are similar to a context-free grammar where the generation is constraint by a set of production rules. A schematic of the new architecture of the program is available on the following figure.

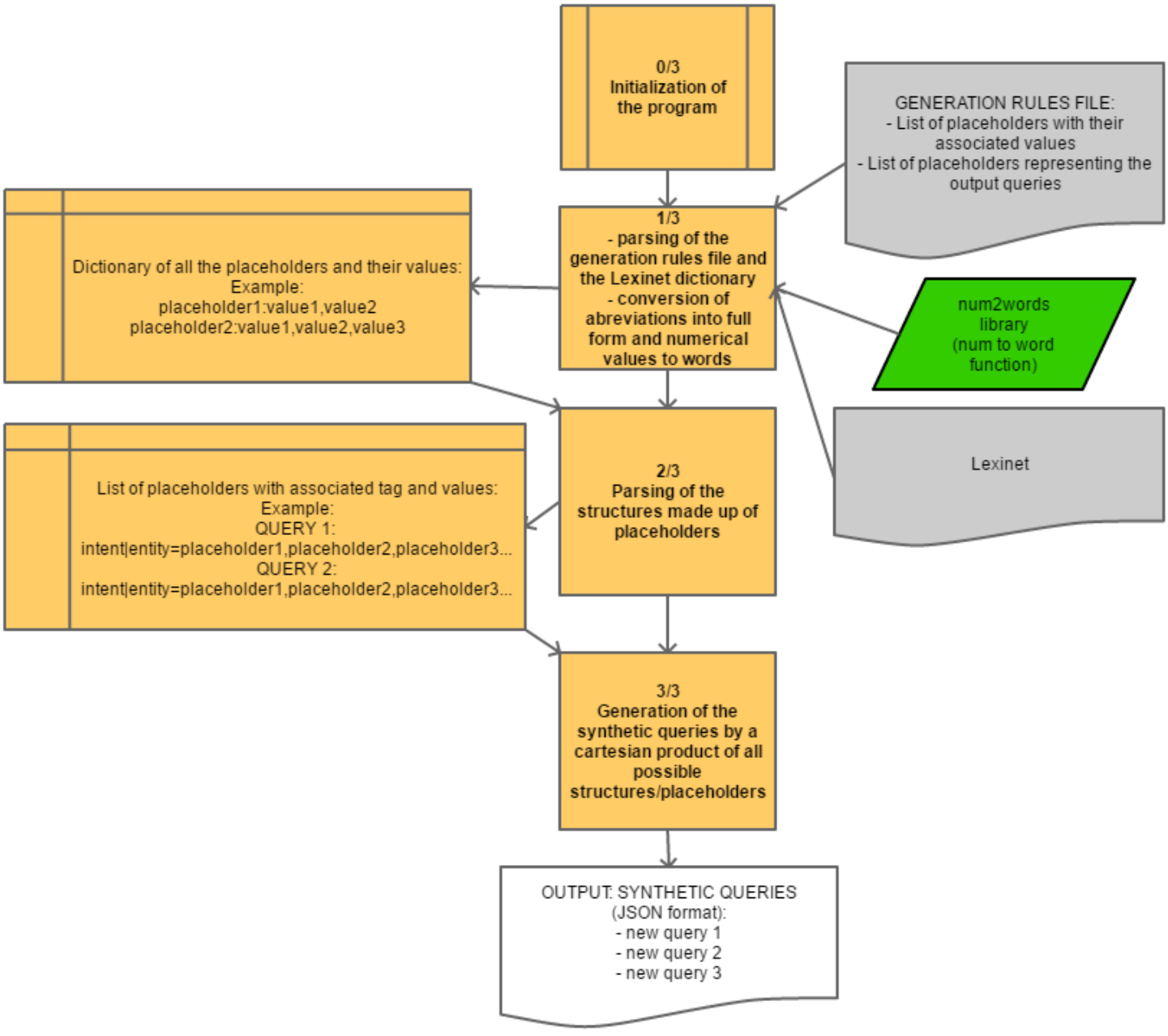


Figure 4: Schematic representation of the third and last version of the generation program

In this version of the program, the "generation rules" file is divided into multiple categories, some related to adding vocabulary and other to the generation itself. Here are some explanations on the elements found in the rules file:

Vocabulary placeholders:

Example: *"verb_show=show,get,display"*

The main purpose of these placeholders is to add vocabulary to the final generation. They store contextually interchangeable elements, which can refer to the paradigmatic variation part of the program.

"Keyword" placeholders:

Example: *"FOR_DITCHING|keyword=ditching,ditch"*

They are mostly similar to the previous placeholders, but the main difference is that they are identified as "keywords" which means that they carry the most important piece of information of the sentence. Separating the "keyword" elements from the "vocabulary" elements will be useful in order to indicate, in the final output, where is located the keyword element for the retrieval system to search relevant information in a database.

Meta Placeholders:

Example:" *{META}for_lvl3_proc|FOR_=FOR_EXCESS_CAB_ALT, FOR_AP_OFF…"*

Meta Placeholders are simply placeholders of placeholders. They are used to ease the process of creating multiple combinatorial structures for specific, closely related, elements.

Combinatorial structures:

Example:

"[COMBINATORY]=get_problem_procedure|symptom:question_simple,det,procedure_name,for,symptoms"

Structures are lists of placeholders that will be parsed in order to generate queries. For each structures there is an intent (in the example it is "get_problem_procedure") and an entity (in the example, it is "symptom"). Intents are used to identity what the user wants when s/he issues a query, and an "entity" refers to a key element that will help the system to search and find relevant information.

Once every rule has been created, the generation program is ready to be used. With this approach, the program takes between 2 and 3 minutes to generate a dataset of 526,000 queries. The next table

contains an excerpt of random queries produced by the program (the output is normally formatted in JSON but for clarity we decided to only show the queries).

SYNTHETIC QUERIES
• apparently engine one fail what is the procedure • show me brief description for main fuel pump system in a350 • i'm currently at flight level six hundred what is minimum speed with weight of seventy-four tons in a320 • i've got air bleed leak what is procedure page • display me note for fastener symbols in a350 • i'm currently at flight level three hundred and fifteen what is optimal speed with weight of seventy-four tons in a320 • i'm currently at flight level zero what is maximum speed with weight of sixty-four tons in a350 • i'm at flight level one hundred and forty-five what is optimum speed with weight of thirty-nine tons

Table 8: Examples of queries from the final generation program

6. The final output of the question generation program

Using the last version of the question generator program, many datasets have been produced. At the beginning our tests were only focused on 2 intents: "*get_problem_procedure*" and "*get_system_reference*", respectively related to finding an official manual for a specific situation and a reference or specification pertaining to a system in a specific aircraft. At the end of our tests, we decided to generate queries on 4 different intents: "*get_document*", "*excerpt*", "*structured*" and "*reasoning*." Those intents are the ones we described at the end of chapter 3. In the following table, we can find information about the last generation file used by the program and the corresponding created queries.

TOTAL NUMBER OF PLACEHOLDERS	200
TOTAL NUMBER OF PLACEHOLDERS VALUES	2,491
TOTAL NUMBER OF INTENTS	4

Table 9: statistics on the final generation rules file used by the generation program

INTENTS	AMOUNT OF QUERY GENERATED

get_document	69,718
excerpt	156,057
structured	240,144
reasoning	60,408
TOTAL	526,327

Table 10: list of all of the intents generated by the program

At the end of our experiments, the program was capable of generating a total of 52,6000 queries, with the “reasoning” intent being the less important in term of query per intent and the “structured” the more proficient.

These differences in terms of quantity can be explained by the number of placeholders each structures owns and the number of elements theses placeholders are referring to. For instance, some placeholders have just one element, such as the placeholders referring to the particle “to”. But some other are carrying a lot more elements, such as the placeholder “systems” which contains more than 1,200 elements extracted from operational documentations. In order to have a full view of the system’s functioning, the full script is available in annex 5.

Chapter 5

–

Using and testing the synthetic dataset

1. Why integrating machine learning into the project

In the research and development team, a decision has been made to use machine learning-based system from the beginning of the development of the virtual assistant. The main purpose of this decision is to create a flexible system where the developers will be allowed to add more data in order to improve and diversify the range of needs the virtual agent will be covering. In the selection process of the right software to use, numerous solutions were contemplated, for instance: Watson (IBM), API.ai, Wit.ai. At the end of this selection, an open-source software was chosen: RasaNLU.

The reason for choosing RasaNLU is that it is a popular library, available across multiples languages, easy-to-use, easy to implement and overall allow for quick prototyping and experiment, which is what the project needs at its earliest stages.

2. RasaNLU: The Natural Language Toolkit

Rasa NLU (for Natural Language Understanding) "is a tool for intent classification and entity extraction"[15]. It aims to extract the semantic information from free texts in order to parse it as structured content, allowing it to be used by a system in order to search information in a database for instance. The following figure shows the processing chain of the program applied to our use-case in a virtual assistant system. Here, "EYDN" indicate that the department focus on the NLU part of the virtual assistant.

[15] https://github.com/RasaHQ/rasa_nlu Information on RasaNLU (last consultation on 09/01/2017)

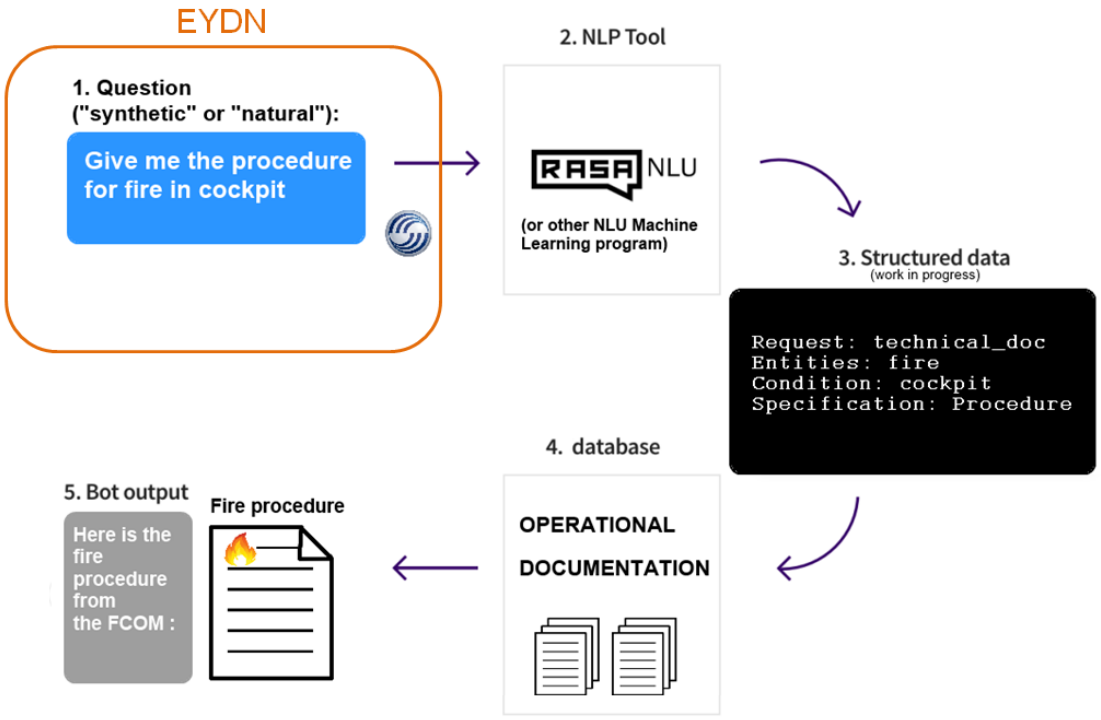


Figure 5: Processing chain of RasaNLU

2.1. How does the library work?

2.1.1. RasaNLU

RasaNLU is an open-source tool aimed toward the development of "conversational AI"[16], it offers full customization by providing various back-ends in order to target specific needs. RasaNLU is mainly written in python (compatible with Python2 and Python3) and use an extensive list of dependencies in order to function. Each dependency has a specific importance in the program. For instance, the library "jsonschema" is used for implementing the parsing of JSON formatted files, the dependency "future" is used to render the program compatible with both python2 and python3. Once installed, Rasa-NLU requires specific back-ends in order to be used. The library offers a choice between multiples programs such as "MITIE" which includes Machine Learning functionalities and Natural Language Processing capabilities, "Spacy" which is a natural language processing software and "scikit-learn" a machine learning library. All of these backends are compatible with Python. The R&D team from Airbus as well as the RasaNLU website[17] recommends the use of "MITIE and scikit-learn" for maximum efficiency. Here is a short description of the two backends:

[16] https://rasa.ai/ Explanation of the advantages of RasaNLU (last consultation on 09/01/2017)

[17] https://rasa-nlu.readthedocs.io/en/latest/installation.html Information on the backends from RasaNLU (last consultation on 09/01/2017)

2.1.2. MITIE

MITIE is described as a "state-of-the-art information extraction tools"[18]. The open-source software includes tools capable of performing "named entity extraction" and "binary relation detection". The library also includes multiples pre-trained models for various languages such as "English, Spanish of German" built on top of various linguistic resources such as "ACE", "Wkikipedia" or "Freebase".

2.1.3. Scikit-learn

Scitkit-learn (often refered as sklearn) is an open-source tool for data mining and data analysis. It encompasses multiple machine learning functionalities such as classification, regression, clustering or model selection capabilities[19].

2.2. Using RasaNLU

Once all of the back-ends and library are operational, we have to input a valid JSON formatted file for the system to train with. A JSON file is described as an "open-standard file format that uses human-readable text to transmit data objects consisting of attribute–value pairs and array data types"[20] The values used by Rasa are indicated in bold in the following figure.


```
{
   "text":"i have excess cabin altitude what is procedure",
   "intent":"get_document",
   "entities":[
      {
         "start":7,
         "end":28,
         "value":"excess cab alt",
         "entity":"system"
      }
   ]
},
```


Figure 6: Screenshot of an output (in JSON format using a beautifier) from the generation program

[18] https://github.com/mit-nlp/MITIE description of MITIE (last consultation on 09/01/2017)

[19] http://scikit-learn.org/stable/index.html information on sklearn (last consultation on 09.01.2017)

[20] https://en.wikipedia.org/wiki/JSON explanations on the JSON format (last consultation on 09.01.2017)

2.2.1. Explanations on the input taken by RasaNLU

In figure 6, we can see multiple categories and values. Here is a description of their different meanings:

- **"text":** store the full sentence we use as an input.
- **"intent":** indicate the type of action referring to the input sentence, here its "get_document" in order to find operation documentation.
- **"entities"** hold several values:
    - **"start"** and **"end"**: mark the beginning and the end of the entity storing the information that we want to extract from the text.
    - **"value":** indicate the tokens that carry the most important information in the sentence, it's the tokens that will allow the information retrieval system to search for relevant documents in the database.
    - **"entity":** categorized the "value" element, in our generation program, "entity" is the name of the placeholders related to the "value" element, which is also the name of its category.

2.3. The training

The training is performed by using a configuration file allowing the user to customize the back-ends used and the data chosen to be used in the training process. For instance, some parameters allow the user to set the language to train the model. For now, RasaNLU support by default English, German and Spanish. The configuration file also allows to tweak more technical parameters such as the maximum number of ngrams to use when augmenting feature vectors for instance. As we explained before, the back-ends we will use for our tests are MITIE and Sklearn. During the training, Sklearn is used to perform intent classification and MITIE for entity recognition. On a dataset comprising 1,500 queries, the training took 2 hours using an intel 5-4200h CPU. The result of the training is a model folder containing files ready to be used with the prediction system. Within the model folder are 5 files, here is a quick description of the functions of these files:

- **Entity_extractor.dat**: contains information about the entities present in the input file.
- **Entity_synonyms.json:** contains the synonyms (if any) to some of the input entities.
- **Intent_classifier.pkl**: contains information on the intent classification.
- **Metadata.json:** containing various information on the training process (language used, backends used and so on).
- **Training_data.json:** contains information on intent and entity associations

2.4. Using the model

Once the model is ready to be used by the system, we launch RasaNLU using all of the necessary metadatas contained in the model directory in order to interpret a new input sentence. More than one model can be used by RasaNLU, but it needs to be specified in the prediction program.

Once an input has been processed by RasaNLU, it returns the entities object which has been detected in the input text. For instance, given our use-case, if we trained the system to recognize the display of a procedure page, it should output the intent "procedure" and the entities the keyword that match the type of procedure we are looking for. Every prediction is associated with a confidence score, allowing the system to discriminate between multiple probable results.

Every time RasaNLU is called to make a prediction on a new input sentence, it stores the predictions it made into log files. This allow the user to keep track of the predictions yielded so far which is useful to see what part of the system is to improve.

INPUT	"apu on fire procedure"
OUTPUT	{'**entities**': [{'**end**': 11, '**entity**': 'symptom', '**extractor**': 'ner_crf', '**start**': 0, '**value**': 'apu on fire'}], '**intent**': {'**confidence**': 0.91305273459423542, '**name**': 'get_problem_procedure'}, '**intent_ranking**': [{'**confidence**': 0.91305273459423542, '**name**': 'get_problem_procedure'}, {'**confidence**': 0.086947265405764598, '**name**': 'find_system_references'}], '**text**': 'apu on fire procedure'}

Table 11: Example of a prediction using RasaNLU

In table 11 we can see an example of a prediction made by the system. We can observe that for each intent detection the system output a corresponding confidence score based on the model used.

2.5. Conclusion:

Even if at first glance RasaNLU seems like a complex tool to use, its online tutorial as well as its popularity made it a quite simple, even if a bit heavy, toolkit. For the moment, the Research and

Development team of Airbus has trained the system with the MITIE and Sklearn backends, which appears to be the most effective ones to use with the given use-cases, although other tests have to be done to ensure that the system would not benefit from using other parameters.

3. A change of program to experiment with the datasets:

3.1. Why we had to stopped using RasaNLU

When using RasaNLU, a problem rapidly occurred when the need to train datasets superior to 100,000 lines emerged: the training was too slow for our computers and we needed to find a faster solution in order to test our different approach on text generation. For instance, a training on a 121,000 queries file took more than 3 days to compute, and we wanted to test multiple approaches on a large number of datasets. As a reason, we decided to develop a series of simple python scripts that would enable us to quickly verify our initial hypothesis: that a synthetic dataset can help generate data in order to improve a system. The baseline program created was written by the Airbus supervisor of this internship. The baseline program is divided into 2 files:

- **training_models.py:** takes as input a JSON formatted file and output a corresponding language model file. The model contains lexical weights that indicate, for all intent *i* in set *I*, how likely a word *w*[21] can be found in a query *q* belonging to intent *i* :

$$weight\ (w,i) = \frac{freq\ (w,i)}{\sum_{i' \in I} freq\ (w,i')}$$

Equation 1:Weight computing in baseline model creation

$$freq\ (w,i) = \frac{\sum_{q\ \in\ i} occ\ (w,q)}{\sum_{q\ \in\ i} |q|}$$

Equation 2:Frequency computing in baseline model creation

occ(w,q) being the number of times word *w* occurs in query *q* and |*q*| being the total number of words in *q*.

- **test_models_intents.py:** takes as input the previously created model file as well as a reference file and output predictions. The predicted intent *i* for a query *q* is the intent with the maximum score:

$$intent(q) = \max_{i \in I} \left(\sum_{w \in q} weight\ (w,i) \right)$$

Equation 3: Prediction of an intent in the baseline program

[21] Tokenization is done by splitting on spaces. Words are set to lower case. All stop words are excluded.

It should be noted that these scripts are only used to test intent classification and not entity recognition.

3.2. The Model creation

All of the scripts we developed were using the JSON formatted files outputted by the synthetic question generator program. For building language models, we used a basic script that take in input a sentence and its intent, for instance:

- **Query:** *"i've got wing a.ice system fault in a350"*
- **Related Intent:** *"get_document"*

The other information contained in the JSON files were not used; such as the entities and their indexes.

After getting this information, the program create a language model based on the lexical weight of each token present in the input dataset according to its intent. The more a token appears in a query of a specific intent, the more weight it has in relation with this intent. Equation 1 and 2 presented part 3.1. explains this distribution. The next table shows an excerpt of the output for a language model based on a 100 queries file.

WORD	**DOC_ OCC**	**STRUCT_OCC**	**EXCERPT_ OCC**	**REAS_OCC**	**DOC_ FREQ**	**STRUCT_ FREQ**	**…**
limited	0.0	0.0	0.0	1.0	0.0	0.0	…
valve	0.0	0.0	0.0	1.0	0.0	0.0	…
show	2.0	0.0	2.0	1.0	0.0134 22818 7919	0.0	…

Table 12: Example of three entries in a language model generated by the baseline system

3.3. Predictions using the Models

For the prediction part, we simply load our previously created models and use them in addition with a reference dataset where each reference question is paired with its corresponding intent. Once the reference file and the model files are loaded, we parse the reference file lines and tokenize them as we did for the model creation part. After this step, we look for every token in the tokenized queries to see if it's also present in the model file. If the token is present, we save its corresponding model score along with the corresponding intent, and then we repeat the process for every token in the string.

Once we have reached the end of the string, we look for the intent which maximize the query we parsed. An equation showing this maximization can be found in part 3.1 of this chapter. Following this simple mechanic, we can then compare the predicted intent classification and the reference one, allowing us to do extensive tests on the synthetic dataset.

3.4. Conclusion on the baseline model creation and predictions

Even if these scripts are simplistic, they allowed us to quickly perform different sets of tests without having to rely on a heavier library. The simplicity of these programs architecture needs to be taken into account in order to considerate the predictions yielded by the system. For instance, as the intent models are only based on frequencies of isolated words (and not on their combinations or n-grams) we can contrast the results by saying that the context of the elements in the sentences, as well as the syntax in general, is not taken into account.

Chapter 6

–

Evaluation

1. The evaluation metrics and tools

1.1. Precision:

For our experiments, the precision is the number of correctly recognized intents divided by the total numbers of predictions made by the system. Precision scores range between 0 (worst) and 1 (best).

$$\text{Precision} = \frac{tp}{tp + fp}$$

Equation 4: Precision

In the equation, *"tp"* means true positives (here, a relevant predictions) and *"tp"* false positives (an irrelevant predictions).

1.2. Recall:

The recall score is the number of correctly recognized intents divided by the total number of relevant intent. In our use-cases the recall score is calculated by taking the total number of intents "A" predicted and divide this number by the total number of "A" present in the reference dataset. Recall scores ranges between 0 (worst) and 1 (best).

$$\text{Recall} = \frac{tp}{tp + fn}$$

Equation 5: Recall

In the equation, "*tp*" means true positives, "*fn*" false negatives and "*fp*" false positives.

1.3. F1 Score

The F1 score can be considered as weighted average of the precision and recall. It is used to measure the overall performance of the system on a given intent. Its scores range from 0 (worst) to 1 (best).

$$F_1 = 2 \cdot \frac{1}{\frac{1}{\text{recall}} + \frac{1}{\text{precision}}} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Equation 6: F1 score

1.4. (Standardized) Type Token Ratio:

The Type Token Ratio is used to evaluate the lexical complexity of a corpus. In this evaluation, *"token"* refers to individual words and *"type"* to the amount of unique words. The Type Token Ratio is the division of those two elements. In our evaluation procedure, we will also use the Type bigrams and Trigrams ratio, which works exactly the same way as the TTR by selecting unique bigrams and trigrams and dividing them by their total number in their respective datasets. As it is not pertinent to compare TTR of different sizes of datasets, we will use the STTR (for Standardized Type Token Ratio) which is the weighted average of *n* TTR calculated on *m* excerpts.

$$TTR = \frac{UniqueTokens}{TotalTokens}$$

Equation 7: Type-Token Ratio

1.5. Confusion Rate:

The confusion rate is used to measure the degree of incorrect predictions the system made. A confusion rate of 1 means that the system is totally inaccurate and a confusion rate of 0 means that all of the systems predictions were right. This metric will be used to compare the effectiveness of models between themselves, then we will be able to investigate further in order to see which factor(s) is likely to cause models to be effective or not.

$$CF = \frac{Fp + Fn}{Fp + Fn + Tp + Tn}$$

Equation 8: Confusion Rate equation

In the equation, "*Tp"* means true positive, *"Tn"* true negative, *"Fp"* false positive and *"Fn"* false negative.

1.6. Confusion Matrix

A confusion matrix is a visualization tool that allows having a comprehensive view of a system's performance compared to a reference one. This kind of matrix is mostly used in machine learning on classification problems, allowing for a quick information intake on the prediction results.

		PREDICTED CONDITIONS	
		PREDICTED CONDITION 1	PREDICTED CONDITION 2
REFERENCE CONDITIONS	CONDITION 1	VALID	FALSE
	CONDITION 2	FALSE	VALID

Table 13: Example of a confusion matrix

2. The reference dataset

In order to experiment on the datasets classification, we need to first have a reference dataset to which we are going to compare the output of our system. For the need of the experimentations, the company supervisor designed a reference dataset including 25 queries for each of the 4 intents, adding up to 100 queries.

These queries were designed while keeping in mind the intents and possible use-cases associated with them, for more accuracy, when possible they were based on actual procedures and information found in operational documentation. A full view of these reference queries are available in annex 6.

3. Evaluation results

We are now going to study the results produced by our different tests on all of the datasets generated by our program. For these tests, we decided to focus on the 4 intents we wanted to study:

- **"get_document"**: to get any full operational documentation.
- **"structured":** to get a structured form of information, such as values from array, by giving all of the necessary information in the query.
- **"reasoning":** to express a query where there is not enough information for the virtual agent to directly propose an answer.
- **"excerpt":** to express a need for a short information, such as a definition or a quick description for instance.

3.1. Results on the maximum dataset

The biggest dataset generated comprises 526,327 queries, including, as seen in table 8, 69,718 queries belonging to the "*get_document*" intent, 240,144 queries from the "*structured*" intent, 156,057 from the intent "*excerpt*" and 60,408 from the "*reasoning*" intent. Overall, the dataset comprises 1,263 unique tokens. Its confusion rate is 0.52.

		SYNTHETIC MODEL				
		GET_DOCUMENT	**STRUCTURED**	**EXCERPT**	**REASONING**	**TOTAL**
REFERENCE DATA	**GET_DOCUMENT**	17	0	7	1	25
	STRUCTURED	10	8	2	5	25
	EXCERPT	2	1	17	5	25
	REASONING	3	3	13	6	25
	TOTAL	32	12	39	17	100

Table 14: Confusion Matrix of the synthetic model and reference data results on intent classification

The confusion matrix in table 14 compares all of the predicted intents with the reference intents from the reference dataset. As we can see, the results vary considerably depending on the tested intents. We are first going to describe the most confused intents within this table. For instance, the intent "*get_document*" is confounded 7 times with the intent "*excerpt*", which is not surprising considering that both are directly linked to the same words from the vocabulary of operational document, sharing an extensive list of systems names comprising of 3,797 tokens.

Then, if we look at the intent "*structured*", we can notice that it was mainly confounded with "*get_document*" with 8 true positive and 10 false positive with the "*get_document*" intent. Contrary to the previous comparison between the confounding of "*get_document*" and "excerpt" we can here note that, in the generation rules, a lot of structural placeholders were shared between these two intents. For instance, structures beginning with "*what is*" are very common: 4,092 "*get_document*" and 30984 "*structured*" queries are beginning with it.

For the "*excerpt*" intent recognition, we can see that the system is mostly confounding it with the "*reasoning*" intent but not by far, with 5 false positive into reasoning, 2 into "*get_document*" and 1 in "*structured*". A lot of placeholders used to create the "*excerpt*" queries are exclusive to this kind of intent, such as the one asking for a definition "*what do* x *signify*?" or the one asking for a specific system explanation with "*what is the purpose of* x *in* x?". It's also the only intent using color names, adding a few unique token to its collection, making its discrimination from the other intents easier.

The intent that was the least well recognized is the "*reasoning*" intent. If we look at its confusion matrix we can observe that it was vastly confounded with the "*excerpt*" intent with 13 false positive representing more than 2 times the number of true positive. These confusions can be easily explained

by the fact that, in our generation rules, the placeholders used to create "*reasoning*" queries weren't exclusive to this kind of intent, sharing the same words (both structural and semantic) with most of the other intents.

	PRECISION	RECALL	F1
GET_DOCUMENT	0.53	0.68	0.6
STRUCTURED	0.67	0.32	0.43
EXCERPT	0.44	0.68	0.53
REASONING	0.35	0.24	0.28

Table 15: Precision, Recall and F1 of the intents in the maximum dataset

In table 15, we can see the precision, recall and F1 score on the maximum dataset. As we saw in the last paragraph, the intent getting the best recognition is "get_document" with a F1 score of 0.6. Then comes the "excerpt" intent with a F1 of 0.53, the "structured" intent with 0.43 and lastly the "reasoning" intent with 0.28.

Overall the results yielded by these experiments are mediocre from an operational standpoint. For instance, if we take the global confusion rate of 0.52 we can deduce that our system gives a wrong classification half the time, while still doing better than a random classification whose confusion rate would be 0.75 (1 chance out of 4 to give the good answer on 100 queries to classify).

To contextualize these results, we can look at an article written by Intento company in 2017[22] presenting an intent detection benchmark while comparing some of the most popular NLU libraries. The dataset used in the test had "15,600 queries" (about 2,000 queries per intent), "7 intents", and "340 000 total symbols". On the detection performance all of the system tested yielded a F1 score ranging between 0.987 and 0. 965.Here is the list of the intents tested in this benchmark:

- SearchCreativeWork
- GetWeather
- BookRestaurant
- PlayMusic

[22] https://fr.slideshare.net/KonstantinSavenkov/nlu-intent-detection-benchmark-by-intento-august-2017 intent classification benchmarks (last consultation on the 09.03.2017)

- AddToPlaylist
- RateBook
- SearchScreeningEvent

As we can see, all of these intents are fairly easy to distinguish by an human, which is not the case with the intents we used in our experiments, but we will return to this point later in this document.

3.2. Results on balanced dataset excerpts

While analysing the results yielded by our baseline scripts is a good way to isolate the most difficult intents to classify, we should not forget that our classification program is very simple and only based on statistical occurrences of terms most commonly associated with intents. Our main goal is to try and show if a synthetic dataset can have beneficial prospects on the natural language understanding part of a system, which is here expressed by intent classification. In order to verify our hypothesis, we randomly generated 10 datasets per sizes ranging from 1,000 queries to 50,000 queries, so that for every given size of a dataset created we would have 10 instances of it. Each of these datasets were balanced by having the exact same amount of intents, with 1 quarter of "*get_document*" intent, 1 quarter of "*structured*" intent and so on. For every dataset, we then generated their corresponding language model with our baseline scripts, these models were tested with our queries classification program and we finally merged the results for each datasets of equivalent size in order to have balanced results for each specific size of our sets of data.

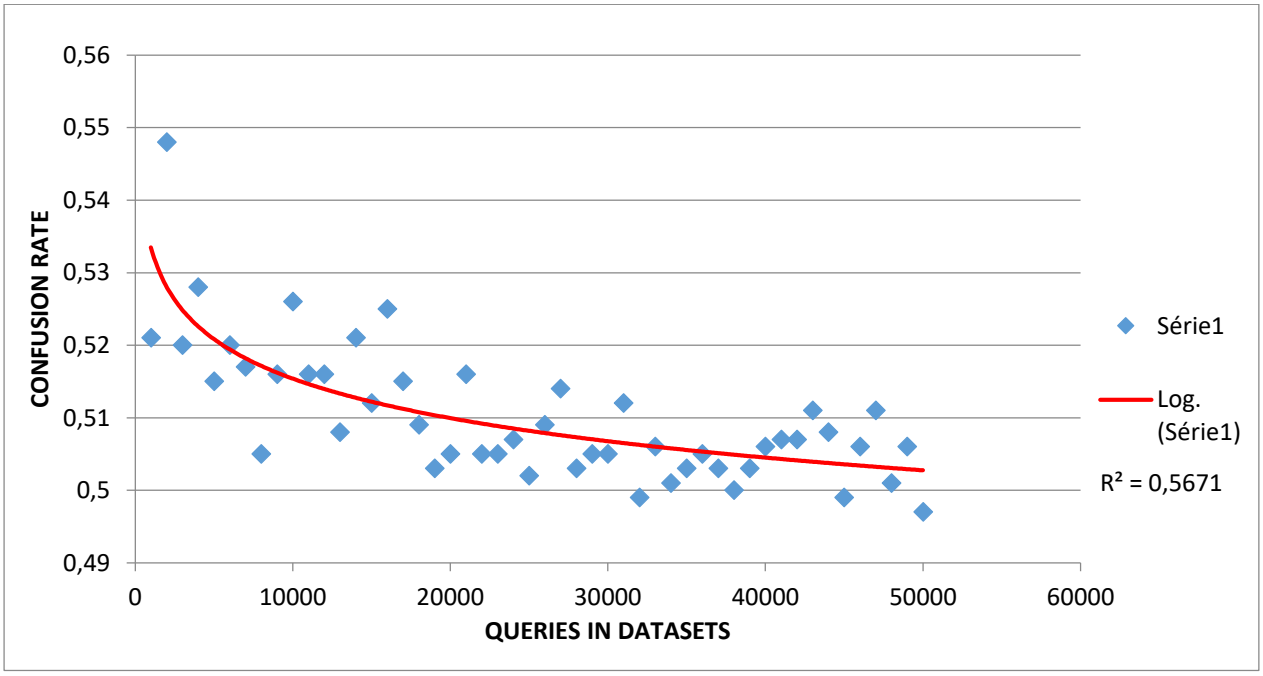


Figure 7: confusion rate compared to datasets sizes

In figure 7, we can see a comparison between confusion rates and datasets sizes. As we can see the size of the datasets tends to lower the confusion rate, with the highest confusion rate being 0.528 for the dataset of 2,000 queries and the lowest confusion rate being at 0.497 for the dataset of 50000 queries. To better represent the tendency for the confusion rate to drop while using bigger datasets we indicated (in red) the logarithmic trendline of the graph. We chose the logarithmic visualisation (rather than linear or other) because it maximized the coefficient of determination (R^2) at 0.56, meaning that datasets sizes can alone explained at 56% the confusion rates.

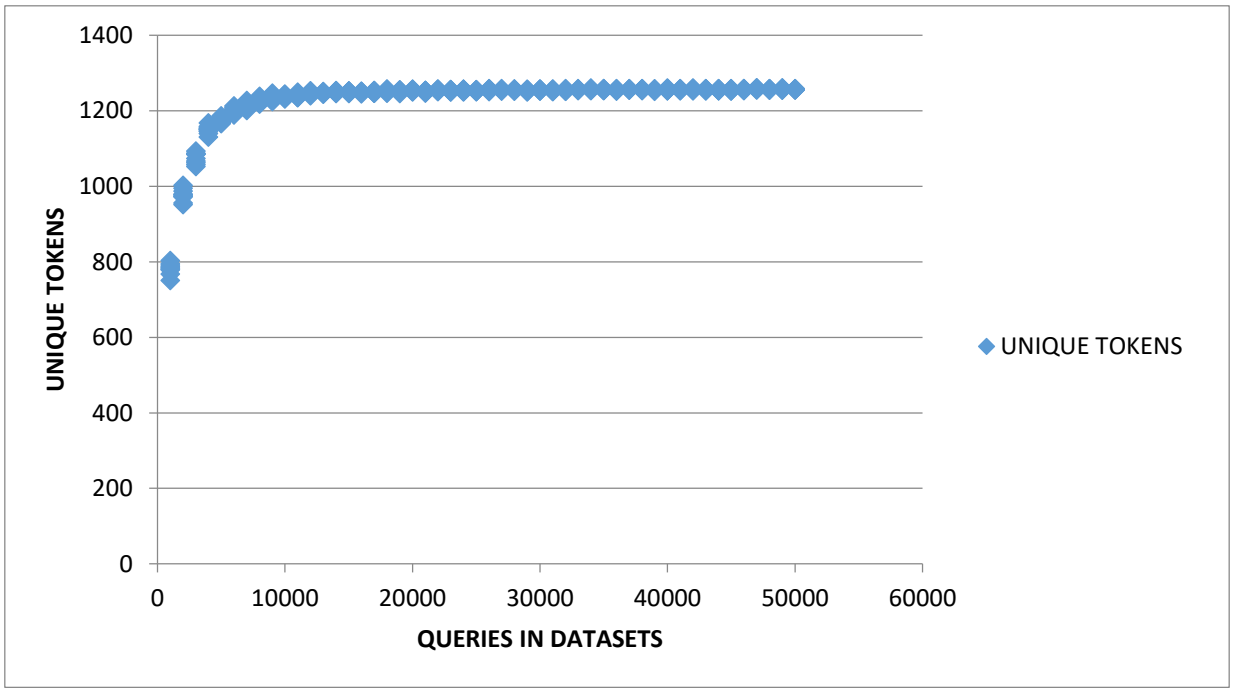


Figure 8: Unique tokens compared to number of queries in datasets

Even if we noted some decrease in the confusion rates along with the increase in datasets sizes in figure 7, we can also see that it seems to quickly cap around 20,000 queries per datasets and higher. To try and explain the existence of this cap we can analyse the figure 8 which compares unique tokens found in datasets as well as datasets sizes. In this graph, quickly after 10,000 queries the vocabulary from the generation rules file seems to be quickly depleted, ranging from 789 to 1,239 unique tokens between the 1,000 and 10,000 queries datasets and only from 1,239 to 1,257 between the 10,000 and 50,000 queries datasets.

These results can give some insights on the evaluation performance, indicating that the vocabulary will not change a lot from one dataset to another past 10,000 queries, rending the task of intents discriminations more difficult for the program due to less token to process and less unique token per intent. After 10,000 queries, the only new information provided by the dataset is the possible word combinations. As the classification script does not take this information into account, the confusion rate stops dropping.

SIZE	TTR (standardized)	T2G (standardized)	T3G (standardized)
526327	0.5758047	0.81351557	0.9097814
40000	0.6610188	0.88971334	0.9565293
30000	0.6600549	0.88920553	0.9565407
20000	0.6606412	0.8911527	0.957551
10000	0.6605792	0.88960527	0.9560356
5000	0.6569484	0.88583013	0.9543212
1000	0.6575	0.89033189	0.9588192

Table 16: Type Token, Bigrams and Trigrams ratio of different sized datasets

If we look at the type token/bigram/trigram ratios showed in table 16 for the different kind of datasets we can notice that from the 1000 queries dataset to the 40,000 ones, the values displayed stays generally the same. The biggest gap between datasets can be found between the ones previously mentioned and the biggest dataset of 526,000 queries. However, even in this case, the gap compared to the other dataset is minimal in view of the important difference in size between those datasets. These observations seem to point to a lack of diversity in the vocabulary contained in the generation rules file.

3.3. Comparison between the human and machine classification

In order to see why our system's classification was not very effective, we did a few tests to see how well humans would perform in such a classification task. For these tests we instructed 3 subjects (including the person that designed the reference dataset) to classify the reference queries without any help. The only information that was given was the names of the intents to fill in the cases and a full oral and written description of these intents accompanied by examples queries for each one.

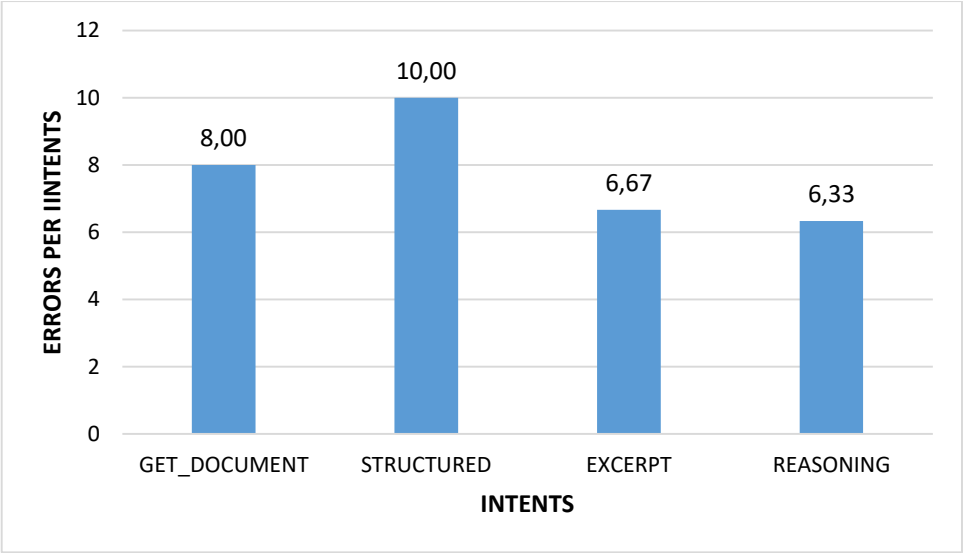


Figure 9: Averaged errors per intents based on 3 subjects

On figure 9, we can see the averaged errors per intents made by human subjects during the classification task. As we can see, some intent seems to be easier to classify than others. For instance, the intents "*excerpt*" and "*reasoning*" seems to be less difficult to classify than the others with an average errors rate of 6 compared to the 8 and 10 of the "*get_document*" and *"structured*" intents. Even the most structurally controlled queries such as the "*structured*" ones which are the only ones indicating values seems to be the hardest to classify in the reference dataset. This could also point out the difference in terms of phrasing between the reference dataset and the synthetic dataset. The full results per subject for the human classification task is available in annex 7.

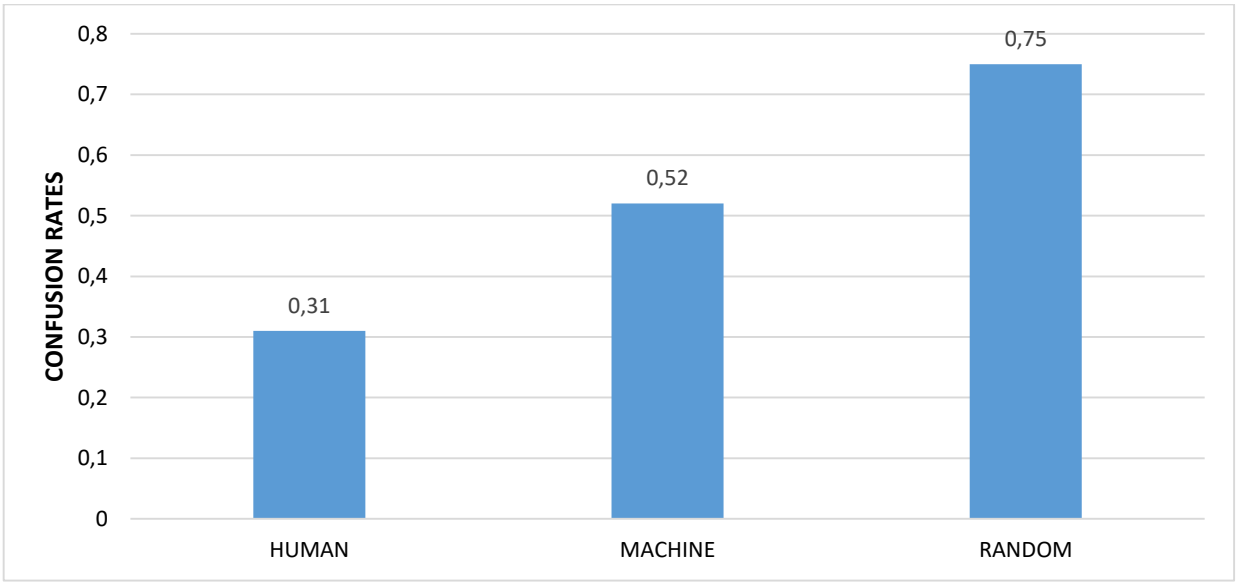


Figure 10: Comparison of the humans, machine and random confusion rates

On Figure 10 we present a comparison of the confusion rate between the 3 human subjects we tested on the classification, the confusion rate of our classification system and the confusion rate a “random” classification would get (with 1 chance out of 4 to get the correct answer). As we can see, even if we only had 3 test subjects, the “human” and “machine” confusion rates are not too far from each other with a 0.31 score for the human classification and 0.52 for the machine. This give us information on an important fact: the four intents seem to be inherently hard to distinguished from themselves in the reference dataset.

4. Conclusion on the evaluations

Before concluding on the results of the evaluations part, we have to keep in mind that the main goal of our experiments was to test the efficiency of a synthetic dataset in the classification of intents. The results we obtained may not be good enough for an operational system to use our models, but we found some trends and behavior patterns that would be considered useful for a future enhancing of the system.

On our tests comparing datasets sizes and confusion rates, we were able to see that confusion rates tend to decrease the bigger the datasets get, even if it was not consistent. These results must be nuanced by a lot of constraining factors: first, our baseline model-based system of intent classification is very simple in its architecture, it only takes into account the tokens contained in the queries, and not their combinations. Secondly, the predictions are returning only one intent, with no entity selection or pre-treatment to refine the predictions. Secondly, we saw that when generating queries the vocabulary from the generation rules file was quickly exhausted which reduce even more the systems performance and capability in terms of classification.

More importantly, we saw that by tasking 3 persons to classify the dataset the results we obtained very poor and not far from the ones yielded by our system, indicating a more important problem: the classification in itself is an ambiguous task where it seems not possible to get consistent results.

Conclusion

1. Reflection on the internship

The core research problem of the whole internship was to experiment on the generation of a synthetic dataset in order to see if it can be efficient in an intent classification task. As we illustrated in the experimentation results, we can say that this method seems promising but we cannot deliver a final statement on the matter because of the lack of resources and the simplicity of the script used to experiment with the data.

However, even if the initial hypothesis needs deeper researches, we can list the elements that contributed to Airbus, and the virtual assistant project overall:

- **The creation of a Synthetic Queries Generator.** The last version of this program is stable and its architecture does not require any programming knowledge in order to get it to run or to add more vocabulary and resources into it. Giving its versatile nature, it can be used to generate anything apart from queries. Moreover, its output in JSON format can also be useful to use the synthetic dataset with most the existing machine learning libraries.
- **Methodologies** were created to collect natural data from end-users. Even if they were not tested on real subjects, they are fully completed and may be useful in the next stages of the natural language understanding enhancing part of the virtual assistant project.
- **Specific intents were created**. At the beginning of the internship, only two intents were designed for the virtual assistant: "*get_document*" and "*find_system_reference*" which we merged into one "*get_document*" intent. After having discussed with experts we designed the "*excerpt*" "*reasoning*" and *"structured*" intent, because we felt that it was would be highly request by the targeted audience. Even if we demonstrated that it's very difficult, even for humans, to classify these information, it highlights specifics needs that should be addressed.

2. Perspectives

In order to enhance the intent classification and the system in general, a few perspectives are opened to investigation for the future of the virtual assistant project, here are some of these ideas:

- As we said before, it is extremely important to have a clear idea of what the end-users have in mind when using a virtual assistant technology. This is why the methodologies created during this internship should be put to the test and modified if necessary in order to collect the maximum amount of information on the kind of queries the virtual assistant should answer to, and what are the intents behind these queries.

- Following this collection of (as large as possible) datasets of natural data, we must then design multiple intents that could, for instance, be divided into categories and sub-categories in order to lessen the probability of confusion. For instance, we could have the following classification shown in table 16:

DOCUMENTS	• Procedures • Good practices • References
EXCERPT	• Definition • Diagrams

Table 17: Example of a new way of classifying intents

A system designed this way would be more reliable and prone to evolve as the classification process would have to go through multiple categories in order to succeed.

- After having collected the new sets of natural data and intents, more intent classification test should be realized in order to see if these intents are sufficiently different from one another.
- The previously mentioned "multi-category" system would have to be implemented with a machine learning software such as RasaNLU in order to remedy to the problems due to our baseline testing script being too simple in its architecture. Tests on classification requiring extensive use of token positions, contexts (ngrams for instance) or stemming and lemmatization to associate words to specific lexical fields would be interesting optimizations to the program.
- Beside the correct intent classification, a more efficient system should be able to correctly recognize "entities" in order to isolate the keyword elements of a query and to send structured data to an information retrieval system.
- Another possible approach on the generation program would consist in experimenting further more on the first versions of the program using Synsets and directly paraphrasing natural queries received in input. For instance, once the script has yielded all the possible synthetic queries, we could use languages models (created with operational documents for instance) to filter the irrelevant queries. Such filtering could be, of course, tweaked with different delimiters in order to reduce or reinforce the required "aeronautic-like" quality of the final dataset.

Bibliography & Webography

Nakov, Hearst, 2008. Solving Relational Similarity Problems Using the Web as a Corpus, School of Information, University of California, Berkeley, USA.

Preece J, Rogers Y, Sharp H, Benyon D,Holland S,Carey T, 1994. Human-Computer Interaction: concepts and design. Addison Wesley, Massachussetts, USA.

Schank, Goldman, Rieger III, Riesbeck, 1975. Inference and Paraphrase by Computer. Yales University, New Haven, USA.

Yuval Marton, Chris Callison-Burch, Philip Resnik, 2009. Improved Statistical Machine Translation using Monolingually-Derived Paraphrases. Baltimore, USA.

https://fr.wikipedia.org/wiki/Airbus Various information on Airbus, last consultation: 08.31.2017.

http://searchmicroservices.techtarget.com/definition/human-factors-ergonomics Definition of the term “human factor”, last consultation: 08.31.2017.

https://wordnet.princeton.edu/ Definition of the term “Synset”, last consultation: 08.31.2017.

https://www.kantanmt.com/whatisbleuscore.php : Explanation of BLEU, last consultation: 08.31.2017.

http://www.dictionary.com/browse/paradigmatic Definition of “paradigmatic”, last consultation: 08.31.2017.

http://www.dictionary.com/browse/syntagmatic Definition of “syntagmatic”, last consultation: 08.31.2017.

https://www.techopedia.com/definition/24183/fuzzy-matching Definition of Fuzzy Matching”, last consultation: 08.31.2017.

https://www.skybrary.aero/index.php/Electronic_Centralized_Aircraft_Monitor_(ECAM) Explanations on the ECAM display, last consultation: 08.31.2017.

http://mathworld.wolfram.com/CartesianProduct.html Explanation on Cartesian Products, last consultation: 01.09.2017.

https://github.com/RasaHQ/rasa_nlu Information on RasaNLU, last consultation on 09/01/2017

http://scikit-learn.org/stable/tutorial/basic/tutorial.html Information on Scikit-Learn, last consultation on 09/01/2017.

https://github.com/mit-nlp/MITIE Information on the MITIE library, last consultation on 09.01.2017.

https://spacy.io/ Information on the Spacy library, last consultation on 09.01.2017.

https://rasa-nlu.readthedocs.io/en/latest/installation.html Information on the rasaNLU backends, last consultation on 09.01.2017.

https://rasa.ai/ Explanation of the advantages of RasaNLU, last consultation on 09/01/2017.

http://scikit-learn.org/stable/index.html information on sklearn, last consultation on 09.01.2017.

https://en.wikipedia.org/wiki/JSON information about JSON format, last consultation on 09.01.2017.

https://fr.wikipedia.org/wiki/Matrice_de_confusion information on confusion matrices, last consultation on 09.03.2017.

https://en.wikipedia.org/wiki/False_positives_and_false_negatives information on false positives and negatives, last consultation on 09.03.2017.

https://fr.slideshare.net/KonstantinSavenkov/nlu-intent-detection-benchmark-by-intento-august-2017 intent classification benchmarks by the Intento company, last consultation on the 09.03.2017

Glossary and Acronyms

A/C:	Aircraft
APU:	Auxiliary Power Unit
ATC:	Air Traffic Control
BLEU:	Bilingual Evaluation Understudy
Docland:	The Airbus information center.
EASA:	European Aviation Safety Agency
ECAM:	Electronic Centralised Aircraft Monitor**:** system that displays aircraft functions and messages to the pilot: details of failures encountered, procedure to follow in specific situation…
End-user:	Final user, the one who will use the software.
EYDN:	Entity within Airbus comprising the Engineering System, General System and Human Factor sectors
FAA:	Federal Aviation Administration
FCOM:	Flight Crew Operating Manual**:** List all of the procedures and information relevant to a specific aircraft in an operational situation.
JSON:	JavaScript Object Notation
NLP:	Natural Language Processing
NLTK:	Natural Language Processing ToolKit
NLU:	Natural Language Understanding
OOV:	Out of Vocabulary
Paradigmatic:	"pertaining to a relationship among linguistic elements that can substitute for each other in a given context."
R&D:	Research and Development
Runway:	Path where planes lands or take off.
SMT:	Statistical Machine Translation
SSTR:	Standardized Type Token Ratio
Synset:	In the Wordnet project, a Synset is a group of interchangeable words carrying the same semantic meaning.
Syntagmatic:	"pertaining to a relationship among linguistic elements that occur sequentially in the chain of speech or writing."
Taxi-way:	Path where planes are transported in order to take off or be parked.
TTR:	Type Token Ratio

VA: Virtual Agent/Virtual Assistant

WordNet: Lexical Database of English terms.

Figures - table of contents

Figure 1: Schematic view of the three approaches 24
Figure 2: Example of an ECAM display (picture from skybrary.aero) 36
Figure 3: Schematic representation of the V1 and V2 versions of the generation program 37
Figure 4: Schematic representation of the third and last version of the generation program 40
Figure 5: Processing chain of RasaNLU 46
Figure 6: Screenshot of an output (in JSON format using a beautifier) from the generation program 47
Figure 7: confusion rate compared to datasets sizes 59
Figure 8: Unique tokens compared to number of queries in datasets 60
Figure 9: Averaged errors per intents based on 3 subjects 62
Figure 10: Comparison of the humans, machine and random confusion rates 62
Figure 11: subject A classification 90
Figure 12: subject B classification 90
Figure 13: subject C classification 91

Equations - table of contents

Equation 1:Weight computing in baseline model creation 50
Equation 2:Frequency computing in baseline model creation 50
Equation 3: Prediction of an intent in the baseline program 50
Equation 4: Precision 54
Equation 5: Recall 54
Equation 6: F1 score 54
Equation 7: Type-Token Ratio 55
Equation 8: Confusion Rate equation 55

Tables - table of contents

Table 1: pros and cons of the validation of a set of question by pilots 26
Table 2: pros and cons of using specific scenarios in user driven approach 28
Table 3: pros and cons of using open questions in an user driven approach 29
Table 4: Excerpt from a preliminary document used to isolate relevant use-cases 30
Table 5: Excerpt from the document listing potentially valid queries creates and their origins 31
Table 6: Example of a Cartesian product in the generation program 38
Table 7: Examples of output produced by the first versions of the generation program 39
Table 8: Examples of queries from the final generation program 42
Table 9: statistics on the final generation rules file used by the generation program 42
Table 10: list of all of the intents generated by the program 43
Table 11: Example of a prediction using RasaNLU 49
Table 12: Example of three entries in a language model generated by the baseline system 51
Table 13: Example of a confusion matrix 56
Table 14: Confusion Matrix of the synthetic model and reference data results on intent classification 57
Table 15: Precision, Recall and F1 of the intents in the maximum dataset 58
Table 16: Type Token, Bigrams and Trigrams ratio of different sized datasets 61
Table 17: Example of a new way of classifying intents 66

Annexes – table of contents

Annex 1: Internship planning 75

Annex 2: Queries validated by experts 76

Annex 3: Final natural dataset 77

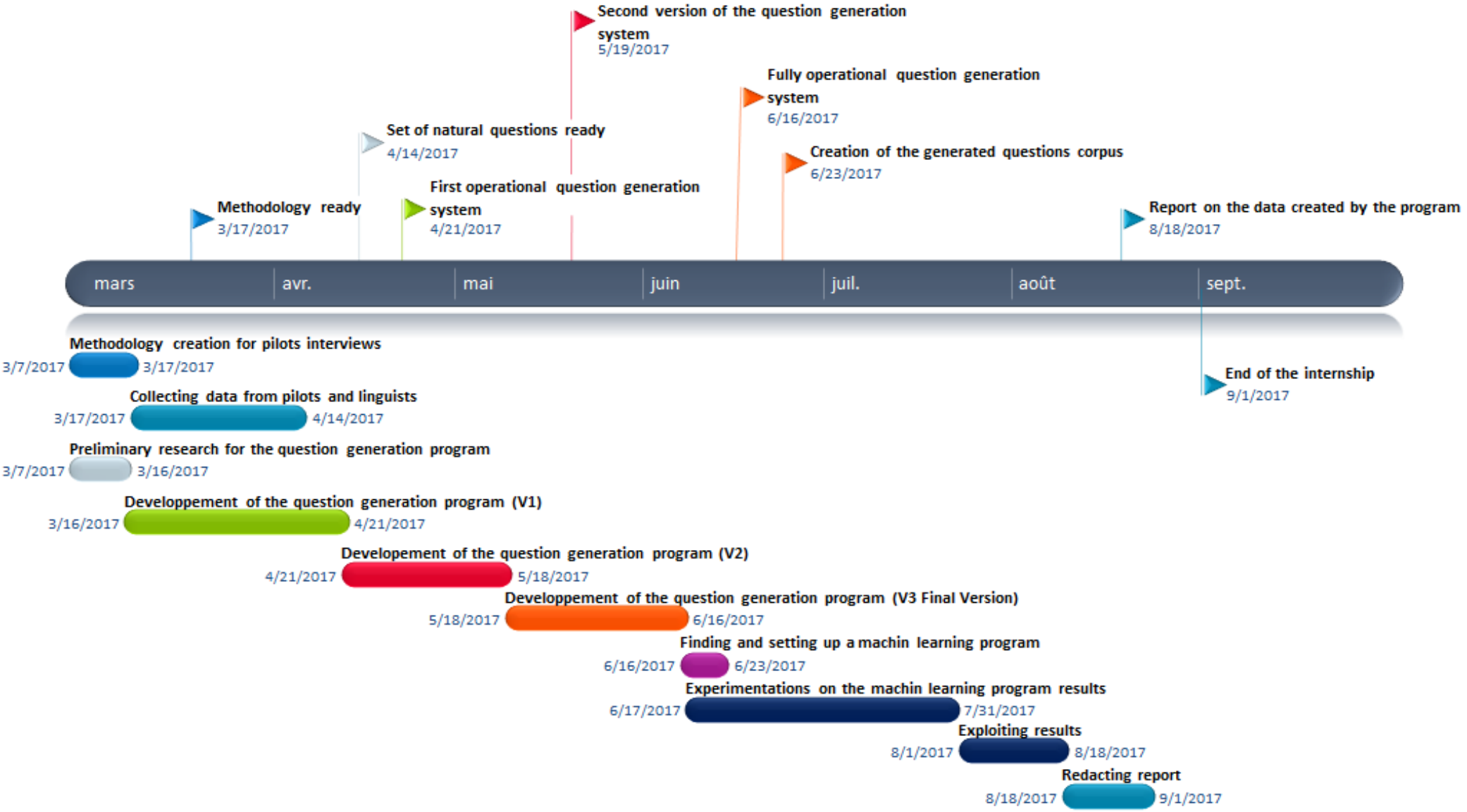
Annex 4: Excerpt from a Flight Operating Crew Manual (FCOM) page 79

Annex 5: Full Python script of the Generation Program 80

Annex 6: full reference dataset 86

Annex 7: human intent classification 90

Annex 1: Internship planning



This planning has been made at the very beginning of the internship, in order to keep track of the advance of the traineeship.

Annex 2: Queries validated by experts

These are all of the queries that has been considered as valid and likely to be ask to an on-board virtual agent by experts in aeronautics.

ARTIFICIAL QUESTIONS (BASED ON FCOM)	POSSIBLE LOCATION OF ANSWER (IN OPERATIONAL DOCUMENTATION)
What is the minimum runway width for a 180 degree turn?	PRO-NOR-SOP/Taxi
What is the maximum demonstrated crosswind for takeoff/landing?	PRO-NOR-SPO-60 Op on Narrow Runways
What is the Fuel On Board maximum discrepancy?	SOP/Before pushback or start
What are the conditions to activate anti-ice systems?	SP/ Ice ?
What is the maximum taxi / takeoff / landing weight?	LIM/Weight and center of gravity limits
What are the center of gravity limits at ACT 1/2/3/4/5/6 ?	LIM/Weight and center of gravity limits
What is the maximum flight level?	?
What is the procedure in case of disabled radar ?	?
What to do when avionic bay is cold soaked?	SOP/Adverse Weather
What are the speed limits with ice accretion?	SOP/Adverse Weather

Annex 3: Final natural dataset

Here are all of the queries composing the final natural dataset. They were either created and validated by experts or they originated from sample questions produced by airline pilots.

- Display me the page
- Show me the fire procedure
- Give me more information about the engine
- what is the procedure in case of hot engine
- What is the minimum runway width for a 180 degree turn?
- What is the maximum demonstrated crosswind for takeoff/landing?
- What is the Fuel On Board maximum discrepancy?
- What are the conditions to activate anti-ice systems?
- What is the maximum taxi / takeoff / landing weight?
- What are the center of gravity limits at ACT 6 ?
- What is the maximum flight level?
- What is the procedure in case of disabled radar ?
- What to do when avionic bay is cold soaked?
- What are the speed limits with ice accretion?
- Hey we have vibrations
- what do you think of vibrations
- Tell me What's possible for vibrations
- Give me procedure for title of engine failure as per ECAM
- vibrations for unsensed engine failure
- Give me a user guide for navigation system
- I want a full description of navigation system
- Give me a short view of navigation system
- Explain context for vibrations
- We have vibrations Explain strategy
- Explain strategy for vibrations
- Define flight limitations for engine failure and context
- Explain performance impact
- we have vibrations what should we do
- How to use navigation system
- Define best technique for engine failure for take off
- Define best technique for engine failure for descent
- Define best technique for engine failure for landing
- we have engine failure define final configuration

- I need Unreliable Airspeed figures for level flight for flaps 1 for 120 tons
- I need Unreliable Airspeed figures for level descent for flaps 1 for 120 tons
- I need Unreliable Airspeed figures for level approach for flaps 1 for 120 tons
- Describe subpart of the navigation system
- Specify AC
- Define AC for abbreviation explanation
- Explain AC for in depth description
- Find limitations for engine problem
- Find Indicators for engine problem
- Find failures for engine problem
- Explain fire procedure
- what the hell is fire procedure

Annex 4: Excerpt from a Flight Operating Crew Manual (FCOM) page

This is an example of a FCOM page describing a specific system that can be found in an aircraft. Sometimes the explanation is found along with a schematic representation of the described system.

LATERAL CONSOLES

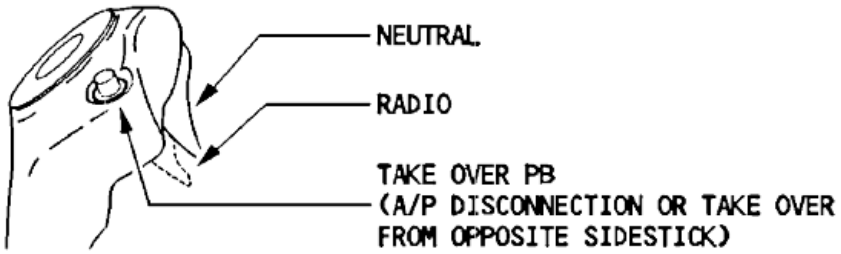
Applicable to: ALL

SIDESTICKS

Each pilot has on his lateral console a sidestick he can use to control pitch and roll manually. Each sidestick is springloaded to neutral.
When the autopilot is engaged, a solenoid-operated detent locks both sidesticks in the neutral position. If the pilot applies a force above a given threshold (5 daN in pitch, 3.5 daN in roll) the stick becomes free and the autopilot disengages.

The hand grip has two switches:

- Autopilot disconnect and sidestick takeover pushbutton.
- Push-to-talk button.



Sidestick priority logic

- When only one pilot operates the sidestick, it sends his control signals to the computers.
- When the pilots move both side stick simultaneously in the same or opposite direction and neither takes priority, the system adds the signals of both pilots algebraically. The total is limited to the signal that would result from the maximum deflection of a single sidestick.

Note: *In the event of simultaneous input on both sidesticks (2 ° deflection off the neutral position in any direction) the two green SIDE STICK PRIORITY lights on the glareshield come on and "DUAL INPUT" voice message is activated.*

Annex 5: Full Python script of the Generation Program

Here is the full script developed in Python 3.6 of the Generation Program. The libraries used are num2words (for number to letters conversion) and NLTK (for tokenization and ngrams).

The inputs of the program are a file containing the generation rules and vocabulary as well as another file containing a list of abbreviations paired with their full names.

The output of the program is a JSON formatted file containing the synthetic queries.

```
import time
start_time=time.clock()
import re
import nltk
import copy
from datetime import datetime
from itertools import product
from collections import Counter
from num2words import num2words
from nltk import ngrams, word_tokenize
from collections import OrderedDict, defaultdict
############################
# VARIABLES DECLARATIONS  #
############################
# COUNTERS
cpt=0 # number of unique questions generated
cpt_dupli=0 # keep track of repetitions avoided in the output file
# DICTIONNARIES
tok_tag_dict={} # contain terms associated with equivalent terms
dict_entity={} # contains the associations of values and entities
gen_rules=defaultdict(list) # store the content of the generation rules file
possible_combinations=defaultdict(list) # contains the possible combinations of categories

meta_repertory=defaultdict(list) # contains meta categories names and their categories
meta_suf={} # contains meta suffixes
lexinet={} # contains fullnames of lexinet abreviations
# COUNTER
cnt_intents=Counter() # number of intents in the output file
# LISTS
list_kwd=[] # contains terms indicated as being "keywords"
aircrafts=[] # list of aircraft denominations
# SET
query_set=set() # contain generated queries, safeguard against duplicates
# BOOLEAN
processing=False # indicate when the question generation has begun
def norma_res(a_string):
    """
    Format numbers to words and abreviations to fullnames
    input = any string
    output = formatted string
    """
    a_string=a_string.lower()
    new_str=""
    split_str=a_string.split(" ")
    for v in split_str:
        if v in lexinet: # we look into Lexinet to see if the term already exists
            if new_str:
                new_str+=" "+lexinet[v]
            else:
                new_str=lexinet[v]
        else:
            if new_str:
                new_str+=" "+v
            else:
                new_str=v
    split_str=new_str.split(" ")
```

```
    final_str=""
    for v in split_str:
        if re.match("\d+",v):
            new_number=num2words(int(v)) # if the token is a number, we translate it
            if final_str:
                final_str+=" "+new_number
            else:
                final_str=new_number
        else:
            if final_str:
                final_str+=" "+v
            else:
                final_str=v
    final_str=final_str.replace(",", "")
    return final_str # return of the final string

################################################################################
# STEP 1/3 : PARSING OF LEXINET
################################################################################
print("### STEP 1/3: Parsing of Lexinet ###\n\tprocessing...")
abrev_lex=open('res/abrev_lex.csv','r',encoding='utf-8')
for line in abrev_lex:
    if (re.search('^#',line)): # we skip the comment lines
        pass
    else:
        split_line=line.split(";")
        lexinet[split_line[0].lower()]=split_line[1].lower() # storing of abbreviations and fullnames
print("Done.")
################################################################################
# STEP 2/3 : PARSING THE GENERATION RESOURCES FILE
################################################################################
print("### STEP 2/3: Parsing of the generation file ###\n\tprocessing...")
rules_file_path='res/gen_rules_4_intents.txt'
rules_file=open(rules_file_path,'r',encoding='utf-8') # opening of the generation rules file
for cat in rules_file:
    if (re.search('^#',cat)): # we skip the comment lines
        pass
    elif(re.search('^\{META\}',cat)): # parsing of the meta categories (categories of categories)
        stripped_cat=cat.strip("{META}")
        get_meta_cat=stripped_cat.split("=")
        categories=get_meta_cat[1] # categories containg in the meta category
        meta_name_suf=get_meta_cat[0] # name of the meta category
        if(re.search("|",meta_name_suf)):
            get_suf_meta=meta_name_suf.split("|")
            meta_name=get_suf_meta[0]
            suf_name=get_suf_meta[1] # isolation of suffixes, in order to delete them later
        else:
            meta_name=get_meta_cat[0]
        list_cat=categories.split(",")
        for catego in list_cat:
            meta_repertory[meta_name].append(catego.strip()) # append to a meta dict
            if suf_name:
                meta_suf[catego.strip()]=suf_name

    elif(re.search('^\{AC\}',cat)): # parsing of the aicraft acronyms
            stripped_cat=cat.strip("{AC}")
            get_name_elem=stripped_cat.split("=")
            name=get_name_elem[0]
            ac_names=get_name_elem[1]
            ac_names=ac_names.strip()
            ac_names=ac_names.lower()
            if (re.search(',',ac_names)):
                aircrafts_names=ac_names.split(',')
                for ac_type in aircrafts_names:
                    aircrafts.append(ac_type) # append to list of aircraft names
```

```
                    gen_rules[name].append(ac_type) # aicraft names are also considered as lexical terms
            else:
                aircrafts.append(ac_names)
                gen_rules[name].append(ac_names)

    elif(re.search('^\[COMBINATORY\]',cat)): # parsing of the combinatorial rules
        get_intent_combi=cat.split("=")
        intent_combi=get_intent_combi[1]
        get_intent=intent_combi.split(":")
        intent_entity=get_intent[0].strip()
        if re.search("\|",intent_entity):
            get_entity=intent_entity.split("|")
            intent=get_entity[0] # name of the intent associated with the query
            entity=get_entity[1] # same, but for the entity name
        else:
            intent=intent_entity
        combinatory=get_intent[1]
        every_combi=combinatory.split(";")
        for pos in every_combi:
            pos=pos.strip()
            if(re.search(",",pos)): # if there are multiple categories into one structure, we iterate over them
                get_each_cat=pos.split(",")
                for catego in get_each_cat:
                    if catego in meta_repertory.keys(): # we check if a category is a meta category
                        for val in meta_repertory[catego]:
                            new_pos=pos.replace(catego,val)
                            possible_combinations[intent].append(new_pos)
                            if entity:
                                dict_entity[new_pos]=entity
                    elif pos not in possible_combinations[intent]: # else we add the combination as it was
                        possible_combinations[intent].append(pos)
                        if entity:
                            dict_entity[pos]=entity
            else: # else, we do the same for unique elements
                if pos in meta_repertory.keys():
                    for val in meta_repertory[pos]:
                        new_pos=pos.replace(pos,val)
                        if entity:
                            dict_entity[pos]=entity
                            for v in meta_repertory[pos]:
                                dict_entity[v]=entity
                        possible_combinations[intent].append(new_pos)
                elif pos not in possible_combinations[intent]:
                    possible_combinations[intent].append(pos)
                    if entity:
                        dict_entity[pos]=entity

    elif(re.search('=',cat)):
        is_a_keyword=False
        cat_struct=cat.split("=")
        if (re.search('\|keyword',cat_struct[0])): # we check if the parsed element is a keyword
            is_a_keyword=True # we set the boolean to "true"
            split_bar=cat_struct[0].split("|")
            cat=split_bar[0]
        else:
            cat=cat_struct[0]
        struct=cat_struct[1] # we store the corresponding structures for every category
        if (re.search(',',struct)): # if the element has a "," then there is multiple equivalent for the same category
            list_struct=struct.split(",") # we isolate every element in the list of equivalent terms
            for elem in list_struct: # we append them to the list of multiple possibility for a category
                elem=elem.strip() # normalization
```

```
                elem=norma_res(elem)
                # print(elem)
                if elem:
                    gen_rules[cat].append(elem)
                if is_a_keyword is True:
                    list_kwd.append(elem)
        else: # if a category has only one element, we simply store it in the correspondin
g dictionnary
            struct=struct.strip()
            if is_a_keyword is True:
                list_kwd.append(struct)
            if struct:
                gen_rules[cat].append(struct)
rules_file.close() # we close the generation rules file
print("Done.")
################################################################################
# STEP 3/3 : GENERATION OF ALL THE POSSIBLE OUTPUT
################################################################################
print("\n### STEP 3/3: Generating output ###\n\tprocessing...")
print("\tFile used:",rules_file_path)
time_mark=datetime.now().strftime('%d-%m-%Y_%H-%M-
%S') # time mark to identity output files
output_json=open('output/synth_corpus_'+time_mark+'.json','w',encoding='utf-
8') # header of the synthetic jason dataset
output_json.write('{"rasa_nlu_data":\n\t{"common_examples":\n\t\t[')
aug_gen_rules=copy.deepcopy(gen_rules) # used to iterate over the first one and save new e
quivalent in the new, copied one
for intent_name in possible_combinations: # we read the keys of the combination dictionnar
y
    for struct in possible_combinations[intent_name]:
        # print("PARSING STRUCT:",struct)
        # print("\tTotal queries so far:",cpt)
        if struct in dict_entity:
            entity_name=dict_entity[struct]
        if (re.search(',',struct)): # we test if there is multiple structures for a combin
ation
            list_struct=struct.split(",")
            cmd_product="product(" # initialization of the cartesian product command
            for elem in list_struct: # iteration over every structures
                for v in gen_rules[elem]:
                    tok_tag_dict[v]=elem

                cmd_product+="aug_gen_rules['"+elem+"'],"" # increment the cartesian produc
t command
            cmd_product+=")" # end of the command
            eval_cmd=eval(cmd_product) # eval of the command as a python3 statement
            for elem in eval_cmd: # iteration over every generated combinations in the lis
t
                synthetic_query="" # we initiate the variable that we will write as a fina
l output
                lemma=""
                ac_specified=False
                for term in elem: # for possible terms (may they be multiple or single)
                    if term in aircrafts:
                        ac_specified=True
                        name_of_ac=copy.copy(term)
                    if term in list_kwd:
                        term_to_find=copy.copy(term)
                        if tok_tag_dict[term] in meta_suf:
                            suf=meta_suf[tok_tag_dict[term]]
                            lemma = tok_tag_dict[term]
                            lemma = re.sub(meta_suf[tok_tag_dict[term]],"",lemma)
                            lemma = re.sub("_"," ",lemma)
                            lemma=lemma.lower()
                        else:
                            lemma=copy.copy(term.lower())
                        if synthetic_query:
                            synthetic_query+=" "+term
                        else:
```

```
                            synthetic_query+=term
                    else:
                        if synthetic_query:
                            synthetic_query+=" "+term
                        else:
                            synthetic_query+=term
                if synthetic_query not in query_set:
                    query_set.add(synthetic_query)
                    synthetic_query=synthetic_query.lower()
                    start_idx=synthetic_query.index(term_to_find.lower())
                    end_idx=start_idx+len(term_to_find)
                    cnt_intents[intent_name]+=1 # adding to the counter of intents
                    if processing is True:
                        output_json.write(',')
                    processing=True
                    output_json.write('\n\t\t\t{"text": "'+synthetic_query+'", "intent": "
'+intent_name+'", "entities": [{"start": '
                        +str(start_idx)+', "end": '+str(end_idx)+', "value": "'+lemma+'",
"entity": "'+entity_name+'"}')
                    if ac_specified is True: # specify the a/c type if it was indicated
                        start_idx_ac=synthetic_query.index(name_of_ac)
                        end_idx_ac=start_idx_ac+len(name_of_ac)
                        output_json.write(', {"start": '+str(start_idx_ac)+', "end": '+str
(end_idx_ac)+', "value": "'+name_of_ac+'", "entity": "aircraft"}]}')
                    else:
                        output_json.write(']}')
                    cpt+=1 # increment the counter of unique queries
                else:
                    cpt_dupli+=1
        # same treatement as above, but for structure with only one element, ex : "element
|tag=one_struct" in the scenario
        # we assume that the end user will only ask about keyword/meaningful words (so no
aircraft specifications)
        else:
            for elem in aug_gen_rules[struct]:
                synthetic_query=""
                if elem in list_kwd:
                    term_to_find=copy.copy(elem)
                    if elem in meta_suf:
                        suf=meta_suf[elem]
                        lemma = elem
                        lemma = re.sub(meta_suf[elem],"",lemma)
                        lemma = re.sub("_"," ",lemma)
                        lemma=lemma.lower()
                    else:
                        lemma=copy.copy(elem.lower())
                    if synthetic_query:
                        synthetic_query+=" "+elem
                    else:
                        synthetic_query=elem
                if synthetic_query not in query_set:
                    query_set.add(synthetic_query)
                    synthetic_query=synthetic_query.lower()
                    start_idx=synthetic_query.index(term_to_find.lower())
                    end_idx=start_idx+len(term_to_find)
                    if processing is True:
                        output_json.write(',')
                    processing=True
                    output_json.write('\n\t\t\t{"text": "'+synthetic_query+'", "intent": "
'+intent_name+'", "entities": [{"start": '
                        +str(start_idx)+', "end": '+str(end_idx)+', "value": "'+lemma+'",
"entity": "'+entity_name+'"}]}')
                    cpt+=1
                else:
                    cpt_dupli+=1

output_json.write("\n\t\t]\n\t}\n}") # we write the final information(s) to the output fil
e
output_json.close() # we close the output file
```

```
303. print("Done.")
304. print("\n-> Number of unique queries generated:",cpt,"\n\t-
    > duplicates avoided:",cpt_dupli)
305. print("\nNumber of queries per Intents:")
306. pprint.pprint(cnt_intents)
307. end_time = time.clock() # calculate the runtime of the program and output it in the shell

308. final_time=end_time-start_time
309. if final_time > 60:
310.     time_mn=final_time/60
311.     print("\n Time elasped:",round(time_mn,3),"minutes(s)")
312. else:
313.     print("\n Time elasped:",round(final_time,3),"second(s)")
```

Annex 6: full reference dataset

This dataset, written by the company supervisor, was used when performing performance tests in order to compare our system classification to a reference one. It is composed of 100 queries envenly devided into 4 intents: “get_document”, “excerpt”, “structured” and “reasoning”.

am i on overspeed what is the procedure	**get_document**
apu on fire procedure	**get_document**
autopilot off procedure	**get_document**
avionics smoke procedure	**get_document**
bat 2 overcharged page	**get_document**
brakes are unretracted how do i do the config	**get_document**
cargo door not closed procedure	**get_document**
cargo smoke procedure	**get_document**
display all turbine overheating procecures	**get_document**
show all pages for turbine overheat	**get_document**
all proc pages about overheat of turbines	**get_document**
show me all about lavatory lightening	**get_document**
there is something wrong with lavatory smoke detection	**get_document**
how does wather draining work	**get_document**
the food in galley is not refrigerated anymore how can we solve this situation	**get_document**
display general air conditionning info please	**get_document**
i am not sure the low fuel pressure indicator is fine	**get_document**
floor panel ref page please	**get_document**
reference page for wings	**get_document**
something is broke with the forwad galley	**get_document**
what are the engine welding restrictions	**get_document**
search reference for welding restrictions of engine	**get_document**
propellers ref page	**get_document**
show rotor system reference	**get_document**
i want to check the master wire list reference	**get_document**
what is my current maximum landing distance	**structured**
what is the pictch att for take off with unreliable speed indication	**structured**

the speech is indication is not reliable at what position should i set n one	**structured**
should i have autothrust on for cat two approach	**structured**
what is the current speed and thrust recommandation there are turbulences	**structured**
what is the recommended thrust setting when at fl hundred and servere turbulence	**structured**
what is my current autoland distance	**structured**
what correction should i appy to my landing distance the runway will be wet	**structured**
what is the correction to landing distance without autobrake in configuration three	**structured**
tell me what are the operating speeds with a weight of forty thousands kilograms	**structured**
what is the flaps lever position for landong with total hydraulic failure	**structured**
tell me the increment to v ref for approach speed with dc bus 2 failure	**structured**
i have flaps and slats at zero what is the increase to landing distance	**structured**
i am at fl three hundred ten with lost prot what should be my max speed	**structured**
what gravity fuel feeding ceiling should i use	**structured**
tell me my maximum speed with slats jammed	**structured**
the flaps are jammed is there a speed limitation	**structured**
what is the speed limit with slats and flaps jammed	**structured**
what is the correction to appply to landing distance runway is icy	**structured**
i have two reversers operative and runway covered with snow show me the land distance correction	**structured**
what is maximum altitude with one engine out	**structured**
show me operating speed vref	**structured**
what is the correction to maximum altitude	**structured**
my gross weight is sixty tons and flight level is hundred fifty what should be my thrust setting	**structured**
do class one failure have operational consequences	**structured**

show me a diagram of the overhead panel	**excerpt**
is there a diagram for rmp	**excerpt**
how does ground mode work	**excerpt**
what does mcdu stand for	**excerpt**
what is the mcdu	**excerpt**
show me the bus equipment list	**excerpt**
i want the general description of the auto flight system	**excerpt**
what are the auto flight modes of operation	**excerpt**
why is the mech button flashing amber	**excerpt**
how do i inhibit the vor	**excerpt**
what is the basic principle of flight control	**excerpt**
what do sec computers control	**excerpt**
what is the general architecture of flight control	**excerpt**
i need a description of the potable water system	**excerpt**
what is the purpose of the central maintenance	**excerpt**
how is the fuel system controlled	**excerpt**
show me a description of the fuel system control	**excerpt**
how is the green hydraulic system pressurized	**excerpt**
give me a list of the filters that clean the hydraulic fluid	**excerpt**
give me a description of the ice and rain protection system	**excerpt**
i have the wing a ice message displayed in green what does it mean	**excerpt**
why is the wing a ice message pulsing	**excerpt**
what does e c a m mean	**excerpt**
where is the ecam control panel	**excerpt**
tell me what is the meaning of the ecam messages colors	**excerpt**
i think we must do a diversion	**reasoning**
someone is having a seizure	**reasoning**
i am uncertain of our position it seems the instruments are wrong	**reasoning**
are we about to stall	**reasoning**
i am not sure of the fuel reserve	**reasoning**
we have a pilot incapacitation	**reasoning**

do i have enough fuel to divert to stockholm	**reasoning**
how do the crew deal with a water leak in the toilets	**reasoning**
the vibrations are very strong what should i do	**reasoning**
is the speed too high i am not sure	**reasoning**
do i have time for a go around	**reasoning**
the wind is strong should i recompute the fuel reserves	**reasoning**
i expect fog at arrival what is the probability that i have to do a cat three landing	**reasoning**
we have a food poisoning problem first officer is ill	**reasoning**
are we experiencing hypoxia what are the symptoms check cabin pressurization	**reasoning**
is there specific conditions for emergency descent in high traffic density situations	**reasoning**
why is the vhf sound so bad	**reasoning**
i have too many alarms keep only the most important	**reasoning**
explain why there is this cavalray charge sound	**reasoning**
is there an adiru failure what happens in that case	**reasoning**
why should i set engine one bleed off explain	**reasoning**
help me evaluate the criticality of the situation	**reasoning**
the visibility is poor should i use cat three approach	**reasoning**
what does this sound mean explain this alarm	**reasoning**
why is the auto pilot disengaged	**reasoning**

Annex 7: human intent classification

Here the figure corresponding of the intent classification made by human subjects. Subject C is the person that developed the baseline reference dataset, hence having the best performance compare to the other two.

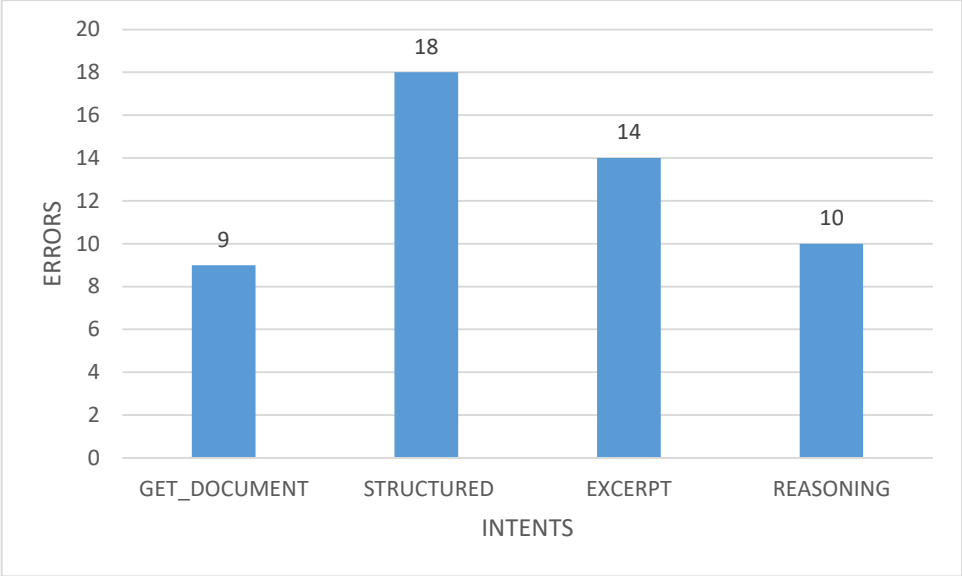


Figure 11: subject A classification

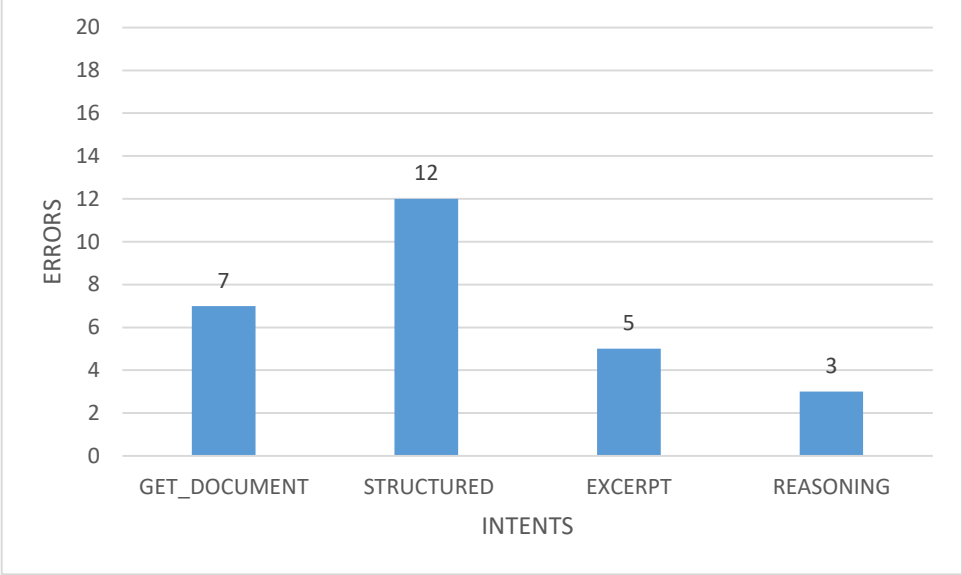


Figure 12: subject B classification

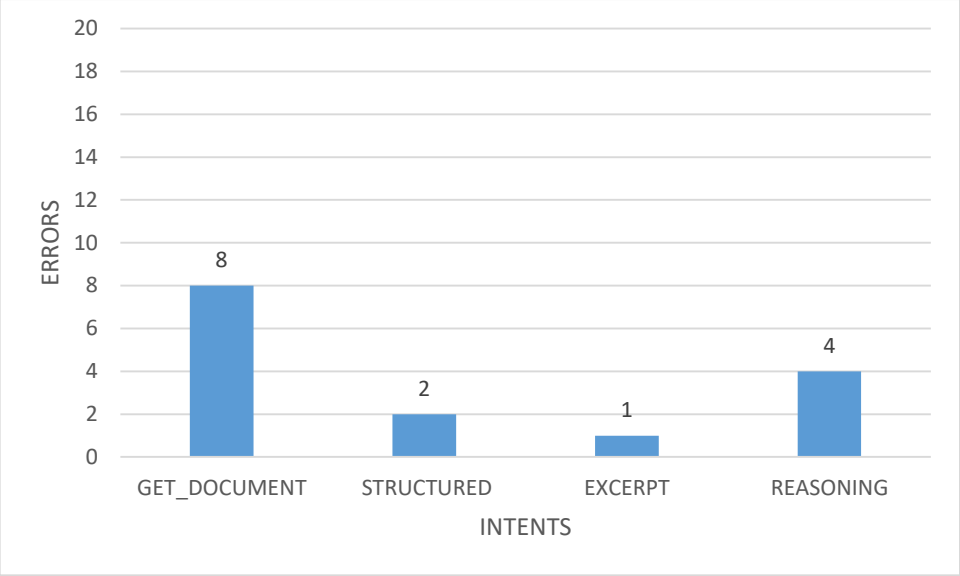


Figure 13: subject C classification

Table of contents

Introduction **8**

1. A virtual assistant technology at Airbus: 9
2. Internship goals: 10
3. Outline 11

Chapter 1 – Context and environment of the internship **12**

1. The Airbus Company 13
2. The Human Factor department: missions and purpose within Airbus 13

Chapter 2 – Report on paraphrasing techniques **15**

1. Introduction 16
2. Computer paraphrasing and generation 16
3. Conclusion 19

Chapter 3 - The Natural Questions Dataset **20**

1. Advices from experts on the methodology for collecting/creating questions 21
   1.1. Interview with a PhD student in cognitive science 21
   1.2. Interview with linguistics and human factor specialists 21
2. The methodologies for natural question collection 22
   2.1. Settings and general specificities of the interviews 23
3. Developer driven approach 25
   3.1. Validation of a set of queries with a pilot 25
4. User Driven Approach 26
   4.1. Use of specific scenarios during interviews with pilots 26
   4.2. Use of open type questions during interviews with pilots 28
   4.3. Problem Encountered 29
   4.1. Solution found to create a functional baseline dataset 30

Chapter 4 - The Synthetic Question Generator **33**

1. Specifications of the question generation program 34
2. Resources used by the program 35
   2.1. The Flight Operating Crew Manual (FCOM): 35
   2.2. Lexinet: 35
   2.3. Electronic Centralized Aircraft Monitoring (ECAM) Messages 36
3. Early design of the program[2] 37

4. Why we changed our approach 39
5. The final version of the synthetic question generator program 40
6. The final output of the question generation program 42
Chapter 5 – Using and testing the synthetic dataset 44
1. Why integrating machine learning into the project 45
2. RasaNLU: The Natural Language Toolkit 45
2.1. How does the library work? 46
2.2. Using RasaNLU 47
2.3. The training 48
2.4. Using the model 49
2.5. Conclusion: 49
3. A change of program to experiment with the datasets: 50
3.1. Why we had to stopped using RasaNLU 50
3.2. The Model creation 51
3.3. Predictions using the Models 51
3.4. Conclusion on the baseline model creation and predictions 52
Chapter 6 – Evaluation 53
1. The evaluation metrics and tools 54
1.1. Precision: 54
1.2. Recall: 54
1.3. F1 Score 54
1.4. (Standardized) Type Token Ratio: 55
1.5. Confusion Rate: 55
1.6. Confusion Matrix 55
2. The reference dataset 56
3. Evaluation results 56
3.1. Results on the maximum dataset 56
3.2. Results on balanced dataset excerpts 59
3.3. Comparison between the human and machine classification 61
4. Conclusion on the evaluations 63
Conclusion 64
1. Reflection on the internship 65
2. Perspectives 65

Bibliography & Webography **67**

Glossary and Acronyms **69**

Figures - table of contents **71**

Equations - table of contents **72**

Tables - table of contents **73**

Annexes – table of contents **74**

MOTS-CLÉS : Airbus, Génération Automatique, Aéronautique, Traitement Automatique du Langage Naturel, Compréhension du Langage Naturel, Agent Virtuel, Facteur Humain

RÉSUMÉ

Ce document synthétise un stage de 6 mois passé dans le service Facteur Humain du constructeur aéronautique Airbus à Toulouse sur une problématique de développement d'assistant virtuel. L'objet de ce stage était de vérifier l'hypothèse selon laquelle un système de catégorisation de requêtes d'utilisateurs pouvait bénéficier de la création de jeux de données synthétiques. Pour vérifier cette hypothèse nous avons dans un premier temps créé une méthodologie de récolte de questions dites "naturelles" auprès des utilisateurs finaux de l'agent virtuel. Ensuite nous avons développé un programme permettant de générer des questions synthétiques en se basant sur le jeu de données naturelles préalablement collectées. Pour finir nous avons expérimenté à l'aide de plusieurs outils sur les jeux de données générés. Les résultats obtenus ont permis d'ouvrir de nouvelles pistes de recherches et d'amélioration sur le sujet de la compréhension du langage naturel par le système.

KEYWORDS: Airbus, Generation, Aeronautic, Natural Language Processing, Natural Language Understanding, Virtual Agent, Human Factor

ABSTRACT

This document summarizes a 6 months internship that took place within the Human Factor department of the Airbus Company in Toulouse, France. This internship was centred on a virtual agent research thematic. Our initial hypothesis was that a system of automatic intent categorization could benefit from using synthetic “natural-like” data. In order to validate or invalidate this hypothesis we decided, first, to create a methodology that would help us collect natural questions from end-users. Then we used the “natural” data we previously collected along with a synthetic question generator we designed in order to output synthetic questions that feels “natural-like” if compared to the input ones. Lastly, we experimented on the synthetic datasets using various tools in order to put to the test our initial hypothesis. The results we obtained from these tests allowed us to open new perspectives on the natural language understanding part of the virtual agent system.