Gustavo Moreira Carreira

Desenvolvimento de um Chatbot para Apoiar a Aprendizagem de Programação

Dissertation in the context of the Master in Informatics Engineering, specialization in Intelligent Systems, advised by Professor Hugo Gonçalo Oliveira and Professor António José Mendes and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

September of 2022

1 2 9 0

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE Ð
COIMBRA

DEPARTMENT OF INFORMATICS ENGINEERING

Gustavo Moreira Carreira

Desenvolvimento de um Chatbot para Apoiar a Aprendizagem de Programação

Dissertation in the context of the Master in Informatics Engineering, specialization in Intelligent Systems, advised by Professor Hugo Gonçalo Oliveira and Professor António José Mendes and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

September 2022

Acknowledgements

Words cannot express my gratitude to professors Hugo Gonçalo Oliveira, António José Mendes, and Leonardo Soares e Silva, who generously provided knowledge and expertise, editing help, feedback sessions, quick response times, and moral support. Their invaluable patience can not go unnoticed either.

I am also grateful to my family, especially my parents, and sister. Their belief lifted me up when I came across what it looked like a dead end.

Abstract

As society's reliance on technology continues to increase, the demand for programmers will continue to rise, leading to an increase in the number of individuals seeking to learn how to program. Programming is regarded as a difficult subject to learn, and as the size of classes and the number of students enrolled in online courses continues to grow, providing individualised support to each student may appear impossible, leading to frustration, lack of motivation, and an increase in dropout rates. A solution could be the implementation of chatbots for the purpose of assisting such students. The benefits of using this technology would not only include the possibility of assisting multiple students at once, but also include 24-hour availability, instant feedback, and a means for shyer students to voice their doubts. The present work proposes the development of a Portuguese chatbot, Pyo, to assist students of introductory programming courses by providing explanations and examples of introductory programming concepts, assistance with the exercises, and guidance towards the resolution of errors in the algorithms. With the Rasa framework, a rule-based approach, and the help of other Python libraries, a chatbot was constructed, integrated into an online introductory programming platform, and then evaluated by real novice programmers. The evaluation was skewed toward the positive, with students identifying Pyo as beneficial to their learning journey, but it also uncovered a strong preference for seeking assistance from peers or professors rather than the agent as the students preferred more straightforward assistance.

Keywords

Natural Language Processing, Chatbot, Virtual Agent, Rasa, Introductory programming

Resumo

O aumento contínuo da dependência da sociedade relativamente à tecnologia tem fomentado um procura crescente por programadores. Este aumento cria incentivos, a um número crescente de pessoas, a desenvolver competências na área. A programação é considerada uma disciplina desafiante, cuja complexidade tende a aumentar com o número the alunos matriculados devido à falta de suporte individualizado, resultando em frustração, falta de motivação e num maior número de desistências. No ambiente educacional online, os chatbots oferecem uma maneira de colmatar este problema ao proporcionar apoio especializado a vários alunos em simultâneo. O presente trabalho propõe o desenvolvimento de um chatbot português, Pyo, para auxiliar alunos de cursos introdutórios à programação, oferecendo explicações de conceitos introdutórios, auxílio nos exercícios e orientação na identificação de erros. Desenvolvido com base em bibliotecas de Python, na framework Rasa, e uma abordagem baseada em regras, o chatbot criado foi integrado numa plataforma online de introdução à programação, onde foi usado pelos seus alunos. Da avaliação efetuada conclui-se que apesar da maioria dos alunos ter identificando o Pyo como benéfico para a sua aprendizagem, continua a existir uma preferência por assistência direta providenciada por colegas ou professores.

Palavras-Chave

Processamento de Linguagem Natural, Natural Language Processing, Chatbot, Agente Virtual, Rasa, Introdução à programação

Contents

Acronyms

ACM Association for Computing Machinery.

AI Artificial Intelligence.

AIML Artificial Intelligence Markup Language.

AST Abstract Syntax Tree.

BERT Bidirectional Encoder Representations from Transformers.

DIET Dual Intent and Entity Transformer.

FAQ Frequently Asked Questions.

GPT Generative Pre-trained Transformer.

IR Information Retrieval.

KB Knowledge Base.

NER Named Entity Recognition.

NLP Natural Language Processing.

NLU Natural Language Understanding.

POS Part-of-Speech.

QA Question Answering.

QG Question Generation.

seq2seq Sequence to Sequence.

SQUAD Stanford Question Answer Dataset.

List of Figures

List of Tables

Chapter 1

Introduction

Our world has been and continues to be revolutionized by technology. It has found its way into almost every aspect of our lives, such as transportation, food, education, healthcare, and socialization, which, consequently, increases the search for individuals with programming knowledge.

A study conducted in the United States in 2022[2] revealed that the demand for computer and information science professionals is projected to increase by 22% from 2020 to 2030, which is much faster than the average for all occupations.

Given this growth, it is not surprising that enrollment in computer science courses has, not only increased but, according to the British Computer Society, has seen the "largest increase of any university subject in the United Kingdom" in 2022[3].

On the other hand, this high number of new students is also followed by an equally high drop-out rate, with at least 9.8% of undergraduates dropping out before finishing their degrees, one of the highest rates when compared to other courses, with 33% of the students citing the difficulty of the course as the main reason[4].

There are a number of potential causes for such complications. For instance, an increase in class size without a corresponding increase in teachers[5] makes it more difficult to provide individualized support, which may have a significant impact on learning outcomes.

[2]https://www.bls.gov/ooh/computer-and-information-technology/computer-and-information-research-scientists.htm

[3]https://www.bcs.org/articles-opinion-and-research/record-numbers-have-applied-for-uk-computer-science-degrees-this-year/

[4]https://www.computerweekly.com/news/252467745/Computer-science-undergraduates-most-likely-to-drop-out

[5]http://cacm.acm.org/magazines/2019/10/239667-the-cs-teacher-shortage/fulltext

1.1 Problem and Motivation

The lack of individualized support can hurt the learning path of a student as it can lead to misconceptions going unnoticed. They would greatly benefit from close supervision, allowing immediate response to the given problems, and explanations and discussions on challenging concepts, hence the benefits of using a chatbot for support.

As defined by Dale [2016], a chatbot is "any software application that engages in a dialog with a human using natural language". This technology can deal with multiple students' questions at a time, providing constant support and some comfort to students who feel intimidated to ask a question to a teacher due to the fear of feeling judged.

The evolution of intelligent tutoring systems, particularly chatbots, has prompted researchers to employ this technology to address the issue of insufficient individualized support for novice programmers [Chinedu and Ade-Ibijola, 2021; Hobert, 2019]. However, it is still a field with a perceptive lack of contributions and novelty [Chinedu and Ade-Ibijola, 2021].

1.2 Objectives and Approach

Regarding the objectives of this thesis, two are proposed:

- The primary objective is the implementation of a chatbot to assist introductory programming students. The chatbot must communicate in Portuguese, motivated not only by the platform's language, where the chatbot was integrated but also due to the proximity of this dissertation to Brasilian and Portuguese universities, where Portuguese is the mother tongue. Its assistance should include the clarification of introductory concepts, errors in the students' algorithms, and assistance solving the exercises.
- The second objective is the integration of the chatbot into an online platform, that provides introductory programming courses, in order to conduct an evaluation with real students to identify any implementation flaws and determine if the chatbot is, in fact, regarded as a useful tool in the students' learning process.

In order to achieve the previously mentioned objectives, a multi-step action plan was developed.

In the initial phase, familiarization with Natural Language Processing (NLP) and chatbot technology was carried out, followed by an analysis of chatbots to identify the employed methodological strategies. It was determined that the majority consisted of introductory concept explanations.

In a second phase, the exercises, documentation, and back end of the platform were analyzed with three goals in mind:

- Examining what assistance it already provided and how the chatbot could add to it.
- Determining what concepts were being taught in the course to prepare the functionalities of the chatbot accordingly.
- Regulating what and how pertinent data could be passed to the chatbot. For instance, information regarding the errors in the algorithms of the students.

In light of this investigation and the analysis of other chatbots in the domain of introductory programming, three primary functionalities were developed:

- The feature known as concept definitions provides explanations and examples for introductory programming courses.
- The second, error guidance, was defined as a dialogue consisting of multiple-choice questions and clarifications that assists students in identifying and correcting errors in their algorithms
- The third was inspired by a feature analyzed in a related work that provided pseudo code algorithm examples, thus inspiring the exercise assistance functionality. It consists of providing assistance by prompting the student to arrange the pieces of a natural language-written skeleton representing one possible solution.

Additionally, two secondary functionalities were defined: question suggestion, which assists the student in determining what questions, related to the exercise at hand, are pertinent to ask the chatbot, and hints regarding previously solved similar exercises and small details that are frequently forgotten.

After defining the functionalities, an investigation into the required implementation tools was conducted. In regards to chatbot platforms, the Rasa[6] platform was selected due to its popularity and greater flexibility than others, followed by language models to automate question answering and the generation of questions and Python libraries. To familiarize oneself with the operation of each tool, each was initially employed independently.

Implementing the chatbot and integrating it in the platform was the third step. The chatbot was created by gluing together the tools from the previous step, despite the fact that a few had to be discarded because they were not a "perfect fit". The developed system and its features were subsequently evaluated in a real introductory programming course, where the conversations were analyzed and students were asked to complete a questionnaire.

1.3 Contributions

The main contributions of this thesis are:

[6]`https://rasa.com`

- The creation of a Python-specific Portuguese teaching assistant chatbot for an introductory programming course. Whose qualities are enhanced by a proactive willingness to offer assistance when deemed necessary. As will be referenced throughout the dissertation, the chatbot assistant was given the name **Pyo**, derived from the Portuguese word "Pio," as it is devoted to its mission of teaching programming. The letter "y" alludes to Python, its language of focus.
- The evaluation of the chatbot with real students, who were asked to complete a questionnaire to assess its usefulness and effectiveness in assisting their learning process.

The development of Pyo, with an emphasis on its educational features, was reported in a scientific article regarding the XXIV International Symposium of Education on Computers in Education, which focuses on systems, platforms, pedagogies, and practice-based education in e-learning and b-learning [Carreira et al., 2022].

1.4 Outline

This document is organized in sections that describe the phases of the work developed throughout the year, from the review of the literature to the evaluation. The remainder of the dissertation is organized as follows:

1. **Background:** touches on the programming difficulties a student might encounter, describes the platform where the chatbot was integrated, and introduces pertinent concepts for a better understanding of the work, particularly in regards to NLP, chatbots, information based question answering, and transformers.
2. **Chatbot Specification:** describes the functionalities from a conceptual standpoint and presents the initial and final architecture of Pyo.
3. **Implementation:** describes and explains the implementation procedure of the chatbot and its functionalities.
4. **Evaluation and Discussion:** presents and discusses the results of the analysis of the students' conversations with the chatbot, as well as the results of a questionnaire administered to the students.
5. **Conclusion:** conducts an overall evaluation of the work performed and a reflection on the expected and achieved objectives and contributions. Additionally it describes the next steps proposed to complement the performance and utility of the assistant.

Chapter 2

Background

This section presents the theoretical concepts necessary for a complete comprehension of the work described in this dissertation. Section 2.1 discusses problems encountered by students at the beginning of their programming journeys, as well as potential causes and solutions, followed by a description of the RESPE platform into which Pyo was integrated (section 2.2). Section 2.3 discusses Natural Language Processing (NLP), the technology at the core of chatbots, and the fundamental components of NLP that a message may go through as to retrieve relevant information.

In section 2.4, a discussion of chatbots' concept, history, categories, approaches, development platforms, advantages and disadvantages, as well as chatbots developed for the education domain, with a focus on those for the introductory programming domain, is presented. Additionally, in section 2.5, a description of a method for automatic question answering (Information Retrieval Based Question Answering) is provided, followed by an explanation of the transformers architecture, which is widely used for various NLP taks and applications (section 2.6).

2.1 Programming Learning Difficulties

Having in mind that the goal of the project is to develop a chatbot with the intent to support students in learning how to program, it is fundamental to have a general understanding of the difficulties experienced by novice programming learners as well as possible ways to overcome them. Therefore, a study of such was conducted and is presented in this section.

As defended by multiple researchers, learning how to program involves three different types of knowledge: syntactic, conceptual, and strategic [McGill and Volet, 1997].

Syntactic knowledge refers to the understanding of the rules and basic features of a given programming language, for example, knowing that Python requires indentation to differentiate between blocks of code, or that in Java, to end a statement, the use of a semicolon is needed.

Conceptual knowledge alludes to the understanding of how programming concepts and principles work, such as why is indentation necessary in Python or what is the reason behind the use of semicolons in some programming languages. Difficulties in this knowledge have a greater impact than the previous mentioned, as it may lead to significant misconceptions related to students' mental models of code execution [Qian and Lehman, 2017]. An example of such errors would be the difficulty of a beginner to understand that A=1 is not equivalent to 1=A.

Strategic knowledge can be summed up as combining syntactic and conceptual knowledge to develop solutions for problems. Such understanding is essential in breaking down a problem, designing a solution, and debugging the program. It is relatively common for novices to display a lack of such knowledge, which demonstrates itself in difficulties when planning, writing, and debugging solutions.

Qian and Lehman [2017] point out that there are numerous potential causes of difficulties, with the following being the most significant:

Task Complexity and Cognitive Load Vainio and Sajaniemi [2007] found that due to this freshness, tracing programs becomes complex and requires a high cognitive load on novices. Thus, increasing the task complexity may lead to confusion because it will require an even higher cognitive load.

Anderson [1993] states that becoming an expert in problem-solving does not come from having a better ability at such, but more from domain familiarization. Thus, novices who are yet to have the time for such extensive learning will increase mistakes when solving problems with higher complexity. Hence, increased task complexity can also be accompanied by a higher number of mistakes.

Flawed Mental Models In general, a mental model can be seen as an internal representation of a given domain. They allow reasoning about a situation not directly experienced, allowing individuals to mentally simulate the behaviour of a program. Having a flawed perception of how a system works, can result in various misconceptions [Qian and Lehman, 2017]. Ma et al. [2007] found that a very high portion of novice programming students held non-viable models of essential programming concepts at the end of their first year and that the small portion of students with viable-mental models displayed significantly better performance in various programming tasks.

Environmental Factors Language features and programming environments may also constitute obstacles to students' success in introductory programming. An example of such is the ambiguity of the addition operator (+). This operator can either add two numbers or concatenate strings in various languages, which allows flexibility to more skilled programmers but can inevitably confuse novices. Another prevalent example results from error messages that are frequently perceived as cryptic due to their lack of information, thereby imposing a substantial barrier on a beginner. [Qian and Lehman, 2017].

Teachers' Instruction and Knowledge Teachers' knowledge can have significant consequences in the students' flowed mental models, and therefore, as mentioned, in their performance, meaning that, if an educator contains incorrect or incomplete knowledge of a given concept, that is what is going to be transmitted to their students and consequently having the same outcome [Ma et al., 2007].

Having understood the main difficulties and their possible sources, Qian and Lehman [2017] also propose different approaches to tackle them, which mainly integrate two levels: at an editor or choice of programming language level, or at a teacher's level. Regarding the first, the authors call for editors that highlight or prevent syntax errors (already supported by various Integrated Development Environments) and using more simple, easily comprehendable, retainable, and self-explanatory programming languages, instead of jumping straight to a choice based on industry popularity as they are usually developed for professional use. As for the second, the authors propose: using clear and straightforward examples; having students explain programs to help educators better perceive possible misconceptions the student may contain; and explicitly teaching helpful programming strategies.

2.2 RESPE

As mentioned throughout the dissertation, Pyo has been integrated into a platform. This platforms was developed as part of a doctorate, and supervised by professor António Mendes. This platform, RESPE[Silva et al., 2022], intends to provide a number of tools to facilitate programming education (one of which is Pyo). It has been utilized to support multiple editions of the same course, including the one in which the evaluated agent has been assessed. Important to note that, despite being mentioned, the course in which the conversational agent was evaluated did not utilize any additional RESPE tools, apart from Pyo.

The course takes place over two moths with a 40 hours duration. Its syllabus is based on the Association for Computing Machinery (ACM) Curricula for fundamental programming concepts, encompassing variables, input/output, conditionals, loop structures, and functions[7].

While participating in a course, students have 24-hour access to the recorded video materials about the programming contents. Students are left learning at their own pace, with a new section unlocked approximately every two weeks. The order followed is variables, conditional, loop structures, and functions.

Regarding the implementation of Pyo, the main focus was on the programming exercises, consisting of a total of 70 (13 variables, 24 conditionals, 21 loops, and 12 functions), which are available on an Open Science repository[8].

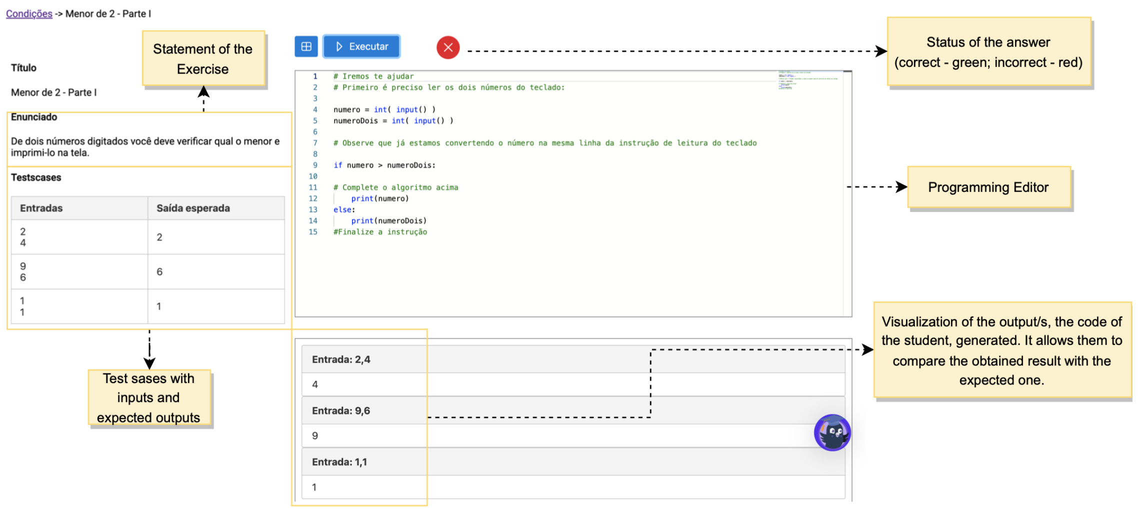
Figure 2.1 displays an example of a programming exercise where the student is asked to complete the algorithm presented initially on the editor. The student has access to a table with test cases to give them the perception of the expected outputs the algorithm should produce. In the example, the student submitted an

[7]`https://www.acm.org/education/curricula-recommendations`

[8]`https://osf.io/tu47x/?view_only=aab2c35ee94d41b1abaf5ececea3f0dd`

answer that did not generate the expected outputs, thus the red circle on the top.

All of the information regarding the student's submission, including their exact algorithm, whether it was correct, and whether it generated a Python error, is stored in a database for later evaluation.



Condições -> Menor de 2 - Parte I

Título

Menor de 2 - Parte I

Enunciado

De dois números digitados você deve verificar qual o menor e imprimi-lo na tela.

Testscases

Entradas	Saída esperada
2 4	2
9 6	6
1 1	1

```python
# Iremos te ajudar
# Primeiro é preciso ler os dois números do teclado:

numero = int( input() )
numeroDois = int( input() )

# Observe que já estamos convertendo o número na mesma linha da instrução de leitura do teclado

if numero > numeroDois:

# Complete o algoritmo acima
    print(numero)
else:
    print(numeroDois)
#Finalize a instrução
```

```
Entrada: 2,4
4
Entrada: 9,6
9
Entrada: 1,1
1
```

Statement of the Exercise
Executar
Status of the answer (correct - green; incorrect - red)
Programming Editor
Visualization of the output/s, the code of the student, generated. It allows them to compare the obtained result with the expected one.
Test sases with inputs and expected outputs

Figure 2.1: Example of a programming Exercise

Figure 2.2 illustrates the architecture of the RESPE platform, after the integration of Pyo. The frontend of the platform is built with the angular framework, which employs HTML, TypeScript, and CSS, while TypeScript and Python are used to perform its logic and communicate with its database. Google's platforms, specifically Google Cloud and Firestore, are used for both hosting and database.

The communication with Rasa is done easily, as Rasa already comes with an endpoint. Therefore, to send messages to Rasa a simple http POST method is required to the "http://<host>:<port>/webhooks/rest/webhook" enpoint, to which Pyo will then return its reponse.

2.3 Natural Language Processing

Chatbots are one of the many applications the field of NLP has. In this section, we discuss this concept and several of its components.

NLP is a subset of Artificial Intelligence (AI) defined by Eisenstein [2018] as "the set of methods for making Human language accessible to computers", meaning enabling systems to communicate with humans in the same language they use to speak and write, also called natural language, as opposed to formal languages.

From machine translation [Nakazawa et al., 2006; Wu et al., 2016], question answering [Ben Abacha and Zweigenbaum, 2015; Choi et al., 2018; Lende

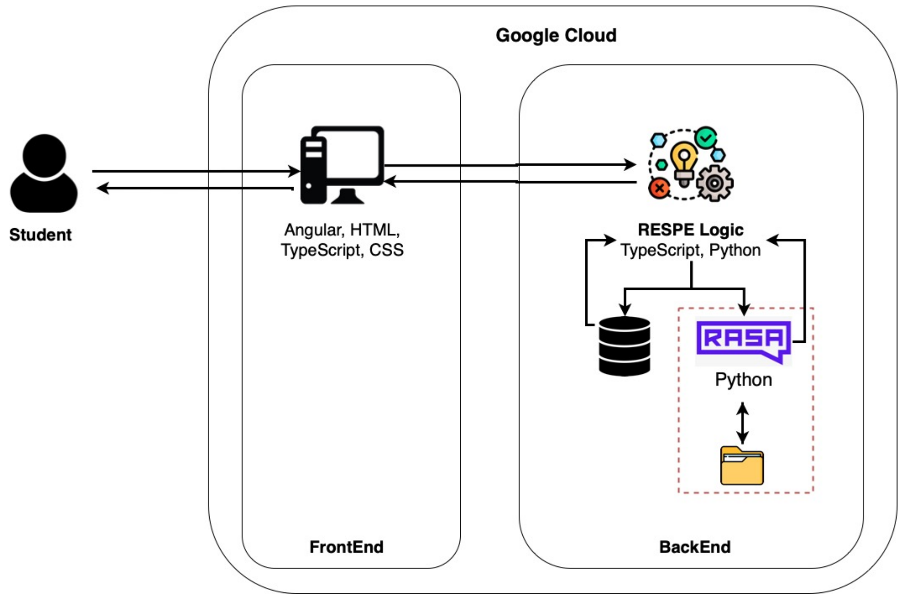


Figure 2.2: Architecture overview diagram for the RESPE platform.

and Raghuwanshi, 2016], question generation [Du et al., 2017], sentiment analysis, text classification [González-Carvajal and Garrido-Merchán, 2021; Liu et al., 2018], auto-correct [Yu and Tsai, 2021], and chatbots, NLP has become an integral part of our lives.

As a subfield of NLP, Natural Language Understanding (NLU) is primarily concerned with programming machines to interpret natural language, derive meaning, recognize context, and draw conclusions. NLP focuses on processing text literally, whereas NLU focuses on extracting the context and intent. Using the phrase "Let's hang out" as an example, NLP will interpret the request as a literal request to hang outside, whereas NLU can infer that the user may intend to spend time together.

NLU is typically implemented in a pipeline, as the tasks are interdependent, and the output of one task serves as the input of the next. The pipeline consists primarily of two stages: text processing and feature extraction.

2.3.1 Text processing

Text processing refers to the tasks that transform unprocessed input into a suitable format for feature extraction. In terms of text processing, a simple pipeline may include the processes of Tokenization, Text Normalization (Stemming and Lemmatization), Part-of-Speech Tagging, and Named Entity Recognition.

Tokenization

Tokenization is typically the first step in the pipeline and a crucial process. It refers to dividing text into smaller units, known as tokens, that convey useful information (usually a word). It is significant because several subsequent steps require tokens as a starting point to extract and provide additional information.

Practical for various use cases is what is called a white space tokenizer. Implied by its name, this approach separates text by white spaces disregarding punctuation. For example, “Hello, how are you?” would yield the array [“Hello”, “how”, “are”, “you”].

Although using a white space tokenizer is common, it is essential to note a few difficulties that a language may present. In English, for example, contractions present a challenge. How should the sentence “These **aren't** green” be segmented? Such difficulties vary from language to language. In Portuguese, for instance, the use of compound words (“guarda-roupa”) and contractions (“daqueles” instead of “de aqueles”) may become problematic. To resolve these issues, it is frequently necessary to develop models that account for these factors and to establish rules for the various exceptional circumstances.

Stopword Removal

Stopwords are frequent words, such as propositions, pronouns, and determinants, that contribute little or nothing to the meaning of a phrase. For instance, in the Portuguese language, “o”, “ele”, and “de” would be two examples of such words. Eliminating stopwords may be advantageous for specific tasks regarding classification, such as, sentiment analysis [Kanakaraj and Guddeti, 2015], and fraud detection [Chen et al., 2017].

Text Normalization

Text normalization is attempting to convert text into a more uniform format to facilitate the association between words that, despite being written differently, may have the same origin. Two processes are used with such intent: Stemming and Lemmatization.

Both stemming and lemmatization aim to make all related words converge into a common term. For instance, *drove*, *drives*, and *driving*, are all rewritten as *drive*.

Stemming accomplishes this with heuristics that aim to remove the ending of words to achieve this goal, mostly involving eliminating derivational affixes. It could be problematic, as it may not always produce an existing word. For instance, *cats* results in *cat*, while *write* may result in *writ*.

Lemmatization refers to using a vocabulary and morphological analysis of words, usually aiming to remove inflectional endings only and to return the dictionary form of a word, known as the lemma.

Part-of-Speech Tagging

Part-of-Speech (POS) Tagging aims to determine the syntactic function of each word in a sentence (which, depending on the context, may be different for the same word). In simple terms, it describes how the antecedence and precedence of particular words influence others.

Ex: The$_{DET}$ dog$_{N}$ is$_{PREP}$ black$_{ADJ}$

Named Entity Recognition

Named Entity Recognition (NER) is the process of detecting, identifying, and classifying entities that contribute to the meaning of a text. From a programming perspective, entities can be seen as variables needed to achieve a task. These entities are categorized according to predefined categories (e.g., person, location, date).

Ex: Kate$_{PERSON}$ would like to book a flight to Lisbon$_{LOCATION}$ for tomorrow$_{DATE}$.

2.3.2 Feature Extraction

The output of the previous stage is a preprocessed vector in text format, which is not the most friendly for the majority of machine learning algorithms. Feature extraction is responsible for converting this vector into numeric format. The features extracted go accordingly to the type of NLP task trying to be accomplished. For instance, the frequency and context of words are extracted as features for the tasks described in this section.

Bag of Words

Regarding the bag-of-words method each word accounts for a feature. This technique involves counting the frequency of each word in a text regardless of its position. Observing the following phrases:

- Phrase 1: Can you give me an example of a for loop?
- Phrase 2: What is an example of an if statement?
- Phrase 3: What is the syntax of a for loop?

Our vocabulary would consist of the following 14 words: “what”, “you”, “the”, “me”, “an”, “example”, “of”, “a”, “give”, “for”, “loop”, “Can”, “is”, “syntax”, “if”, “statement”.

	Phrase 1	Phrase 2	Phrase 3
What	0	1	1
you	1	0	0
the	0	0	1
me	1	0	0
an	1	2	0
example	1	1	0
of	1	1	1
a	1	0	1
give	1	0	0
for	1	0	1
loop	1	0	1
Can	1	0	0
is	0	1	1
syntax	0	0	1
statement	0	1	0
if	0	1	0

Table 2.1: Bag-of-words example

Figuratively shaking the words inside a bag, a possible take of this method is to count the frequency of each word in the set of documents/phrases, as demonstrated in table 2.1.

Finally, the previous example would result in the following sparse vectors:

- Phrase 1: [0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0]
- Phrase 2: [1, 0, 0, 0, 2, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1]
- Phrase 3: [1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0]

Word Embeddings

Embeddings are yet another effective word representation. Its idea is to map every word to a point in space where words with similar meanings are physically closer. The space in which they are present is called an embedding space. One can pre-train an embedding space to save time or use one already pre-trained, with GloVe [Pennington et al., 2014] and WordVec [Mikolov et al., 2013] being the most popular approaches for learning word embeddings, based on neural networks.

The embedding space then maps each word to a vector with dimensions between 50 and 1000 rather than the much longer vocabulary size. In opposition to the bag-of-words vectors, which have several zero-counts, embeddings will be dense vectors, of positive or negative real numbers. This is possible because each entry does not correspond to a word, rendering embeddings not interpretable [Jurafsky and Martin, 2021].

2.4 Chatbots

The topic of this dissertation is chatbots, one of the many application of NLP. This section focuses on this technological advancement. Through research, we attempted to comprehend what a chatbot is, its origin, the various ways it can be categorized, the primary approaches taken, and the benefits and drawbacks of utilizing these systems.

2.4.1 Brief History

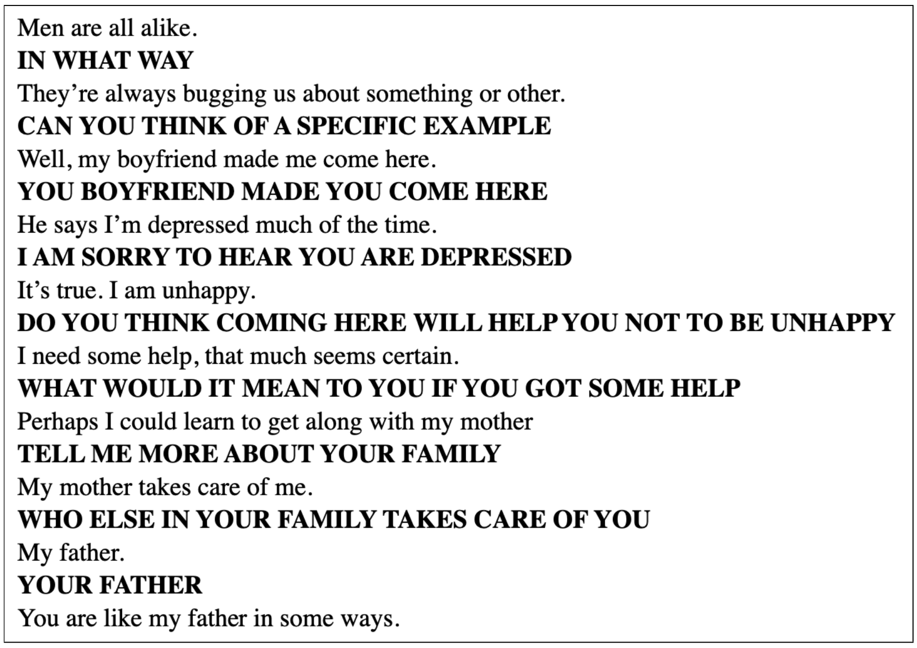
The term conversational agent has come to express a variety of systems with different capacities and motivations, with the fundamental premise that the artificial agent takes part in a human-machine conversation through natural language [Eeuwen, 2017]. Chatbots are the simplest types of such systems.

Communication between computers and humans is as old as the field of computer science itself. Traveling back to the 1950s leads us to a prominent figure named Alan Turing. "I PROPOSE to consider the question, 'Can machines think?'" began Turing in his paper "Computer Machinery and Intelligence" [Turing, 1950], where he then proposed a well-known test, the Turing test, to determine if a machine can be considered intelligent. The Turing test consists of three participants: two humans and a computer. One of the humans plays the role of the interrogator, whose objective is to determine which of the other two contestants is the computer by asking a series of text-based questions. The computer is considered intelligent if the interrogator cannot correctly identify the machine [Jurafsky and Martin, 2021].

Amidst the fuss generated by the Turing test, the birth of the first chatbot, ELIZA, took place. Developed by Weizenbaum [1966], an MIT professor, this conversational agent uses pattern matching and substitution methodology to emulate the operations of a psychotherapist, mainly consisting of returning back the sentences of users in an interrogative form. Despite its straightforward approach and unsuccessful attempt at conquering the Turing test, the program still gave the illusion of understanding the problems of the users and successfully fooled many people, which encouraged a whole community of interest in developing chatbots to defeat the Turing Test [Dale, 2016]. A sample from an example conversation taken from the original paper is displayed in figure 2.3.

Fast forward a decade, Kenneth Colby developed a chatbot named PARRY, described by the creator as "ELIZA with an attitude" [Colby et al., 1971]. This time, the chatbot aimed towards simulating the behaviour of a patient with paranoid schizophrenia, capable of expressing beliefs, fears, and anxieties [Reshmi and Balakrishnan, 2016]. Its structure is similar to the one of ELIZA but with an upgrade in controlling structure, understanding capabilities, and a mental model to simulate emotions. PARRY was evaluated using an adaptation of the Turing test where it managed to fool 52% of the psychiatrists involved.

Another worthy of noting highlight in the history of chatbots is Alice (Arti-



Men are all alike.
IN WHAT WAY
They're always bugging us about something or other.
CAN YOU THINK OF A SPECIFIC EXAMPLE
Well, my boyfriend made me come here.
YOU BOYFRIEND MADE YOU COME HERE
He says I'm depressed much of the time.
I AM SORRY TO HEAR YOU ARE DEPRESSED
It's true. I am unhappy.
DO YOU THINK COMING HERE WILL HELP YOU NOT TO BE UNHAPPY
I need some help, that much seems certain.
WHAT WOULD IT MEAN TO YOU IF YOU GOT SOME HELP
Perhaps I could learn to get along with my mother
TELL ME MORE ABOUT YOUR FAMILY
My mother takes care of me.
WHO ELSE IN YOUR FAMILY TAKES CARE OF YOU
My father.
YOUR FATHER
You are like my father in some ways.

Figure 2.3: Sample ELIZA dialogue from Weizenbaum [1966]

ficial Linguistic Internet Computer Entity), developed by Richard Wallace [Wallace, 2009]. Initially inspired by ELIZA, it was released into the wild in 1995, and it went on to win the Loebner Prize[9] three times. Alice was the first chatbot to use the Artificial Intelligence Markup Language (AIML) language to relate a user input with a Knowledge Base (KB) response. A few more details on AIML will be given on section 2.4.3, since it is a very common language used to create the knowledge bases of chatbots.

In the early 2010s, the idea of conversational agents in our day-to-day lives started to cement with the rise of the well-known Big Four voice-driven assistants: Siri from Apple, Cortana from Microsoft, Alexa from Amazon, and Google's new Assistant [Dale, 2016]. Nevertheless, even though these have been and will continue to be at the forefront of this technology, they are far from being the only ones, as the commercial interest of this technology keeps increasing.

Although the implementation of chatbots started by simply following a patter-matching approach, its recent achievements are due to the use of Deep Learning. First introduced by Cho et al. [2014] for Machine Translation, Sequence to Sequence (seq2seq) techniques have since been used to create several generative systems [Lu et al., 2017]. Recent studies make use of pre-trained models, such as the GPT-2, which are then fine-tuned to be implemented in chatbots [Budzianowski and Vulić, 2019]. SHI et al. [2020] provide an illustration of such a system by implementing a language-learning chatbot that combines a transfer-learning-based training scheme and a high-capacity transformer model.

The discussed selection of chatbots is illustrative and not exhaustive, and it serves as a summary indicating the essential lines in the evolution of this technology.

[9]a competition inspired by the Turing test, defunct in 2020, that awarded prizes to the computer programs considered by the judges to be the most human-like

Criterion	Categories
Knowledge Domain	Open domain Closed domain
Goals	Informative Conversational Task-Oriented
Input Processing and Response Generation	Rule-based Corpus-based Hybrid
Communication Channel	Text Voice

Table 2.2: Types of chatbot classifications

2.4.2 Chatbot Categories

A chatbot can be employed for a variety of purposes. Its classification can and may be categorized differently, depending on the criteria considered. According to the creteirons, knowledge domain, goals, input processing and response generation, and communication channel, table 2.2 displays the various classifications a chatbot can and may have have [Adamopoulou and Moussiades, 2020].

According to the knowledge accessed by a chatbot, it can be categorized as closed or open domain. A closed-domain chatbot is focused on a specific domain. For example, OneRemission [Ayanouz et al., 2020] is a chatbot that provides information about cancer and post-cancer healthcare to survivors, fighters, and supporters. On the other hand, open domain chatbots can converse about general topics. An example would be Blender by Facebook [Liang et al., 2021], which can have sophisticated conversations on nearly any topic.

When it comes to the goals the chatbot aims to achieve, it can either be informative, conversational or task-based. Implied by the name, a conversation agent that aims to deliver information stored beforehand or available from a designated source is designated informative. Usually, they are based on Information Retrieval (IR) algorithms and would either fetch the result of a query from a database or perform string matching. An example would be any that falls in the category of Frequently Asked Questions (FAQ) chatbots [Sethi, 2020]. A conversational chatbot refers to those whose aim is mainly to hold a natural conversation with a user, much like a human-to-human conversation. For example, the Mitsuku [Croes and Antheunis, 2021] and the XiaoIce [Zhou et al., 2020] chatbots, fall into this category. Finally, task-oriented chatbots are responsible for handling a specific task such as bookings (restaurants, flights, hotels) [Garg et al., 2021]. Its actions and flow of events are, normally, predetermined [Nimavat and Champaneria, 2017]. Leggeri et al. [2018] presents an example in which the authors describe a task-oriented conversational agent implemented to help people understand how to use a platform to track an organization's performance and pinpoint losses in real-time.

Regarding input processing and response generation, a chatbot can be divided into two main classes: rule-based chatbots and corpus-based chatbots, which will be examined in greater detail in the following section.

The last type of classification is probably one of the first that comes to mind when classifying a chatbot. A chatbot can carry out a conversation in two ways: text, where the interaction is conducted through messaging, or voice, in which the conversation is carried out orally.

2.4.3 Rule-based vs Corpus-based Approaches

Although the different classifications of chatbots have already been discussed, when thinking of its implementation, two are the main approaches one can take into account, rule-based and corpus-based.

Rule-based approaches act upon pattern-matching to map the input supplied by the user to a given rule pattern, which then selects an answer from a set of responses defined beforehand [Adamopoulou and Moussiades, 2020]. ELIZA, PARRY, and Alice fall into this category.

Because the developer has written the knowledge in conversational patterns, this method is typically rigid because it does not generate new responses. The developer must outline all possible rule paths (dialogue flows) to provide a greater capacity to answer user questions correctly.

Due to the fact that the messages of the user are not subjected to a thorough syntactic or semantic analysis, this method has a quick response time, and its behavior is entirely under the programmer's control. On the other hand, these chatbots lack the uniqueness of human responses. The level of diversity in a conversation is determined by the number of programmed options.

When proceeding with this approach, one of the most common languages is AIML, a programming language developed by Richard Wallace between 1995 and 2000. It serves the primary purpose of building the knowledge base of rule-based chatbots [Mikic et al., 2009]. As we have previously touched on, the first chatbot to use this language was ALICE, which went on to win three Loebner Prizes.

A general example is given in figure 2.4, where if a user says "Hello" to the chatbot, it would respond with "Hello, how are you?"

Chatscipt [Wilcox and Wilcox, 2014], a rule based engine developed by Bruce Wilcox, is also widely used when taking this approach [Finch et al., 2020]. Its main advantage when compared to AIML is the support of longer back and forth exchanges between the user and the chatbot, called gambits, which is what keeps the response of the agent focused on the current topic, but with the drawback of requiring a much more complex implementation that often turns people back to AIML.

Although AIML is widely used because of its simplicity, this is also one of its drawbacks. When implementing a rule-based conversation agent using chatbot

```
<aiml>

   <category>
      <pattern> HELLO </pattern>
      <template> Hello, how are you? </template>
   </category>

   <category>
      <pattern> GOODBYE </pattern>
      <template> Goodbye, see you around! </template>
   </category>

</aiml>
```

Figure 2.4: Simple AIML example.

frameworks such as Dialogflow[10], IBM Watson[11], Rasa[12], and Microsoft LUIS[13], the use of a method that focuses on the intents and entities of a message is becoming increasingly prevalent. Instead of focusing on the message itself, it attempts to link it to its intent. For instance, given the message "Hello", the intent would be "greet", and the chatbot would respond accordingly. Another type of intent would be "What is the weather like in Coimbra?" which could be labeled "weather_forecast". The difference between the two examples is that in the second, we must also extract additional information from the message, namely the entity, which in this case is "Coimbra", the location where the user desires the weather forecast.

The other alternative, to rule-based conversational agents, are corpus-based chatbots, which map human-human conversations [Jurafsky and Martin, 2021] instead of using hard-coded rules. This approach is hugely data-driven. The amount of data for training these types of chatbots depends on various aspects, including the coverage of the bot and the complexity of the domain, but to learn how to respond effectively to different human interactions requires a substantial amount of data. This approach has seen an increase in engagement and success, majorly due to an increase in the availability of extensive public datasets and efficient machine learning models [Serban et al., 2018].

These bots generally generate their responses via either retrieval methods or generation methods. Retrieval methods rely on an IR method which retrieves response candidates from a pre-built index and then ranks the candidates. The calculations of the scores can be achieved using the approach explained in section 2.5. Based on this score, the system selects a reply from the top ranked ones.

As we have mentioned, the generation of a corpus-based chatbot's response is not restricted solely to the use of IR methods. It can instead generate its answers based on the context of the dialogue through an encoder-decoder or language models, which is a model that assigns probabilities to sequences of words

[10]http://cloud.google.com/dialogflow

[11]http://www.ibm.com/pt-en/products/watson-assistant

[12]http://rasa.com

[13]http://www.luis.ai

(e.g. GPT-2) [Jurafsky and Martin, 2021]. The encoder-decoder takes the input sequence (the query usually formed from the entire conversation) as to produce a contextualized representation of it, which is then passed onto the decoder that generates the response (figure 2.5).

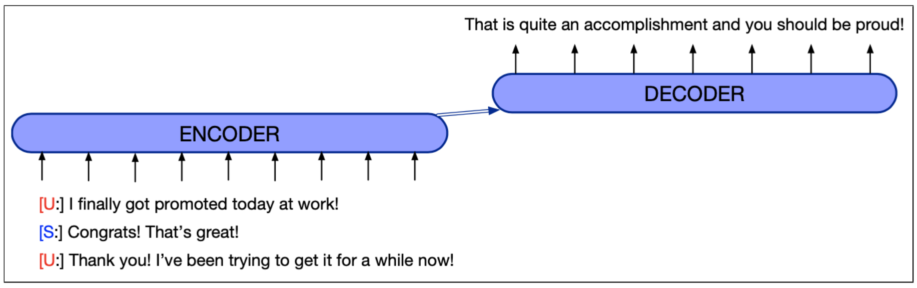

That is quite an accomplishment and you should be proud!
DECODER
ENCODER
[U:] I finally got promoted today at work!
[S:] Congrats! That's great!
[U:] Thank you! I've been trying to get it for a while now!

Figure 2.5: Encoder decoder architecture for response generation, from Jurafsky and Martin [2021].

Unlike rule-based chatbots, these do not require previously defined responses for every possible outcome and take into account the context of the conversation. However, they do require large amounts of training data, which for a particular domain may constitute a difficulty, as it might not be available.

Due to the specificity of the chatbot domain (programming domain), the rule-based approach was more heavily used for the implementation of Pyo, although an initial approach also benefited from the second.

2.4.4 Advantages and Disadvantages of Chatbots

There are numerous benefits for utilizing a chatbot. In a survey conducted by Drift[14], SurveyMonkey Audience[15], Salesforce[16], and myclever[17], participants were asked, "If chatbots were available (and working effectively) for the online services you use, which of the following benefits would you expect them to provide?". The results of said survey are presented on figure 2.6.

Undoubtedly, consumers expect 24-hour service, followed by immediate response, answers to simple questions, and accessible communication, which a conversational agent must be able to provide.

Two years later, another study conducted by Adamopoulou and Moussiades [2020] in 2020 reached a similar conclusion, reporting productivity as the primary reason for using chatbots, with the majority of participants citing ease of use, quick service, no waiting times, and the comfort of not having to interact with a human, as it can be intimidating at times. Other motivations included entertainment, social and relational purposes, and novelty, indicating that some participants were interested in experimenting with this novel technology.

[14]`https://www.drift.com`
[15]`http://www.surveymonkey.com`
[16]`https://www.salesforce.com/`
[17]`http://www.myclever.com`
[18]http://www.drift.com/learn/chatbot/

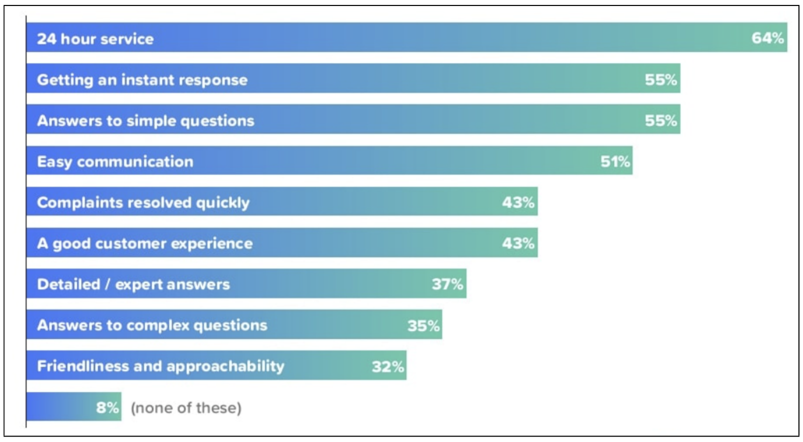


Figure 2.6: Survey answers on the benefits expected from a chatbot[18]

These factors explain why the development of this technology is not anticipated to cease in the near future. Technavio[19] projects that the market for chatbots will increase by 1.73 billion dollars at a compound annual growth rate of 25% between 2021 and 2025.

However, like everything else in life, chatbots have their drawbacks. A clear but drastic example of how things can quickly go wrong is Tay [Davis, 2016], a chatbot released by Microsoft in 2016 that was discontinued after only 16 hours of its release. It used a neural network that learned by interacting with users, which led to its manipulation. It was released on Twitter, and it was intended to emulate a teenage girl and engage with users. Shortly after its release, Tay had sent thousands of texts, primarily abusive and of racist nature, which it had learned from the social platform. This situation came to secure the position several researchers already had that chatbots must be developed in a controlled environment to prevent such situations from repeating themselves. Other disadvantages, now more related to typical conversational agents, include the fact that they may contain a certain degree of coldness, not be programmed to answer more than a question at a time, may not understand the question of a user, or may return incorrect responses, which may cause frustration.

2.4.5 Chatbot Platforms

To investigate the solutions available, research was conducted on the most popular platforms that enable the creation of a chatbot or conversational agent.

The research conducted aimed to simulate a study conducted by an individual with the intention of implementing a chatbot with similar goals to those proposed in this document. The research resulted in the exploration of the platforms used for the implementation of chatbots whose aim is to assist introductory programming students (presented in the following chapter) and the cross-referencing of articles that collected the most widely used platforms.

Research revealed the existence of solution for every purpose. Certain

[19]http://www.technavio.com/report/chatbot-market-industry-analysis&nowebp

platforms permit greater customization by involving the client in the process, whereas others provide their services and/or tools for the client to develop their project.

At the time of writing, Dialogflow by Google, Amazon Lex, Watson Assistant by IBM, and Rasa have been identified as the most popular platforms [15] for the development of chatbots, despite the fact that hundreds of such platforms exist.

Therefore, these services merit a more comprehensive analysis and comparison of the technologies. Therefore, a brief description of each one's characteristics is provided, followed by the rationale for selecting Rasa.

Dialogflow Dialogflow is a Google-developed NLU platform available on Google Cloud. It enables the user to create a chatbot that can be later integrated with other services, including mobile apps and web applications.

It offers two distinct types of services, each with its own agent type, user interface, API, client libraries, and documentation: the Dialogflow CX, which is suitable for more complex agents, and the Dialogflow ES, which is suitable for agents with fewer capabilities.

Both versions utilize machine learning to predict and eliminate intents and entities based on user-supplied training phrases. It contains interfaces designed to simplify the process of creating a chatbot for everyone, including those with limited professional experience.

Amazon Lex Amazon Lex is an AWS service for building voice or text interfaces in applications that use the deep learning engine that supports Amazon Alexa. It offers speech recognition, speech-to-text conversion, and language comprehension, among other features. among others, natural, slot-filling, and intent chaining. Permits development via the console or REST APIs supported by AWS services, enabling scalability of the developed solution.

IBM Watson Assistant IBM Watson Assistant is a platform that enables the creation of artificial intelligence assistants and applications utilizing natural language understanding that enable the user to respond in a manner resembling human interaction.

As with any IBM solution, it was designed for simple integration with the entire IBM ecosystem, enabling rapid scaling of the solutions created.

Rasa RASA is an open-source framework that automates text and/or voice-based assistants using open-source machine learning models and libraries. Since these libraries are accessible to the user, the platform can be used transparently, giving the user complete control over all aspects and processes of their chatbot.

Reasons for selecting a platform can vary depending on the characteristics considered. In analyzing the platforms, the characteristics of the chatbot were considered, which led to the selection of Rasa.

Amazon's Lex platform was initially disregarded for two reasons. The first, and main one, was the lack of Portuguese support, which is of utmost importance given that one of the primary characteristics of Pyo was communication in Portuguese. The second issue was the documentation, as compared to the other three resources, which were very intuitive and straightforward, its documentation was written as if the reader required prior knowledge of chatbots, potentially leading to an higher learning curve.

Rasa's open-source nature was nevertheless the deciding factor in its selection. Rasa, unlike the other two, is not implemented via a provider-held interface; rather, the developer works with code. This may require a steep learning curve for those without prior programming experience, but it grants the user complete control over all chatbot processes, facilitates the connection to other channels and APIs, and enables the user to modify and customize the chatbot's algorithms, setting it apart from competitive solutions.

2.4.6 Chatbots in Education

Regarding the education domain it is common to see conversation agents with a clear service focus assisting on FAQ, for instance, the University of Murcia in Spain, has implemented Lola with the purpose of reducing the volume of student enrolment-related questions during enrolment periods [Muñoz, 2018]. Similarly, Dinus University in Indonesia developed a virtual agent with the purpose of answering admission related questions. However, there is a significant increase in the number of chatbots whose aim focuses on the learning aspect of a specific topic. Used for both informal and formal education, these conversational agents serve the purpose of interacting and helping the students as a human educator would [Pérez et al., 2020].

Researchers have discovered that engaging with chatbots increases students' interest in learning, which has further prompted the creation of educational chatbots [Johnson, 2001]. Recent examples whose objective is knowledge acquisition include the work of Shorey et al. [Shorey et al., 2019], where a virtual agent was created to imitate a patient, in order to teach nursing students how to speak more effectively, and ChatBot [Chen et al., 2020] a conversational agent developed with the purpose of teaching Chinese.

The field of introductory programming was one of the many taken upon by researchers, and as to form a conscience on the chatbots already developed, several publications were analyzed with a specific attention on the programming language of focus, the platform used to develop the bot, the approach taken, the functionalities implemented, and the details of the evaluation.

Language Focus Analysis revealed that a big percentage of contributions had their focus on the JAVA programming language [Finch et al., 2020; Hobert, 2019; Ismail and Ade-Ibijola, 2019; Müller et al., 2018]. The widespread use of this language in universities may account for this [Simon et al., 2018]. Despite the apparent preponderance of contributions pertaining to JAVA, other authors have extended the field to other programming languages, such as MATLAB [Verleger and Pembridge, 2018] and Python [Chinedu and Ade-Ibijola, 2021], or programming in general, without specifying the language [Ismail and Ade-Ibijola, 2019].

Approach Due to the specialized nature of the domain, rule-based approaches were universally adopted as the method of implementation. But rather than approaches, the selection of platforms appeared to be the primary determinant of the implementation method. IBM Watson Assistant [Ismail and Ade-Ibijola, 2019; Müller et al., 2018], Microsoft QnA Maker [Verleger and Pembridge, 2018], Chatscript [Finch et al., 2020], and SnatchBot [Chinedu and Ade-Ibijola, 2021] were among the platforms utilized. Other implementations did not use any framework and implemented their chatbot using AIML [Daud, 2020].

Functionalities Regarding the types of questions, all of the conversational agents implemented were programmed to answer to questions related to the introductory concepts of each language [Chinedu and Ade-Ibijola, 2021; Finch et al., 2020; Hobert, 2019; Ismail and Ade-Ibijola, 2019; Müller et al., 2018; Verleger and Pembridge, 2018]. Only the e-java chatbot [Daud, 2020] had a more narrow domain focusing solely on the control structures, such as if-statements and loops. Apart from answering concept related questions, Ismail and Ade-Ibijola [2019] also programmed their conversational agent to generate examples and advise students facing depression. One other common option was the functionality of emailing a professor when an answer could not be reached and then updating the database accordingly [Chinedu and Ade-Ibijola, 2021; Finch et al., 2020; Verleger and Pembridge, 2018].

Evaluation The evaluation procedures mainly consisted of two parts: a quantitative and qualitative examination of the interactions, as well as a student perception questionnaire. Some researchers used only one, while others used both. Regarding the quantitative aspect, developers intended to evaluate variables such as the number of times the chat window was opened and the number of distinct students who interacted with the chatbot, as to understand the evolution of its usage [Finch et al., 2020; Verleger and Pembridge, 2018]. On the qualitative side, in order to determine the accuracy of responses, researchers paid close attention to the questions asked and the chatbot's responses [Finch et al., 2020; Müller et al., 2018]. Another method of evaluation was a questionnaire to determine students' perceptions of the chatbot's user-friendliness, accuracy, usefulness, and suggestions for further enhancements [Chinedu and Ade-Ibijola, 2021; Hobert, 2019; Ismail and Ade-Ibijola, 2019; Müller et al., 2018].

The results appeared mixed. On the one hand, novices considered the chatbots user-friendly and valuable in supporting their programming learning journey [Chinedu and Ade-Ibijola, 2021; Ismail and Ade-Ibijola, 2019; Müller et al., 2018]. On the other hand, some of the agents displayed a lack of capability in answering the majority of the student's answers [Finch et al., 2020; Verleger and Pembridge, 2018], and a few even compared the agent to other resources such as Google, which they deemed an easier way to retrieve the answers [Verleger and Pembridge, 2018].

As already mentioned, the search for relevant literature on chatbots in the field of programming education did not yield an abundance of results. With Pyo, the aim is to make a valuable, well-implemented, and interesting contribution to this domain, given that it was only possible to find one other chatbot that focused on the same programming language as Pyo's (Python). Novelty also comes from the use of Portuguese as the interaction language, and a proactivity aspect, as it was revealed that it was always the student who requested assistance, and never the agent offering help and making its presence known when in need.

2.5 Information Retrieval Based Question Answering

Included in the features of Pyo is the ability to answer questions, which, similar to what was described in section 2.4.3, can be implemented through predefined rules, where there is a predefined answer for a given intent, or through the corpus-based approach employing machine learning models. The route of the rules was followed, but the latter was used initially. As it was developed and can be a characteristic that further enhances the chatbot's capabilities, the method for doing so is explained in this section.

Jurafsky and Martin [2021] present a question answering model (figure 2.7) based on IR that, as we have previously touched on, can be used to acquire similar information in regards to the query of a user. The model comprises two stages: the first stage, regarding the input of a user, returns relevant documents from a collection (IR). In a second stage, possible spans, likely to clarify the question of the user, are extracted from the relevant documents utilising a neural reading comprehension algorithm (Extractive Question Answering (QA)).

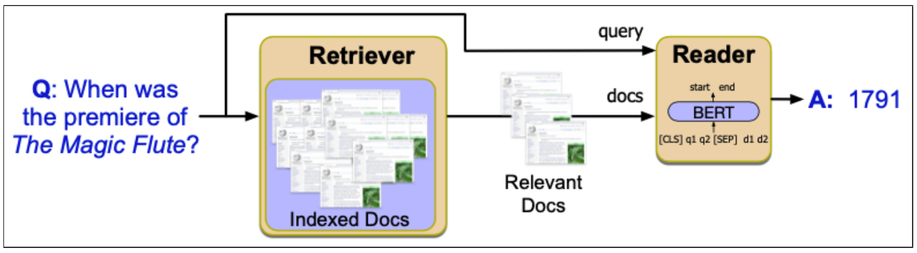


Figure 2.7: IR-based QA model from Jurafsky and Martin [2021]

IR is defined by Jurafsky and Martin [2021] as "the field encompassing the retrieval of all manner of media based on user information needs."

The specific IR task displayed in the model is called ad-hoc retrieval. It takes the query of a user and, based on relevance, returns an ordered set of documents from a given collection [Guo et al., 2016].

The documents with a higher relevance are then passed onto a neural algorithm, where for a given question (q) of n tokens, and a passage (p) of m tokens, computes the probability of each span being the answer, $P(a|q,p) = P_{start}(a_s|q,p)P_{end}(a_e|q,p)$, where a_s is the start of the span and a_e is its end. Thus, for each token p_i in the passage, the probabilities of that given token being the start and the answer's end have to be computed.

Such a standard architecture is represented in figure 2.8. Based on the architecture of transformers described in section 2.6, the figure depicts a BERT encoder [Devlin et al., 2019] that, given a query and a passage, returns an encoding token embedding for each token p_i in the passage.

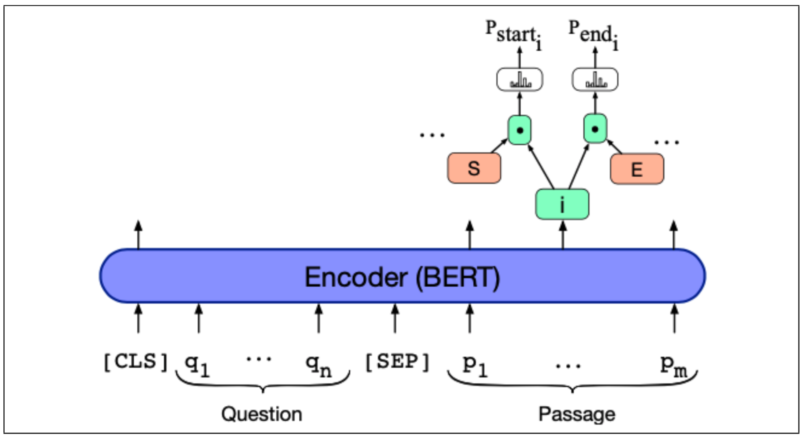


Figure 2.8: Encoder Model using Bert for question answering from Jurafsky and Martin [2021]

A linear layer is added and trained upon the fine-tuning phase to predict the start and end positions of the span containing the answer. Additionally, two embeddings are added and learned in the fine-tuning phase, a span-start embedding (S) and a span-end embedding (E). The span-start (equation 2.1) and span-end (equation 2.2) probabilities, of each token p_i returned by BERT, are then given by the dot product between S and p_i, and E and p_i, respectively normalized by a softmax on all the tokens in the passage [Jurafsky and Martin, 2021].

$$P_{start_i} = \frac{exp(S \cdot p_i')}{\sum_j exp(S \cdot p_j)} \tag{2.1}$$

$$P_{end_i} = \frac{exp(E \cdot p_i)}{\sum_j exp(E \cdot p_j')} \tag{2.2}$$

Finally, the score of a span going from position i to position j is given by $S \cdot p_i + E \cdot p_j$. The model prediction choice is the span with the highest score, attending that j is higher than i.

2.6 Transformers

In section 2.4.3, we have seen that language models can be used for generative approaches when generating the responses of a chatbot, and, in section 2.5, for extracting answers from a set of relevant documents, or similarly to generate questions. Two of the most used models (GPT-2 and BERT) are based on transformers. This architecture is also responsible for many of the recent advances of NLP. Therefore, we deemed valuable an explanation of what transformers are, which we present in this section.

Vaswani et al. [2017a] first introduced the transformer neural network architecture in their paper "Attention Is All You Need". This network operates using encoder-decoder architectures likewise to recurrent neural networks with the difference that input sequences can be processed in parallel.

The encoder and decoder are composed of modules stacked on top of each other multiple times, as described by the **Nx** in figure 2.9.

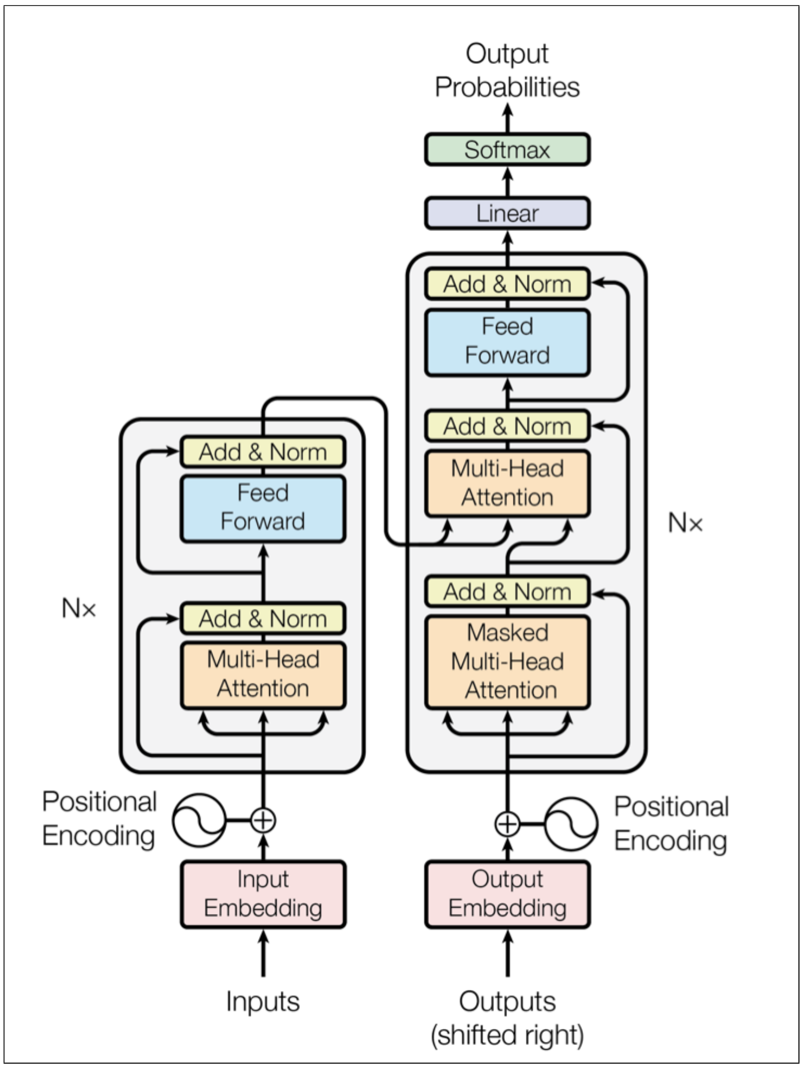


Figure 2.9: The transformer model architecture from Vaswani et al. [2017b]

Each encoder is broken down into two sub-layers, the self-attention layer, and the Feed Forward Neural Network layer.

The performance improvements transformers offer concentrate on the self-attention mechanism. As the model processes each word, self-attention allows it to look at other positions in the input sequence for clues that can help lead to a better encoding of the word. Taking as an example a phrase we would like to translate "John didn't feel like working out as he was too tired", self-attention is what allows the model to associate "he" with "John". The Feed Forward Neural Network applied to all the attention vectors is used in practice to transform the attention vectors into a form that is digestible by the next encoder/decoder block.

The decoder similarly contains both these layers, but with an extra attention layer in between them that helps the decoder focus on relevant parts of the input sentence.

Transformers adopt a form of semi-supervised learning, meaning that they are pre-trained in an unsupervised manner with a large, unlabeled dataset and then they are fine-tuned through supervised training for a downstream task, such as QA.

Based on this Transformer architecture, there are two well-known language models: the Generative Pre-trained Transformer (GPT) (1, 2, and 3) [Brown et al., 2020; Radford et al., 2019] and the Bidirectional Encoder Representations from Transformers (BERT) [Devlin et al., 2019].

Both models are pre-trained using unlabeled data and can be fine-tuned for different NLP tasks. GPT is comprised solely of stacked decoder blocks from the transformer architecture, whereas BERT is comprised only of encoder blocks. However, the significant difference, which Devlin et al. [2019] consider "harmful when applying fine-tuning based approaches to token-level tasks such as question answering, where it is crucial to incorporate context from both directions", is that GPT is unidirectional, whilst BERT is a bidirectional encoder model.

Chapter 3

Chatbot Specifications

Using the acquired knowledge, a chatbot that assists introductory programming students was developed. It uses Natural Language Processing (NLP) to provide additional assistance quickly and on demand, enabling individualized support that may not always be possible due to the large number of students, a difficulty exacerbated by the online regime, or the “learn at your own pace” nature of the course, which can mean the professor may not be available when the student is completing the exercises.

This chapter describes the primary functionalities (section 3.1) of the implemented system and the proposed architectures (section 3.2) throughout its implementation process, thus providing an overview so that implementation specifics can be introduced in the following chapter (Chapter 4).

3.1 Functionalities

Recalling the main objective, Pyo was designed to assist introductory programming students, particularly when completing exercises requiring the creation of an algorithm. Towards this goal specialized functionalities were developed as a means to answer the question “How does one provide assistance to introductory programming students?”.

When stratergizing Pyo’s possible methods of assistance, there were three primary factors considered that could prevent a student from successfully completing an exercise:

- Inability to recall what and how the various introductory concepts are implemented (e.g., variables, conditions, for loops).
- Incapacity of understanding and correcting errors presented in their algorithms.
- Lack of understanding of what is necessary to complete an exercise.

For each of this three points the functionalities referred to as **concept definitions**, **error guidance**, and **exercise assistance**, were developed. In addition, at a later stage of implementation, two secondary features were added: **question suggestions** as a means of suggesting questions related to the concepts embedded in the exercise, and **hints** for additional assistance with the exercises. Important to note that only the first feature was available for all question types, whereas the others were only available for exercises requiring the implementation of a program, as they required information about the algorithm corresponding to the exercise's answer.

3.1.1 Concept Definitions

As stated in the background (section 2.1), one of the causes of difficulty is the rise in cognitive load as tasks become more complex. This can be a hindrance for novice programmers who lack a solid understanding of the concepts and hence easily forget how to apply them. The concept definitions feature consists simply of asking Pyo a question about a particular concept and receiving the response. An example can be seen in figure 3.1 The response begins with an explanation (figure 3.1a), followed by an illustration (figure 3.1b), and, if predefined, a list of recommendations pertaining to the concept in question (figure 3.1c).

3.1.2 Error Guidance

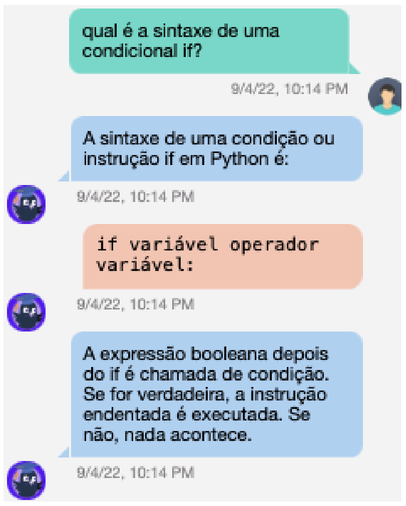
When beginning the process of learning a subject, it is normal to make several mistakes; programming is no different. Beginner programmers typically produce a large number of flawed algorithms at first, and if their conceptual understanding is not solid, moving on to more complex tasks that require them to produce more complex solutions, can lead to frustration and a lack of motivation.

Typically, a programming editor contains error messages with relevant error-related information, which novice programmers may find difficult to interpret. Not only that, but the primary focus of Pyo is assisting students who speak Portuguese, and those messages are typically sent in English. This issue is taken into account by the RESPE platform, which already generates error messages that are more easily comprehended and in Portuguese.

Despite the fact that the messages are presented in a more user-friendly format, some students still find it difficult to understand them. With these students in mind, the "error guidance" functionality was developed.

By error guidance, it is meant that when a student submits code containing syntax errors, Pyo suggests assistance in the form of multiple-choice questions. Figure 3.3 depicts an example of a dialogue pertaining to this functionality.

Importantly, both this feature and the exercise assistance feature are initially triggered. In the context of error guidance, Pyo is triggered when a student submits code containing an error; it then sends the student the message shown in figure 3.2. The use of the functionality starts, however only if the student agrees



qual é a sintaxe de uma condicional if?

9/4/22, 10:14 PM

A sintaxe de uma condição ou instrução if em Python é:

9/4/22, 10:14 PM

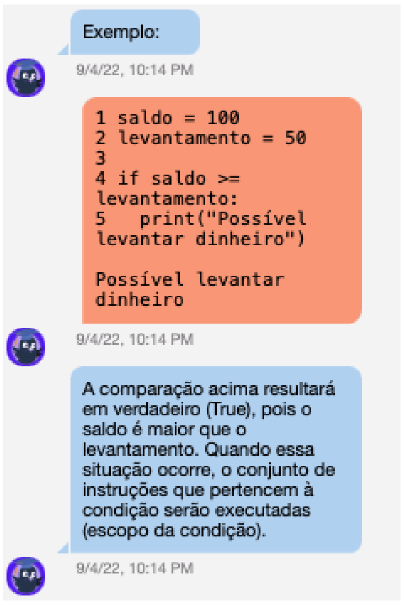
```
if variável operador
variável:
```

9/4/22, 10:14 PM

A expressão booleana depois do if é chamada de condição. Se for verdadeira, a instrução endentada é executada. Se não, nada acontece.

9/4/22, 10:14 PM

(a)



Exemplo:

9/4/22, 10:14 PM

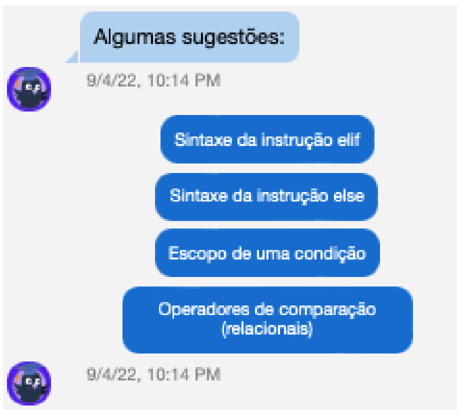
```
saldo = 100
levantamento = 50

if saldo >=
levantamento:
   print("Possível
levantar dinheiro")

Possível levantar
dinheiro
```

9/4/22, 10:14 PM

A comparação acima resultará em verdadeiro (True), pois o saldo é maior que o levantamento. Quando essa situação ocorre, o conjunto de instruções que pertencem à condição serão executadas (escopo da condição).

9/4/22, 10:14 PM

(b)



Algumas sugestões:

9/4/22, 10:14 PM

Sintaxe da instrução elif
Sintaxe da instrução else
Escopo de uma condição
Operadores de comparação (relacionais)

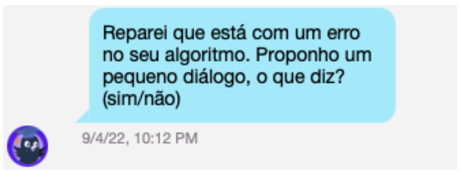
9/4/22, 10:14 PM

(c)

Figure 3.1: Example of the concept definitions functionality.

in participating in a dialog (figure 3.3a). Beginning the dialogue, the student is prompted with a question on the same topic as the error they have committed, after which they are presented with different options (figure 3.3b). The number of questions is limited to a maximum of two so as not to loose the students' attention. At the conclusion of the question/s, an explanation with an emphasis on the error's specifics is provided.

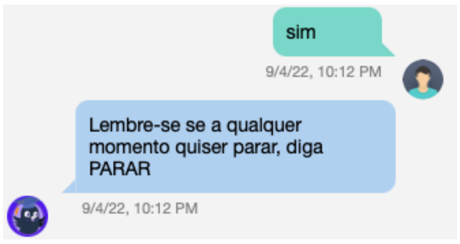
In addition to providing an example of the error and an explanation of any details the student may not be aware of, the questions and options are designed to improve the student's ability to analyze their own code.



Reparei que está com um erro no seu algoritmo. Proponho um pequeno diálogo, o que diz? (sim/não)

9/4/22, 10:12 PM

Figure 3.2: Message suggesting error guidance.



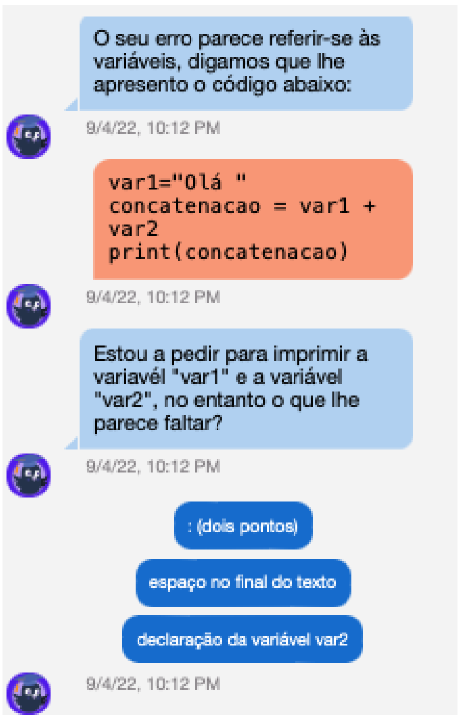
sim

9/4/22, 10:12 PM

Lembre-se se a qualquer momento quiser parar, diga PARAR

9/4/22, 10:12 PM

(a)



O seu erro parece referir-se às variáveis, digamos que lhe apresento o código abaixo:

9/4/22, 10:12 PM

```
var1="Olá "
concatenacao = var1 +
var2
print(concatenacao)
```

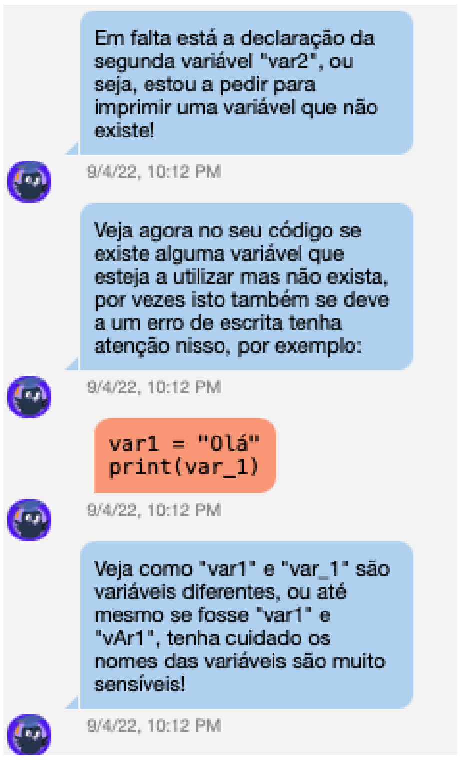
9/4/22, 10:12 PM

Estou a pedir para imprimir a variavél "var1" e a variável "var2", no entanto o que lhe parece faltar?

9/4/22, 10:12 PM

9/4/22, 10:12 PM

(b)



Em falta está a declaração da segunda variável "var2", ou seja, estou a pedir para imprimir uma variável que não existe!

9/4/22, 10:12 PM

Veja agora no seu código se existe alguma variável que esteja a utilizar mas não exista, por vezes isto também se deve a um erro de escrita tenha atenção nisso, por exemplo:

9/4/22, 10:12 PM

```
var1 = "Olá"
print(var_1)
```

9/4/22, 10:12 PM

Veja como "var1" e "var_1" são variáveis diferentes, ou até mesmo se fosse "var1" e "vAr1", tenha cuidado os nomes das variáveis são muito sensíveis!

9/4/22, 10:12 PM

(c)

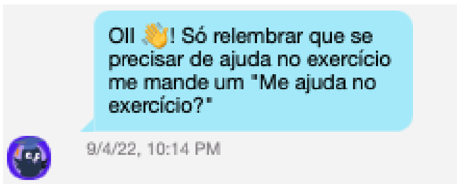
Figure 3.3: Example of the error guidance functionality.

3.1.3 Exercise Assistance

The third main feature is named "exercise assistance". It consists of prompting the student to arrange, in natural language, the components of a possible answer's skeleton. This assistance is intended to provide the student with an understanding of the concepts involved in the exercise without revealing the exact Python syntax for implementing them.

As mentioned in the explanation of the error guidance feature, this functionality is also triggered. This trigger is sent 2 minutes after the student entered an exercise involving the development of an algorithm, to which it then sends the student the message displayed on figure 3.4. This requirement that a minimum of two minutes must have passed before a student may request assistance on an exercise is intended to prompt the student to first think and analyse the exercise.

When the student requests assistance, they are prompt to organize a scrambled list of the constituent parts of a potential answer. In the example shown in

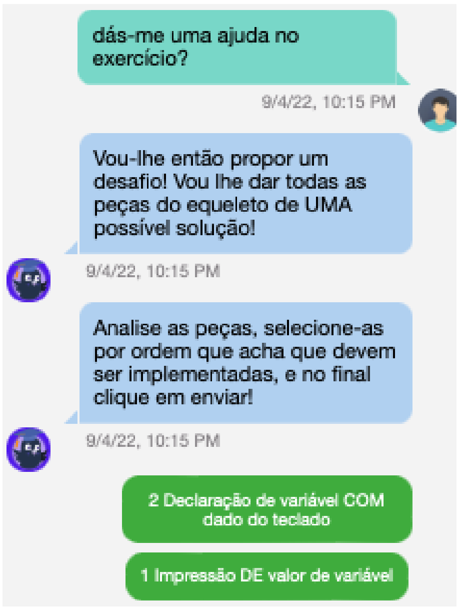


Oll 👋! Só relembrar que se precisar de ajuda no exercício me mande um "Me ajuda no exercício?"

9/4/22, 10:14 PM

Figure 3.4: Message suggesting exercise assistance.

figure 3.5a, the green buttons at the bottom of the image represent the answer's components, and the numbers they contain indicate their selection order. Once the student submits their response, the chatbot provides feedback and the correct skeleton (figure 3.5b).



dás-me uma ajuda no exercício?

9/4/22, 10:15 PM

Vou-lhe então propor um desafio! Vou lhe dar todas as peças do equeleto de UMA possível solução!

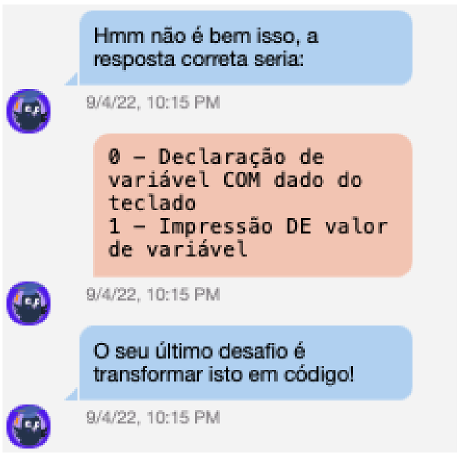
9/4/22, 10:15 PM

Analise as peças, selecione-as por ordem que acha que devem ser implementadas, e no final clique em enviar!

9/4/22, 10:15 PM

2 Declaração de variável COM dado do teclado
1 Impressão DE valor de variável

(a)



Hmm não é bem isso, a resposta correta seria:

9/4/22, 10:15 PM

```
0 - Declaração de
variável COM dado do
teclado
1 - Impressão DE valor
de variável
```

9/4/22, 10:15 PM

O seu último desafio é transformar isto em código!

9/4/22, 10:15 PM

(b)

Figure 3.5: Example of the exercise assistance functionality.

3.1.4 Question Suggestions and Hints

Question suggestions and **hints** are more accurately described as secondary features, as they do not serve a primary function. Regarding the suggestion of questions, when prompt to, as displayed on figure 3.6, Pyo returns a list of questions pertaining to the implementation of all the concepts involved in that particular exercise (e.g., "How to conduct concatenation?" and "How to declare a for loop?").

The second mentioned functionality refers to the return of predefined small hints regarding previously solved similar exercises and small commonly forgotten details observed in previous student submissions of previous course of the platform (figure 3.7).

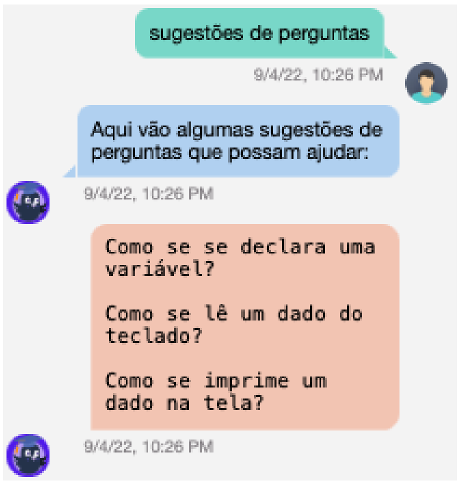

sugestões de perguntas
9/4/22, 10:26 PM
Aqui vão algumas sugestões de perguntas que possam ajudar:
9/4/22, 10:26 PM
```
Como se se declara uma
variável?

Como se lê um dado do
teclado?

Como se imprime um
dado na tela?
```
9/4/22, 10:26 PM

Figure 3.6

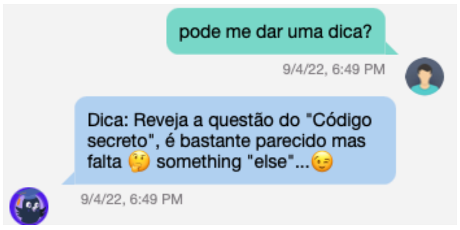

pode me dar uma dica?
9/4/22, 6:49 PM
Dica: Reveja a questão do "Código secreto", é bastante parecido mas falta 🤔 something "else"...😉
9/4/22, 6:49 PM

Figure 3.7

3.2 Chatbot Architecture

This section presents two diagrams as to understand the various components of Pyo, how they connect, and the differences between the initial (figure 3.8) and final approaches (figure 3.9). The tools and resources (documentation) mentioned in this section will be further explained on section 4.1.

The architectures have two common aspects:

- The Natural Language Understanding (NLU) and dialogue management processes, which are responsible for extracting pertinent information from the student's message and predicting the next action, respectively.The Rasa platform was used to manage and implement both of these procedures.
- The block of the **Exercise Assistance** functionality, in which the abstract syntax tree of the answer to the exercise, provided by the platform, is examined with the intent of translating each algorithmic component into natural language and returning the list to the student. This was achieved using the Python library AST.

The differences, however, lie in the initially projected approach of the concept definitions and error guidance versus the final implementation, as well as the addition of two secondary features.

Regarding the initial architecture (figure 3.8):

- The **Concept Definitions** functionality was initially implemented with the intention of automating the retrieval of the answer to a student's question from a collection of documents using information retrieval-based question answering. For information retrieval, the Python library Whoosh was used, while a Portuguese Question Answering (QA) language model was employed for question answering.
- The **Error Guidance** was originally idealized to use question generation, using a Portuguese Question Generation (QG) language model on a set of documents, in an effort to automate this process.

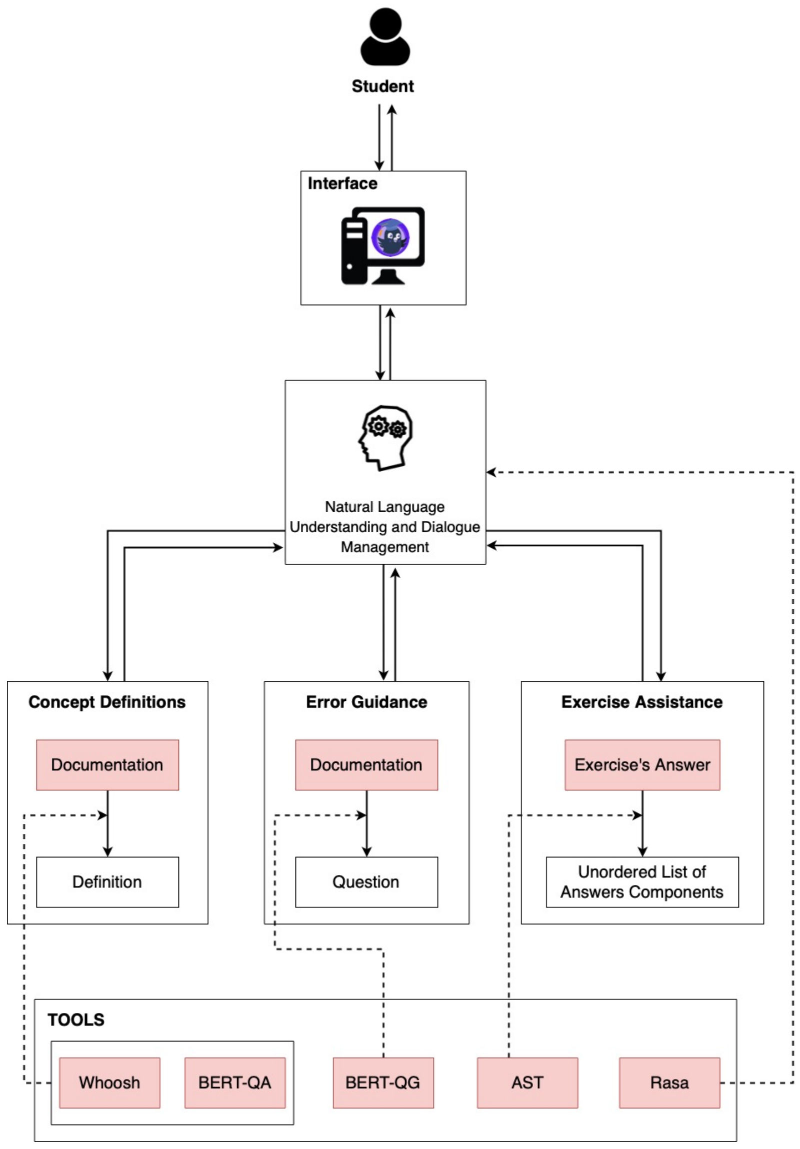


Figure 3.8: Initial proposed architecture overview diagram for the implementation of Pyo.

As will be explained in the implementation chapter, the final strategy took a different route. In regards to the final architecture (figure 3.9):

- Both **Concept Definitions** and **Error Guidance** were developed using a more hard-coded method, in which each has its own collection of documents and the definition and set of questions are pre-defined based on the intent or error information.

- Similarly to the **Exercise Assistance** feature, the **Question Suggestion** functionality performs an analysis of the syntax tree of the exercise's answer. The concepts pertinent to the solution are extracted during analysis and used to form questions in natural language.
- Given the exercise's id (sent by the RESPE platform), the **Hints** functionality sends the student a predefined hint.

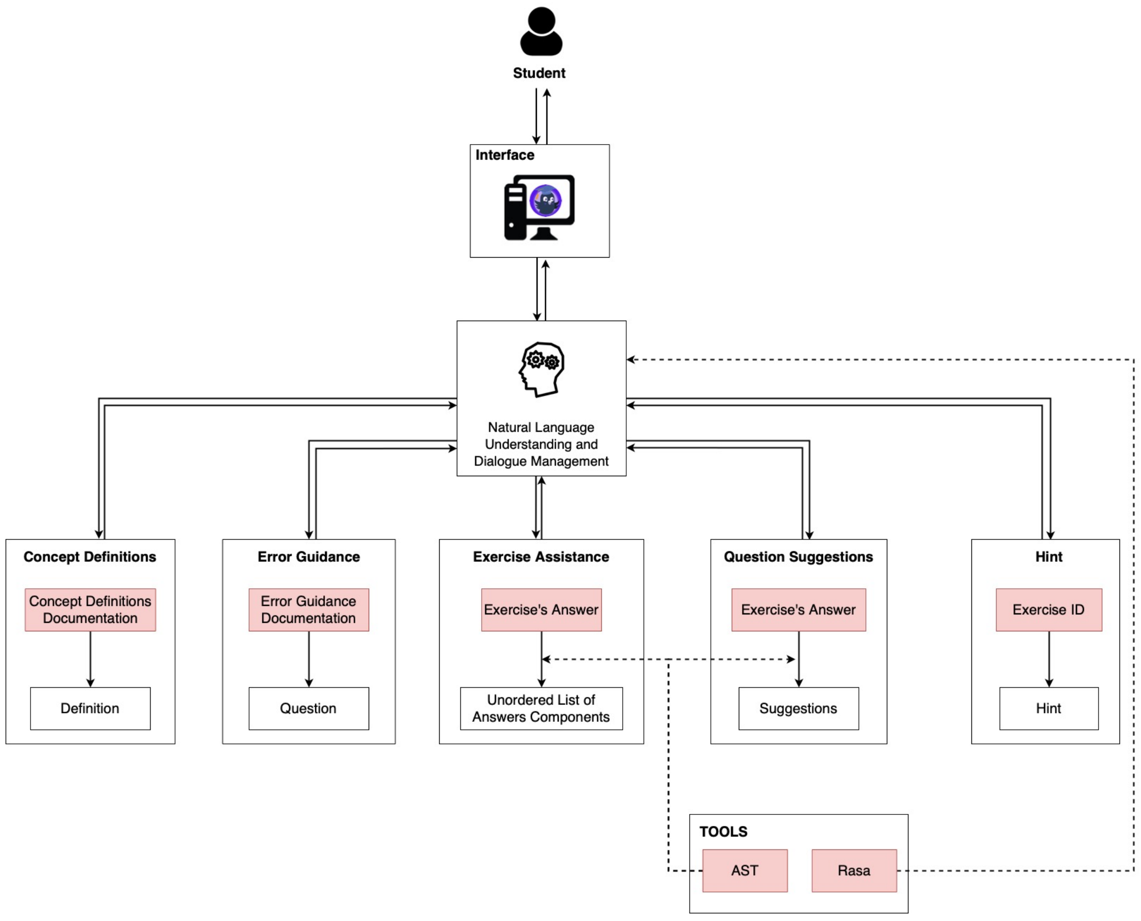


Figure 3.9: Final proposed architecture overview diagram for the implementation of Pyo.

Chapter 4

Implementation

This chapter is intended to introduce the reader to the development strategy according to the functionalities and architecture proposed on the previous chapter. The following sections will describe the resources and tools used throughout the implementation of Pyo (section 4.1), the approaches used to implement the functionalities (section 4.2), the pipeline employed (section 4.3), and the front-end development (section 4.4).

4.1 Resources and Tools

This section discusses the tools and resources used to implement Pyo. Beginning with an explanation of the framework's primary characteristics, and concluding with the language model and Python libraries used throughout the implementation of the agent.

4.1.1 Rasa

Rasa was selected as the framework for a number of reasons, including its open-source nature, the potential for rapid development of context assistants, and its high degree of customizability. As it is a fundamental aspect of the work developed, the explanation of its underlying general concepts and mechanisms is provided in this section. Figure 4.1 demonstrates how its components connect with each other.

General Concepts

In regards to this framework there are general concept that will be mentioned further ahead, which are intents, entities, slots, forms, custom actions, responses, stories, and rules.

Intents When developing a chatbot, it is necessary to specify all the possible in-

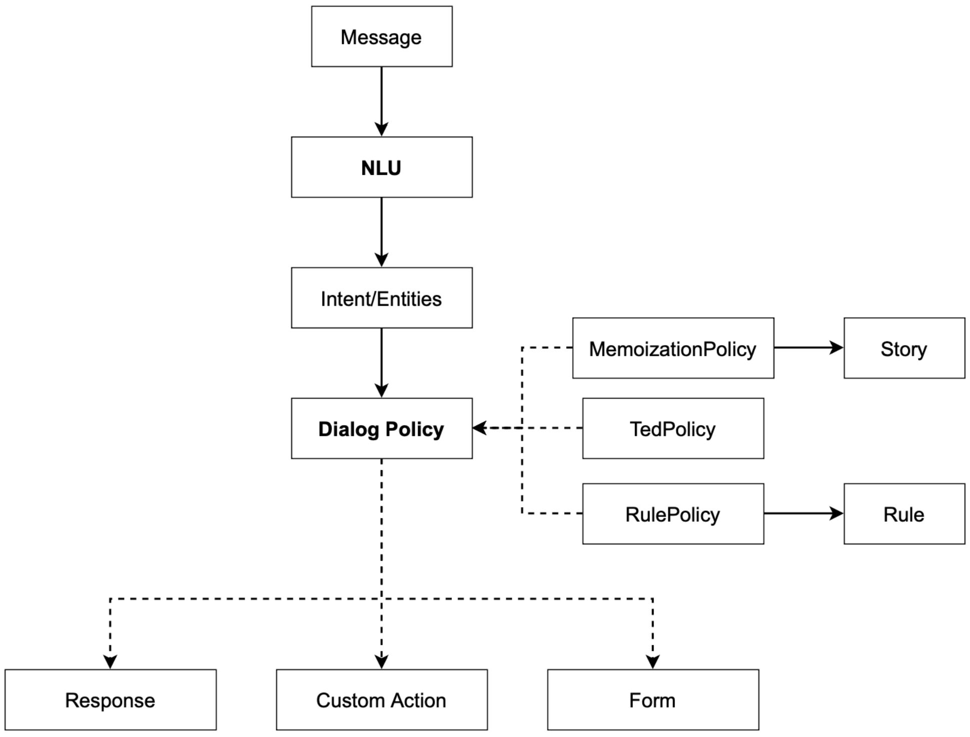


Figure 4.1: Rasa components diagram.

tentions of a user's message and provide training examples for them. Regarding Pyo, examples of intentions include greeting, saying goodbye, requesting the explanation of a conditional, and requesting assistance. An extensive list comprising all of the intents defined can be found in Apendix A as well as their descriptions. Figure 4.2 depicts an example of training data for an intent regarding the concatenation concept.

```
- intent:  ask_question_var_strings_concat
  examples: |
  - como concatenar
  - o que é a concatenção
  - me dá um exemplo de concatenação
  - como se juntam dois textos
  - que operador se usa para uma concatenação
  - como junto duas frases
```

Figure 4.2: Example of training data for an intent.

Entities Entities are structured pieces of information that a chatbot can extract from the users' input. Such information can help the assistant comprehending the question and utilizing the requested information in a particular setting. Figure 4.3 presents a short example of an *ask_definition* intent whose training examples contain the entity *wanted_concept*.

```
- intent:  ask_definition
  examples:  |
  - o que é uma [variavel](wanted_(concept)?
  - o que significa [concatenacao](wanted_(concept)
  - me diz como declarar uma [função](wanted_(concept)?
  - ...
```

Figure 4.3: Example of how the entities are inserted into the training data.

The entity would allow the agent to know what concept the user is referring to without needing to specify all possible concepts, provided that the training data is sufficient for the chatbot to identify it correctly.The initial method necessitated the identification of the entity depicted in the image, whereas the final method did not require the extraction of any entity.

Slots In Rasa, slots represent long-term conversational memory. They store information that the developer may need in the future. They can be used to control the flow of a conversation, or accessed by a custom action.

Slots differ from entities in that they store persistent information and can be used to store data even if no entity has been detected, such as by a custom action or when a specific intent is extracted. Despite this, assigning an entity value to a slot is extremely common.

This component is not represented in figure 4.1 as it is not linked to any other component specifically.

Forms In a variety of situations, it is advantageous to collect information before acting on behalf of a user. For instance, when we ask a student a question for the functionality of "error guidance", Rasa must know that the next input is their response and, unless otherwise specified, ignore the extracted intent. We accomplish this by activating a form that deactivates once a student submits an answer.

Custom Actions Rasa offers a feature referred to as "Custom Action" that triggers a custom Python function instead of a response text.

Using the scenario in which a student requests the definition of a concept, as an illustration, suppose we had the concept but needed to retrieve the answer from a database. This would require more than simply returning a response. Therefore code needs to be executed on behalf of the users, hence the need for custom actions.

Responses Responses are the messages we want the chatbot to send to the user in response to a particular intent. In figure 4.4, the response *utter_goodbye* contains two options that are selected at random when used. Worthy to note that the name convention of the responses is *utter_name*.

Stories Stories serve as training data to instruct the assistant on its next steps. They are examples of conversational flows that will be taught to the chatbot

```
responses:
 utter_goodbye:
     -text:"Goodbye :("
     -text:"Sad to see you go :("
```

Figure 4.4: Example of responses.

so that it can learn and apply them to unseen flows. Figure 4.5 depicts an example of a conversation's flow.

```
stories:
    - story:definition of variable
       steps:
       - intent:ask_variable_definition
       - action:utter_variable_definition
       - action:utter_anything_else
       - or:
              - intent:deny
              - intent:goodbye
       - action:utter_goodbye
```

Figure 4.5: Example of a story.

Rules Rules constitute another type of training data used to train the assistant's dialogue management model. Rules provide a means of describing short conversations that should always proceed in the same manner.

The primary distinction between a rule and a story is that a story can serve as a learning example, whereas a rule is a pattern the assistant must follow.

Figure 4.6 presents an example of a simple rule, where given the intent *greet*, the chatbot must always answer with a *utter_greet* response.

```
rules:
    - rule:Greeting Rule
       steps:
       -intent:greet
       - action:utter_greet
```

Figure 4.6: Example of a rule.

NLU and Dialog Policies

The conversational agent is generally driven by two mechanisms contained within Rasa. The mechanisms of Natural Language Understanding (NLU) and Dialog Policies.

NLU In the context of Rasa, when talking about NLU, we typically referring to the portion of the system that accepts raw text and outputs machine-readable information. This typically denotes that the component accepts text and converts it into intents and entities.

This can be rule-based, wherein we might employ a Regex or a neural network. Rasa includes a neural network architecture called DIET, which is a multitasking architecture for classifying intent and recognizing entities based on the examples provided [Bunk et al., 2020]. The architecture is based on a transformer that is shared for both tasks.

Rule-based approaches are typically more lightweight and necessitate greater domain expertise. Neural approaches typically require more training data and processing power, but they are better at handling novel situations.

Dialog Policies The component of the system that predicts the next course of action is referred to as the dialogue policy. Sometimes, in order to determine the next action, it is necessary to be aware of the entire conversation thus far.

Again, policies may be based on rules or neural methods. Rasa enables the definition of lightweight rules for determining what must occur. Rasa also provides a neural network called TED that selects the next best turn based on the current conversation and all the conversations it was trained on.

It contains three different dialog policies:

- The **RulePolicy** manages interactions that correspond to predefined rule patterns.
- The **MemoizationPolicy** examines the current conversation to determine if it matches any of the scenarios in the training data. If so, it will predict the subsequent move of the matching story.
- The **TEDPolicy** predicts the next-best action using machine learning.

These policies predict the next action in parallel, and the policy with the highest degree of certainty determines the subsequent action. When two policies have equal confidence, Rasa has a mechanism for prioritizing the decision. Policy priority determines how an assistant makes decisions when multiple policies accurately predict the next action.

Priority by default in Rasa is:

- 6 for the RulePolicy
- 3 for the MemoizationPolicy
- 1 for the TEDPolicy

RulePolicy and MemoizationPolicy have a higher priority due to the fact that they are directly based on designer input. The added rules and stories serve as examples of how conversations should proceed. The TEDPolicy is a machine-learning-based policy that will attempt to generalize so that it can handle conversations not defined in the training data.

4.1.2 Documentation

Throughout the explanation of the functionalities, the term documentation appears multiple times; therefore, it is necessary to provide an explanation of what it entails. Initially, when using the term documentation, we were referring to a Word book draft written by the developer of the RESPE platform. The document contained information regarding variables, conditions, and the different types of Python errors. Given that the course also included repetitions and functions, information from a source suggested by the original documentation's author was added[20].

The existence of this documentation was what originally inspired the search for language models regarding Question Answering (QA) and Question Generation (QG).

4.1.3 BERT Language Models

From the beginning of the implementation of Pyo, it was clear that it would need to address any introductory concept questions a student might have regarding the course offered by the RESPE platform. Early on in the development of the chatbot, a BERT-QA language model was employed. Although it was not included in the final product, as we will see in the following chapter, it still represents a significant step to be considered in future work.

At the beginning of the implementation, the creator of the RESPE platform provided us with a document containing an introduction to the Python programming language capable of answering multiple questions regarding introductory concepts. With this in mind an Extractive Question Answering model, specifically a language model, was sought to automatically extract, from the document, the answer to the questions of the students.

The choice of model was made based on three main reasons: first, because it is based on a state-of-the-art architecture (BERT/transformer), second, because it is easy to use as it is available on the Hugging Face portal[21], which allows access through Python by utilizing the transformers[22] library, and third because it is dedicated exclusively to the Portuguese language.

The research on question answering models led us to the "Portuguese BERT base cased QA (Question Answering), finetuned on the Portuguese version of the

[20]`https://algoritmosempython.com.br`
[21]`http://huggingface.co`
[22]`http://huggingface.co/docs/transformers/index`

Stanford Question Answer Dataset (SQUAD) v1.1"[23] [Guillou, 2021], referred to throughout this thesis as BERT-QA. For reference, SQUAD is a dataset consisting of questions asked by crowdworkers in a set of Wikipedia articles. A Portuguese version is available and was published by the group Deep Learning Brasil[24].

This model is the result of fine-tuning the pre-trained Portuguese BERTimbau Base language model ("bert-base-portuguese-cased") on the Portuguese version of the SQUAD dataset.

Important to note that the model is applied to a particular type of QA, Extractive QA, and, as discussed in section 2.5, it extracts a potential answer from a text segment. It cannot extract the answer from lengthy documents, so the documentation had to be divided into multiple files, each based on the subject matter of the information they contained, before the method outlined in section 2.5 could be applied; hence the need for an Information Retrieval (IR) library, which we will discuss next.

Another attempted approach was in regards of the "error guidance" functionality. As we had on out hands useful documentation the thought process passed by the automation of the generation of questions using a language model. For this purpose we found the "bert2bert_shared-portuguese-question-generation model"[25], also fine-tuned on the pre-trained Portuguese BERTimbau Base language model, referred to in this dissertation as BERT-QG. Although the model was tested, its use was abandoned at the beginning of the implementation and as further discussion on the implementation of the functionalities was conducted. The documentation lacked information regarding syntax errors, and a further modification did not justify the use of the model due to the preference for multiple-choice questions. This preference was based on the student's ability to reflect on the various options and make the connection between them and their flawed code.

4.1.4 Whoosh

Whoosh[26] is a Python library that indexes and searches a collection of documents, facilitating the implementation of ad-hoc information retrieval. Its purpose was to retrieve the most pertinent document pertaining to the concept referenced by the student's query. This document would then be forwarded to the BERT-QA model discussed previously.

Throughout the creation of the index, the search based on a query, and the results themselves, it is important to keep in mind the desired end result when utilizing Whoosh. Regarding the creation of an index, the following steps are taken:

1. **Designing a schema** – The schema lists the fields of the index. A field is a

[23]`http://huggingface.co/pierreguillou/bert-base-cased-squad-v1.1-portuguese`

[24]`http://www.deeplearningbrasil.com.br`

[25]`http://huggingface.co/mrm8488/bert2bert_shared-portuguese-question-generation`

[26]`http://whoosh.readthedocs.io/en/latest/`

piece of information associated with each document in the index, such as the path or text content of the document. It is also required, when creating a Schema object, to map the field names to their respective types, which determines what is indexed and searchable. Whoosh includes several predefined field types, including ID, KEYWORD (space or comma separated keywords), TEXT, NUMERIC, BOOLEAN, and DATE. For TEXT fields, Whoosh requires an analyzer, which is a function or callable class that accepts a unicode string and returns a generator of tokens, which may include regex tokenizers, lowercase filters, stemmers, and stop lists.

2. **Creating an index and adding the documents** – The remainder of the process consists of creating an index and adding documents to said index, with each document containing the initially created fields; therefore, if the schema is composed of path and text content, we must specify the document's path and its content for each document.

After the index has been created, one can proceed to the search phase. A simple "searcher" requires the index, the field we wish to search on (for example, the document's content), and the query terms.

Whoosh enables the development of more sophisticated "searchers" with the ability to modify standard aspects and add additional features for manipulating search results. The selection of scoring algorithms is an illustration of the modification of standard features. Typically, result document lists are ordered by score. Its scoring module implements a variety of scoring algorithms, such as BM25F and TF-IDF.

Once searched a Result object is returned which acts like a list of the matched documents. It can be used to retrieve and display the stored fields of each document hit. Given the search results, Whoosh offers additional features for manipulating the search results:

- Sorting results according to the value of an indexed field rather than by relevance.
- Highlighting the search terms in excerpts from the original documents. For example, rather than the entire document, one could use a context fragmenter, which finds matched terms and then pulls in surrounding text to form fragments, or a sentence fragmenter, which attempts to fragment the text based on sentence punctuation.
- Expanding the search terms based on the most relevant documents discovered.
- Paginating the results (e.g. "Showing results 1-20, page 1 of 4").

4.1.5 AST

The ast[27] python library module allows the conversion of Python code into Abstract Syntax Tree (AST), where each node of an AST is represented by a different Python class. This library allows the analysis of code in which by iterating the tree we are able to find whether, for instance, a conditional, variable, repetition, and function is being declared. As explained ahead in greater detail, this library is essential for the implementation of the exercise assistance functionality.

4.2 Implementation of the Functionalities

The purpose of this section is to first explain the further modifications made to the collection of documents, and the implementation process of the functionalities explained on section 3.1

4.2.1 Documentation Alterations

Regarding the final implementation of "concept definitions" and "exercise guidance", two new sets of documents were compiled for each of the aforementioned functionalities. This because the path chosen was a more hard-coded approach.

Introductory Concept Definitions This collection of documents contains information on the concepts mentioned and required for resolving the exercises on the RESPE platform. The data was assembled by adapting and completing the initially provided information with details from a source suggested by the owner of RESPE.

In most instances, the explanations on each concept consists of:

- Definition of the referenced term
- Illustration and its description
- Suggestions of related questions

An example of the organization of the files containing information about variable concepts is shown below.

[27] `https://docs.python.org/3/library/ast.html`

```
ConceptDefinitions
└─ variables
    ├─ arrays.txt
    ├─ bools.txt
    ├─ floats.txt
    ├─ ints.txt
    ├─ strings.txt
    ├─ variables_general.txt
    └─ variable_types.txt
```

Error Guidance Similar to the approach taken for the feature utilizing the previously described collection of documents, a hard-coded approach was taken regarding this feature, with the questions, choices, answers, and explanations predefined.

Below is an example of the structure of files containing data we want to return regarding the assistance of an error.

```
ErrorGuidance
└─ Função
    ├─ faltaDoisPontos.txt
    ├─ faltaParentesis.txt
    └─ faltaVirgula.txt
```

Regarding the naming convention, the chosen method requires that the folder containing the files be named based on the type of error, whether it was related to variables, functions, loops, or conditionals. Furthermore, each of the files must correspond to the error's exact simplified error message generated by the platform. Both the type and message are transmitted to Pyo by RESPE.

Each error message is accompanied by one or two multiple-choice questions. Each block regarding a question is composed by:

- Question statement
- Choices
- Answer
- Explanation when the given answer was incorrect
- Final explanation with a greater emphasis on the error's cause

The final point is presented to the student only after each question has been posed. Figure 4.7 depicts an example of a file containing a question, its alternatives, the correct response, and an explanation with an emphasis on the error's cause, for an attempt to declare a variable with two equal signs.

Pergunta:O seu erro parece ser na declaração de uma variável, digamos que queriamos declara a variavel var1, com a string Olá, como o faria?

Opções:var1 = "Olá"<sep>var1: "Olá"<sep>var2 == "Olá"

Resposta:var1 = "Olá"

Checkpoint:Por vezes pode haver uma certa confusão entre as variáveis e as condições, no que toca às igualdades. Nas variaveis apenas uma igualade é usada! Olhando agora para o seu algoritmos, veja se talvez esse tenha sido o seu erro.

Figure 4.7: File containing the information regarding the syntax error of using two equal signs to declare a variable.

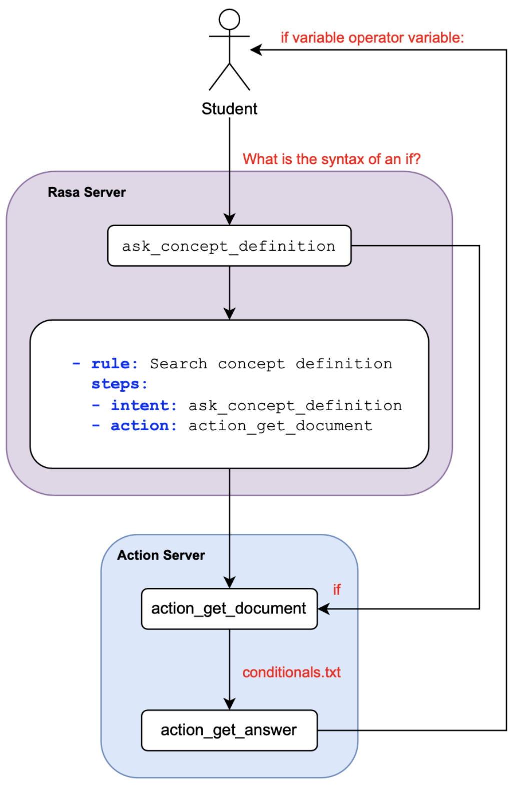
4.2.2 Concept Definitions

The concept definitions functionality was initially implemented using Information Retrieval Based Questions Answering. Although it was successfully implemented, the low level of accuracy and the modifications the documentation required to increase it did not justify its use. The final version of the implementation made use of a more hard-coded method. Both approaches are explained here, as the initial approach remains a vital component of future work despite not being chosen (section 6.5).

Initial Approach

In addition to Rasa, the initial strategy used the BERT-QA model and the Whoosh library described in the preceding section. An example of the steps the system would take to extract and return the answer are depicted in figure 4.8:

1. **Intent and entity extraction –** Given this method, it was only necessary to define one intent, *ask_concept_definition* for this feature, for which multiple training examples were provided. If asked "What is the syntax of an if?", if correct, the system would extract the mentioned intent as well as a defined entity named *wanted_concept*.



```
- rule: Search concept definition
  steps:
  - intent: ask_concept_definition
  - action: action_get_document
```

Figure 4.8: Concept definition feature's procedure (initial method).

2. **Rule mapping –** Because the interaction is straightforward, i.e., given a query about a concept, we would always want to return its definition, therefore, Rasa would apply the RulePolicy, specifically the rule "Search concept definition", which would then move to the action *action_get_document*.

3. **Information Retrieval –** The custom action *action_get_document* was responsible for IR. It would take the initially extracted entity "if" and use Whoosh to appoint the most relevant document, "conditionals.txt".

4. **Extractive Question Answering –** The preceding custom action would forward the document to a subsequent action, *action_get_answer*, which would apply the BERT-QA model using the student's query and the relevant document as parameters. Then the span with the highest probability of being the answer would be returned to the student, in the case of the example it would simply return *if variable operator variable:*.

This approach had an advantage over the second one because it automated the process of adding new information. As long as Rasa could accurately predict the intent based on its training data and the document contained information on the subject, it would not be required to identify all the possible concepts, in the form of intents, the student could ask about.

On the other hand there were a few reasons why this approach had to be put aside:

- The returned response would be the **span** of text with the highest probability percentage, but regarding novice programmers, a straightforward answer without exemplification or context would not be optimal.
- The model was **insufficiently accurate** to be presented to real introductory programming students. In contrast to Whoosh, which was accurate when returning the document containing the answer, the model's inaccuracy was considerable regarding queries containing Python-specific words that are English by nature. Nonetheless, modifications were made to the documents to enhance the performance of the model, but the numerous changes did not justify the possibility of the model still returning the incorrect answer or favoring this approach over a rule-based method.

Table 4.1 displays the number of incorrect and accurate predictions made by the BERT-QA model in response to 78 test queries requesting explanations and illustrations of conditionals . The test consisted of determining whether the span predicted by the model was contained within the phrase deemed to contain the answer. Approximately 68% of the 78 prompted questions were incorrect.

	Correct	Incorrect
if	14	12
else	5	21
elif	6	20
total	25	53

Table 4.1: Results of BERT-QA model accuracy testing regarding conditional queries.

Table 4.2 illustrates examples of incorrect and correct answers regarding the syntax of the different types of conditions.

Final Approach

Given the problems with the previous method, a more hard-coded strategy was adopted, with the advantage that the returned answer would be transmitted in a more controlled manner.

Rasa's objective, given a concept-related user utterance, is to extract the user's specific intent regarding the concept. Then, a custom action uses the intent's name as the key of a predefined dictionary whose value is the answer's path. Examples of intents include:

- ask_opearators_arithmetic
- ask_loop_for
- ask_conditional_elif
- ask_variables_ints

	Question	Answer	Status
if	Qual a sintaxe das instruções if?	pela palavra-chave "else" seguida por dois pontos (:)	Incorrect
if	Qual a sintaxe das condicionais if?	palavra-chave "if", duas variáveis para comparação, o operador dessa comparação, e finalizando com dois pontos (:)	Correct
else	Qual a sintaxe de um else?	pela palavra-chave "if"	Incorrect
else	sintaxe instrução else?	pela palavra-chave "else" seguida por dois pontos (:)	Correct
elif	Qual a sintaxe de uma condicional elif?	pela palavra-chave "else" seguida por dois pontos	Incorrect
elif	Qual a sintaxe de uma instrução elif?	dada pela palavra-chave "elif", duas variáveis para comparação, o operador dessa comparação, e finalizando com dois pontos (:)	Correct

Table 4.2: Examples of correct and incorrect answers given by the BERT-QA model.

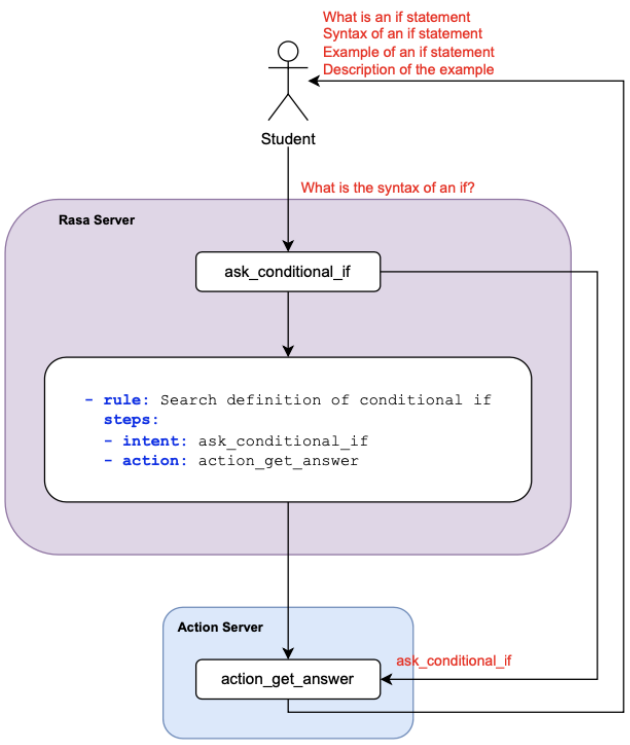
Figure 4.9 depicts the system procedure of the "concept definitions" feature:

1. **Intent extraction –** Rasa runs the message through its pipeline, resulting in an extraction of intent, in this case, the specific *ask_conditional_if* intent, when given the same example as the initial approach, e.g., a student asks, "What is the syntax of an if". As we are not using the BERT-QA model, the Whoosh library serves us no purpose, therefore no entities are required as we do not have to extract any keyword to pass to Whoosh.

2. **Rule mapping –** As with the implementation of all features, Rasa applies the RulePolicy, in this case the rule "Search definition of conditional if", and then executes the custom action *action_get_answer*. Herein lies one of the most significant disadvantages of this method, as not only multiple intents had to be declared to correspond to the multiple concepts the student could ask about, but also a rule had to be specified for each intent, even though the action called is common to all of them.

3. **Answer retrieval –** The action *action_get_answer* is then intended to retrieve the file path associated with the intent and return its contents. These files constitute the previously described collection of **introductory concept definitions**. The intent's name serves as the key to a dictionary containing the predefined path as its value.

Control over what is sent to the user is a big advantage of this approach, but having chosen this hard-coded path has a significant disadvantage in that specific data must be defined when adding new information, such as the definition



```
- rule: Search definition of conditional if
  steps:
  - intent: ask_conditional_if
  - action: action_get_answer
```

Figure 4.9: Concept definition feature's procedure (final method).

of the intent, its training examples, the creation of a rule pointing to the action *action_get_answer*, and the addition of a new key to the paths dictionary pointing to the desired file.

4.2.3 Error Guidance

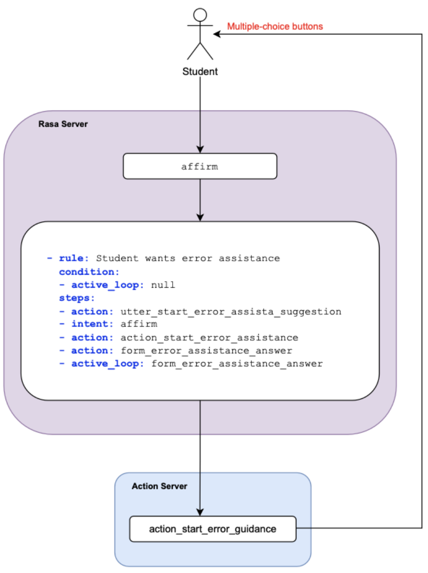
As mentioned previously on section 3.1.2 this functionality is triggered when a student submits code containing syntax errors. The platform inserts into the conversation an intent *EXTERNAL_ERROR_MESSAGE* with the entities *error_type* and *error_message* which are programmed to fill identically-named slots to be later accessed. The *error_type* defines the folder and the *error_message* the file where the wanted information is contained, as explained in the previous section for the **Error Guidance** collection. The result of this trigger is a message suggesting assistance.

Figure 4.10a depicts the system's procedure for initiating error assistance, whereas figure 4.10b depicts the examination of the answer.

The starting procedure for the assistance is as follows:

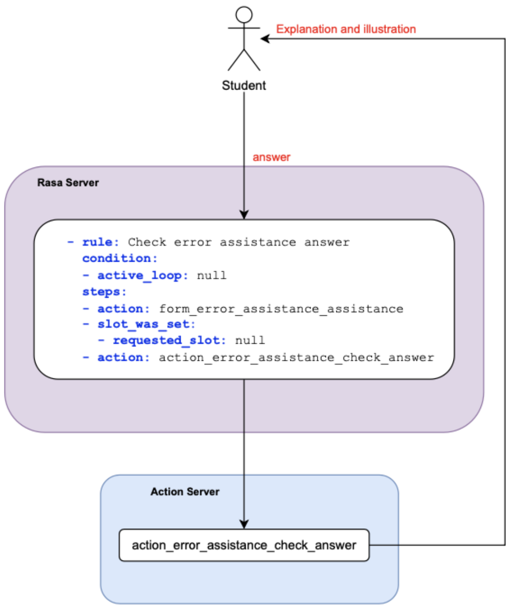
1. **Intent extraction –** Given the example, Rasa takes the "yes" message input and proceeds to predict the intent *affirm*.

2. **Rule mapping –** The intent is then be mapped to the rule "Student



```
- rule: Student wants error assistance
  condition:
  - active_loop: null
  steps:
  - action: utter_start_error_assista_suggestion
  - intent: affirm
  - action: action_start_error_assistance
  - action: form_error_assistance_answer
  - active_loop: form_error_assistance_answer
```

(a) Error guidance feature's initialization procedure.


Student
Explanation and illustration
answer
Rasa Server
Action Server
action_error_assistance_check_answer

```
- rule: Check error assistance answer
  condition:
  - active_loop: null
  steps:
  - action: form_error_assistance_assistance
  - slot_was_set:
    - requested_slot: null
  - action: action_error_assistance_check_answer
```

(b) Error guidance answer verification procedure.

Figure 4.10: Error guidance feature's procedure.

wants error assistance". This is because the message triggered by the platform corresponds to the *utter_start_error_guidance*, which is the first step of the defined rule, and as the student responded positively to the suggestion, it matches to the said rule. The following steps of the rule are the custom action *action_start_error_guidance* and the activation of the form *form_error_assistance_answer*, which informs Rasa that the user's subsequent message corresponds to the answer and not another intent. Notable that a message informing the student that the "stop" command terminates the interaction is sent.

3. **Question retrieval –** The objective of the action *action_start_error_guidance* is to use the slots *error_type* and *error_message* to locate the file containing the guidance's specifics. The data is then organized as required, sending the question statement and answer options to the student and storing the response along with the subsequent data in a slot.

For checking the answer and, if defined, sending a second question the procedure is the following:

1. **Answer acquirement –** The next input provided by the student after activating the form *form_error_assistance_answer* is interpreted as the student's response, which can be provided via a button click or in text format. Upon retrieving the answer, the form is stopped and the action *action_error_assistance _check_answer* is predicted as the next action when applying its corresponding rule.

2. **Answer verification –** As the answer is passed to the custom action, the slot, where the real answer was stored on, is retrieved and then compared to the answer provided by the student. If the answer is correct, it sends a predefined motivational message; otherwise, it sends an explanation of the differences between the choices. If there is a second question, the function is also responsible for delivering the question and re-starting the form, which will then trigger this same rule "Check error assistance answer".

4.2.4 Exercise Assistance

As touched on in section 3.1.3, this feature is not accessible until two minutes after the student has entered a programming exercise. By inserting an external intent *EXTERNAL_EXERCISE_INFO* with the entity *code_answer* and *ex_id* onto the conversation, a message will be triggered after the specified time to inform the student that the functionality is now available. The first entity carries the code for a solution to the exercise, while the second stores the id of the exercise. When Rasa detects it, it stores the information in slot so that it can be retrieved later in the conversation.

When the student requests the assistance, a few explanation messages are returned, followed by buttons containing the answer components. After the student's response, whether correct or incorrect, the ordered solution is displayed.

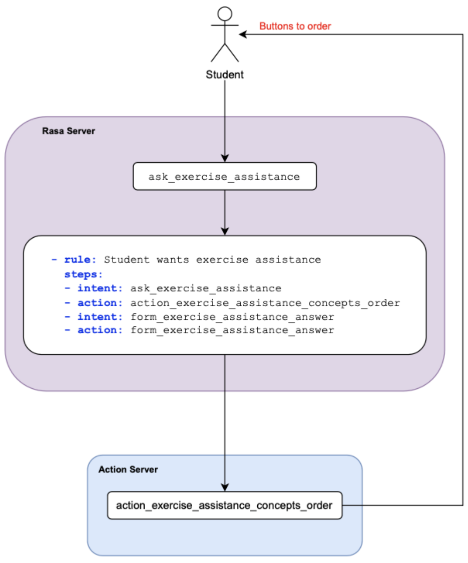
Figure 4.11a depicts the system's procedure for initiating exercise assistance, whereas figure 4.11b depicts the examination of the answer.

The starting procedure for the assistance is as follows:

1. **Intent extraction –** The student initially receives a message proposing help that instructs them to say *help me* if they need assistance. The procedure, therefore, begins when the intent *ask_exercise_assistance* is predicted.

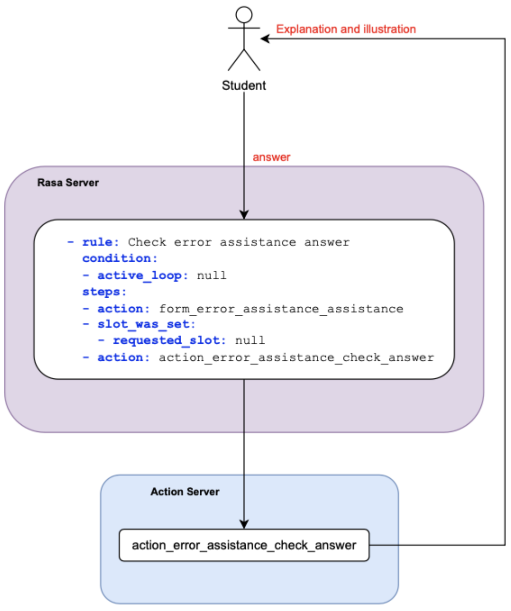
2. **Rule mapping –** In accordance with the rule "Student wants exercise assistant", the assistant predicts the execution of the action *action_exercise_assistance_concets_order* followed by the action *form_exercise_assistance_answer*, which activates the form and notifies the chatbot that the next input is an answer.

3. **Concepts –** The purpose of the action *action_exercise_assistance_concepts_order* is to retrieve the answer code previously stored in a slot and run it through another function that iterates its AST, converted using the previously mentioned ast library, and checks for the use of concepts including inputs, variables, functions, conversion functions, types of variables, various types of conditionals, and repetitions. This enables us to pass it the answer code passed by RESPE to Pyo, extract its components using ast, and generate the buttons without having to pre-define the pieces for the student to order for each exercise. The unshuffled buttons are stored in another slots, while a shuffled version is



```yaml
- rule: Student wants exercise assistance
  steps:
  - intent: ask_exercise_assistance
  - action: action_exercise_assistance_concepts_order
  - intent: form_exercise_assistance_answer
  - action: form_exercise_assistance_answer
```

(a) Exercise assistance feature's initialization procedure.


Explanation and illustration
Student
answer
Rasa Server
Action Server
action_error_assistance_check_answer

```yaml
- rule: Check error assistance answer
  condition:
  - active_loop: null
  steps:
  - action: form_error_assistance_assistance
  - slot_was_set:
    - requested_slot: null
  - action: action_error_assistance_check_answer
```

(b) Exercise assistance answer verification procedure.

Figure 4.11: Exercise guidance feature's procedure.

Code	Elements
var1 = "Olá "	Declaração de variável COM texto (string)
var2 = input()	Declaração de variável COM dado do teclado
concat = var1 + var2	Declaração de variável COM concatenação de variáveis
print(concat)	Impressão DE valor de variável

Table 4.3: Example of algorithm translated into natural language phrases.

sent to the student. An example of an algorithm translated into natural language pieces is demonstrated on table 4.3.

The procedure for validating the response is nearly identical to that described in the error guidance section, but now, the answer is provided by selecting the buttons in the order the student believes to be correct. The platform then stores the button's text in an array and passes it to Rasa as the answer. The agent predicts *action_error_assistance_check_answer* based on the rule policy, which compares the answer array to the one stored in the slot.

4.2.5 Question Suggestions and Hints

Regarding *question suggestions*, when the agent predicts the intent *ask_suggestions* it applies the RulePolicy, predicting and returning the custom action *action_return_suggestions*. This custom action utilizes the ast library to extract the concepts presented on the answer code, which is stored in a slot, and generates natural language questions using the same method as the one used for the "exercise assistance" functionality. It analyses the abstract syntactic tree of the algorithm corresponding to the answer and extracts the concepts involved. Finally, it transforms the concepts into questions. Examples are:

- What is an input?
- How do I convert a String to Integer?
- How to a conduct a subtraction?

In respect to the *hints/tips* feature, responses containing messages with simple tips were generated for various exercises. Upon detecting the intent *ask_hint*, the RulePolicy is similarly applied, and the action *action_return_hint* is returned. This action retrieves the exercise's id from the slot *ex_id* and returns the predefined response with the id as its name.

4.3 Pipeline

Initially, as described in section 4.2.2, two distinct approaches were utilized, each with its own pipeline (figures 4.12 and 4.13).

For each pipeline, the types of components are the same:

- Tokenizer
- Featurizers
- Intent Classifier
- Entity Extractor

The difference lies in the choice of the tokenizer and the featurizers:

Tokenizer As an entity needed to be extracted, both its position and syntactic function were essential. Consequently, a more Portuguese-centric strategy was required. To accomplish this, the spaCy structures needed to be initialized using the Poruguese model "pt_core_news_md"[28], upon which all subsequent spaCy components depended. This was used in the initial method, which employed a spaCy tokenizer. On the final approach, however, since only the intent is required, a simple white space tokenizer was used.

Featurizers As already mentioned, when extracting features, the initial method took a more entity-aware approach, and as a result, it utilized additional featurizers that the final method did not. The supplementary characteristics were:

[28]`https://spacy.io/models/pt`

```
pipeline:
    - name: SpacyNLP
      model: "pt_core_news_md"
      case_sensitive: False
    - name: SpacyTokenizer
    - name: SpacyFeaturizer
    - name: RegexFeaturizer
    - name: LexicalSyntacticFeaturizer
      "features": [
        [ 'BOS', 'EOS', 'pos'],
      ]
    - name: CountVectorsFeaturizer
    - name: CountVectorsFeaturizer
      analyzer: char_wb
      min_ngram: 1
      max_ngram: 4
    - name: DIETClassifier
      epochs: 100
      constrain_similarities: true
    - name: EntitySynonymMapper
    - name: FallbackClassifier
      threshold: 0.65
      ambiguity_threshold: 0.1
```

Figure 4.12: Pipeline utilized on the initial version of Pyo.

```
pipeline:
    - name: WhitespaceTokenizer
    - name: RegexFeaturizer
    - name: CountVectorsFeaturizer
    - name: CountVectorsFeaturizer
      analyzer: char_wb
      min_ngram: 1
      max_ngram: 4
    - name: DIETClassifier
      epochs: 100
      constrain_similarities: true
    - name: FallbackClassifier
      threshold: 0.65
      ambiguity_threshold: 0.1
```

Figure 4.13: Pipeline utilized on the final version of Pyo

- **SpacyFeaturizer -** Unlike the following featurizers, this feature extractor generates dense features for entity extraction and intent classification using the spaCy featurizer.

- **LexicalSyntacticFeaturizer -** This feature extractor generates sparse features for entity extraction. It moves a sliding window over each token in the user message and generates configuration-specific features. The configuration chosen was, BOS, EOS, and pos. The first determines whether the token is at the beginning of the phrase, the second whether it is at the end, and the third retrieves the token's Part-of-Speech tag.

Due to the use of entities in the initial strategy for concept definitions, a more cautious pipeline was required regarding entity extraction. However, since the final strategy did not require entity extraction, a more straightforward and lightweight pipeline with the same efficiency was chosen.

In addition, the following characteristics are shared and adopted by the second strategy:

- **RegexFeaturizer** This featurizer creates attributes for use in entity and intent classification. During training, the RegexFeaturizer generates a list of regular expressions defined in the training data. For each regex, a feature will be set indicating whether this expression was found in the user message or not. All features will eventually be fed to an intent classifier/entity extractor to facilitate classification (assuming the classifier has learned during the training phase that a particular set of features indicates a particular intent/entity).
- **CountVectorsFeaturizer** Creates classification features for intent. Using sklearn's CountVectorizer[29], this method produces a bag-of-words representation of the user's message, intent, and response.

Intent and Entity Classifier Rasa provides distinct classifiers for entity classification and intent classification, for which its multi-task architecture Dual Intent and Entity Transformer (DIET) was selected. It outputs entities, intents, and the ranking of the intents based on dense and/or sparse features.

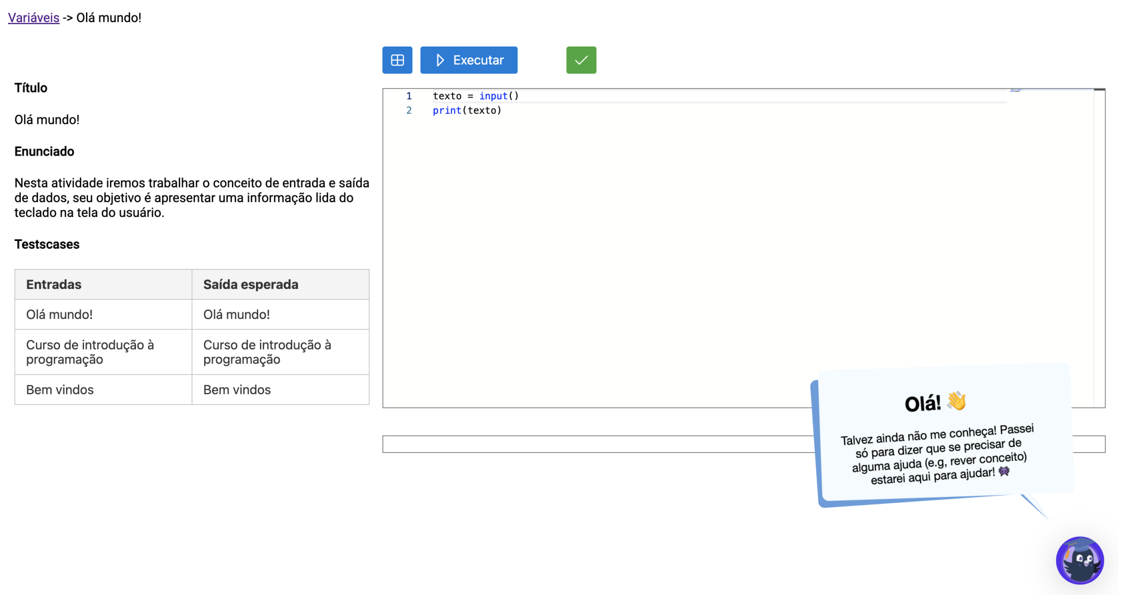
Fallback Classifier If the NLU intent classification scores are ambiguous, this classifier assigns the message the intent "nlu fallback". The confidence value is set to match the fallback threshold value. In response to this intent there was created a rule in which Pyo returns an apology message followed by the suggestion of questions.

4.4 Front End

Even though it is not the project's primary focus, the chatbot's user interface is crucial because it serves as the conduit between users and Pyo. The icon of a bird represents the chatbot, following the concept of Rasa itself, a perfect fit as its name is pronounced as per the Portuguese word "pio", a word used when

[29]`https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html`

imitating the sound of birds. During research on chatbot user interfaces, a free-to-use image[30] was discovered and selected for the icon of Pyo, which is depicted at the bottom right of figure 4.14.



Título

Olá mundo!

Enunciado

Nesta atividade iremos trabalhar o conceito de entrada e saída de dados, seu objetivo é apresentar uma informação lida do teclado na tela do usuário.

Testscases

Entradas	Saída esperada
Olá mundo!	Olá mundo!
Curso de introdução à programação	Curso de introdução à programação
Bem vindos	Bem vindos

```
texto = input()
print(texto)
```

Olá! 👋

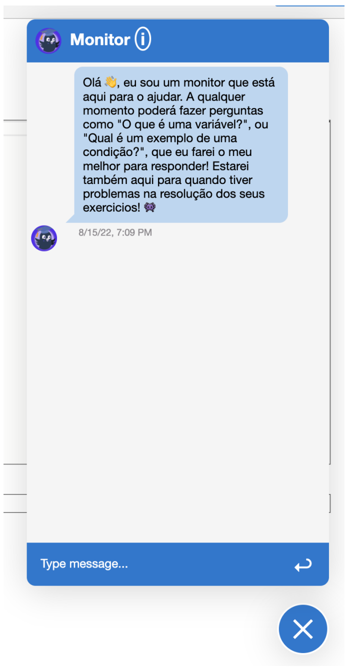
Talvez ainda não me conheça! Passei só para dizer que se precisar de alguma ajuda (e.g, rever conceito) estarei aqui para ajudar! 👾

Figure 4.14: Pyo's pop-up for awareness of its existance.

On figure 4.14 we are focusing on Pyo's appearance when a student enters an exercise, which displays a balloon for 10 seconds to indicate its existence, while figure 4.15 depicts the appearance of the chat window when the student initiates communication with the chatbot. Initially, as represented in the image, the student is presented with an introductory message containing example questions that describe the general purpose of the chatbot.

As the chatbot was integrated in the RESPE platform it made use of the same technologies, meaning angular, HTML, CSS, and TypeScript. The front end and back end of Pyo was conducted using simple http requests as Rasa already provides a endpoint (http://<host>:<port>/webhooks/rest/webhook) to send and receive messages.

[30]`https://github.com/JiteshGaikwad/Chatbot-Widget/blob/main/static/img/botAvatar.png`


Monitor

Olá 👋, eu sou um monitor que está aqui para o ajudar. A qualquer momento poderá fazer perguntas como "O que é uma variável?", ou "Qual é um exemplo de uma condição?", que eu farei o meu melhor para responder! Estarei também aqui para quando tiver problemas na resolução dos seus exercicios! 👾

8/15/22, 7:09 PM

Figure 4.15: Pyo's conversation window and introductory message.

Chapter 5

Evaluation and Discussion

A technological solution, regardless of how complete and ready it may appear, is only valuable if it achieves its goals when used by its intended audience. In order to validate the implemented agent, Pyo was exposed to real RESPE platform students. In this section the evaluation process will be explained and its results discussed.

5.1 Methodology

An online introductory programming course offered by the RESPE platform was advertised on the Pernambuco Federal Institute of Education's, a Brazilian institution of higher education, website to evaluate Pyo in a real-world setting. The course spanned two months and 40 hours and was designed for individuals with little to no programming background.

Throughout the course, the chatbot's interactions were analyzed on a daily basis in order to quickly correct any errors encountered or conduct pertinent modifications on its features based on their usage. Then, a questionnaire was developed to collect the students' opinions and feedback regarding the chatbot and its features.

Ten of the 65 enrolled students completed more than 50% of the course, with just one student completing more than 90%, similarly with previous RESPE's previous courses. Only nine students responded to the survey that was prompted three weeks before the course's conclusion.

Students were made aware of the chatbot and its experimental nature. Additionally they were warned of the storage and analysis of their conversations.

It is important to note that Pyo was present in every question, regardless of whether it required the development of an algorithm, but exercise assistance and error guidance were only available for programming questions.

5.2 Pyo Usage

Throughout the course, an analysis of the conversations was conducted almost daily in order to identify and correct any errors quickly. This section describes the implementation errors as well as the interaction and functionality usage statistics.

The percentage of errors in both the implementation and intent predictions was highest during weeks one through three. The number of errors increased as the number of active students (students actively participating in the course) increased during the first week.

The most significant implementation error was a flaw in the code that stored the conversations, which caused the students to receive messages regarding other conversations. Due to the lack of testing prior to the course, Pyo was unprepared to manage multiple students. As a result of storage errors, the first week's analysis of the conversation was omitted from subsequent analyses.

Regarding intent prediction, Pyo encountered a total of 12 incorrectly predicted intents, which were subsequently added to the agent's training examples. Following is an examination of the usage of Pyo and its characteristics, weekly, from May 23 to July 31.

Figure 5.1 depicts the evolution of the proportion of active students who engaged at least once with Pyo. The week with the highest percentage of students interacting with the agent was the second week, with 67% of students engaged in the course interacting with the agent. Inspired by the results referred to in section 2.4.4, the novelty of this technology can explain the level of student interest.

The weeks with the lowest percentage, inferior to 30%, were week 7 and week 10. The percentage of week 7 can be explained by a problem with the virtual machine which caused Rasa to crash. This issue could not be resolved until the following week due to a lack of access to a machine.

The percentages follow the increases and decreases in the number of active students throughout the weeks (figure 5.2), except for week 4 that saw a decrease in number of active students but and increase in the percentage of students interacting with Pyo.

Regarding the weekly analysis of each feature's usage, it was impossible to draw any conclusions because the data did not appear to follow any interesting pattern (figure 5.3). Therefore, table 5.1 depicts the absolute values of the number of times a student utilized a particular feature throughout the whole course. The most frequently utilized feature was the assistance on the exercise, followed by the explanation of concepts, hints, error guidance, and finally, the suggestion of questions.

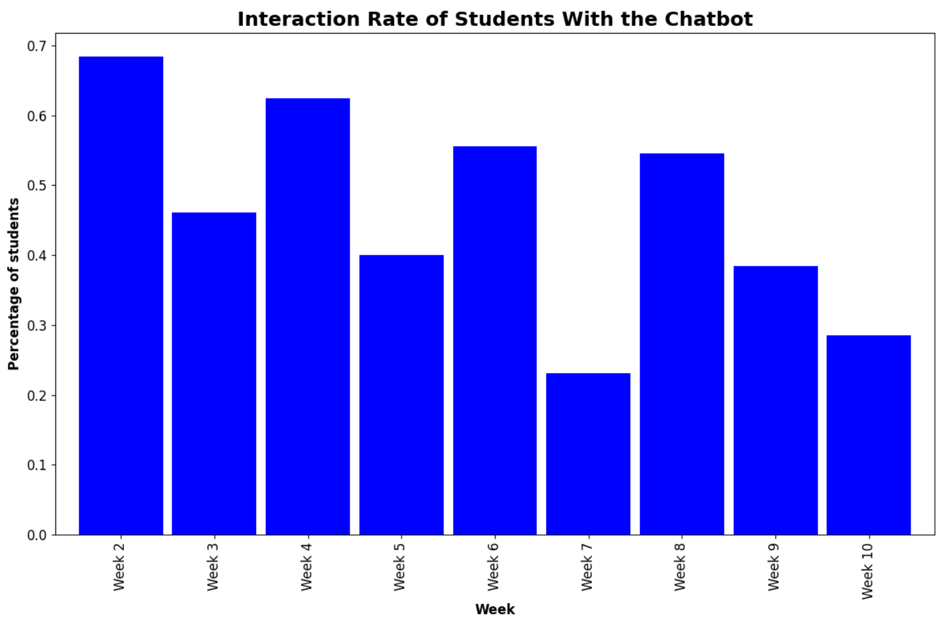


Figure 5.1: Weekly number of different students that interacted with Pyo.

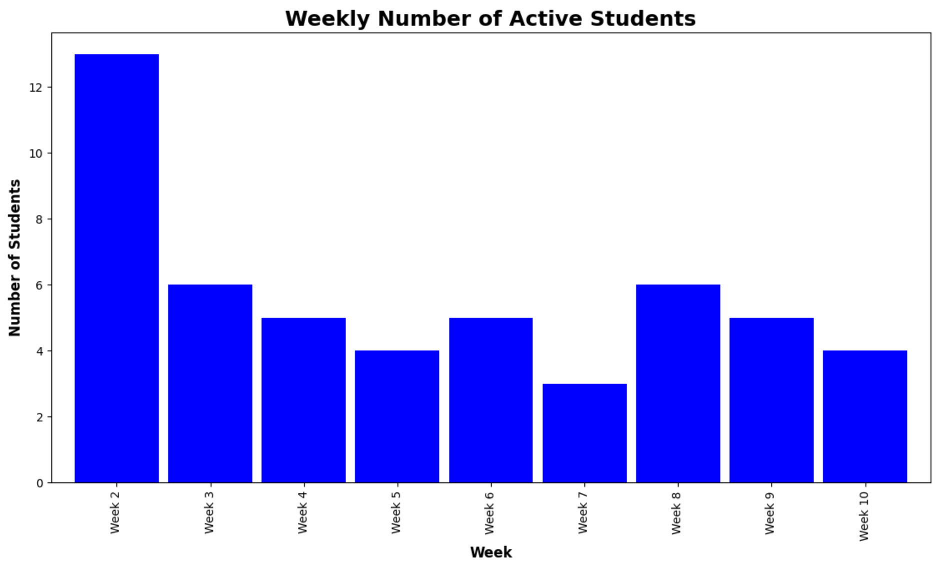


Figure 5.2: Weekly evaluation of the number of active students.

Functionality	**Number of Times Used**
Concept Definition	53
Error Guidance	33
Exercise Assistance	61
Question Suggestions	15
Hints	49

Table 5.1: Functionalities utilization frequency.

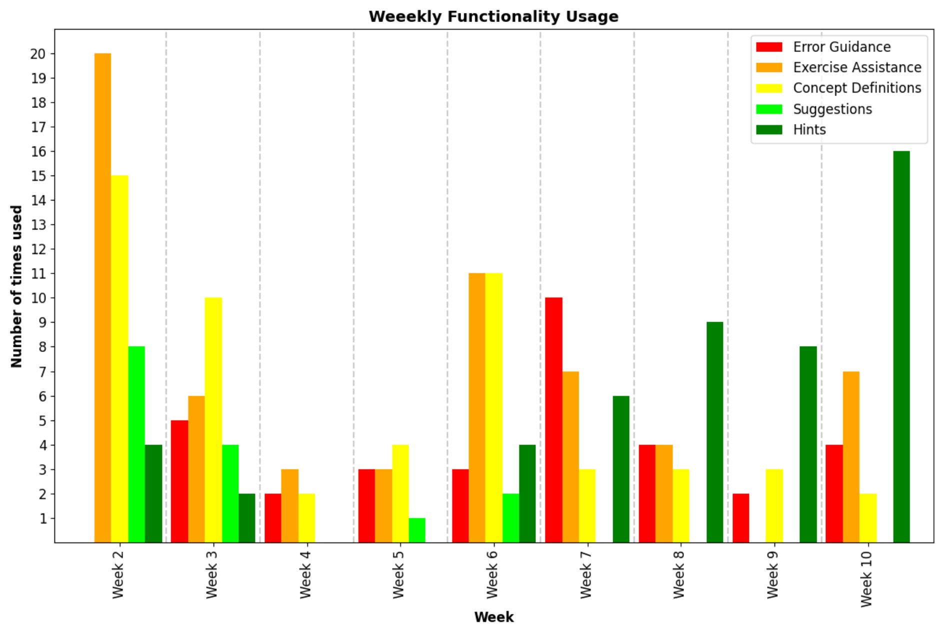


Figure 5.3: Weekly functionality utilization frequency.

5.3 Student Perceptions of Pyo

In an effort to enhance the user experience and gain insight into students' perceptions of Pyo, a questionnaire was developed and conducted using Google Forms. The questionnaire enables the identification of student feedback, allowing future modifications to be made accordingly. As stated in the methodology, the questionnaire was proposed three weeks prior to the end of the course in order to prevent additional dropouts and obtain a sufficient number of responses, however, only nine answers were obtained which corresponds to a total of approximately 14% of the initially registered users.

Despite the lack of a standard form or specific guidelines for its creation, a questionnaire inspired by the ones used in the evaluation of the reviewed chatbots in the same domain (section 2.4.6) was developed.

Two questionnaires were developed, the "Questionnaire 1 (No)" and the "Questionnaire 2 (Yes)", which would be displayed to the depending on whether the student would respond negatively or positively to the initial question, 'Do you use the assistant?'. Questionnaire 1 was designed to determine why students were not using Pyo.

Table 5.2 displays the form's questions and their respective evaluation scales. The original form and the original answers regarding the open questions are presented in Apendix B.

Observing the pie chat corresponding to the question - "Do you use the assistant?" - (figure 5.4), it was determined that only six out of the nine people that answer the questionnaire had interacted with the chatbot.

Question Id	**Question Statement**	**Scale**
Question 1a	Why did you not use the assistant?	Text field
Question 1b	Do you have any suggestions for how the assistant could be improved?	Text field
Question 2a	The assistant was easy to use	1, 2, 3, 4, or 5
Question 2b	The assistant possessed a realistic and engaging personality	1, 2, 3, 4, or 5
Question 2c	The assistant elucidated its functionalities and purpose in an enlightening manner	1, 2, 3, 4, or 5
Question 2d	The assistant accurately answered my questions	1, 2, 3, 4, or 5
Question 2e	The assistant handled errors and misunderstandings competently	1, 2, 3, 4, or 5
Question 2f	The assistant facilitated my learning	1, 2, 3, 4, or 5
Question 2g	I prefer interacting with my peers/professor over the assistant	1, 2, 3, 4, or 5
Question 2h	I feel uncomfortable knowing that my conversations are being analysed	1, 2, 3, 4, or 5
Question 2i	The assistant contained several features. Which do you consider to be more advantageous?	Options boxes
Question 2j	What aspects do you believe should be improved?	Text field

Table 5.2: Specifics regarding the form's questions (translated) and their respective evaluation scales.

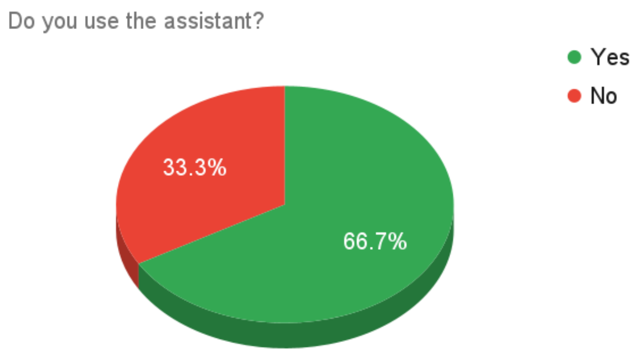


Figure 5.4: Results to question "Do you use the assistant?" of the questionnaire.

The majority of the options presented to the users were based on a Likert scale (1 - Completely Disagree, 2 - Disagree, 3- Indifferent, 4 - Agree, 5 - Completely Agree). The purpose of questions 1a and 1b was to discover why students did not interact with Pyo and if they had any suggestions for its enhancement.

Regarding the second questionnaire the aim was to incorporate qualitative characteristics such as functionality and affection (questions 2a, 2b, and 2c)

[Radziwill and Benton, 2017]. Followed by questions regarding the students' perceptions of precision and error management (questions 2d and 2e). The most obvious and crucial question regarding Pyo's goals was whether or not it had actually facilitated their learning (question 2f) [Chinedu and Ade-Ibijola, 2021]. The students were also asked about their stance on interacting with their peers/professor over the agent (question 2g), followed by a question devised to determine if their awareness that their conversations were being analyzed had any bearing on their interactions (question 2h). Finally, the students were queried regarding their preferred functionalities and suggestions for enhancements (questions 2i and 2j)[31].

5.3.1 Results

In this section the results pertaining to the questions of the two parts of the questionnaire ("No", and "Yes") are presented.

Why did you not use the assistant?

When asked for the reason on why have not they used Pyo, two of the students simply said that they had yet to need its assistance, while the third student thought of the interactions of the chatbot as being forced, as they prefer more examples with the content they are currently studying. This response can be attributed to a lack of comprehension of Pyo's capabilities, as the chatbot provides an example when asked for one via the concept definitions functionality. However, their feedback could be considered for the future addition of a feature that focuses solely on examples equipping the agent with numerous illustrations of the different concepts.

Do you have any suggestions for how the assistant could be improved?

Similar to the previous question, two students responded negatively, whereas the third student provided a more detailed answer. Now focusing on the first part of their answer, the student suggested a higher interval where the exercise suggestion message is presented, as two minutes is not enough time, for every question, to solve the exercise. A further suggestion was the addition of preferences, such as the ability to disable or enable suggestions, a particular feature, or the use of a "do not bother me" checkbox.

The assistant was easy to use

Figure 5.5 reveals that, of the six students who used the chatbot, two were indifferent to the statement "The assistant was easy to use", while the remaining

[31]For the observations purposes, the form remains open via the url: `https://forms.gle/6r7BpJoX3kdq3Yg17`

four were entirely in agreement with the statement. Since no student wholly disagreed, the results can be considered positive. The 'indifference' may be a result of Pyo's initial difficulties, which caused confusion in the interactions.

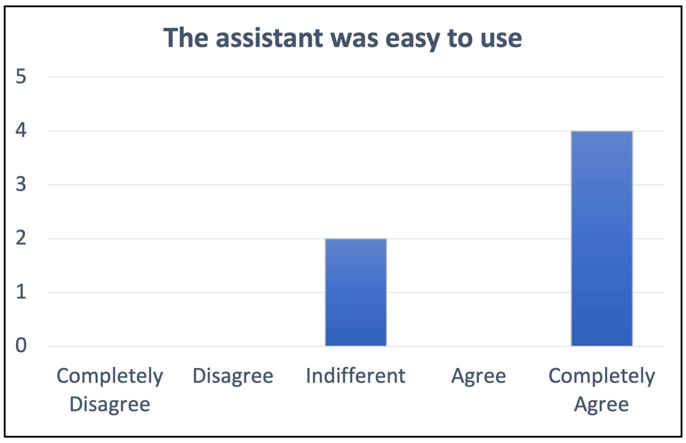


Figure 5.5: Results to question "The assistant was easy to use" of the questionnaire.

The assistant possessed a realistic and engaging personality

Observing the responses in figure 5.6, it was determined that one student disagreed, two agreed, and three completely agreed with the statement. We can, therefore, conclude that the majority of students who responded to the survey found Pyo's personality captivating.

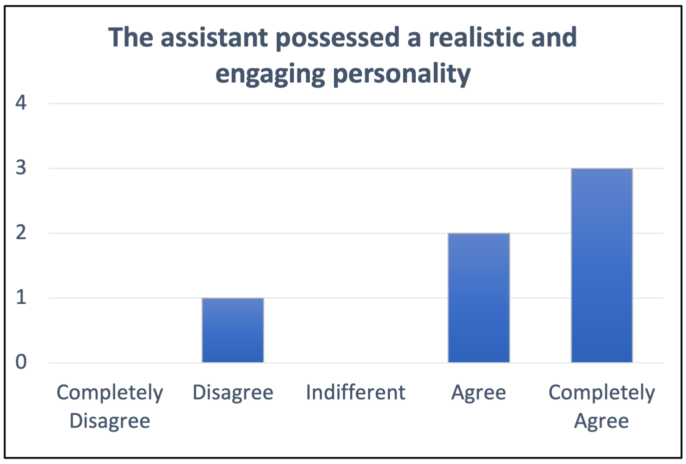


Figure 5.6: Results to question "The assistant possessed a realistic and engaging personality" of the questionnaire.

The assistant elucidated its functionalities and purpose in an enlightening way

Regarding the purpose of the chatbot, the feedback was deemed satisfactory (figure 5.7), with one student indifferent, three agreeing, and two completely agreeing with the statement "The assistant elucidated its functionalities and purpose in an enlightening manner".

Although still positive, it was not as positive as the responses to statement 2a. This was to be expected, as it was evident that not all students were aware on what the functionalities offered, for instance, that the exercise assistance would allow them to be presented with the skeleton of a possible answer, which would then only require translation into Python. And if the student had forgotten how to convert the concept into code, they could refresh their memory by asking Pyo.

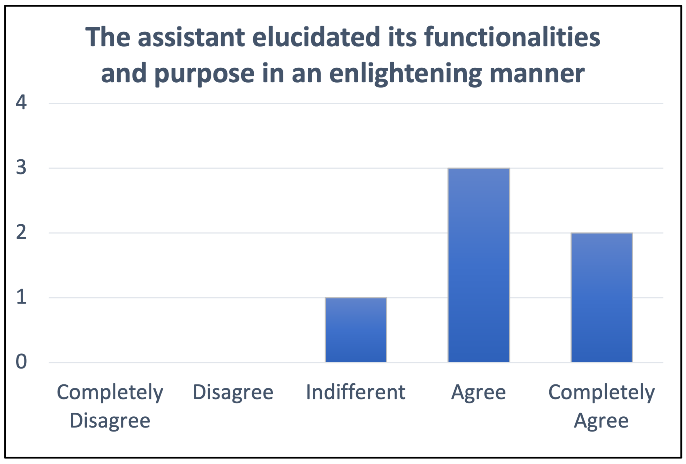


Figure 5.7: Results to question "The assistant elucidated its functionalities and purpose in an enlightening way" of the questionnaire.

The assistant accurately answered my questions

To this statement four of the students opted for the indifferent response 5.8. The conclusion drawn is that students had both positive and negative interactions with the chatbot.

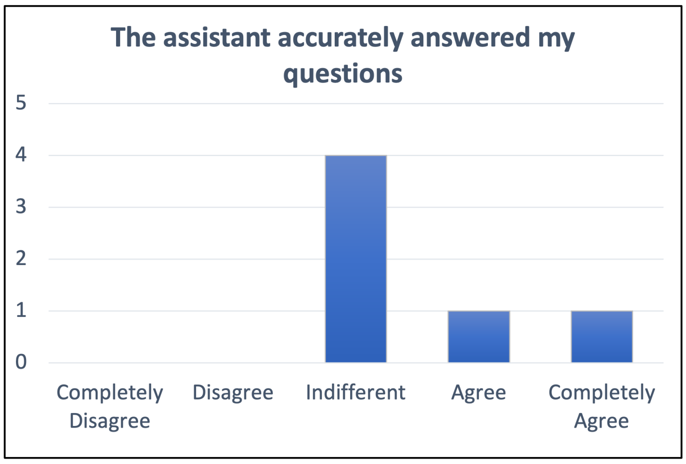


Figure 5.8: Results to question "The assistant accurately answered my questions" of the questionnaire.

The assistant handled errors and misunderstandings competently

As depicted in figure 5.9, 2 students responded indifferently to the statement about how Pyo handled errors and misunderstandings, while 3 agreed and 1 totally agreed.

As stated previously, conversations were analyzed daily to quickly correct any incorrectly predicted intentions or other issues, which may account for the majority of positive feedback. The fallback intent may be another reason, which in the cases where the agent was unable to predict an intent Pyo would return question suggestions.

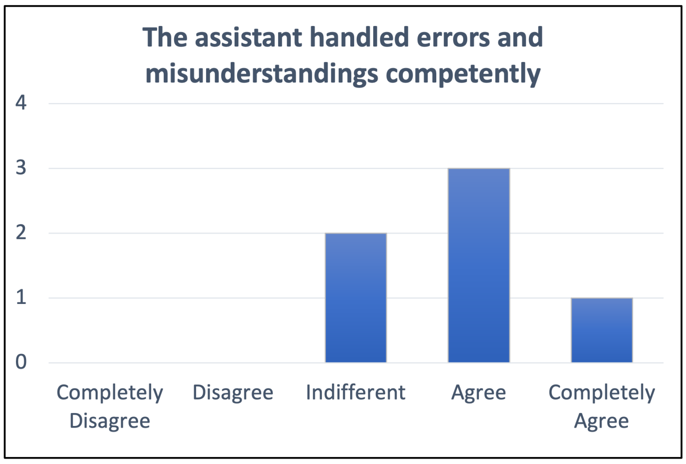


Figure 5.9: Results to question "The assistant handled errors and misunderstandings competently" of the questionnaire.

The assistant facilitated my learning

The majority of responses to the statement "the assistant facilitated my learning" were positive, with 4 students agreeing with the statement (figure 5.10). Although 2 students still exhibited apathy, the conclusion is positive and significant, as this is the chatbot's primary objective.

I prefer interacting with my peers/professor over the assistant

Throughout the course, the student had access to a WhatsApp group where they could communicate with their classmates and instructor. The students frequently turned to the group, which may have been due to the fact that Pyo did not provide an exact solution for how to solve an exercise or a straightforward solution for how to solve an error; instead, it provided the pieces for the student to reach the solution independently. They would then consult the group, where their peers would provide the exact solution. The majority of respondents favored interacting with their peers and professor (figure 5.11).

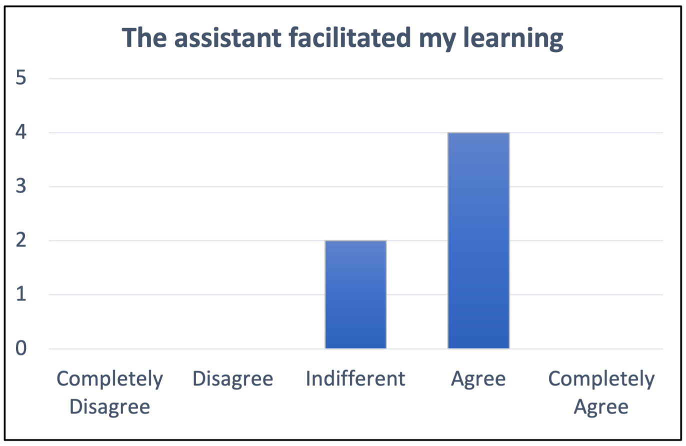


Figure 5.10: Results to question "The assistant facilitated my learning" of the questionnaire.

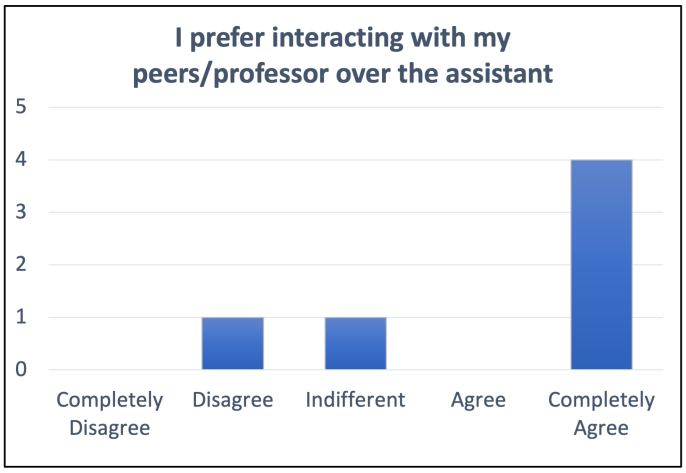


Figure 5.11: Results to question "I prefer interacting with my peers/professor over the assistant" of the questionnaire.

I feel uncomfortable knowing that my conversations are being analysed

As a means of determining whether the results obtained on the previous question could be due to discomfort on the part of the student, the student was prompted to provide their response to the statement, "I feel uncomfortable knowing that my conversations are being analyzed". It was determined that 66.7% of students strongly disagreed with the statement, while the rest were neutral (figure 5.12). This suggests that the student knowing of analysis of their conversations did not influence the preference on asking peers for assistance.

The assistant contained several features. Which do you consider to be more advantageous?

The students were also asked which of the features they believe to be most advantageous; they were required to select at least one but were permitted to select more. The results, displayed in figure 5.13, indicate that error assistance

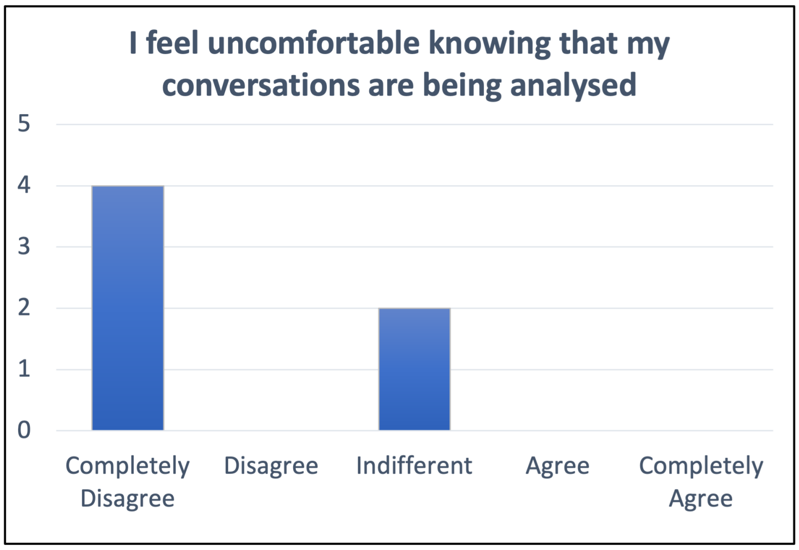


Figure 5.12: Results to question "I feel uncomfortable knowing that my conversations are being analysed" of the questionnaire.

was deemed the most important factor, followed by error assistance, question suggestion, and hints.

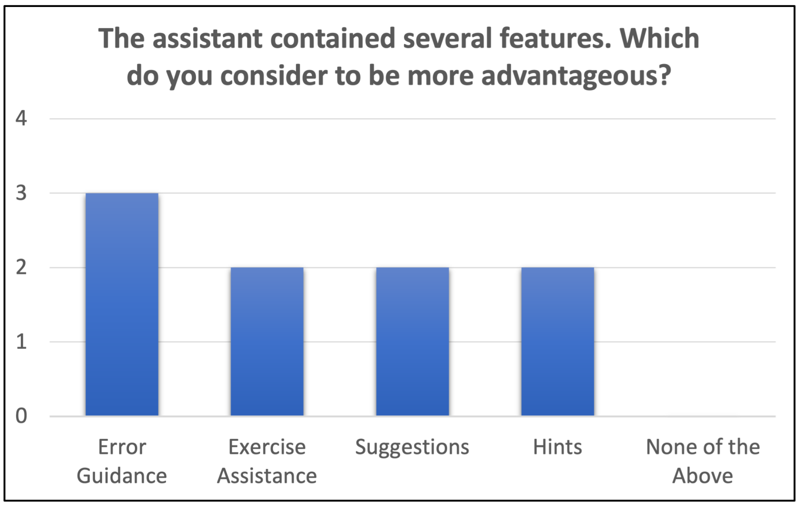


Figure 5.13: Results to question "The assistant contained several features. Which do you consider to be more advantageous?" of the questionnaire.

What aspects do you believe should be improved?

When asked for suggestions, as this question was not mandatory, only 4 students answered. The feedback was the following:

- **Translated:** Focus on the exercises that we are solving, especially for us students who have not had contact with programming. An example: if the error is in line "4", the program could alert us that there is an error in the respective line, and present the concepts related to the error committed, we could review and find the error, as if a teacher were nearby, guiding. And if it is an error in the logic or typing of the programming, the program could say "Paulo, your mistake is in not having declared a data" or "you forgot to print your data, use the print() command to do so", or "In line 5 you didn't insert the quotes - (could come as the general concept)". I think this is the program's proposal, because the student keeps trying, and can't implement the right logic in the exercise or due to a typing error, and when he uses the program he can't understand where the error is and ends up getting lost.

- **Original:** "Focar nos exercícios que estamos a resolver, principalmente, nós alunos que não tivemos contato com programação. Um exemplo: se erro for na linha "4", o programa poderia nos alertar que há um erro na respectiva linha, e apresentar os conceitos relacionadas ao erro cometido, poderíamos revisar e encontrar o erro, como se um professor estivesse ali perto, nos orientando. E caso for um erro na lógica ou digitação da programação, o programa poderia dizer "Paulo o seu erro está em não ter declarado um dado" ou "você se esqueceu de imprimir seu dado, para isso use o comando print()", ou "Na linha 5 você não inseriu as aspas – (poderia vir conceito geral)". Acho que a proposta é essa do programa, porque o aluno vai tentando, e não consegue implementar a lógica certa no exercício ou devido a algum erro na digitação, e quando recorre ao programa não consegue compreender onde está o erro e acaba se perdendo."

- **Translated:** Well, the assistant itself is great, but I believe that it could be improved by adding a second monitor to display the questions, shortening the response time, and increasing student interaction.
- **Original:** "Bom, o monitor em si é ótimo, acho que o que poderia melhorar seria ter mais um monitor para ver as perguntas, ter um tempo de resposta um pouco mais rápido, ter uma interação maior aos alunos."

- **Translated:**Always end the service in the face of the questions asked, whether conclusive or not, for instance, it was not possible to identify the question.
- **Original:** "Sempre finalizar o atendimento diante das perguntas feitas, seja conclusivas ou não, tipo, não foi possível identificar a perguntar"

- **Translated:** More examples.
- **Original:** "Da mais exemplos"

In contrast to multiple-choice questions, the first recommendation suggests a more straightforward approach for error guidance. For example, attempting to pay greater attention to the code that is being developed and adding support for logical errors or missing steps. The second suggestion concerns the assistant's speed, which could be enhanced. The third was considered illogical, as even when it was impossible to predict the student's intent, Pyo would inform them of this fact. The reason can be that the student may still be referring to the mistakes made in the first week. Additionally as similarly suggested on the "no" side of the questionnaire is the suggestion of the addition of more examples.

These suggestions are a great insight into the enhancements that can be conducted regarding Pyo's functionalities.

5.3.2 Conclusion

The questionnaire allowed us to identify the perception of students regarding different aspects of Pyo. On the one hand, the chatbot was deemed user-friendly, realistic, possessing an engaging personality, perceptive regarding its features, adept at handling misunderstandings, and useful, thus achieving the primary objective of "being able to assist introductory programming students".

The students, on the other hand, viewed the chatbot as neither accurate nor inaccurate and showed a strong preference for seeking assistance from their peers and professor. Possible causes include the presence of critical errors at the beginning and a lack of individualized assistance based on the student's level of knowledge.

Nonetheless, the analysis demonstrated the utility and value of Pyo, as the majority of questionnaire responses were positive. It was also extremely helpful in identifying aspects in need of improvements to conduct in future work, described in the final chapter.

Important to consider the major limitation of this evaluation, as only twelve of the 65 students enrolled in the course reached the final week, and only nine completed the questionnaire, preventing us from drawing conclusions that are representative of the entire population, in this case the students.

Chapter 6

Conclusion

In this section, a review of all the completed work, the resulting contributions, and the upcoming work is presented.

This project's initial phase consisted of an analysis of existing tools and applications in order to achieve the proposed objectives. The objective of this research was to acquire the knowledge required to implement a chatbot to aid introductory programming students. The implementation of error guidance, exercise assistance, and concept definitions followed an analysis of the RESPE platform to determine how Pyo could assist its students. Not forgetting the later addition of question hints and suggestions

Initially the implementation of the different features included the process of Information Retrieval Question Answering (QA) and Question Generation (QG), which were omitted from the final version due to their unreliability. To counter this and achieve the goals of this dissertation a rule-based approach was taken.

Finally, recalling the objectives proposed as a result of this thesis:

- Development of a chatbot able to answer questions about introductory concepts, provide assistance with the exercise, and provide guidance on mistakes
- Integration of the chatbot into an online platform for teaching introductory programming
- Evaluation of the assistant by real world users

Given the results of the daily analysis and of the questionnaire, it is believed that the work performed was successful and that all objectives were met. It can be regarded as a significant and useful contribution to the field of chatbots, despite the need for further development.

Despite the drawn conclusion, we are aware of certain limitations and the fact that there is significant room for improvement. For future endeavors, it is intended to continue the entire current procedure. Future work would concentrate on training a QA model specifically for the programming domain in order to accurately automate the agent's answer-providing capabilities.

Regarding the functionalities, the question suggestions would remain unchanged, as students, particularly at the beginning, may not know what to ask for or what concepts are involved in the exercise. The exercise's assistance would be enhanced in terms of its design's intuitiveness.

The error guidance feature would be modified to account for frequently occurring logical errors and missing steps in the student's algorithm, for which a more direct approach would be taken by returning the definitions of the involved concepts. Keeping in mind that the world of logical errors easily turns into a "snowball", it could take a considerable amount of time to implement this feature correctly.

In addition to the previously mentioned factors, the students' participation in the course should be analyzed to determine ways to personalize the Pyo's support. Additionally, the front-end could be improved by creating settings that give students control over the notifications they receive from the chatbot.

Upon the project's objective, an article was written, "Pyo, a Chatbot Assistant for Introductory Programming Students" accepted to the XXIV International Symposium on Computers conference, focusing on the educational aspect of the agent.

On a final note, my overall impression of the internship has been quite positive. This was a time of tremendous professional and personal development. This year established a crucial link between education and the working world. It undoubtedly revealed areas that required improvement, such as time management, but, on the other hand, allowed me to improve my research skills and learn and get a great insight into the world of intelligent systems. Not only was this dissertation an excellent addition to my academic career, but also to my personal life.

References

Eleni Adamopoulou and Lefteris Moussiades. Chatbots: History, technology, and applications. *Machine Learning with Applications*, 2, 2020. ISSN 26668270. doi: 10.1016/j.mlwa.2020.100006.

John R Anderson. Problem solving and learning. *American psychologist*, 48(1):35, 1993.

Soufyane Ayanouz, Boudhir Anouar Abdelhakim, and Mohammed Benhmed. A smart chatbot architecture based NLP and machine learning for health care assistance. In *Proceedings of the 3rd International Conference on Networking, Information Systems Security*, NISS2020, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450376341. doi: 10.1145/3386723.3387897. URL `https://doi.org/10.1145/3386723.3387897`.

Asma Ben Abacha and Pierre Zweigenbaum. Means: A medical question-answering system combining NLP techniques and semantic web technologies. *Information Processing Management*, 51(5):570–594, 2015. ISSN 0306-4573. doi: https://doi.org/10.1016/j.ipm.2015.04.006. URL `https://www.sciencedirect.com/science/article/pii/S0306457315000515`.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

Paweł Budzianowski and Ivan Vulić. Hello, it's GPT-2 – how can i help you? towards the use of pretrained language models for task-oriented dialogue systems, 2019. URL `https://arxiv.org/abs/1907.05774`.

Tanja Bunk, Daksh Varshneya, Vladimir Vlasov, and Alan Nichol. DIET: Lightweight language understanding for dialogue systems, 2020. URL `https://arxiv.org/abs/2004.09936`.

Gustavo Carreira, Leonardo Silva, António Mendes, and Hugo Gonçalo Oliveira. Pyo, a chatbot assistant for introductory programming students. In *Proceed-*

ings of XXIV International Symposium on Computers in Education (SIIE), page (accepted), 2022.

Hsiu-Ling Chen, Gracia Vicki Widarso, and Hendri Sutrisno. A chatbot for learning chinese: Learning achievement and technology acceptance. *Journal of Educational Computing Research*, 58(6):1161–1189, 2020. doi: 10.1177/0735633120929622.

Yuh-Jen Chen, Chun-Han Wu, Yuh-Min Chen, Hsin-Ying Li, and Huei-Kuen Chen. Enhancement of fraud detection for narratives in annual reports. *International Journal of Accounting Information Systems*, 26:32–45, 2017. ISSN 1467-0895. doi: https://doi.org/10.1016/j.accinf.2017.06.004. URL `https://www.sciencedirect.com/science/article/pii/S1467089516300343`.

Okonkwo Chinedu and Abejide Ade-Ibijola. Python-bot: A chatbot for teaching Python programming. *Engineering Letters*, 29:25–34, 02 2021.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation, 2014.

Eunsol Choi, He He, Mohit Iyyer, Mark Yatskar, Wen tau Yih, Yejin Choi, Percy Liang, and Luke Zettlemoyer. QuAC : Question answering in context. *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2018. doi: 10.18653/v1/D18-1241.

Kenneth Mark Colby, Sylvia Weber, and Franklin Dennis Hilf. Artificial paranoia. *Artificial Intelligence*, 2(1):1–25, 1971.

Emmelyn A. J. Croes and Marjolijn L. Antheunis. Can we be friends with Mitsuku? a longitudinal study on the process of relationship formation between humans and a social chatbot. *Journal of Social and Personal Relationships*, 38(1): 279–300, 2021. doi: 10.1177/0265407520959463. URL `https://doi.org/10.1177/0265407520959463`.

Robert Dale. The return of the chatbots. *Natural Language Engineering*, 22(5):811–817, 2016.

Siti Hawa Mad Daud. e-java chatbot for learning programming language: A post-pandemic alternative virtual tutor. 2020.

Ernest Davis. Ai amusements: The tragic tale of Tay the chatbot. *AI Matters*, 2 (4):20–24, dec 2016. doi: 10.1145/3008665.3008674. URL `https://doi.org/10.1145/3008665.3008674`.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding, 2019.

Xinya Du, Junru Shao, and Claire Cardie. Learning to ask: Neural question generation for reading comprehension, 2017.

Milan Van Eeuwen. Mobile conversational commerce: messenger chatbots as the next interface between businesses and consumers. *University of Twente*, 2017.

Jacob Eisenstein. *Natural Language Processing*. MIT press, 2018. ISBN 9780198520115.

Dylan Keifer Finch, Stephen H. Edwards, and Mukund Rajagopal. Using a pedagogical agent to support students learning to program. *ASEE Annual Conference and Exposition, Conference Proceedings*, 2020-June, 2020. ISSN 21535965. doi: 10.18260/1-2--35447.

Rishma Garg, Riya Riya, Sahil Thakur, Nancy Tyagi, Kasunuru Nawaz Basha, Dinesh Vij, and Gurjot Singh Sodhi. NLP based chatbot for multiple restaurants. In *2021 10th International Conference on System Modeling Advancement in Research Trends (SMART)*, pages 439–443, 2021. doi: 10.1109/SMART52563.2021.9676218.

Santiago González-Carvajal and Eduardo C. Garrido-Merchán. Comparing BERT against traditional machine learning text classification, 2021.

Pierre Guillou. Portuguese BERT base cased QA (question answering), finetuned on SQUAD v1.1. 2021.

Jiafeng Guo, Yixing Fan, Qingyao Ai, and W. Bruce Croft. A deep relevance matching model for Ad-Hoc retrieval. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, CIKM '16, page 55–64, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450340731. doi: 10.1145/2983323.2983769. URL `https://doi.org/10.1145/2983323.2983769`.

Sebastian Hobert. Say hello to 'coding tutor'! design and evaluation of a chatbot-based learning system supporting students to learn to program. In *ICIS 2019 Proceedings. 9*, 2019.

Mohammed Ismail and Abejide Ade-Ibijola. Lecturer's apprentice: A chatbot for assisting novice programmers. *2019 International Multidisciplinary Information Technology and Engineering Conference (IMITEC)*, pages 1–8, 2019.

W. Lewis Johnson. Pedagogical agent research at carte. *AI Mag.*, 22(4):85–94, oct 2001. ISSN 0738-4602.

Daniel Jurafsky and James H Martin. *Speech and Language Processing: An introduction to speech recognition, computational linguistics and natural language processing.* 2021.

Monisha Kanakaraj and Ram Mohana Reddy Guddeti. NLP based sentiment analysis on twitter data using ensemble classifiers. In *2015 3rd International Conference on Signal Processing, Communication and Networking (ICSCN)*, pages 1–5, 2015. doi: 10.1109/ICSCN.2015.7219856.

Serena Leggeri, Andrea Esposito, and Luca Iocchi. Task-oriented conversational agent self-learning based on sentiment analysis. In *NL4AI@AI*IA*, 2018.

Sweta P. Lende and M. M. Raghuwanshi. Question answering system on education acts using NLP techniques. In *2016 World Conference on Futuristic Trends in Research and Innovation for Social Welfare (Startup Conclave)*, pages 1–6, 2016. doi: 10.1109/STARTUP.2016.7583963.

Weixin Liang, Kai-Hui Liang, and Zhou Yu. HERALD: An annotation efficient method to detect user disengagement in social conversations, 2021.

Hui Liu, Qingyu Yin, and William Yang Wang. Towards explainable NLP: A generative explanation framework for text classification, 2018. URL `https://arxiv.org/abs/1811.00196`.

Yichao Lu, Phillip Keung, Shaonan Zhang, Jason Sun, and Vikas Bhardwaj. A practical approach to dialogue response generation in closed domains, 2017. URL `https://arxiv.org/abs/1703.09439`.

Linxiao Ma, John Ferguson, Marc Roper, and Murray Wood. Investigating the viability of mental models held by novice programmers. *SIGCSE Bull.*, 39(1): 499–503, mar 2007. ISSN 0097-8418. doi: 10.1145/1227504.1227481. URL `https://doi.org/10.1145/1227504.1227481`.

Tanya J. McGill and Simone E. Volet. A conceptual framework for analyzing students' knowledge of programming. *Journal of Research on Computing in Education*, 29(3):276–297, 1997. doi: 10.1080/08886504.1997.10782199. URL `https://doi.org/10.1080/08886504.1997.10782199`.

Fernando A. Mikic, Juan C. Burguillo, Martín Llamas, Daniel A. Rodríguez, and Eduardo Rodríguez. Charlie: An aiml-based chatterbot which works as an interface among ines and humans. *20th EAEEIE Annual Conference, EAEEIE 2009 - Formal Proceedings*, 2009. doi: 10.1109/EAEEIE.2009.5335493.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013. URL `https://arxiv.org/abs/1301.3781`.

Sarah Müller, Bianca Bergande, and Philipp Brune. Robot tutoring: On the feasibility of using cognitive systems as tutors in introductory programming education: A teaching experiment. In *Proceedings of the 3rd European Conference of Software Engineering Education*, ECSEE'18, page 45–49, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450363839. doi: 10.1145/3209087.3209093. URL `https://doi.org/10.1145/3209087.3209093`.

Andrés Pedreño Muñoz. Lola, el chatbot inteligente que triunfa entre los estudiantes. `https://elpais.com/retina/2018/11/30/innovacion/1543580663_865121.html`, 2018.

Toshiaki Nakazawa, Kun Yu, Daisuke Kawahara, and Sadao Kurohashi. Example-based machine translation based on deeper NLP. In *Proceedings of the Third International Workshop on Spoken Language Translation: Evaluation Campaign*, Kyoto, Japan, nov 2006. URL `https://aclanthology.org/2006.iwslt-evaluation.9`.

Ketakee Nimavat and Tushar Champaneria. Chatbots: An overview. types, architecture, tools and future possibilities. *International Journal for Scientific Research Development*, 10 2017.

Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1162. URL `https://aclanthology.org/D14-1162`.

José Quiroga Pérez, Thanasis Daradoumis, and Joan Manuel Marquès Puig. Rediscovering the use of chatbots in education: A systematic literature review. *Computer Applications in Engineering Education*, 28(6):1549–1565, 2020. doi: https://doi.org/10.1002/cae.22326.

Yizhou Qian and James Lehman. Students' misconceptions and other difficulties in introductory programming: A literature review. *ACM Trans. Comput. Educ.*, 18(1), oct 2017. doi: 10.1145/3077618. URL `https://doi.org/10.1145/3077618`.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 8(1):9, 2019.

Nicole M. Radziwill and Morgan C. Benton. Evaluating quality of chatbots and intelligent conversational agents, 2017. URL `https://arxiv.org/abs/1704.04579`.

S. Reshmi and Kannan Balakrishnan. Implementation of an inquisitive chatbot for database supported knowledge bases. *Sadhana - Academy Proceedings in Engineering Sciences*, 41, 2016. ISSN 09737677. doi: 10.1007/s12046-016-0544-1.

Iulian Vlad Serban, Ryan Lowe, Peter Henderson, Laurent Charlin, and Joelle Pineau. A survey of available corpora for building data-driven dialogue systems: The journal version. *Dialogue Discourse*, 9(1):1–49, 2018. doi: https://doi.org/10.5087/dad.2018.101.

Farhana Sethi. FAQ (frequently asked questions) chatbot for conversation. *INTERNATIONAL JOURNAL OF COMPUTER SCIENCES AND ENGINEERING*, 8:7–10, 10 2020.

Nuobei SHI, Qin Zeng, and Raymond Lee. XAI language tutor - a XAI-based language learning chatbot using ontology and transfer learning techniques. *International Journal on Natural Language Computing*, 9:1–21, 10 2020. doi: 10.5121/ijnlc.2020.9501.

Shefaly Shorey, Emily Neo Kim Ang, John Yin Gwee Yap, Esperanza Debby Ng, Siew Tiang Lau, and CheeKong K. Chui. A virtual counseling application using artificial intelligence for communication skills training in nursing education: Development study. *Journal of Medical Internet Research*, 21, 2019.

Leonardo Silva, António Mendes, and Anabela Gomes. Fostering programming students regulation of learning using a computer-based learning environment. In *Proceedings of XXIV International Symposium on Computers in Education (SIIE)*, page (accepted), 2022.

Simon, Raina Mason, Tom Crick, James H. Davenport, and Ellen Murphy. Language choice in introductory programming courses at australasian and uk universities. *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, 2018.

Alan M. Turing. I.—COMPUTING MACHINERY AND INTELLIGENCE. *Mind*, LIX(236):433–460, 10 1950. ISSN 0026-4423. doi: 10.1093/mind/LIX.236.433. URL `https://doi.org/10.1093/mind/LIX.236.433`.

Vesa Vainio and Jorma Sajaniemi. Factors in novice programmers' poor tracing skills. *SIGCSE Bull.*, 39(3):236–240, jun 2007. ISSN 0097-8418. doi: 10.1145/1269900.1268853. URL `https://doi.org/10.1145/1269900.1268853`.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017a.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017b.

Matthew A. Verleger and James J. Pembridge. A pilot study integrating an AI-driven chatbot in an introductory programming course. *2018 IEEE Frontiers in Education Conference (FIE)*, pages 1–4, 2018.

Richard S. Wallace. *The Anatomy of A.L.I.C.E.*, pages 181–210. Springer Netherlands, Dordrecht, 2009. ISBN 978-1-4020-6710-5. doi: 10.1007/978-1-4020-6710-5_13. URL `https://doi.org/10.1007/978-1-4020-6710-5_13`.

Joseph Weizenbaum. ELIZA—a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36–45, 1966.

Bruce Wilcox and Sue Wilcox. Making it real: Loebner-winning chatbot design. *Arbor*, 189:a086, 12 2014. doi: 10.3989/arbor.2013.764n6009.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation, 2016.

Meng-Lin Yu and Meng-Han Tsai. ACS: Construction data auto-correction system—taiwan public construction data example. *Sustainability*, 13(1), 2021. ISSN 2071-1050. doi: 10.3390/su13010362.

Li Zhou, Jianfeng Gao, Di Li, and Heung-Yeung Shum. The design and implementation of XiaoIce, an empathetic social chatbot. *Computational Linguistics*, 46(1):53–93, 03 2020. ISSN 0891-2017. doi: 10.1162/coli_a_00368. URL `https://doi.org/10.1162/coli_a_00368`.

Appendices

Appendix A

Rasa Training Data

This appendix contains all the intents defined as well what purpose they serve.

- **General**
  - greet
    - The user intends to greet.
  - goodbye
    - The user intends to say goodbye.
  - affirm
    - The user intends to affirm or answer positively.
  - deny
    - The user intends to deny or answer negatively.
  - bot_challenge
    - The user intends to ask Pyo whether they are talking to a person or not.
- **Variables**
  - ask_question_var
    - The user intends to ask for the definition or an example of a variable.
  - ask_question_var_name
    - The user intends to query about what names can a variable be given or what words and symbols can it not contain.
  - ask_question_var_get_type
    - The user intends to ask of what types can a variable be or how to get its type.
  - ask_question_var_strings

    - The user intends to ask for the explanation or example of a string.
  - ask_question_var_strings_convert
    - The user intends to ask how to convert a value to a string.
  - ask_question_var_strings_concat
    - The user intends to ask what is a concatenation or how it is conducted.
  - ask_question_var_ints
    - The user intends to ask for the explanation or example of an integer.
  - ask_question_var_ints_convert
    - The user intends to ask how to convert a value to an integer.
  - ask_question_var_floats
    - The user intends to ask for the explanation or example of a float.
  - ask_question_var_floats_convert
    - The user intends to ask how to convert a value to a float.
  - ask_question_var_bools
    - The user intends to ask for the explanation or example of a boolean.
  - ask_question_var_arrays
    - The user intends to ask for the explanation or example of an array.

- **Conditions**
  - ask_cond_general
    - The user intends for the explanation or purpose of a conditional.
  - ask_cond_if
    - The user intends to ask for an explanation, syntax, or example, of a conditional if.
  - ask_cond_else
    - The user intends to ask for an explanation, syntax, or example, of a conditional else.
  - ask_cond_elif
    - The user intends to ask for an explanation, syntax, or example, of a conditional elif.
  - ask_cond_indent
    - The user intends to ask for an explanation of what the scope of a conditional is.

- **Repetitions**
  - ask_loop_general

    * The user intends for the explanation or purpose of a repetitions.
  - ask_loop_for
    * The user intends to ask for an explanation, syntax, or example, of a repetition for.
  - ask_loop_while
    * The user intends to ask for an explanation, syntax, or example, of a repetition while.
  - ask_loop_indent
    * The user intends to ask for an explanation of what the scope of a repetition is.

- **Functions**
  - ask_func_general
    * The user intends to ask for an explanation or example of a function without parameters.
  - ask_func_parameters
    * The user intends to ask for an explanation or example of a function with parameters.
  - ask_func_return
    * The user intends to ask how to have a function return data.
  - ask_func_indent
    * The user intends to ask for an explanation of what the scope of a function is.

- **Operators**
  - ask_op_arithmetic
    * The user intends to ask what operations are allowed.
  - ask_op_arithmetic_add
    * The user intends for an example or explanation on how to conduct an addition.
  - ask_op_arithmetic_sub
    * The user intends for an example or explanation on how to conduct an subtraction.
  - ask_op_arithmetic_multi
    * The user intends for an example or explanation on how to conduct a multiplication.
  - ask_op_arithmetic_div
    * The user intends for an example or explanation on how to conduct a division.

  - ask_op_arithmetic_rem
    * The user intends for an example or explanation on how to get the remainder or how to verify if a number is multiple of another.
  - ask_op_arithmetic_exp
    * The user intends for an example or explanation on how to calculate the power of a number.
  - ask_op_square_root
    * The user intends for an example or explanation on how to calculate the square root of a number.
  - ask_op_logic
    * The user intends to ask what are the logic operators.
  - ask_op_logic_and
    * The user intends to ask an explanation/example of the logic operator "and".
  - ask_op_logic_or
    * The user intends to ask an explanation/example of the logic operator "or".
  - ask_op_logic_not
    * The user intends to ask an explanation/example of the logic operator "not".
  - ask_op_comparison
    * The user intends to ask an explanation/example of comparison operators, such as to verify if a number is greater than another.
  - ask_op_order
    * The user intends to ask for the order or priority of the arithmetic operators.

- **Input and Output**
  - ask_input
    * The user intends to ask how to read a value from the keyboard, or simply what is an input.
  - ask_output
    * The user intends to ask how to print a value, or simply what is an output.

- **Exercises**
  - ask_help
    * The user intends to ask for assistance on the exercise
  - ask_hint

    - The user intends to ask for a hint.
  - ask_suggestions
    - The user intends to ask for suggestions of questions.

- **Errors**
  - ask_name_error
    - The user intends to ask what is a NameError as well as possible causes.
  - ask_syntax_error
    - The user intends to ask what is a SyntaxError as well as possible causes.
  - ask_type_error
    - The user intends to ask what is a TypeError as well as possible causes.

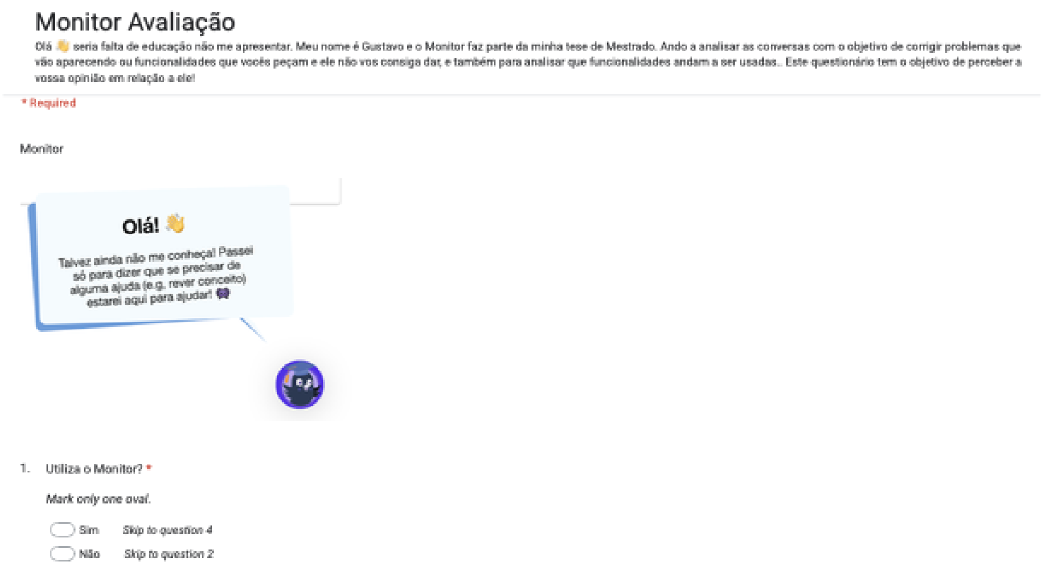
- **External** The purpose of these intents are to send data, regarding information about the exercise or errors, from the platform to Pyo, and not to represent an intent of the student.
  - EXTERNAL_ERROR_MESSAGE
  - EXTERNAL_CODE_MESSAGE
  - EXTERNAL_CODE_INFO

Appendix B

Questionnaire

This appendix includes both the original questionnaire and the untranslated responses to the open questions.

Figure B.1 displays the original question, and whether the students responded affirmatively (figure B.2) or negatively (figure B.3), they would be displayed another set of questions.



Monitor Avaliação

Olá 👋 seria falta de educação não me apresentar. Meu nome é Gustavo e o Monitor faz parte da minha tese de Mestrado. Ando a analisar as conversas com o objetivo de corrigir problemas que vão aparecendo ou funcionalidades que vocês peçam e ele não vos consiga dar, e também para analisar que funcionalidades andam a ser usadas.. Este questionário tem o objetivo de perceber a vossa opinião em relação a ele!

* Required

Monitor

Olá! 👋

Talvez ainda não me conheça! Passei só para dizer que se precisar de alguma ajuda (e.g. rever conceito) estarei aqui para ajudar!

1. Utiliza o Monitor? *

Mark only one oval.

◯ Sim *Skip to question 4*

◯ Não *Skip to question 2*

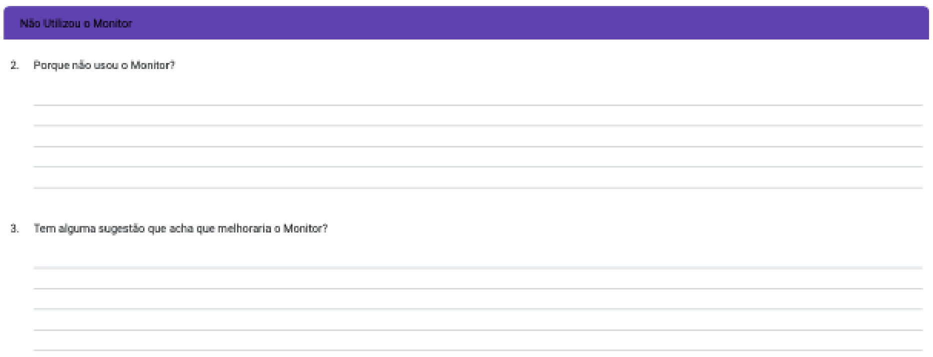
Figure B.1: Original student questionnaire, initial question.

Regarding the first question on the "no" part of the questionnaire (figure B.1) the answers were:

- Acho a Interação dele forçada. Prefiro pesquisa exemplos com o conteúdo que estou estudando para melhor entender
- Ainda não precisei
- Não precisei usar, pois não fiquei com dúvidas ainda

In regards to the second, their answers were:

- Não
- Ainda não
- Poderia ter um intervalo de tempo maior em que ele manda uma mensagem e outra, as vezes o enunciado é um pouco grande e acabo ficando mais tempo escrevendo o código e ele fica mandando mensagem de minuto em minuto, também seria bom um botão com a utilidade de desabilitar ele para ele não ficar aparecendo na tela se você ativasse o botão


Não Utilizou o Monitor

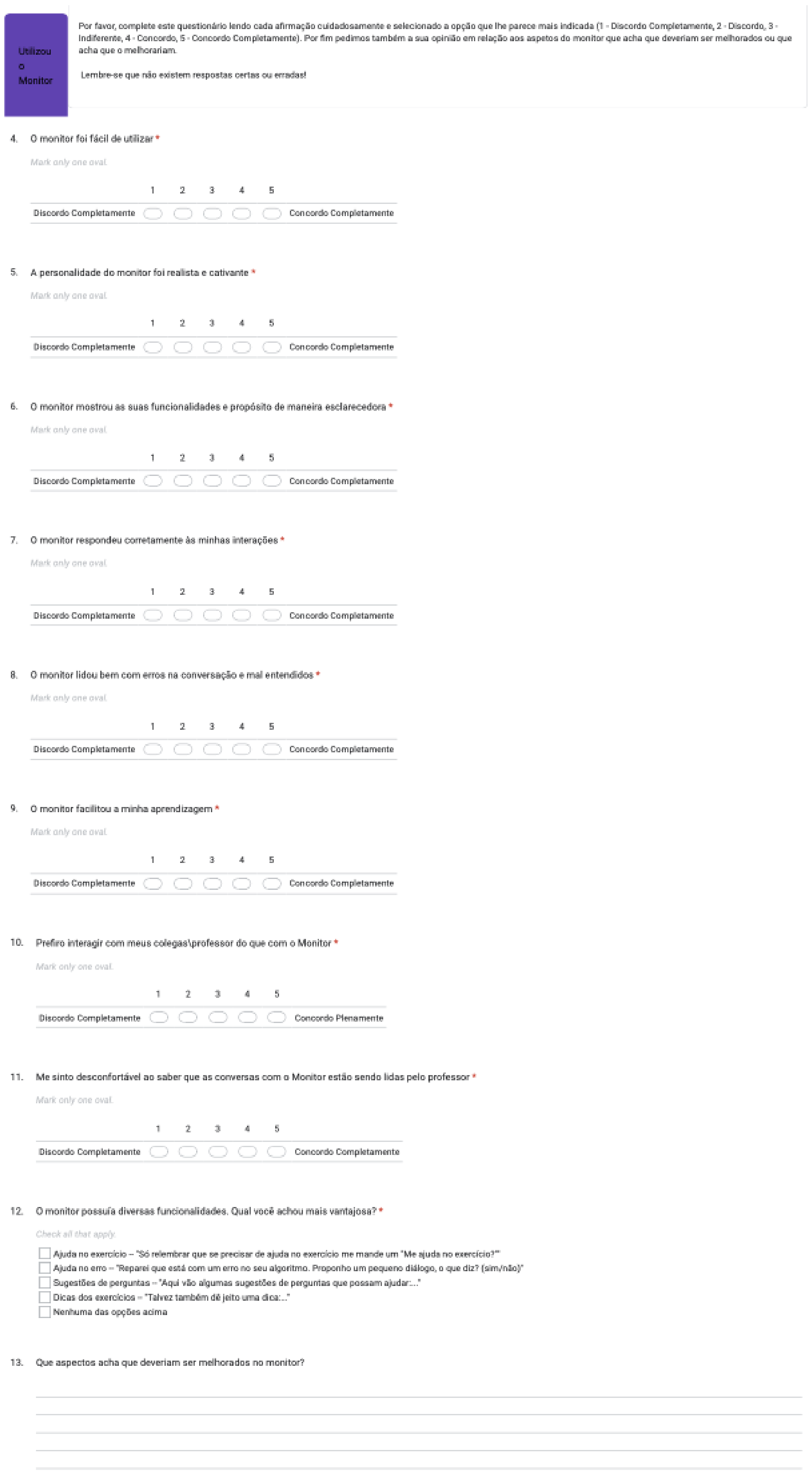
2. Porque não usou o Monitor?

3. Tem alguma sugestão que acha que melhoraria o Monitor?

Figure B.2: Original student questionnaire, questions to students that had not interacted with Pyo.

For the "yes" part of the questionnaire, the last question also prompted suggestions, in which the students gave the following answers:

- Focar nos exercícios que estamos a resolver, principalmente, nós alunos que não tivemos contato com programação. Um exemplo: se erro for na linha "4", o programa poderia nos alertar que há um erro na respectiva linha, e apresentar os conceitos relacionadas ao erro cometido, poderíamos revisar e encontrar o erro, como se um professor estivesse ali perto, nos orientando. E caso for um erro na lógica ou digitação da programação, o programa poderia dizer "Paulo o seu erro está em não ter declarado um dado" ou "você se esqueceu de imprimir seu dado, para isso use o comando print()", ou "Na linha 5 você não inseriu as aspas – (poderia vir conceito geral)". Acho que a proposta é essa do programa, porque o aluno vai tentando, e não consegue implementar a lógica certa no exercício ou devido a algum erro na digitação, e quando recorre ao programa não consegue compreender onde está o erro e acaba se perdendo.
- Da mais exemplos
- Bom, o monitor em si é ótimo, acho que o que poderia melhorar seria ter mais um monitor para ver as perguntas, ter um tempo de resposta um pouco mais rápido, ter uma interação maior aos alunos
- Sempre finalizar o atendimento diante das perguntas feitas, seja conclusivas ou não, tipo, não foi possível identificar a perguntar



Utilizou o Monitor

Por favor, complete este questionário lendo cada afirmação cuidadosamente e selecionado a opção que lhe parece mais indicada (1 - Discordo Completamente, 2 - Discordo, 3 - Indiferente, 4 - Concordo, 5 - Concordo Completamente). Por fim pedimos também a sua opinião em relação aos aspetos do monitor que acha que deveriam ser melhorados ou que acha que o melhorariam.

Lembre-se que não existem respostas certas ou erradas!

4. O monitor foi fácil de utilizar *

Mark only one oval.

	1	2	3	4	5	
Discordo Completamente	◯	◯	◯	◯	◯	Concordo Completamente

5. A personalidade do monitor foi realista e cativante *

Mark only one oval.

	1	2	3	4	5	
Discordo Completamente	◯	◯	◯	◯	◯	Concordo Completamente

6. O monitor mostrou as suas funcionalidades e propósito de maneira esclarecedora *

Mark only one oval.

	1	2	3	4	5	
Discordo Completamente	◯	◯	◯	◯	◯	Concordo Completamente

7. O monitor respondeu corretamente às minhas interações *

Mark only one oval.

	1	2	3	4	5	
Discordo Completamente	◯	◯	◯	◯	◯	Concordo Completamente

8. O monitor lidou bem com erros na conversação e mal entendidos *

Mark only one oval.

	1	2	3	4	5	
Discordo Completamente	◯	◯	◯	◯	◯	Concordo Completamente

9. O monitor facilitou a minha aprendizagem *

Mark only one oval.

	1	2	3	4	5	
Discordo Completamente	◯	◯	◯	◯	◯	Concordo Completamente

10. Prefiro interagir com meus colegas\professor do que com o Monitor *

Mark only one oval.

	1	2	3	4	5	
Discordo Completamente	◯	◯	◯	◯	◯	Concordo Plenamente

11. Me sinto desconfortável ao saber que as conversas com o Monitor estão sendo lidas pelo professor *

Mark only one oval.

	1	2	3	4	5	
Discordo Completamente	◯	◯	◯	◯	◯	Concordo Completamente

12. O monitor possuía diversas funcionalidades. Qual você achou mais vantajosa? *

Check all that apply.

- ☐ Ajuda no exercício – "Só relembrar que se precisar de ajuda no exercício me mande um "Me ajuda no exercício?""
- ☐ Ajuda no erro – "Reparei que está com um erro no seu algoritmo. Proponho um pequeno diálogo, o que diz? (sim/não)"
- ☐ Sugestões de perguntas – "Aqui vão algumas sugestões de perguntas que possam ajudar:..."
- ☐ Dicas dos exercícios – "Talvez também dê jeito uma dica:..."
- ☐ Nenhuma das opções acima

13. Que aspectos acha que deveriam ser melhorados no monitor?

Figure B.3: Original student questionnaire, questions to students that had interacted with Pyo