UNIVERSITÀ DI PISA

Master of Science in
Artificial Intelligence and Data Engineering

Automatic multi-choice quiz generation and evaluation using Generative AI and NLP

G.Bello, J.Niccolai, A.Ruggieri

June 6, 2023

Contents

Chapter 1

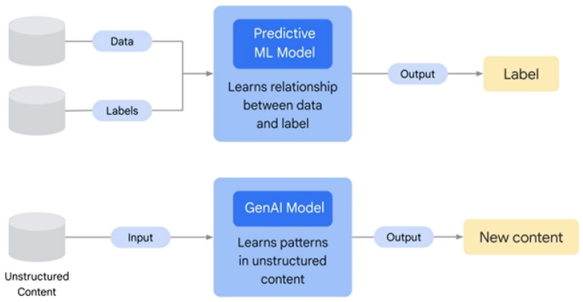
Introduction description

1.1 Generative AI

1.1.1 Introduction to Generative AI

Artificial Intelligence is the branch of computer science that deals with the creation of intelligence agents, which are systems that can reason, learn and act autonomously. Essentially, AI has to deal with theories and methods to build machines that thinks and acts like humans.
Subfield of AI is Machine Learning, which aim is model training from input data to gain as output useful predictions on never before seen data, drawn from the same ones used to train the model, which gives the computer the ability to learn whitout actually programming. Since Machine Learning requires handcrafted features to be extracted from data before learning process begins, Deep Learning extract relevant features directly from raw data, avoiding need for manual feature engineering.
Two main Deep Learning model types are:

1. **Discriminative**
    - Used to classify and predict
    - Typically trained on a labeled dataset (supervised learning)
    - Learnes the relationship between features of the data points and labels
2. **Generative**
    - Generates new data instances similar to data it was trained on
    - Understand the distribution of data and how likely a given example is
    - Predict next word in sequence



In conclusion, wave of Neural Network (highly explored since 2012) can help on objects distinction and results prediction but Generative AI coming allow users to generate own contents as text and multimedia (e.g. images, audio and so on), helped by programs trained with very large data from multiple sources across internet to provide answers simply asking question typed into a prompt (even verbally talking to a prompt).

- Training: learning process from existing content, which results in the creation of a statistical model
- When given a prompt, GenAI uses it to predict what an expected response might be and generates new content

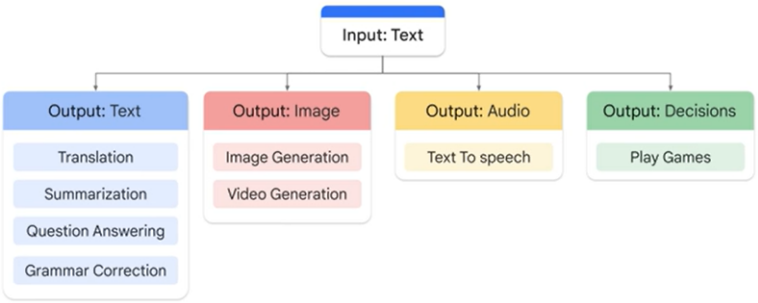
$$y = f(x) \tag{1.1}$$

Where:

- y is model output
- f is model function
- x are input data

When can properly call it Generative AI when y is Natural Language or Image/Audio.

1.1.2 Generative AI applications



Since there are many usage models, two main categories are the ones for images and text:

1. **Generative Image Models**: given a prompt or related imagery, they transform random noise into images or generates new images from prompts
2. **Generative Language Models**: given some text, predict what comes next
    - Text-to-Text Model: take natural language input and produce text as output; these models are trained to learn and mapping between pair of text
    - Applications:
        - **Simulation and Scenario Generation**: Generative AI can create simulated scenarios for testing and training purposes.
        - **Creative Content Generation**: Generative AI enables the production of novel and creative content. It can generate realistic images, imaginative artwork, or even generate entirely new text passages that resemble human-written content.
        - **Creative Assistance and Inspiration**: Generative AI can serve as a tool for creative professionals, offering assistance and inspiration.
        - **Classification**, **Summarization** and **Clustering**
        - **Extraction** and **(Re)Search**
        - **Content editing** and **Translation**

1.2 Natural Language Processing

1.2.1 Introduction to NLP

Natural Language Processing is the field of Artificial Intelligence that focuses on the relationship between computers and human language. Combining the study of computational linguistics with statistical, machine and deep learning models, the aim is let computers efficiently process human language in the form of the text or voice data and understand its fully meaning, complete with user intentions. Even if human language is filled with ambiguities that make it incredibly difficult to write software that accurately determines the intended meaning of text or voice data, referring to homonyms, homophones, sarcasm, idioms, metaphors, grammar and usage exceptions, programmers must teach natural language-driven applications to recognize and understand accurately from the start, if those applications are going to be useful.

1.2.2 NLP applications

The importance of NLP lies in its ability to unlock valuable insights, automate tasks, and facilitate communication between humans and machines. It has become a critical technology in various applications, including:

- **Information Retrieval**: NLP enables efficient searching and retrieval of relevant information from large volumes of text data. Search engines utilize NLP techniques to understand user queries and provide accurate and relevant search results

- **Text Classification and Sentiment Analysis**: NLP allows for the classification of text documents into predefined categories or the analysis of sentiment and opinions expressed in text
- **Machine Translation**: NLP plays a crucial role in machine translation, enabling the automatic translation of text from one language to anotherd
- **Question Answering Systems**: NLP techniques are employed in question answering systems, where machines attempt to understand user questions and provide relevant and accurate answers
- **Text Summarization**: NLP can automatically generate concise summaries from long documents or articles, providing a quick overview of the main points
- **Speech Recognition and Language Generation**: NLP techniques are employed in speech recognition systems that convert spoken language into written text

While NLP has made significant progress in recent years, challenges remain. Ambiguity, sarcasm, cultural context, and understanding subtle nuances in language pose ongoing research problems. Additionally, ethical considerations, such as bias detection and fairness in NLP applications, are crucial for responsible development and deployment.

1.3 Generative AI on quiz generation

1.3.1 Aim of the project

Multiple-choice quizzes and tests are widely recognized and utilized as valuable assessment tools in various fields, including work, school, and academia. Their popularity stems from their efficiency, objectivity, ability to promote critical thinking, and adaptability across diverse subject areas. By presenting predefined options and requiring respondents to select the most accurate answer, multiple-choice assessments offer a time-saving and scalable method to measure comprehension and evaluate.
Due to the large necessity of new exam text for schools every year, one of most useful application of Generative AI in academic field is the generation of test. Starting from an input text, for example lecture notes, Generative AI is able to read it learning information about, then to produce any kind questions.
This application can bring many advantages for both professors and candidates: since students can try to test their own abilities using simple tools to prepare exam-like exercises, handling some specific parameters, professors can decide same level of difficulty for every call.
By extracting relevant information from academic papers, a quiz generation system can create quizzes that cover a wide range of topics and levels of difficulty. These quizzes can be designed to test students' comprehension, critical thinking skills, and ability to apply knowledge in real-world scenarios. By incorporating real academic content, the quizzes not only provide a comprehensive assessment but also expose students to the types of texts they will encounter in their academic journeys.

Furthermore, by automating the quiz generation process, the system can save significant time and effort for educators. Instead of manually creating quizzes, teachers can utilize the power of artificial intelligence to generate personalized quizzes tailored to the specific needs of their students.
Our team is supposed to find a simple solution of this issue providing an application with, given an input text, generates multiple choice quiz.

1.3.2 Bloom's taxonomy

The Revised Bloom's Taxonomy is a widely recognized and used framework in academic field that categorizes learning objectives and cognitive processes into a hierarchical structure.
The taxonomy consists of six levels, each representing a different cognitive domain. These levels, arranged in ascending order of complexity, provide educators with a structured framework to design instructional activities, set learning objectives, and assess students' progress.
The levels of the Revised Bloom's Taxonomy are:

1. Remembering: these questions mainly check if the student has base knowledge of topic's key-words, focusing on the ability to recall or recognize facts, information, or concepts. It involves remembering previously learned material, such as definitions, facts, or lists

2. Understanding: require the skill of combining different keywords definitions, memory is not enough without comprehension

3. Applying: Need application of definitions on specific situations.

4. Analyzing: Given a scenario as input, student must know which skills, definitions, information use to analyze the situation. It involves examining the underlying structure, identifying patterns, and drawing conclusions

5. Evaluating: this level requires students to critically evaluate ideas, arguments, or solutions and provide reasoned judgments or recommendations

6. Creating: highest level of the taxonomy, creating, involves generating new ideas, products, or solutions. It requires students to use their knowledge and skills to produce original work, develop innovative solutions, or create new interpretations or designs

This project uses Bloom's taxonomy to categorize questions and let the user ask for exact number of every level, which can for example help to difficulty regulation.
Since taxonomy has been subject of discussion for decades, there are different levels for each different version, starting from the five in 1950's until the six in 2010's. This project uses by default the Revised Bloom's taxonomy by 2012, but there's the possibility to let the user apply favourite taxonomy by changing the number, and so the relative definitions, of the levels in `main.py` class.

Chapter 2

Description of the process

2.1 Requirements and Dataset

This program is implemented using 3.10.11 Python version and requires some specif libraries, for example the ones for NLP language processing, which names are included in requirements.txt provided at following github link:
For the dataset, test were done with multiple academic materials, starting from simple files with .txt extension to PDF, but mind that this kind of file is useful only if text can be read by Python script, due to the images reading issue which is extremely difficult to handle.
To launch the application, an API key is need, is possible to generate one via `https://platform.openai.com/`.
Full project folder is available at
`https://github.com/BaffoBello14/QuizGenAIrator`

2.2 Application structure

- Input folder
  - `extracted_plain_text.txt`, text extracted from document given as input
  - `language_query.txt`, query to extract language from document given as input
  - `main_query.txt`, query for quiz generation
  - `refactor_query.txt`, refactor query for final quiz generation
- Output folder
  - `final_quiz.txt`, final quiz with:
    - Question
    - Possible answers and correct answer
    - Bloom's taxonomy
    - Score
  - `raw_quiz.txt`, temporary quiz with maximum number of question generated for each level (format might not be the correct one)

- Results folder
  - `answer.txt`, all answers
  - `question.txt`, all questions
- Python classes:
  - `main.py`
    - User interface, must specify:
      - file input path via application dialogue
      - number of questions for every level of Bloom's taxonomy
      - API key

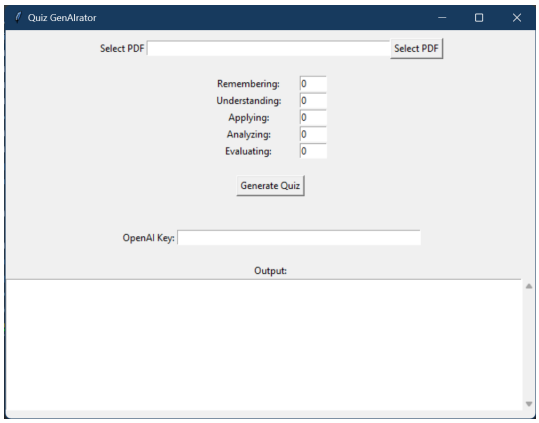


Figure 2.1

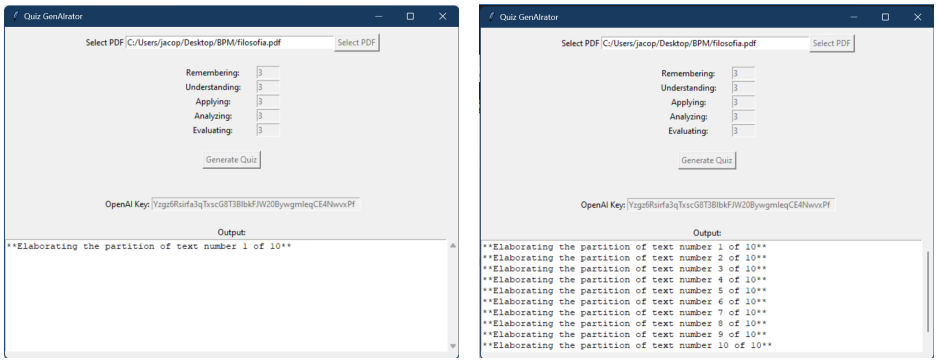
    - Reads document given as input using `pdf_reader.py`
    - Giving as input list of Bloom's taxonomy levels and number of question for every level, calls `quiz_generator.py`
    - Giving as input quiz extracted from `quiz_generator.py` and document taken as starting input at the beginning, calls `quiz_analyzer.py`
  - `pdf_reader.py`
    - Reads the text given as input from `main.py`
    - Writes in `extracted_plain_text.txt` text just read
  - `question_class.py`
    - Allow to instantiate Question object with relatives functions. `Question` has following attributes:
      - `text`: question's text
      - `answers`: list of possible answers
      - `correct_answer`: index of correct answer
      - `level`: index of Bloom's taxonomy level
      - `score`: relative score, set zero by default
  - `quiz_analyzer.py`
    - Text and quiz language detection and relative library installation

    - Compute calculation of parameters needed for quiz evaluation
- `quiz_class.py`
    - Allow to instantiate Quiz object with relatives functions.
      Quiz has following attributes:
        - language: language detected
        - Questions: list of Question objects
    - This class also provides methods to:
        - Select best questions for every level using score
        - Write
            1. All questions in `questions.txt`
            2. All answers in `answers.txt`
            3. Final quiz in `final_quiz.txt`
        - Results generation
- `quiz_generator.py`
    - Contains the reference to the openai API key
    - Uses `tiktoken` library to count tokens and split text by chars
    - Generates quiz
    - Combining refactor query and raw quiz, refactors quiz

2.3 Application workflow

1. `main.py`
    (a) Starts application
    (b) Declares Bloom taxonomy levels
    (c) Asks user to specify document input path
    (d) Instantiates a PDFreader `pdf_reader` object given as input the file path just specified and calls `process_pdf()` function
2. `pdf_reader.py`
    (a) `process_pdf()`:
        - Removing useless spaces and empty lines, reads the document
        - Writes in file `extracted_plain_text.txt` the text
3. `main.py`
    (a) Instantiate an object `quiz_generator` taking as input number of question for each Bloom's taxonomy level and list of relatives levels
    (b) Calls `quiz_generator.generate()` method
4. `quiz_generator.py`
    (a) `generate()`:
        - Divides the text into overlapping partition

- Text is divided into partitions due to the Chat API limit of number of tokens read for the conversation. If this limit of maximum tokens is not respected, application will not provide questions about not considered tokens
- Overlapped partitions are calculated to consider all text: from each partitions program will provide questions, so if for example a chapter cannot be completely read in only one partition this one will be split into two sequential partitions. Overlapped partitions are introduced to avoid this problem and ensure the user to have questions of all text

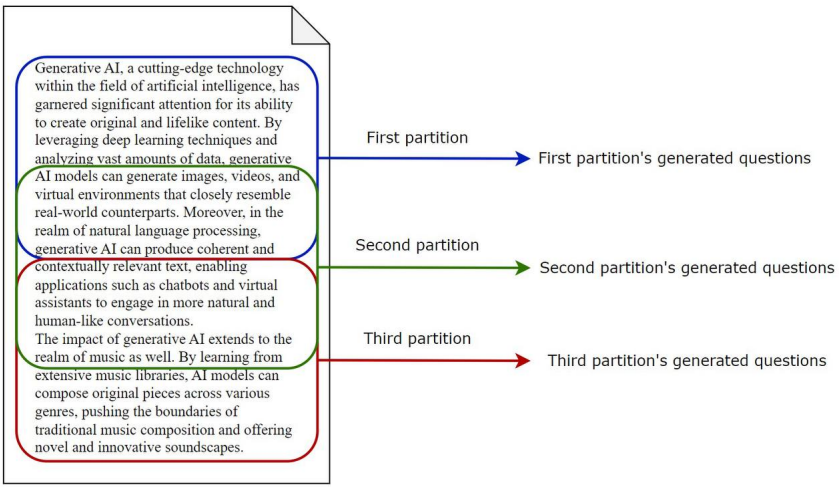


(a) (b)

- Calculates the total number of questions needed for the given number of partitions
  - Considering partitions $1, ..k$, levels $1, .., i$ and $p \in [1, k], m \in [1, i]$, for every partition Number of Questions z for m level per partition p is

$$z_m = \lceil \frac{\text{Number of questions per level}_m}{\text{Number of partition}} \rceil \tag{2.1}$$

    * This computation allows the programm have the minimum number of questions for every level. Starting form these, application will provide a standing of the questions for every level and select the best ones
  - `temp_query` is the result of concatenation of string to represent the query the user want to provide to Generative AI API
- Determines the number of questions for each level in each partition and updates the `temp_query`
  e.g. "Generate" + `number_of_question_for_level` + "questions classified by Bloom taxonomy:"
- Determines the number of questions for each level in each partition and update the `temp_query`
- Generates questions for each partition
  - Giving complete query, calls Chat API to get the AI response



  - Writes the generated content to the output file `raw_quiz.txt`
  - For every partition calls partition_refactor(), giving as input the number of question for each level
  - After calling `partition_refactor()`, verifies if number of questions is correct:
    - If it is correct, appends partition questions on `raw_quiz.txt`
    - If it is not, recomputes the function
- `partition_refactor()`:
  - Calculates the total number of questions needed for refactoring
  - Creates a prompt for the refactoring step
  - Queries the AI model for refactoring the quiz

5. `main.py`

   (a) Quiz object declaration passing to it the language in which the quiz is written

   (b) QuizAnalyzer declaration passing to it the generated quiz and the starting text

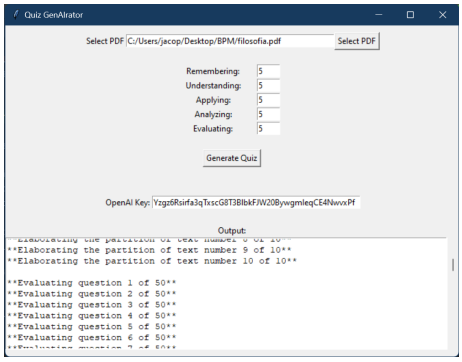
   (c) Calls `quiz_analyzer.calculate_weighted_standing()`

6. `quiz_analyzer.py`

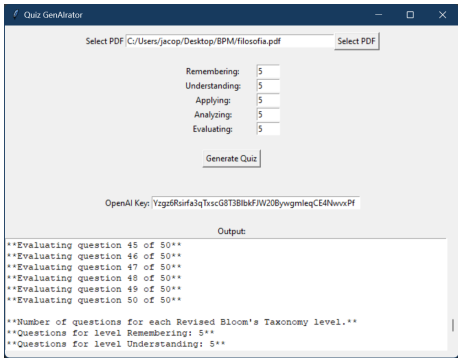
   (a) `calculate_weighted_standing()`:

   - Computes parameters for test evaluation:
     - **Similarity**: Comparison between input text and questions + answers
       - Weight = 0.6
     - **Coherence:** Comparison between each question and relative answers
       - Weight = 0.3

- **Clarity**: Calculates the average token vector similarity within the question
  - Weight = 0.1
- Calculates the weighted standing
- Setting the score on the Question object



Quiz evaluation (1)



Quiz evaluation (2)

7. `main.py`
   (a) Implicit selection of desired number of questions for each level from the quiz (avoiding lowest graduated ones)
   (b) Calls `quiz.generate_files()` by `quiz_class.py`
8. `quiz_class.py`
   (a) Writes all questions in `questions.txt`
   (b) Writes all answers in `answers.txt`
   (c) Writes final quiz in `final_quiz.txt`

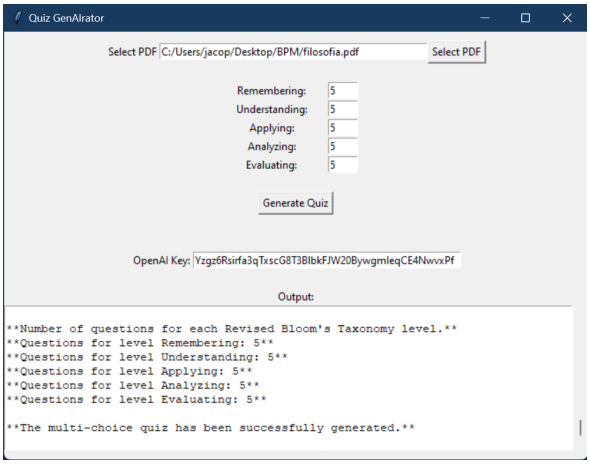


Figure 2.2

2.3.1 Queries

Main Query

```
Provide me multiple choice quiz with questions that respects the following
requirements:
each question must be unique,
for each question must generate 4 possible answers of which only one must be
the correct one.
Specify after the fourth answer the correct answer and the level of Bloom Taxonomy.
The format for the lists must be the same as following:
1.  < question text > ?
A) < option A >
B) < option B >
C) < option C >
D) < option D >
Correct answer:  < correct option >
Revised Bloom's Taxonomy level:  Understanding
Only empty line after the correct answer, before the next question.
You don't have to put an empty line between last option and the correct answer.
Questions must be impersonal, must not be related to the previous text but
only must be related to the following text:
```

then the program append text read by input text.

Refactor Query

```
Starting from the previous quiz
you have to remove any empty line between last option and the correct answer.
The format for the single question must be as it follows:
1.  < question text > ?
A) < option A >
B) < option B >
C) < option C >
D) < option D >
Correct answer:  C
Revised Bloom's Taxonomy level:  Understanding

2.  < question text > ?
A) < option A >
B) < option B >
C) < option C >
D) < option D >
Correct answer:  A
Revised Bloom's Taxonomy level:  Remebering

and so on for the correct number of questions.
Do it for each question and do not delete any question.
The questions must be the same provided,
so the number of questions must be
```

then the program append the correct number of questions.

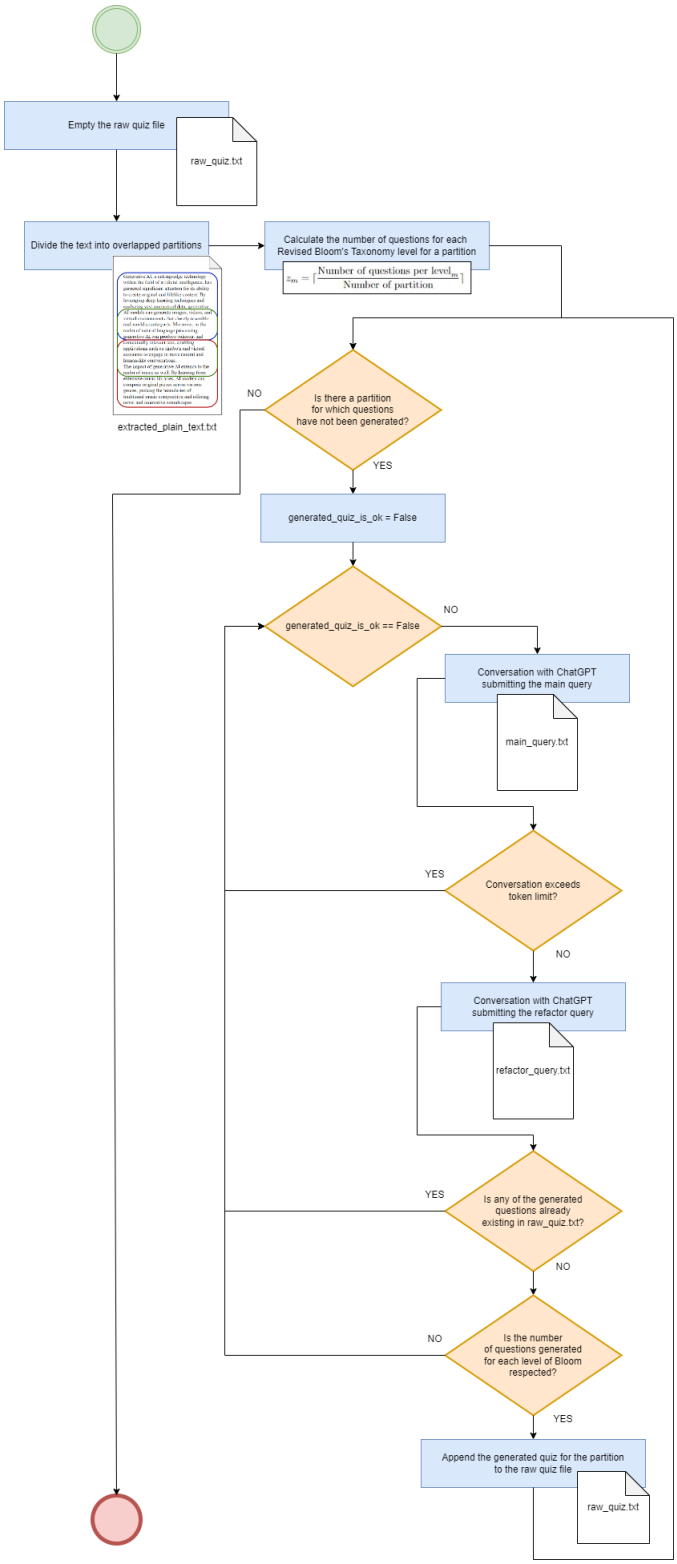


Figure 2.3: **Application flowchart for questions and answers generation**

Chapter 3

Discussion on the result

3.1 Testing data set

For testing phase, many documents were used as:

- IOT university lecture presentation, starting from 5 up to 20 slides
- Business Project Management lecture presentation, different modules between 20 and 30 pages
- Philosophy introduction, 15 pages

Some test files are available in `PDF` GitHub folder.

3.2 Results

3.2.1 Relationship between number of partitions and number of questions

$$f(x) = \lceil \frac{x}{k} \rceil * k - x \tag{3.1}$$

Where:

- k = number of partitions
- x = number of question for a Bloom Taxonomy level

This function provides difference between number of generated and required questions for a Bloom Taxonomy level.

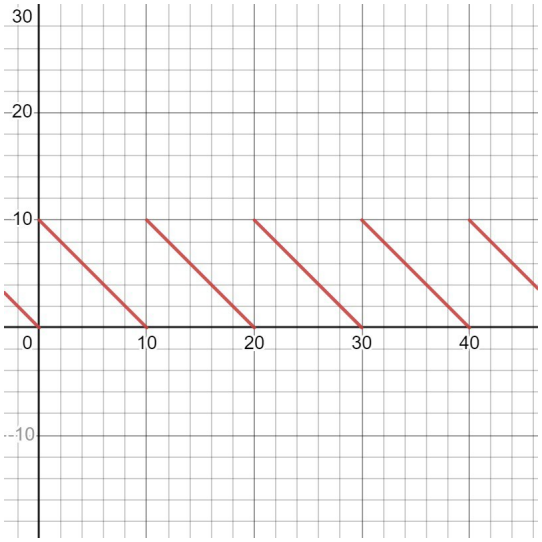


Figure 3.1: $f(x)$ computed for $k = 10$

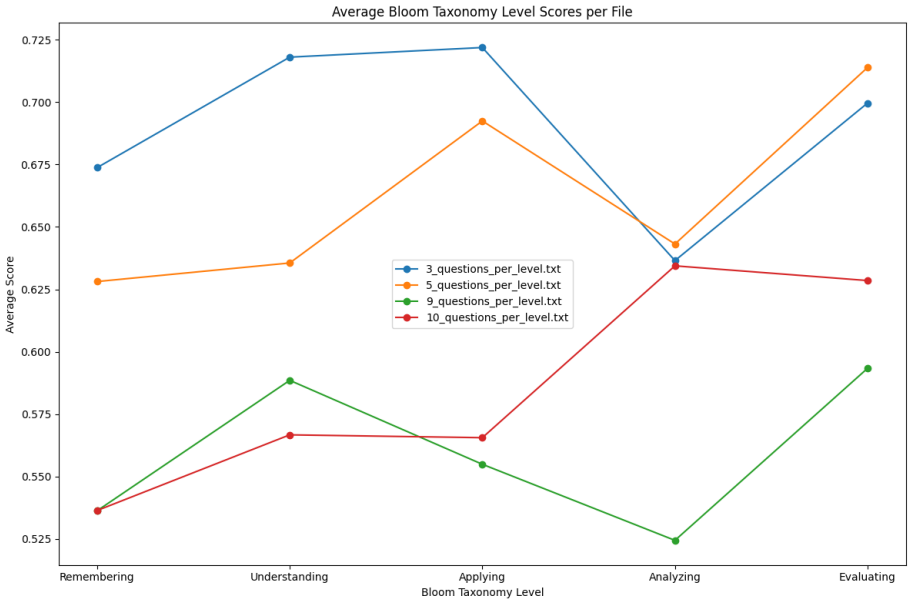
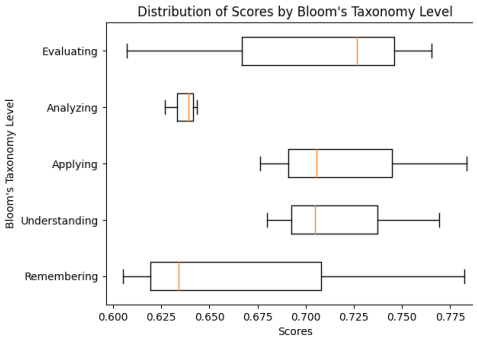
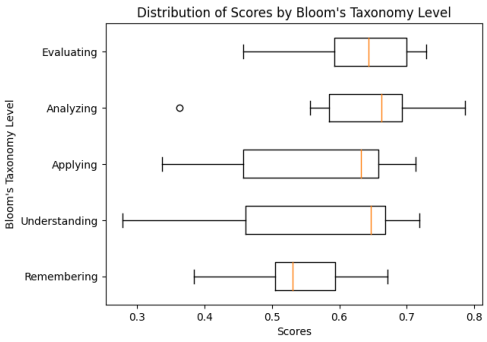


Figure 3.2: Plot shows average score per every Bloom Taxonomy level for every file

Fig.3.2 test, using number of questions x in the interval $x \in (0, 10]$, shows that increasing of questions' number in a range between $(i * k, (i + 1) * k]$ with i starting from zero, average score decreases. This is the reason why the lowest average score corresponds to the maximum interval value and the highest score represents the lowest interval value.
Increasing x, number of questions for a level in the relative interval, the difference between generated and selected questions decreases: the less is this difference the less will average score be.



3 question per level philosophy



10 question per level philosophy

Presence of outlier in 10 question case is due to the number of questions for Bloom Taxonomy level: since in both ways were used same input text so same number of partitions $k = 10$, coming to number of questions x equal to 10 proves previous statement

$\Longrightarrow$ Increasing x and bring it near k value has two main consequences:

1. Average score decreases
2. Presence of outliers

3.2.2 Analysis of different topics behaviour

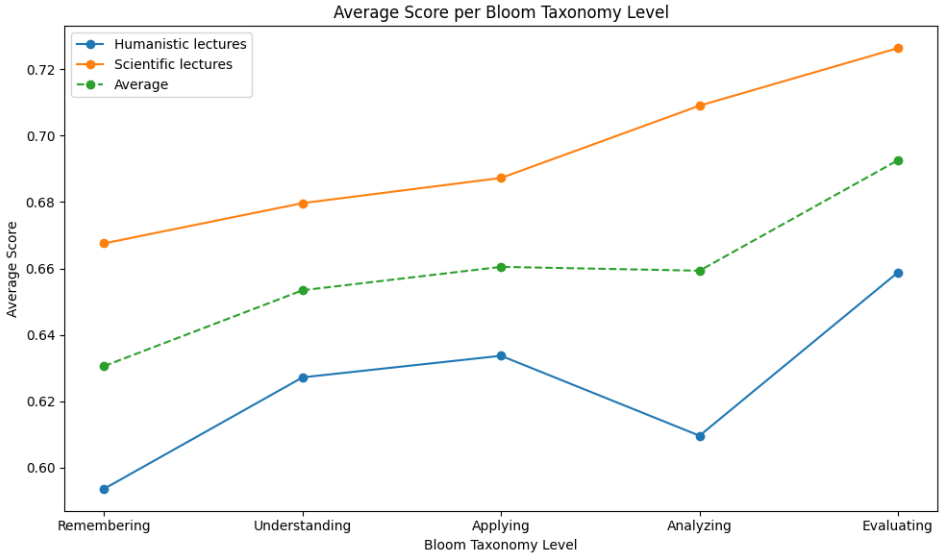


Figure 3.3: Plot of average score per Bloom Taxonomy level for different topics

Since humanistic area has lower average scores, scientific lectures seem easier to processing maybe due to the necessity to use specific words to explain concepts, there are no multiple ways to explain same statement.

3.3 Considerations

3.3.1 Academic application

Thanks to results obtained, application can be considered ready to run academic application of Generative AI wave, bringing a standardized method for tests generation.

- **Evaluation**: AI-generated quizzes can be used as formative or summative assessments to evaluate students' understanding of course material. These quizzes can cover a wide range of topics, provide immediate feedback to students, and help instructors identify areas where students may be struggling
- **Study and practice**: Students can use AI-generated quizzes as study aids to reinforce their learning and practice their knowledge. These quizzes can be tailored to specific topics or chapters, allowing students to test their understanding and identify areas that require further review
- **Personalized learning**: By generating quizzes based on individual student needs and performance, AI can support personalized learning. Adaptive quizzes can be designed to challenge students at an appropriate difficulty level, provide targeted feedback, and track their progress over time

3.3.2 Economic application

One real-life economic application of generating multiple-choice quizzes using AI is in the field of human resources and job interviews. Many companies use multiple-choice questionnaires as part of their screening and selection process for job applicants. These quizzes can assess candidates' knowledge and understanding of specific job-related topics or industry-related concepts.
The application can analyze job descriptions, required skills, and industry knowledge to generate questions that accurately evaluate candidates' qualifications. This approach ensures consistency and objectivity in the assessment process, allowing companies to compare candidates fairly and efficiently.
Also curriculum enhancement can be helped by this application: AI-generated quizzes can contribute to the development and enhancement of curriculum materials. Instructors can use the generated quizzes to identify gaps in their teaching materials, assess the effectiveness of instructional strategies, and refine their course content

3.4 Future updates

- **Files processing improvements:** improve the PDF processing functionality to handle different types of PDF documents, such as scanned PDFs or documents with complex formatting, to avoid the problem of non-selectable text in files and enlarge the wide area of useful documents as input. Consider using OCR (Optical Character Recognition) techniques to extract text accurately
- **Language support:** implementing automatic translation from input text language and user selected language to allow users to specify the language of the input document and generate quizzes in multiple languages
- **Answer validation:** Implement a mechanism to validate answers provided by users, involving NLP techniques to compare user answers against model answers
- **Customization Options:** implement a mechanism to validate answers provided by users, involving NLP techniques to compare user answers against model answers
- **Performance optimization:** optimize the application's performance, especially for large documents or when dealing with a high volume of questions. This could involve optimizing algorithms, implementing parallel processing, or utilizing cloud-based resources for scalability
- **Security and privacy:** Ensure proper security measures are in place to protect user data and confidential documents. Implement measures such as data encryption, secure file handling, and user authentication
- **Community:** create a community for both professor and students where is possible to find quiz already generated from other users about same topic. A community can also help to boost Network Effect, increasing sales and creating a feedback mechanism speeding up the application improvements