AutoQuizzer Report

Léo TORRADO

Introduction

This is my report for my machine learning project on the AutoQuizzer API and my implementation to a Web application that generates quizzes only using an url given by the user.
I chose to work on this project because I believe it has a real-world use in helping people test and evaluate knowledge. For example, to make it easier and faster for teachers to create exam questions or for students who wish to evaluate their own knowledge to prepare for an exam.
In this report, I will explain different aspects of the project but will prioritize the functioning of the machine learning components.

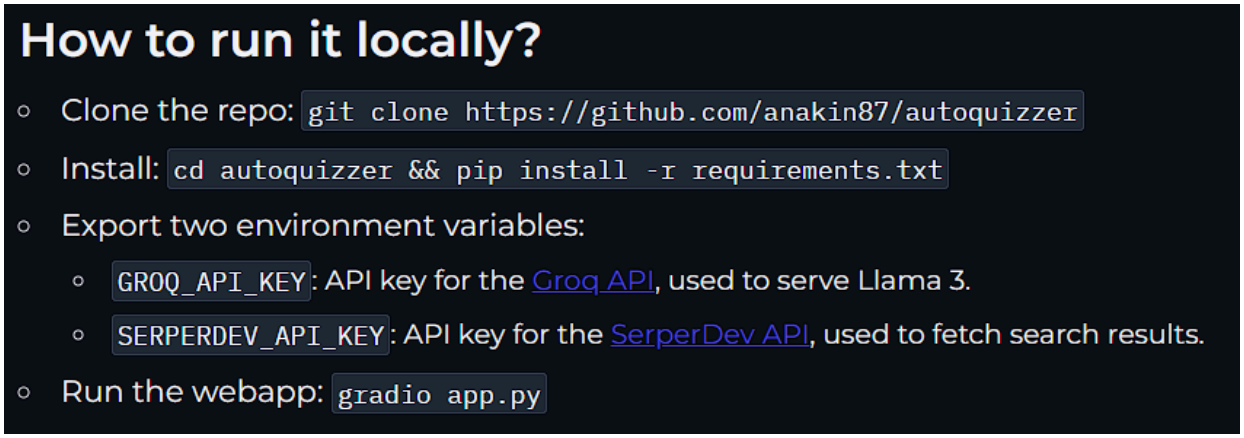
First, let's introduce deepset, the company that created the AutoQuizzer API. They published it as a Space on HuggingFace in order to advertise one of their own products which they used to create AutoQuizzer. This product is Haystack, the framework developed by deepset, specialized to build AI systems that uses LLMs. Examples of such systems and uses include advanced RAGs, chatbots/agents, multi-modal generation (image, text, sound) and information extraction.
[What is Haystack? | Haystack (deepset.ai)]

Let’s move on to SerpAPI, the Google Search API developed by Serper. SerpAPI is a ‘fast-and-easy’ API used to make searches on Google to supply it to a program that will make use of the information obtained through those searches. Some examples of its uses are the ability to gather information on websites using parameters given by the user such as a url, keywords or a location.
SerpApi: Google Search API

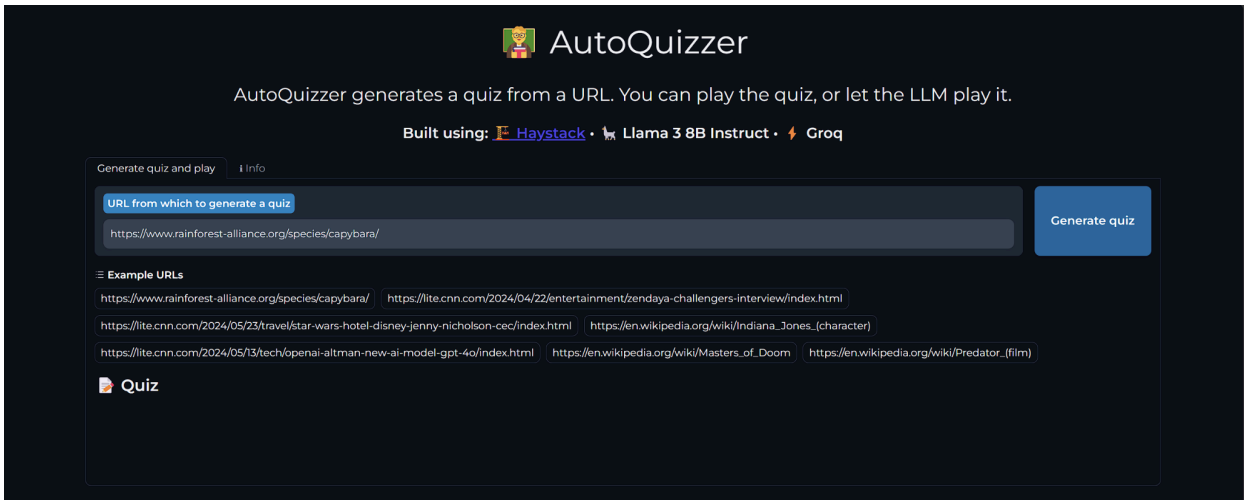
Now, let’s take a look at the more important components of both the project and the future of machine learning. Groq is a revolutionary company for AI as they are responsible for building the currently fastest AI inference technology, about 10 times faster than concurrents (GPT). They achieved this by building LPUs (Language Processing Unit), these are chips similar to the CPUs and GPUs which run LLMs and Inference Engines except that they are built specifically for the purpose of AI technology. That way, Groq was able to create processing units that overcome two major LLM bottlenecks : compute density and memory bandwidth. The company developed their own Inference Engine to showcase the superiority of their technology.
Groq is Fast AI Inference

Finally, we have the LLM that powers the API, LLAMA 3. LLAMA 3 is the latest published version of the LLM developed by Meta AI. Like other Big Tech companies, Meta's goal is to get ahead of other companies in AI technology and build the most performant LLMs and Inference Engine, For this, they have developed LLAMA 3, a direct concurrent to OpenAI's GPT.
Meta Llama 3

Installation, implementation, development and explanations



After cloning the AutoQuizzer repo, installing the requirements and generating and exporting both my Groq and SerpAPI keys, I was ready to implement the API to my Web application. From there, I was able to run it locally and did some basic testing to see if the installation was done correctly so I could get started on implementing and developing features for the Web solution.



The main feature of the webapp is to generate a quiz of 5 questions using the url given by the user. The process for this functionality and the app in general is as follows :

1. The webapp is created using Gradio which is a Python package that focuses on creating interfaces and UI for learning models and APIs. As such, the webapp is coded in Python and uses the Gradio library for the custom user interface.

2. The url given by the user is fetched using SerpAPI which extracts the html source code of the corresponding Web page. From this html code, the webapp

separates the usable content of the Web site from the rest, like <html> tags and creates a text document of the information found.

3. The next step is using a Haystack component called PromptBuilder. This allows us to write a template to give to the LLM, containing detailed instructions in natural language as well as an example of what a good answer is, in our case, a JSON file containing 5 questions and exactly 4 different possible answers, among these exactly 1 answer is the correct one for each question. (see screenshot below)

```
quiz_generation_template = """Given the following text, create 5 multiple choice quizzes in JSON format.
Each question should have 4 different options, and only one of them should be correct.
The options should be unambiguous.
Each option should begin with a letter followed by a period and a space (e.g., "a. option").
The question should also briefly mention the general topic of the text so that it can be understood in isolation.
Each question should not give hints to answer the other questions.
Include challenging questions, which require reasoning.

respond with JSON only, no markdown or descriptions.

example JSON format you should absolutely follow:
{"topic": "a sentence explaining the topic of the text",
 "questions":
  [
    {
      "question": "text of the question",
      "options": ["a. 1st option", "b. 2nd option", "c. 3rd option", "d. 4th option"],
      "right_option": "c"  # letter of the right option ("a" for the first, "b" for the second, etc.)
    }, ...
  ]
}

text:
{% for doc in documents %}{{ doc.content|truncate(4000) }}{% endfor %}
"""
```

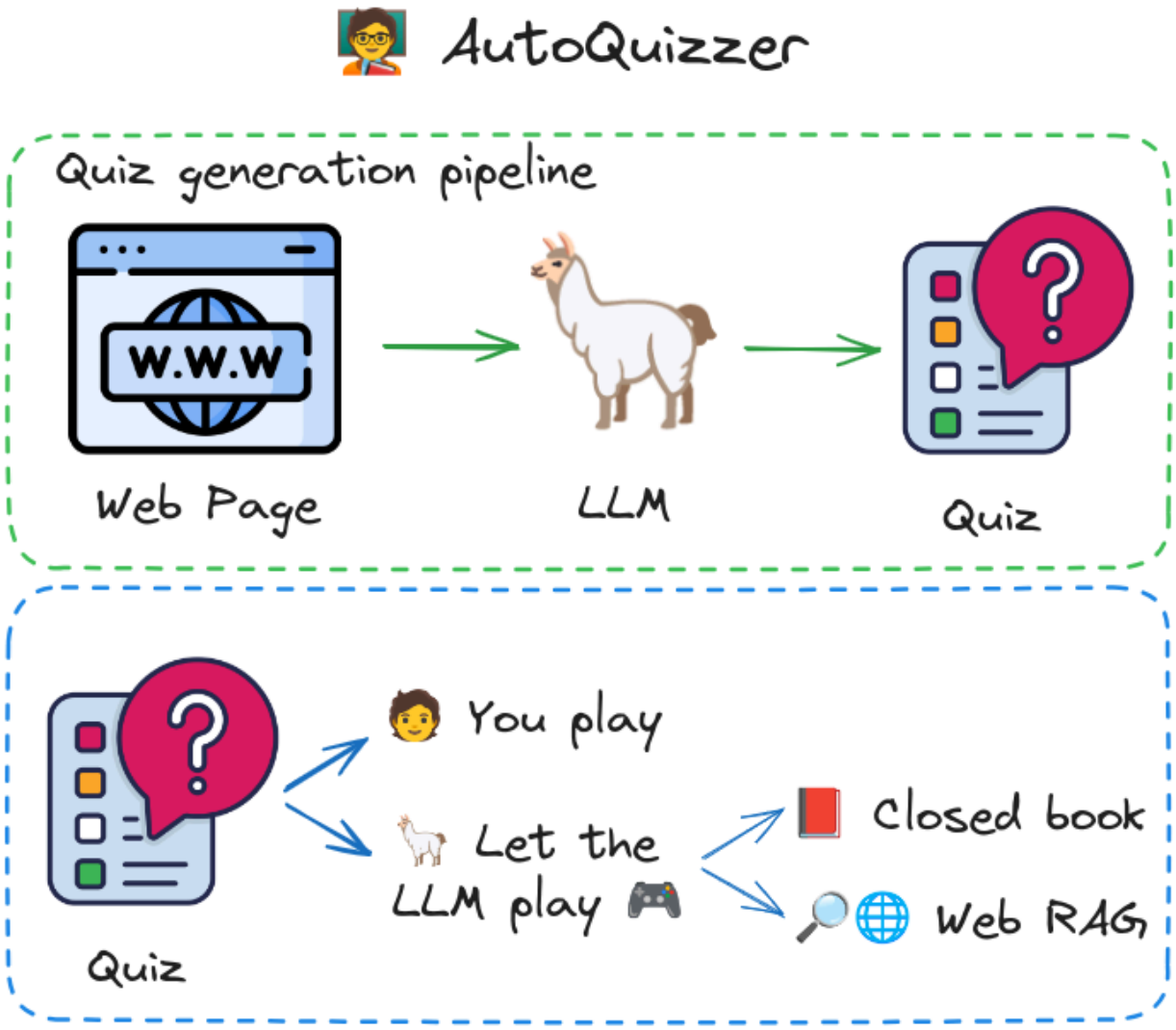
4. After this, it is up to the LLM to do the work. Llama 3, served by Groq, goes through the process of inferring using the prompt and the text document then returns a corresponding quiz including the questions, the possible options and the answer of each question.

5. Finally, the webapp displays the interface which allows the user to choose and submit their answers. It then checks how many questions the user answered correctly and displays their score. From here, the user can choose to stop if they are satisfied with their knowledge or regenerate the quiz to get new questions, they can also enter another url to change the source of the information, making sure that the questions are more diverse.

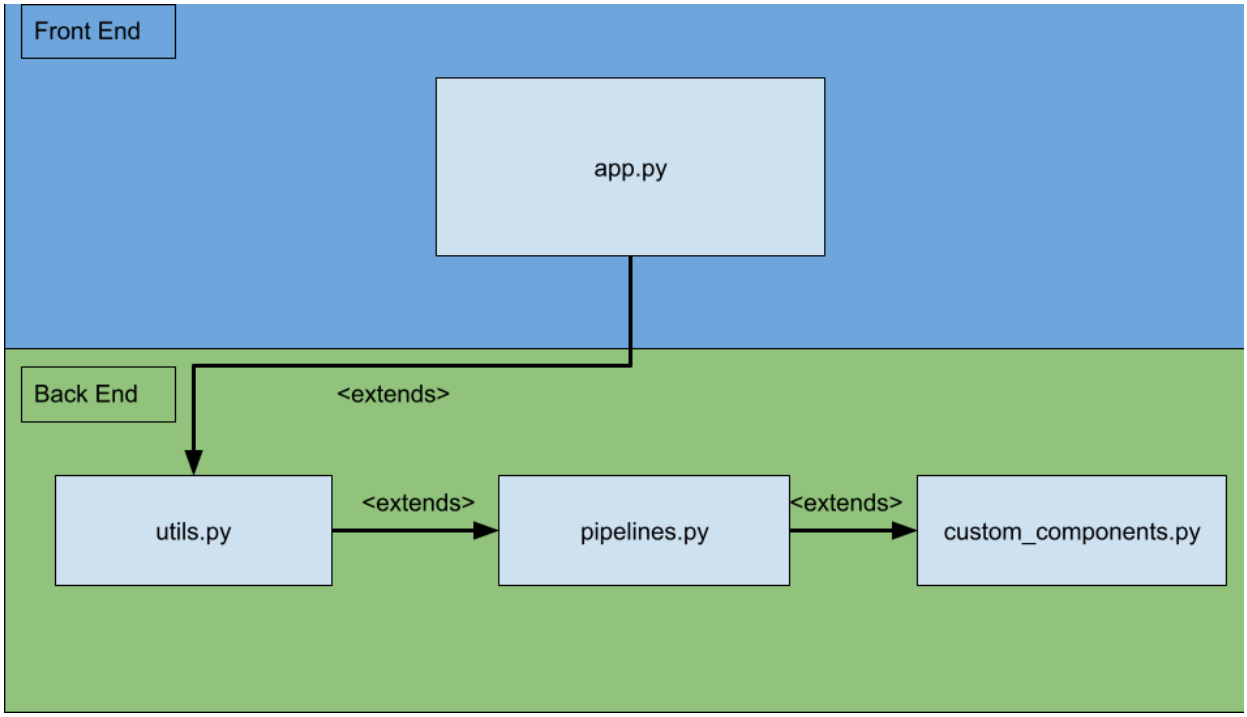
6. Additionally, the user can then let the LLM play instead. It will then attempt to answer the questions it asked itself, however, the LLM will only be able to answer the question using its parametric knowledge and reasoning abilities. This means that it will not be able to use the url of the Web page or search the Internet at all and will have to rely solely on the knowledge it gained during its training phase (supervised and unsupervised) as well as its capacity to use reason to deduce the right answer.
   The result of this feature was particularly interesting to me because it can be a way to guess what resources and information an LLM was trained on. For example, during my tests, I have found that Llama 3 accuracy was very good on topics related to technology, especially on machine learning but struggle when answering questions on animals like their anatomy or behavior.

7. For the third functionality, I wanted to use the SerpAPI again but this time, instead of directly giving the webapp an url to fetch, I would enter a topic and the app would respond by giving me a very quick answer. This could have been done by using the automated questions/answers that Google searches provide. These questions are created once enough users asked the same question in Google which then attempts to answer them looking for the information in the top 3 website results of the search. I wanted for the LLM to access these automated questions/answers to provide the user with the quickest answer possible but unfortunately, I haven't been able to make it work without hard-coded values.

   (see illustrated process below)



Architecture


Front End
app.py
Back End
<extends>
utils.py
<extends>
pipelines.py
<extends>
custom_components.py