**NANYANG TECHNOLOGICAL UNIVERSITY**

**SCHOOL OF COMPUTER SCIENCE & ENGINEERING**



**Assignment 02: Study basics of Google Cloud SaaS**

**Chan Eu Ching**

**U2021000F**

# Introduction

In this rapidly evolving landscape of technology education, the fusion of cloud computing services and artificial intelligence (AI) presents a transformative opportunity for enhancing learning experiences. The significance of integrating cloud-based platforms and LLMs in educational tools cannot be overstated. Cloud platforms like Google Cloud and others offer unparalleled scalability, flexibility, and access to powerful computing resources on demand, making them ideal for hosting sophisticated AI-driven applications.

This project delves into the intersection of Google Cloud’s Software as a Service (SaaS) offerings, particularly Cloud Shell, with the advanced capabilities of Large Language Models (LLM) such as Google Gen-AI. The primary focus is to develop a quiz generator (specifically Cloud Computing quizzes), leveraging on latest advancements in AI and cloud technology. The application of the quiz generator will also be discussed. Lastly, prompt engineering and LLM functions were experimented with to gain more insights into the importance and significance of their presence in building a quiz generator as well as in LLM models.

# Quiz Generator with Google Gen-AI and Cloud Run

Google Gen-AI represents a pioneering initiative by Google Cloud, aimed at harnessing the power of Gen-AI to create and deploy a variety of applications including tools like Quiz Generator. Gen-AI stands out for its ability to understand and process natural language, enabling it to generate quiz questions and answers that are both relevant and challenging.

Google Run is a managed platform that enables developers to deploy containerized applications. It simplifies the deployment process, by running the “gcloud beta run deploy <project-id>” command line in the terminal.

The first step to building a quiz generator in Gen-AI is to set up the development environment. This is done by activating the cloud shell, then setting up a new project and ensuring that the billing is enabled for the project. Then enable the necessary APIs such as:

* cloudbuild.googleapis.com
* artifactregistry.googleapis.com
* aiplatform.googleapis.com
* run.googleapis.com

Then, the access to Google Gen-AI will have to be allowed. Prompt templates were created. The data of the lecture content were extracted from the PDF files and concatenated into the prompt template to ensure that the questions generated were relevant to the content that was needed. The solutions to the questions are then generated using the inbuilt LLM. Templates were also fed in the LLM to guide the model to generate the quiz in the desired format. Once the quiz generator is done, it is packed into a Docker container and deployed to Cloud Run.

# Quiz Generator on Cloud-Based Software

Utilizing cloud-based software platforms to develop a quiz generator offers scalability, flexibility, and access to advanced technologies. In this project, deploying the quiz generator generates a service URL that allows users to call the application as an API. This service URL allows users to append any arguments defined in the main function, to regenerate the content with respect to the appended arguments.

## Google Cloud

In this assignment, users have the flexibility to modify the `topicNo` argument, which specifies the topic number of the lecture slides that the quiz should cover. The lecture slides are uploaded to Google Cloud first. After users specify the topic numbers they prefer, the relevant PDF will be loaded, and the quiz will be generated. This link provides the Google Cloud deployed quiz generator: <https://quiz-generator-6famce7bfq-uc.a.run.app/>. If the user wants to generate a quiz covering Lectures 1 to 3, an array [1, 2, 3] will be the argument of the topicNo. Hence the new quiz generator link address will be [https://quiz-generator-6famce7bfq-uc.a.run.app/?topicNo=[1,%202,%203]](https://quiz-generator-6famce7bfq-uc.a.run.app/?topicNo=%5b1,%202,%203%5d), where ‘%20’ represents spaces. Otherwise, the quiz generator will generate quizzes on Lectures 1 to 7 by default. The web interface will display the questions, answers, and files that the questions were generated from. This could be further improvised by creating a simple web application to take in inputs for the lecture content desired to be covered by the quiz.

## Nemobot Platform

Another exploration done in this assignment was to call the service URL generated by Cloud Run as an API in the Nemobot platform where the chatbot is integrated. Integrating the quiz generator into this platform provides a more user-friendly interface for the users to generate quizzes. In this Nemobot exploration, quiz generation with both OpenAI and VertexAI (Google) will be explored.

Here is the list of LLM Functions defined in the Nemobot:

* GenerateQuiz({userMessage}, {memory}) 🡪 detect whether the student wants to generate a quiz through lecture contents of OpenAI. If OpenAI, then it will generate a quiz using prompts stated on the platform.
* getLectureChoice({userMessage}) 🡪 retrieve the ‘topicNo’ wanted to generate the quiz. Returns array format i.e. [1, 2, 3] for Lectures 1 to 3 (space between comma and number is necessary!)
* ConvertToMarkdown({content}) 🡪 organize the quiz generation format by topics

The way that the Nemobot will behave is as such:

1. The chatbot will detect whether the user wants to generate a quiz from OpenAI or a specific lecture content quiz.
2. If it is OpenAI, details regarding the subject, number of questions, difficulty level and whether is it MCQ or short answered will be asked. Then the relevant quiz will be generated via the API.
3. If it is a lecture content quiz, the chatbot will request for the topics to be included in the quiz. The chatbot will then fetch the service URL acting as an endpoint to run the quiz generator and produce the quiz in the chatbot itself.

Since the Nemobot platform has an in-built API call for OpenAI API, the LLM functions that are used to generate the quiz will employ OpenAI by default. As long as the prompt for the function to generate a quiz is properly instructed, the chatbot will be able to produce high-quality quizzes.

In this project, Nemobot acts as software to prompt the users for the lecture contents to be included in the text. The input will then be passed into the LLM function which appends to the endpoint and generates the quiz accordingly. However, the fetching of the API took a bit of time if there was more than 1 lecture slide required for the quiz. This invoked a time-out error from Vercel, the cloud platform that deploys Nemobot, which does not allow a fetch call with no response for more than 25 seconds. Since it is also not a JavaScript fetch timeout error, it would not be effective to implement a timeout abortion on JavaScript as well.

Hence, to solve this issue, the quiz generator is coded to stream a set of questions and answer topic by topic to prevent a time-out error. This is done by having an inner function that uses a generator to generate data then invoke that function and pass it to a response object. The inner function will include a yield function and each yield expression is directly sent to the browser.

Nemobot also has a restriction on the length of response that is allowed for its response, hence there is a limitation of 3 questions per topic and the user can only call for a maximum of 3 topics at a time.

Below is a snippet of the quiz generation in Nemobot.

A screenshot of a chat

Description automatically generated

# Implementing Google Gen-AI Quiz Generator in LLM functions

After running through the [Quiz Generator tutorial](https://codelabs.developers.google.com/cloud-genai#0) to build a simple quiz generator using prompts, the generation of high-quality quizzes for this module was explored. To ensure that the quizzes were relevant to the course, the course lecture slides had to be taken in by the LLM model to generate both the questions and the answers.

## Loading content

Firstly, the lecture contents are loaded into the server by PyPDFLoader. It will then go through a CharacterTextSplitter encoder to convert the contents into chunks. The chunk\_size defines the maximum size (in characters) of each text chunk after the splitting operations. It is crucial to set a balance between being small enough to comply with LLM’s limitations and large enough to maintain the context and coherence necessary for the model to generate meaningful outputs. Meanwhile, chunk\_overlap mitigates the risk of losing context or cutting sentences in the middle when splitting text into chunks. It refers to the number of characters at the end of one chunk that is repeated at the beginning of the next chunk. After splitting and overlapping, the chunks will be output and fed into the LLM sequentially.

Inspired by [AIAnytime](https://github.com/AIAnytime/Question-Answer-Generation-App/tree/main), the questions and answers will undergo different values of chunk\_size and chunk\_overlap. To increase the quality of the questions generated, more chunk\_size and chunk\_overlap were assigned for question generation to ensure that more context was available for the model, facilitating the generation of questions that are well-informed by text.

However, some PDF files were too large for the LLM model to load even after splitting into chunks. Therefore, in the data processing stage, for every 10 slides, the extracted content will be fed into a TextGenerationModel “text-bison@001” to summarize the content. Since presentation slides usually have very little content per slide in them, grouping the content into 10 slides was considerably reasonable to not miss out on any important content within the file.

## Generating Questions

In this assignment, the approach to generating the quiz was to first generate the questions. Hence, an LLM for the question generator was defined using “text-bison@001” with a temperature of 0.3. This temperature represents the degree of randomness in the token selection, ranging from 0 to 1. By setting the temperature low for question generator LLM, it aims to generate high-quality, relevant, and precise questions that are customized towards the chunks fed.

The LLM model will then be used to generate the questions through the load\_summarize\_chain function that generates a response by iterating over the input documents, progressively refining its answer with each step. For each document, it passes all the non-document inputs, current documents, and the most recent answer to an LLM chain to generate a new response.

## Generating Answers

Before the answers for the LLM-generated questions were produced, the documents for generating answers were embedded in FAISS, into a searchable vector space using embeddings generated by VertexAIEmbedding().

This vector store then serves as a retriever for the RetrievalQA chain. When generating answers to the questions, the system first retrieves the most relevant documents from the vector store. This retrieval is based on semantic similarity between the question embeddings and the document embeddings stored in the vector store. The role of vector store is critical in ensuring efficiency during semantic search. As FAISS is designed to handle very large datasets efficiently, it is deemed suitable for this scenario.

# Exploration of Prompt Engineering and LLM functions

Prompt engineering involves designing and refining the input text provided to an LLM to elicit specific, desired outputs. In the world of quiz generators, this means generating prompts that guide the LLM to produce questions, answers, and distractors that are relevant and tailored to the context’s complexity and scope.

In this section, the exploration of both prompt engineering and LLM functions that are created will be discussed.

## Prompt Engineering

The prompts fed into the LLM model to generate quiz questions were explored.

Case 2: Refined General prompt: Statement with more context.

i.e. You are an expert in generating Cloud Computing quiz…

Generate in JSON array format.

Response:

Case 1: General prompt: Brief statement with no context.

i.e. Generate a quiz about Cloud Computing.

Response:

A screenshot of a computer screen

Description automatically generated A black screen with white text

Description automatically generated

From both cases above, it is not that obvious that there is any difference in the quality of the quizzes. Both are very relevant to Cloud Computing and only a professor would notice which is more relevant to the course. However, for Case 2, the prompt specified the format to be output hence the output is in JSON array format and returns “question” and “correct” which implies that it is not a multiple-choice quiz. This shows how prompt engineering can also achieve a desired output format.

Now we will explore the General prompting (Case 3) and Refined General prompting (Case 4) for prompt templates with extracted lecture information concatenated into them. This enables the model to tune the questions towards the content of the lecture slides. In both Case 3 and Case 4, there is an extra iterative process where the first generation of question sets from the model will be refined by being fed to the same LLM model again (with the extracted texts from PDF) to enhance the quality of the questions.

Case 3 Result:

A close-up of a computer screen

Description automatically generated

Case 4 Result:

A close-up of a text

Description automatically generated

The difference in this scenario is clearer. There are a few ‘questions’ generated in the General prompting (with context) that were statements rather than sentences. With more explicit prompts fed into the LLM, it guides the model to produce more desirable outputs hence being able to produce a more accurate, refined quiz. This is evident to show that prompt engineering plays a crucial role in generating high-quality quizzes.

## LLM Functions

For LLM functions, only the Temperature parameter was explored. Temperature controls the randomness in the model’s responses. Lower temperature leads to more predictable and conservative outputs. This is very essential when generating questions as the questions should be predictable and directed towards the prompt. It was tested that higher temperature not only produced questions that might not be relevant to the lectures, but the answers were also very abstract such that it would not be able to be classified as a legitimate Cloud Computing quiz.

There are many other token-based control functions within LLM such as top-k and top-p sampling that decide how a model selects its tokens for output.

# Conclusion

The exploration and development of a quiz generating using Google Cloud’s SaaS platforms especially Cloud Shell, alongside the capabilities of LLM such as Google Gen-AI, represents a significant stride towards redefining educational tools. By carefully leveraging the advanced text generation capabilities of LLMs, high-quality quiz content that is both engaging and informative was generated.

Furthermore, the exploration of prompt engineering and LLM functions has shed light on the iterative process of optimization that underpins the effective use of AI in education.

In this project, two quiz generators were deployed to Google Cloud Run. [One](https://quiz-generator-rqtuxk7xxq-uc.a.run.app/?) was designed for Nemobot hence it could only generate limited content due to the limitations of the software, and the other [one](https://quiz-generator-6famce7bfq-uc.a.run.app/?) was a web application for users to view the data frame easily. More functions like file uploads could be implemented to generate quizzes of any fields relevant to the uploaded files.