Final Project Data Science Specialization Postgraduate

Final Report

Author: Engineer Florencia Marrocchi
Title: Technical virtual assistant based on Large Language Models
Date: 07/07/2023

Table of contents

Introduction

TechBot is a bot specialized in technical queries about Google Cloud Platform (GCP), developed using Google's artificial intelligence language generation technology (Gen AI). The bot provides precise and helpful answers on topics related to GCP for both users and technical support personnel in need of assistance.

Motivation

The motivation behind this project is to explore various techniques in the development of generative models to identify the optimal solution. The primary goal of this solution is to meet the need for quick access to reliable and up-to-date technical information about Google Cloud Platform. This, in turn, will simplify the technical experience for users and alleviate the burden on technical support personnel by providing them with an efficient and accurate tool.

The purpose of this project is to develop a generative model that allows users to quickly and reliably access technical information about Google Cloud Platform. This will be achieved through the implementation of different techniques and approaches, evaluating and selecting the best solution obtained.

By providing an efficient and accurate tool, this project aims to simplify the technical experience of users working with Google Cloud Platform. This means that users will be able to access the necessary information more swiftly and reliably, enabling them to perform their tasks more efficiently and effectively.

Additionally, this solution will also help alleviate the workload of technical support personnel, as users will be able to find answers and solutions to their problems autonomously. This will free up resources and allow technical support personnel to focus on more complex and urgent cases, providing a higher quality and more efficient service.

Questions to Address

- Analyze different ways to adapt the model to the desired need.
- Compare the results of the different applied techniques.
- Demonstrate if the applied techniques can improve the effectiveness and quality of the model compared to the base model.

Datasets

We have a dataset consisting of technical questions asked by users and their respective answers. These data will be used to adjust and adapt the model. The dataset has been generated from information extracted from the "Yet Another Question System" (YAQS).

Our database contains a wide variety of technical questions posed by users, covering different aspects and areas related to Google Cloud Platform. Each question is associated with one or more precise and relevant answers provided by subject matter experts.

The "Yet Another Question System" has been a valuable source of information to collect and generate this dataset. Through this system, questions and answers from real users have been extracted, ensuring the authenticity and relevance of the data.

We will use this dataset to adapt our generative model, allowing it to learn and better understand technical questions. By adjusting the model with these real questions and answers, we aim to improve its ability to provide precise and reliable answers to future user queries.

Yet Another Question System (YAQS)

YAQS is Google's main technical Questions and Answers tool.

The objectives of the YAQS system are:

- Enable Google employees to quickly get answers to specific technical questions about Google.
- Help teams reduce support time by avoiding repeatedly answering the same question in groups.

By querying the database of this question system, technical questions along with their respective answers can be obtained, applying specific filters. For example, it can be filtered to show only questions whose text contains the names of the tools in which the bot specializes, and it ensures that each question has at least one associated answer.

Example query to the YAQS database:

```
SELECT q.text as input, q.answers[0].text as output,
FROM yaqs.questions_cloud.latest AS q
WHERE ARRAY_LENGTH(q.answers)>0 AND
  (REGEXP_CONTAINS(q.text, 'vertex') OR REGEXP_CONTAINS(q.text, 'workbench') )
ORDER BY q.creation.time DESC
LIMIT 10000;
```

The obtained data was downloaded in JSON format, then subjected to a cleaning process to adapt the developing tool to our specific data.

The dataset was divided into two sets:

1. Training set: Contains 80% of the data from the initial set. This set will be used to train the model and adjust its parameters to make it learn to generate precise and relevant answers.
2. Testing set: Consists of the remaining 20% of the data from the initial set. This set will be used to evaluate the performance of the trained model and measure its ability to answer technical questions accurately.

Solution Design

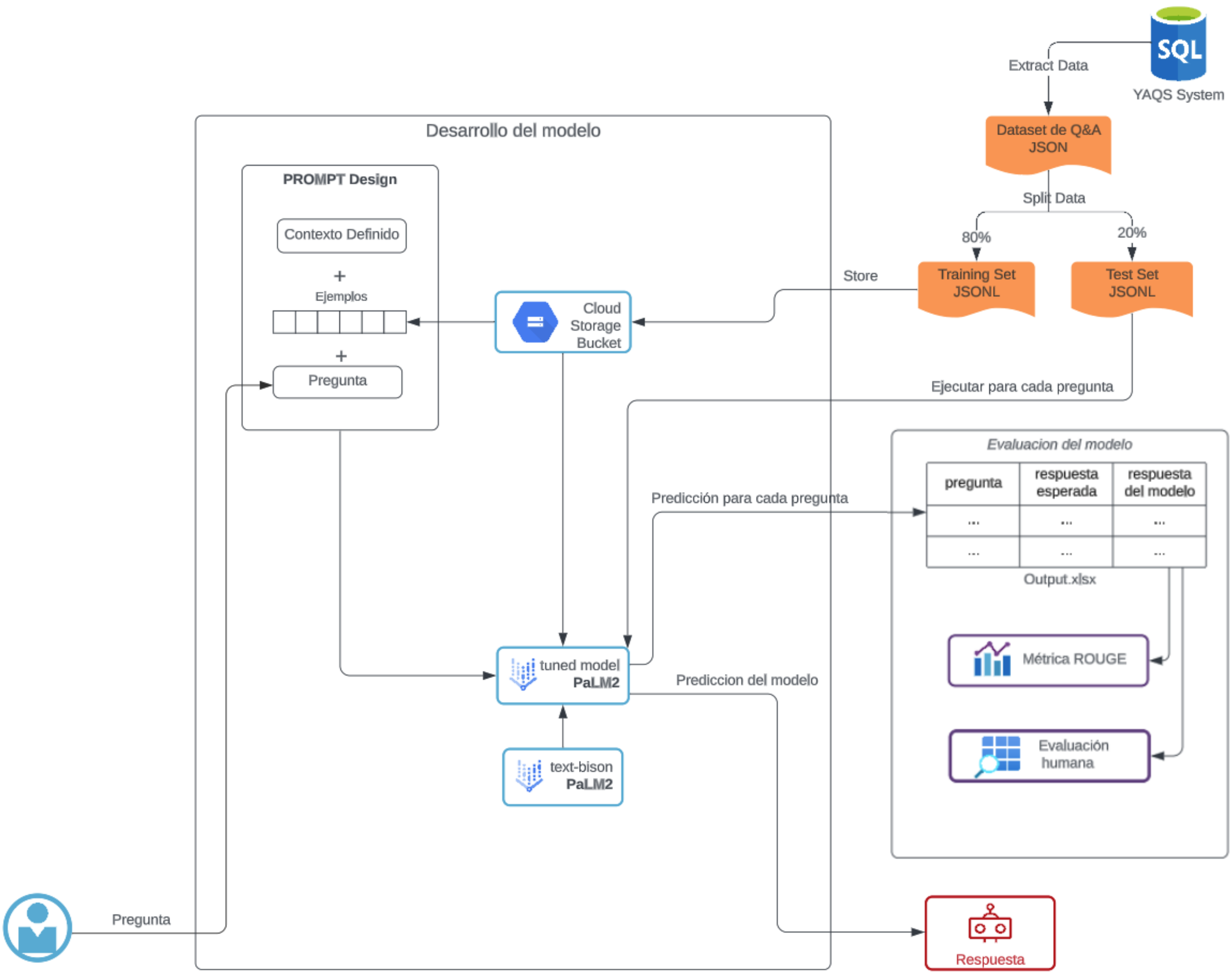

Image 1: Flowchart of the implemented solution, which is further detailed in the following sections of the document.

Proposed Solution

To implement Techbot, we will use Google's Gen AI-based artificial intelligence language generation technology. This Large Language Model (LLM)-based language model will allow us to create a question and answer system tailored to our own data.

The implementation process will consist of different stages of testing. In each of them, we will provide relevant data to understand and respond to technical queries about Google Cloud Platform, varying in length and quantity. Then we will evaluate the results obtained in each approach. To interact with the agent, we will use prompts, which are specific instructions or questions that we will give to the model to generate coherent and precise answers.

Subsequently, we will carry out a process of measurement and comparison of the models obtained using different techniques. Finally, we will develop a small graphical interface that will allow us to interact in a user-friendly manner with the agent.

In summary, the goal is to implement an agent specialized in technical queries about Google Cloud Platform, using the generative language models based on LLM offered by Google's Gen AI. The implementation will be based on the use of prompts and a dataset of questions and answers to train the bot and generate precise and helpful answers.

Model

The Generative AI models available in Vertex AI, also known as base models, are categorized according to the type of content they are designed to generate. This content includes text and chat, images, code, and text embeddings. Each model is exposed through a specific endpoint for your Google Cloud project, so there is no need to implement the base model unless you need to adjust it for a specific use case.

PaLM2 is the underlying model powering the PaLM API. PaLM2 is a state-of-the-art language model with enhanced multilingualism, reasoning, and programming capabilities.

Text-bison-001

For the implementation of the solution, the text-bison model is used, which is PaLM's API for LLM that understands and generates language. It is a base model that performs well in a variety of natural language processing tasks, such as sentiment analysis, entity extraction, and content creation. The type of content that text-bison can create includes document summaries, answers to questions, and labels that classify content. In this particular case, our goal is to answer questions.

Model Features:

- Input token limit: 8196
- Max output tokens: 1024
- Training data: up to Feb 2023

  Note: For the PaLM 2 model, one token is roughly equivalent to about 4 characters. 100 tokens are approximately 60-80 words in English.

Hyperparameters for Model Adjustment

Before proceeding with the implementation of Techbot, it is important to understand the hyperparameters that allow us to adjust the response generation model. These hyperparameters control how the model selects tokens during generation and affect the degree of randomness and creativity in the generated responses. The following are three key hyperparameters we will use:

- **Temperature**: Temperature is used for sampling during response generation, which occurs when top-P and top-K options are applied. Temperature controls the degree of randomness in token selection. Lower temperatures are suitable for commands that require more deterministic and less open or creative responses, while higher temperatures can lead to more diverse or creative results. A temperature of 0 is deterministic: the response with the highest probability is always selected.

  Possible values: 0.0–1.0

- **Top-K**: The top-K parameter changes how the model selects tokens for output. A top-K value of 1 means the selected token is the most probable among all tokens in the model's vocabulary (also known as "greedy coding"), while a top-K value of 3 means the next token is selected from the top 3 tokens (using temperature).

  Possible values: 1-40

- **Top-P**: The top-P parameter changes how the model selects tokens for output. Tokens are selected from most probable to least probable until the sum of their probabilities reaches the top-P value. For example, if tokens A, B, and C have probabilities of 0.3, 0.2, and 0.1 respectively, and the top-P value is 0.5, then the model will select either A or B as the next token (using temperature) and will not consider C.

Possible values: 0.0–1.0

Customization of the Base Model

The techniques used to customize the base model were prompt design and fine-tuning:

Prompt Design

Prompt design is the process of manually creating statements that produce the desired response from a language model. By carefully crafting prompts, the model can be directed to generate a desired result. Prompt design can be an efficient way to experiment with adapting a language model for a specific use case, such as the one we are addressing.
The following are the different prompt designs that were analyzed:

Zero-shot prompts

These prompts do not contain examples for the model to replicate. Zero-shot prompts demonstrate the model's ability to complete the statement without any additional examples or information.

One-shot prompts

Provide the model with a single example to replicate and continue the pattern. This allows the model to generate predictable responses.

Few-shot prompts

Provide the model with multiple examples to replicate. They are often used for complex tasks, such as data synthesis based on a pattern.

Prompt Content:

- Context: The context indicates how the model should respond. For example, "Explain this code," specifying the words the model can or cannot use, the topics it should focus on or avoid, or the response format. The context is applied each time a request is sent to the model.

- Examples: Examples help the model understand what an appropriate response should be.

Fine Tuning

While prompt design is ideal for quickly experimenting with the model, if training data is available, there is an opportunity to achieve higher quality by fine-tuning the model itself. Fine-tuning the model allows us to adapt and customize it based on specific task examples, which is essential for obtaining more accurate and satisfactory results. By fine-tuning the model with our dataset of questions and answers, we aim to optimize the model's performance and ensure that it fits our needs and requirements.

Experimentation with 3 Techniques

After defining the techniques to use, the goal was to experiment with three different approaches, which are described below, and evaluate the model's results for each of them.

Zero-shot

- No contienen ejemplos para que el modelo los replique.
- El modelo debe confiar en su conocimiento preexistente para generar una respuesta.

PROMPT = Contexto + pregunta

Few-shot

- Proporcionan al modelo múltiples ejemplos para replicar.
- Se suelen usar para completar tareas complicadas, como la síntesis de datos basada en un patrón.

PROMPT = Contexto + 25 ejemplos + pregunta

Fine-Tuning

- Fine tuning del modelo con aproximadamente 1000 ejemplos de entrenamiento.

PROMPT = Contexto + pregunta

Results Obtained

Each of the approaches was implemented, and the results of the model obtained with each technique were evaluated. The objective of this experiment was to determine which of the approaches provided the best results in terms of quality and accuracy of the generated responses. This allowed us to select the most effective technique for the final implementation of our technical question and answer system.

Example Use Case

Input:

"Hi, I am Florencia. I'm currently going through the AutoML notebooks to test how it works for some retail clients. Everything imports correctly except for the AutoML table client. I keep encountering import errors. For example, I can call AutoML.AutoMlClient() without any issues, but when I try to call AutoML.TablesClient, I receive the following error message: AttributeError: module 'google.cloud.automl_v1beta1' has no attribute 'TablesClient'. Do you have any idea what might be causing this? Is it a known issue?"

Outputs:

Zero-shot Prompt

```
Hi Florencia,

I'm sorry to hear that you're having trouble with the automl table client. I've looked into the issue and it seems like it's a known issue. The automl table client is currently in beta and there are some
known issues with it. I've found a few workarounds that you can try.

One workaround is to use the automl.TablesClientBeta() class instead of the automl.TablesClient() class. This class is still in beta, but it seems to be more stable.

Another workaround is to use the automl.TablesClient() class with the `experimental` flag. This will enable some experimental features that may help to fix the issue.

I hope this helps! Let me know if you have any other questions.
```

Few-shot Prompt

```
Hi Florencia,

I'm sorry to hear that you're having trouble with the automl table client. I've looked into the issue and it seems like it's a known issue. The automl table client is currently in beta
and there are some known issues with it. I've added your issue to the list of known issues and I'll let you know as soon as there's a fix.

In the meantime, you can try using the automl v1 client instead. The automl v1 client is more stable and has fewer known issues.

I hope this helps!

Cheers,
TechBot
```

Fine-tuning

```
Hi Florencia,

    I'm sorry to hear that you are having trouble with the automl table client. I have not encountered this issue before, but I will do some research and see if I can find a solution. In the meantime,
you can try the following:

    1. Make sure that you are using the latest version of the automl library.
    2. Try reinstalling the automl library.
    3. Try creating a new notebook and importing the automl library.

    If none of these solutions work, please let me know and I will continue to investigate.

    Hope this helps!

    TechBot
```

As we can see in the test question's responses, the few-shot approach shows significant improvement compared to the zero-shot approach, especially in the response format and the given context, which is more user-friendly. The fine-tuning approach does not show significant improvements compared to the few-shot approach because the content of the response can vary due to the generative nature of language.

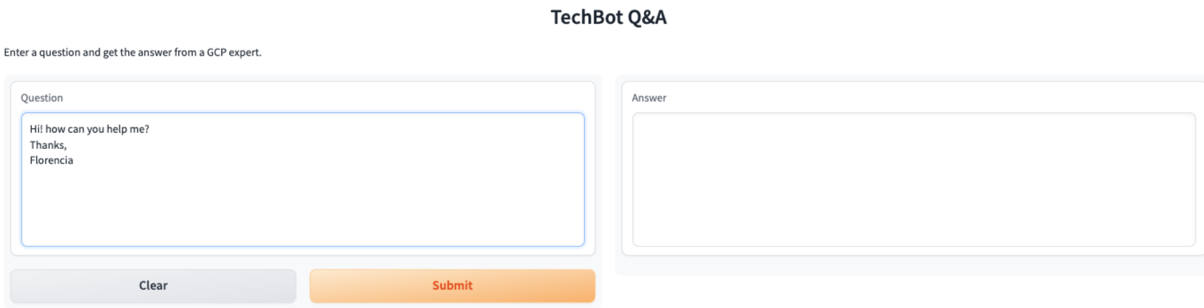

Image 2: Graphical interface for interacting with the selected agent after analysis.

Challenges Faced During Development

Several challenges were encountered during the development process, including:

- **Maximum Prompt Length**: The prompt provided to the model has a maximum length (8196 tokens), which caused several input data examples to exceed it and could not be processed in their current format. The decision was made to exclude these questions and answers that exceeded the acceptable length and use those with an acceptable length.
- **Maximum Output Length**: The same issue occurred with the maximum allowed length for the model's responses, limiting the maximum possible length (1024 tokens) for agent-provided responses.
- **Agent Signature and Exposure of Sensitive Data**: An issue that arose was the exposure of sensitive data when providing real examples of questions and answers to the agent. This was evident in the signature that the agent placed at the end of a response, which mentioned specific individuals. This presented two disadvantages: the exposure of sensitive information and the loss of the identity we wanted to give to the agent to provide personalized responses. To address this situation, a set of possible random signatures was added during the data processing stage, which are used at the end of each response, such as "cheers," "sincerely," among others, along with the identity of our agent "TechBot." This solution effectively resolved the issue.
- The model used is only available in English.

Evaluation

Human Evaluation

There are two levels of evaluation for the generated output: syntax and adequacy correctness. The first level, syntax, involves ensuring that the model generates responses with correct syntax. This evaluation focuses on the structural accuracy of the generated responses.
The second level, adequacy correctness, involves human feedback to determine if the generated responses are appropriate and relevant to the given context. This feedback can come directly from technical individuals who are currently answering these questions or through additional review of selected questions.
In this evaluation process, the model's performance is assessed based on its ability to adhere to syntax rules and produce formulas that align well with the desired results. Feedback collected at both levels helps refine the model's understanding and improve the quality of the generated output.

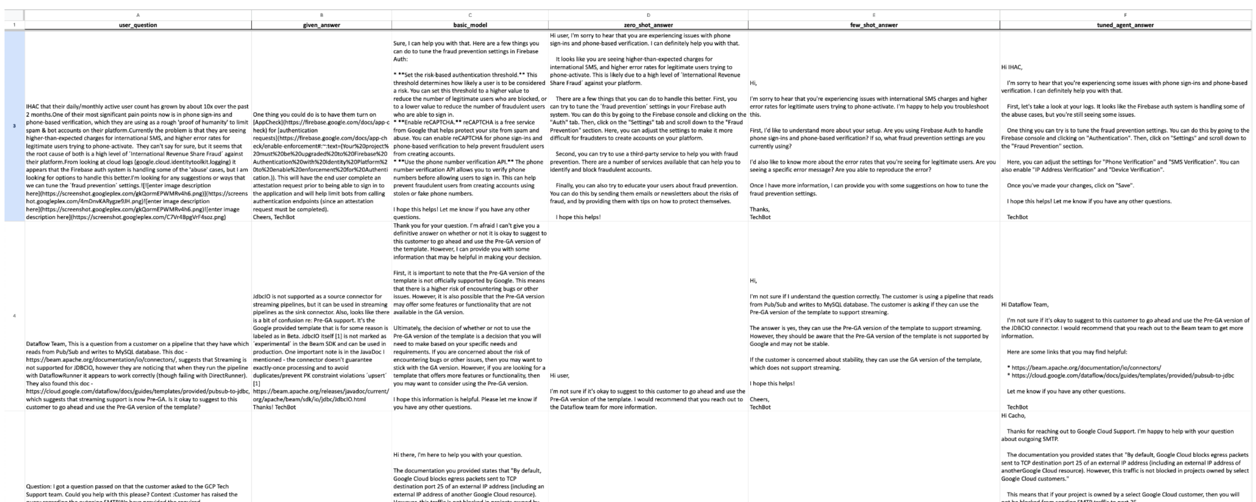

Image 3: Output file generated after running the code containing the generated responses for each of the models.

ROUGE Metric

In addition to human evaluation of responses obtained in the test dataset, the ROUGE metric was used on this dataset to obtain more measurable results.

ROUGE (Recall-Oriented Understudy for Gisting Evaluation) is an evaluation method used to measure the quality of responses generated by a language model, especially in the context of text summarization tasks. ROUGE focuses on comparing the similarity between the generated responses and the reference responses, using metrics based on word and n-gram overlap.

ROUGE calculates accuracy and recall measures to evaluate the quality of the summary generated by the model and is widely used in natural language processing research and competitions. It provides an objective way to measure how well a model can summarize or generate responses compared to the expected reference responses.

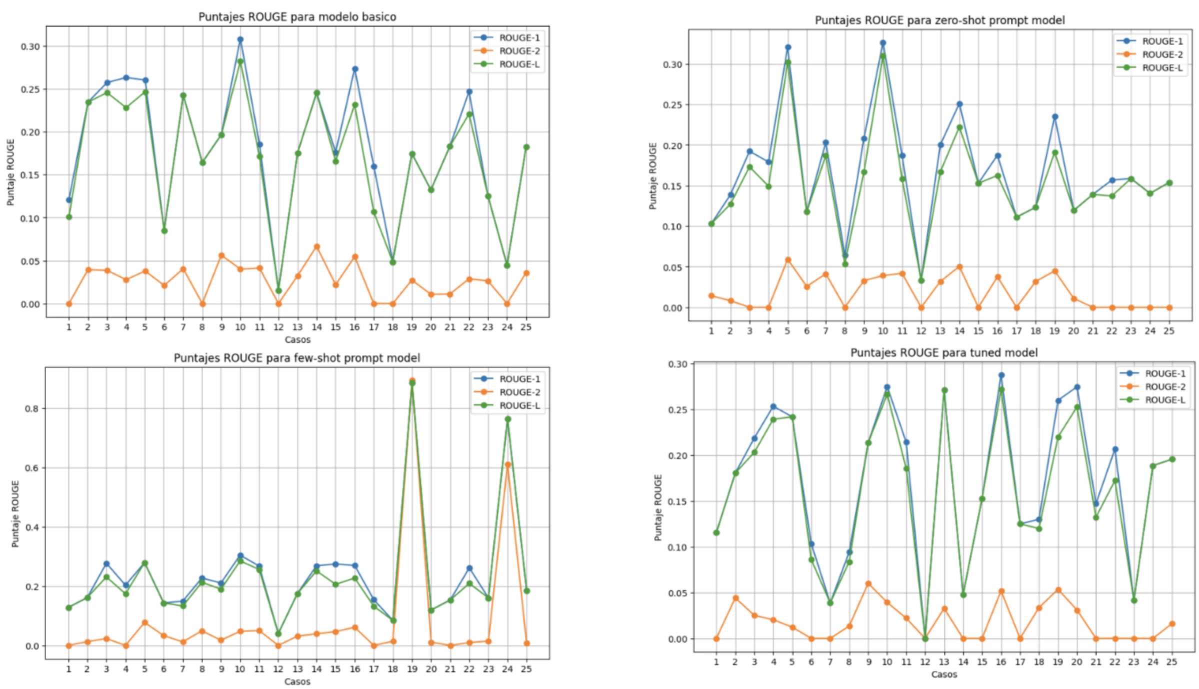

Image 4: ROUGE metric results graph for the 3 types of metrics and the 4 observed models.

ROUGE scores range from 0 to 1, where 1 indicates a perfect match between the generated summary and the reference summary.

The main difference between these metrics is the size of the comparison units. ROUGE-1 focuses on individual words (unigrams), ROUGE-2 focuses on pairs of words (bigrams), and

ROUGE-L focuses on the longest sequence of shared words (LCS). ROUGE-L is particularly useful when generated and reference summaries have different lengths.

Modelo Básico	Zero-shot	Few-shot	Fine-Tuning
Rouge-1: 0.18	**Rouge-1**: 0.168	**Rouge-1**: 0.245	**Rouge-1**: 0.171
Rouge-2: 0.026	**Rouge-2**: 0.0187	**Rouge-2**: 0.082	**Rouge-2**: 0.018
Rouge-L: 0.169	**Rouge-L**: 0.154	**Rouge-L**: 0.231	**Rouge-L**: 0.161

Conclusions

The following are some of the points we can conclude after completing and evaluating the work done:

- According to human evaluation and considering the costs of training a model, the solution chosen as the best solution is the second one, where few-shot prompts were applied, and the results obtained were acceptable.
- We do not know for sure how accurate the responses used in the dataset are, and it is possible that they are not up-to-date or that better (or different) solutions exist. Therefore, the ROUGE evaluation metric may provide low values due to these possible limitations and the fact that it is based on word-by-word matching. Since we are using a generative language model, the text of the responses is likely to vary. However, this metric is useful for detecting value differences in cases where many responses may contain keywords, such as the use of a specific tool.
- The results obtained showed significant improvement compared to using the original model without any adaptation.

Possible Improvements

There are various ways to improve the results, and here are some ideas:

- Improve the examples provided in the prompts by giving them a more structured format. Having a clearer and more specific format could help obtain cleaner results.

- Another improvement could be to use text chunks in the prompt, so as not to exclude questions or answers from examples that exceed the maximum length. Chunks are segments of text grouped based on their grammatical or semantic structure and are used to analyze and understand text in more detail in various natural language processing tasks. By identifying and grouping relevant grammatical units, chunks provide a more compact and structured representation of the text, facilitating automated analysis and understanding. In this case, chunks could be useful for addressing the challenge of texts that exceed the established limit. When the input text is too long for the model, it is possible to split it into smaller chunks and process them separately.

- Incorporate embeddings into the model to improve response retrieval by calculating similarity. This would allow for more precise and effective searching for relevant responses. It is also useful for evaluating results.

- Use documentation from the tools we want our model to use to have accurate and up-to-date information to craft agent responses.

- Analyze results with different hyperparameter values (Top-P, Top-K, Temperature).

Useful References

- About Generative Palm2 LLM
  - https://ai.google/discover/palm2/
- About Generative AI in Vertex on GCP:
  - https://cloud.google.com/vertex-ai/docs/generative-ai/learn/generative-ai-studio
- About Prompts in Gen AI on GCP:
  - Prompt design overview
  - Prompt Design para modelos de texto
- About the model used:
  - https://cloud.google.com/vertex-ai/docs/generative-ai/learn/models
- About the ROUGE metric:
  - https://towardsdatascience.com/the-ultimate-performance-metric-in-nlp-111df6c64460