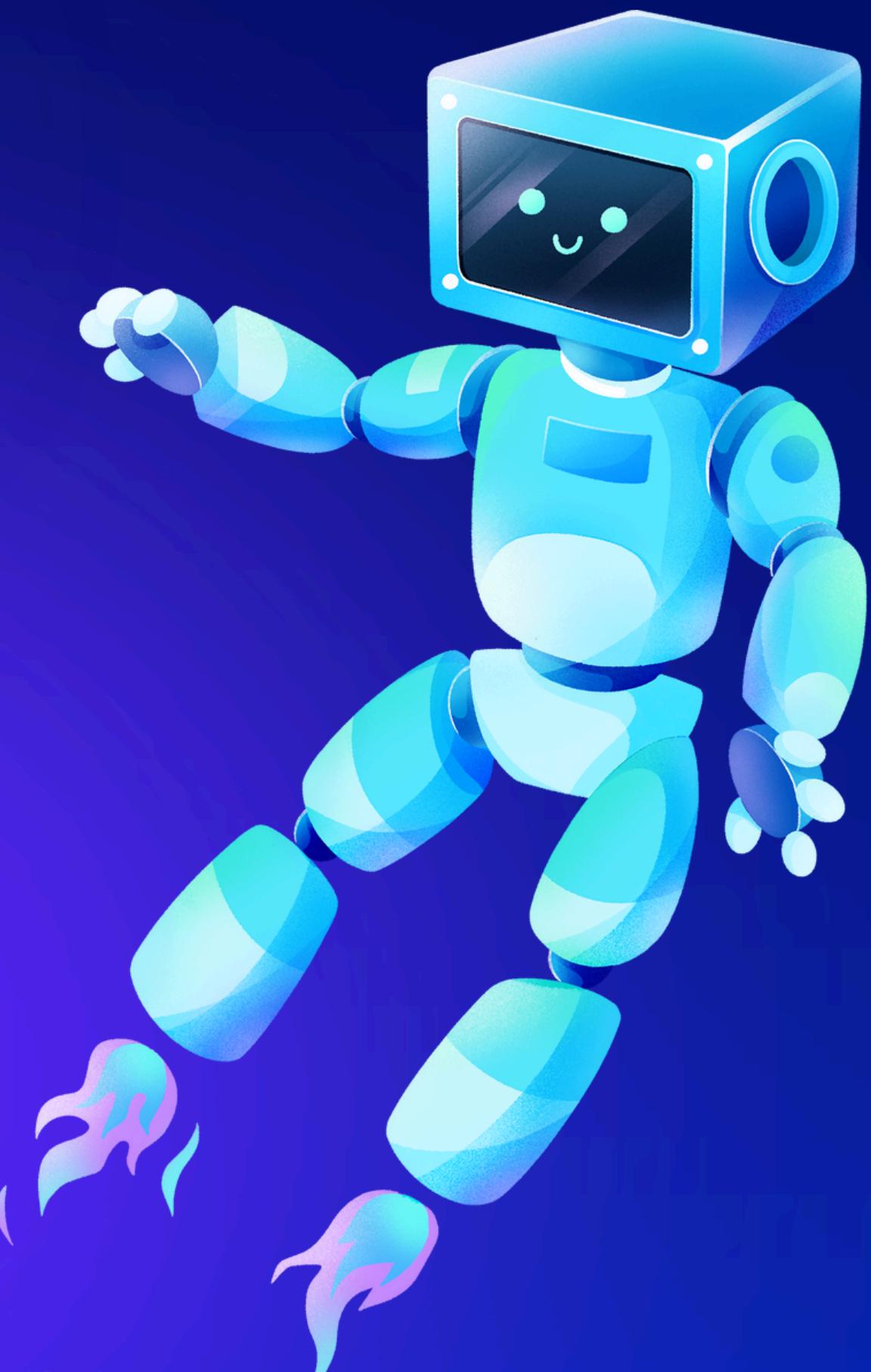


CHATCARE

Presentación Final

20/05/2024

Integrantes:
Gabriele Petroni
Jesús Espadas
Ricardo Palomares





CONTENIDO

- Antecedentes.
- NoteBook ChatBot de partida.
- Resultados de vía 1 de desarrollo.
- Resultados de vía 2 de desarrollo.
- Fine-Tuning.
- Conclusión.
- Ejemplo.



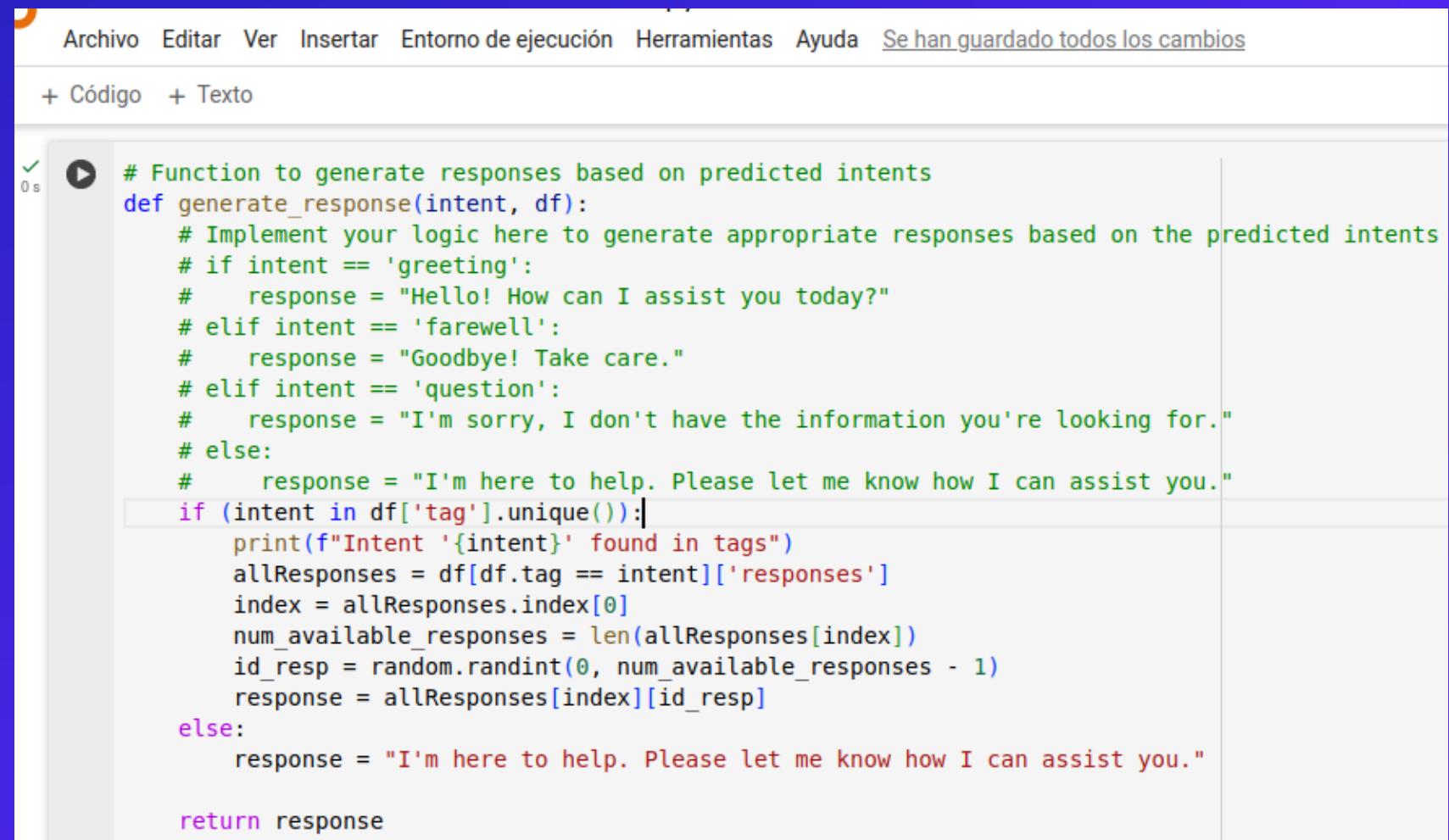
ANTECEDENTES

- Objetivo: crear un ChatBot que apoye a usuarios con síntomas de salud mental.
- Planificado en varias etapas:
 - ChatBot inicial y mejoras con las que crear conversaciones.
 - Conversión a formato de fine-tuning para GPT-3.5.
 - Fine-tuning para generar ChatBot operativo con ChatGPT.



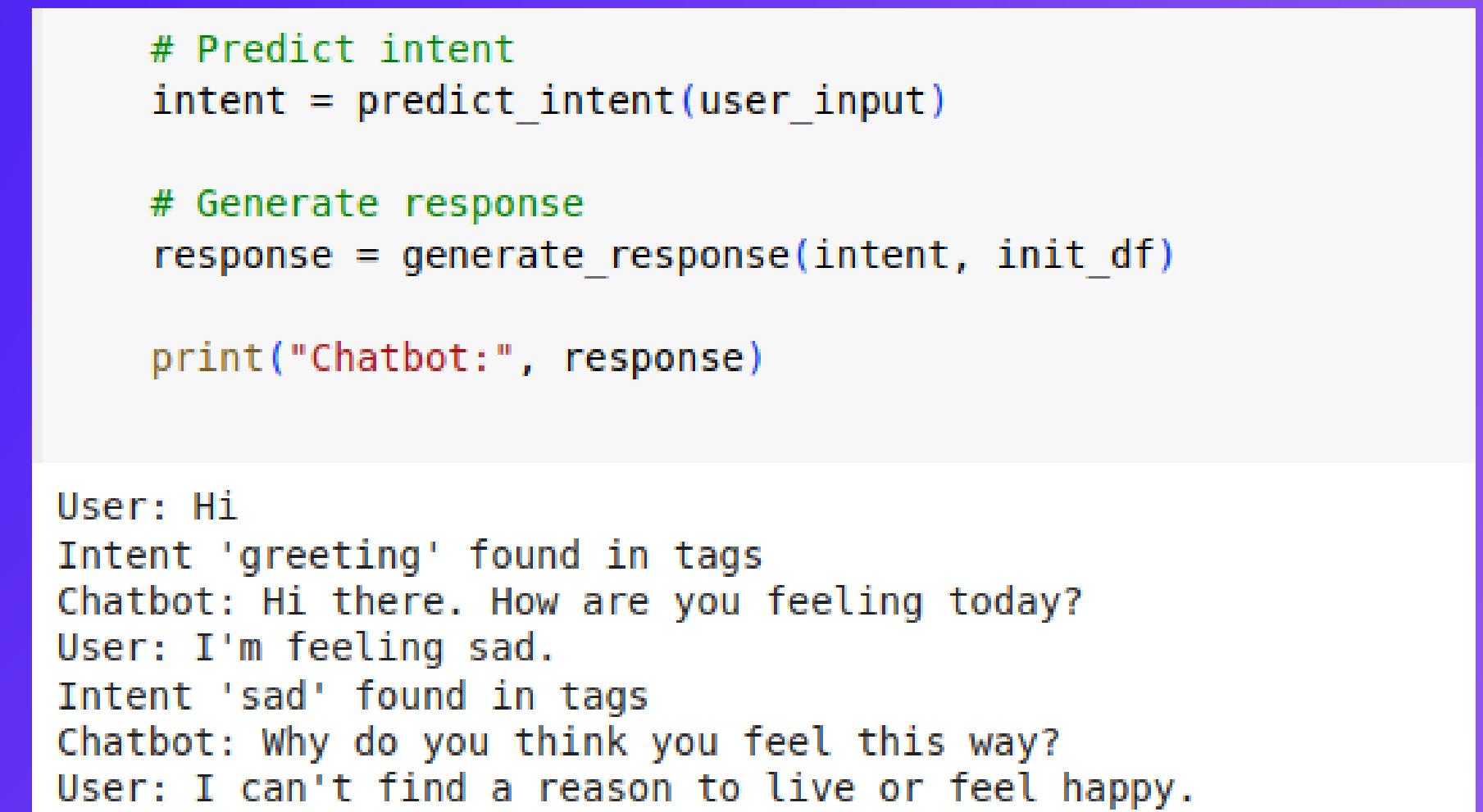
NOTEBOOKS UTILIZADOS

Enlace al chatbot original



The screenshot shows a Jupyter Notebook interface with a toolbar at the top. The code cell contains a function named `generate_response` which takes an intent and a DataFrame `df` as parameters. It handles four main cases: greeting, farewell, question, and other. For the 'greeting' case, it prints a welcome message. For other intents, it finds all responses associated with that intent in the DataFrame and selects a random one. If no intent is found in the tags, it falls back to a general greeting. The code uses `random.randint` to select a response index.

```
# Function to generate responses based on predicted intents
def generate_response(intent, df):
    # Implement your logic here to generate appropriate responses based on the predicted intents
    # if intent == 'greeting':
    #     response = "Hello! How can I assist you today?"
    # elif intent == 'farewell':
    #     response = "Goodbye! Take care."
    # elif intent == 'question':
    #     response = "I'm sorry, I don't have the information you're looking for."
    # else:
    #     response = "I'm here to help. Please let me know how I can assist you."
    if (intent in df['tag'].unique()):
        print(f"Intent '{intent}' found in tags")
        allResponses = df[df.tag == intent]['responses']
        index = allResponses.index[0]
        num_available_responses = len(allResponses[index])
        id_resp = random.randint(0, num_available_responses - 1)
        response = allResponses[index][id_resp]
    else:
        response = "I'm here to help. Please let me know how I can assist you."
    return response
```



The screenshot shows a Jupyter Notebook interface with a toolbar at the top. The code cell contains three main parts: 1) A comment block for predicting intent. 2) A comment block for generating a response. 3) A print statement that outputs the chatbot's response. The code uses `predict_intent` to get the predicted intent and `generate_response` to get the response for that intent. Finally, it prints "Chatbot:" followed by the generated response.

```
# Predict intent
intent = predict_intent(user_input)

# Generate response
response = generate_response(intent, init_df)

print("Chatbot:", response)
```

User: Hi
Intent 'greeting' found in tags
Chatbot: Hi there. How are you feeling today?
User: I'm feeling sad.
Intent 'sad' found in tags
Chatbot: Why do you think you feel this way?
User: I can't find a reason to live or feel happy.

RESULTADOS DE VÍA 1 DE DESARROLLO.

DataSets:

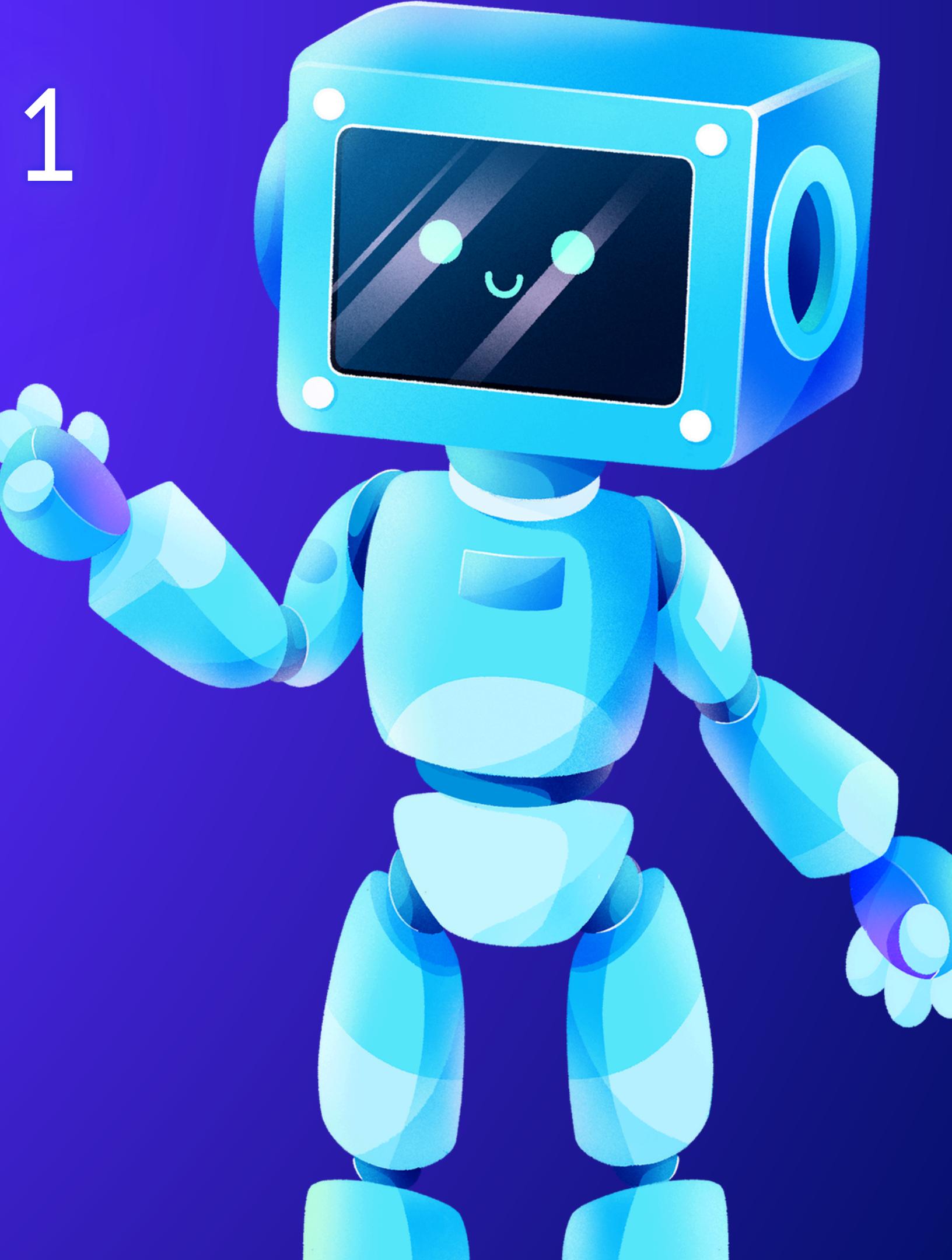
Se han usado 3 DataSets de Hugging Face.

Resultados:

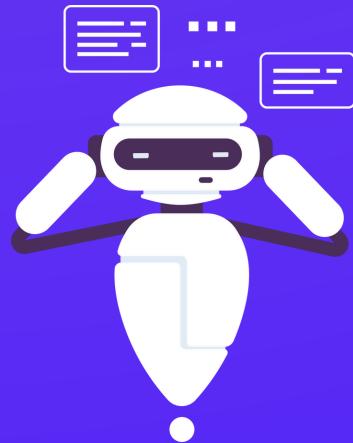
Se han conseguido crear 4 ChatBots, 3 para cada DataSet y un último con la combinación de los DataSets anteriores.

Problemas:

Largos tiempos de entrenamiento, respuestas genéricas y formato de datos.



VÍA DE DESARROLLO 2: ETIQUETADO DE MUESTRAS ADICIONALES



PROBLEMAS:
Inexperiencia con
Python, datos
heterogéneos,
potencia de cálculo

Etiquetado manual

Con una variación del notebook original, se inyectan entradas de otro dataset mayor, se genera la respuesta y se permite al usuario corregirla para aumentar el dataset original. Se han procesado 100 entradas así, creando unas 10 etiquetas nuevas.

Etiquetado automático

Se ha desarrollado un script con PyTorch y Transformers. Mediante pipeline y zero-shot-classification, con la lista de etiquetas del paso anterior, se hace etiquetado automático.

Generación automática de conversaciones

Por último, otra variación del notebook manual se encarga de injectar entradas de otros datasets, obtener la respuesta y volcarla en formato JSONL para fine-tuning de GPT.

FINE-TUNING

Azure OpenAI Studio > Models > ftjob-43156332440f418c80ba9b633f330949

gpt-35-turbo-0125: ftjob-43156332440f418c80ba9b633f330949

Status: Training running

Last updated at: [redacted]
Training file: conv-ft2.jsonl
Base model: gpt-35-turbo-0125

[Download training file](#)

Para probar el modelo, utilizamos el Chat playground que simula una interfaz de Chat para interactuar con nuestro modelo con fine-tuning.

Para el fine-tuning, hemos utilizado la herramienta “Create custom model” de OpenAI. Una vez creado el dataset de entrenamiento en formato .jsonl, lo subimos a la plataforma y lanzamos un entrenamiento.

Chat playground

Setup

Prompt Add your data

Apply changes

Use a system message template

Using templates

Select a template

Start chatting

Test your assistant by sending queries below. Then adjust your assistant setup to improve the assistant's responses.

MODELOS ENTRENADOS

chatchare-1	gpt-35-turbo-0125.ft-43156	1	Standard	1K TPM	Succeede
-----------------------------	--	---	----------	--------	----------

Una vez entrenado el modelo, es necesario desplegarlo a través de la función “deploy” del portal OpenAI. De esta forma, tenemos una instancia accesible del modelo entrenado. Para poder interactuar con este mismo, podemos utilizar APIs o, como se ha enseñado en las diapositivas anteriores, el Playground de OpenAI.

De momento, hemos terminado exitosamente un entreno, cuyos resultados están accesibles a través de este [enlace](#).

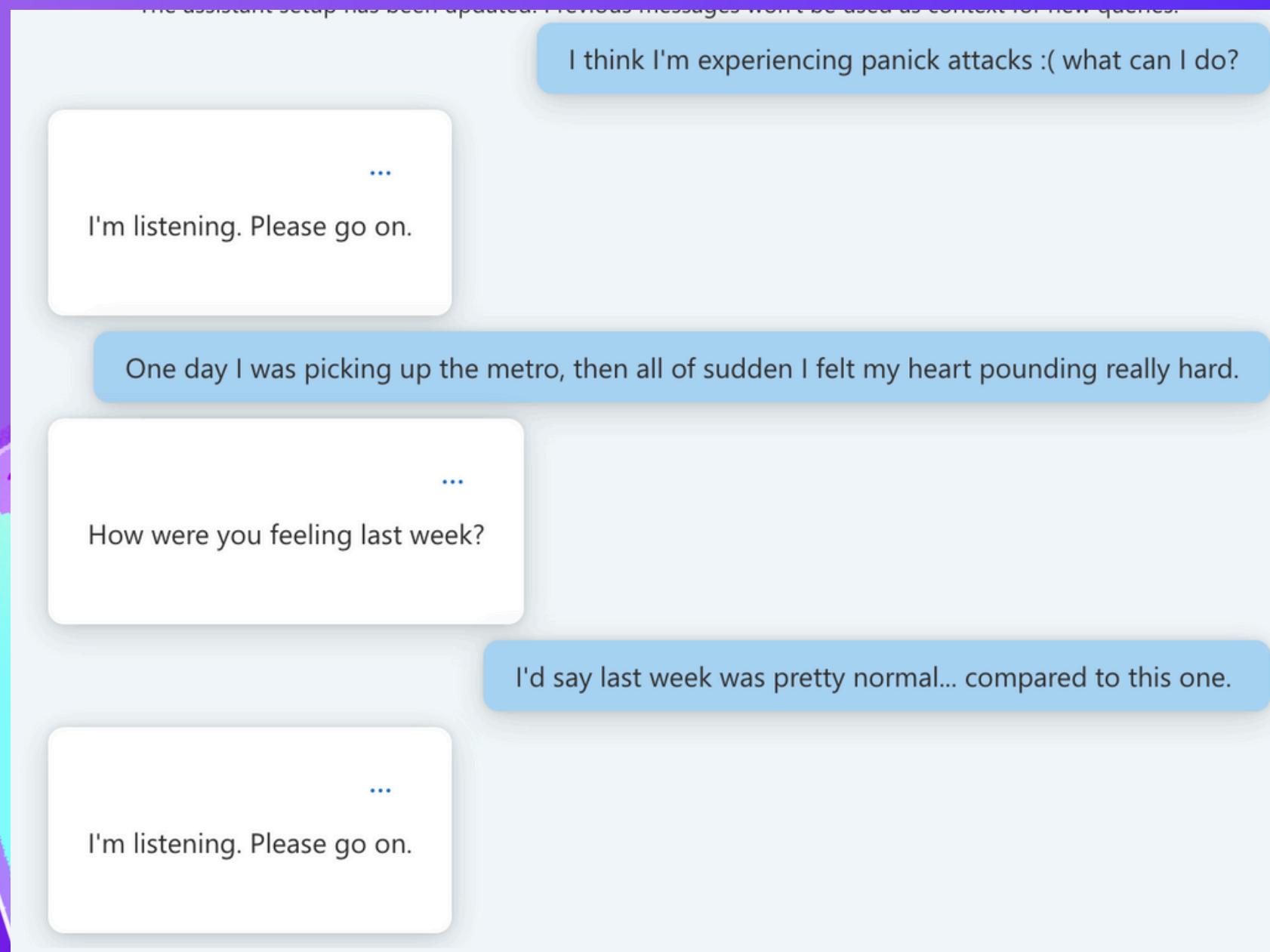
Podemos ver como la pérdida va bajando con cada step de las fases de entreno, desde el step 0 hasta 303.

Es posible también reentrenar el mismo modelo ya entrenado anteriormente para mejorar y perfeccionar el mismo.

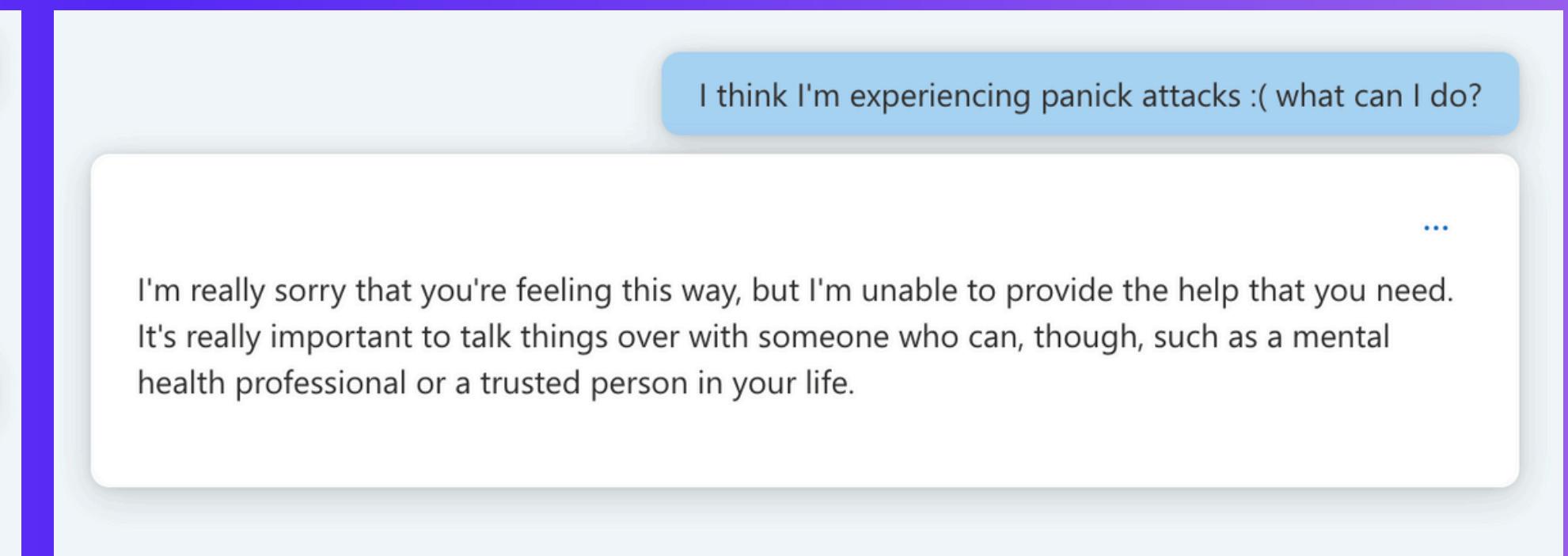
CONVERSACIONES

Ejemplo de conversación con Chatcare

GPT 3.5 con Fine Tuning (Chatcare)

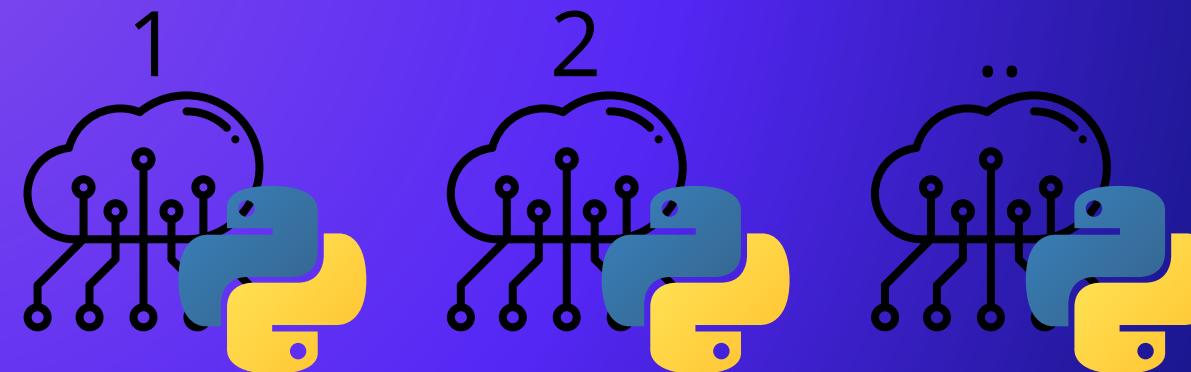


ChatGPT 4 (Sin afinamiento)



RESUMEN - ACTIVIDADES REALIZADAS

1. Bots en Python que nos permiten realizar patrones de respuesta (pareja consulta/respuesta)



2. Revisamos los datos generados para corregirlos si fuera necesario y para convertirlos al formato correcto (JSON)

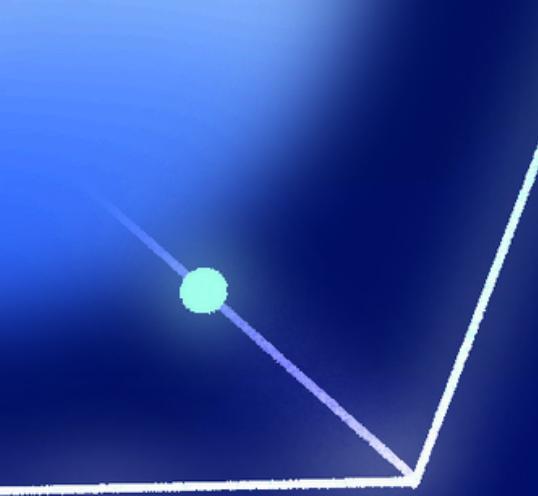


4. El modelo GPT entrenado se queda almacenando en la nube en un entorno Azure y es accesible a través de sus propias APIs



3. Realizamos un programa Python para comunicar con las API de OpenAI y para realizar el fine-tuning





DEMO FINAL

FIN
GRACIAS :)