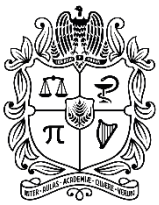

UNIVERSIDAD
NACIONAL
DE COLOMBIA

Manual técnico

Nombre del estudiante: Mallerly Ceballos Trujillo

Universidad Nacional de Colombia
Facultad de Administración
Departamento de Informática y Computación

Manizales, Colombia
2025

Introducción

El modelo Mistral 7B v2 fue seleccionado frente a otros modelos como; llama,bert y los modelos de la familia T5, una de las razones decisivas para esta selección fue su mayor rendimiento en el idioma, a pesar de ser un modelo mucho más grande y que requiere una cantidad considerable de recursos computacionales se encontró un equilibrio entre calidad y recursos computacionales.
Adicionalmente su flexibilidad para ser optimizado mediante cuantización de 4 bits es decir, "reducir el uso de memoria de los parámetros del modelo para que sea más accesible a quienes tienen recursos computacionales limitados", y sus características que van muy alineadas a los requerimientos para la construcción del chatbot, esto fue lo que determinó su selección frente a otros modelos menos flexibles y con un rendimiento inferior.

Plataforma de Implementación:
Para el proceso de fine-tuning del modelo, se evaluaron tres alternativas principales:

1. Google Colab:

   Se planteó ya que es Gratuito y ya se ha usado con anterioridad. Pero no fue seleccionado debido a sus interrupciones frecuentes durante entrenamientos de modelos grandes y que tiene recursos limitados para las sesiones gratuitas.

2. Hugging Face:

   Aunque esta plataforma fue de vital importancia para hallar y evaluar los distintos modelos, y está especializada en ofrecer servicios para el procesamiento y entrenamiento de los modelos, la falta de experiencia en su configuración y uso representaba una dificultad significativa en el desarrollo del proyecto.

   Google Cloud (Seleccionado):

   Se planteó y seleccionó como la opción más adecuada por los siguientes factores, cuenta con una amplia comunidad y una amplia cantidad de recursos digitales de aprendizaje desde conceptos básicos hasta implementaciones más avanzadas, lo que brinda una guía más sólida para el uso de la plataforma y entrenamiento del modelo.

   Adicionalmente la plataforma ofrece un programa de créditos iniciales a nuevos usuarios, lo que permite familiarizarse con la herramienta, y crear el entorno virtual y configuraciones iniciales para el entrenamiento del modelo seleccionado.

Configuración o preparación del entorno.

1. En Google Cloud- en "compute engine" se crea una máquina virtual con las siguientes características:

Se selecciona la máquina con GPU NVIDIA A100 40GB (Para este modelo se recomienda: NVIDIA T4 o A100 Mínimo 32GB RAM) También se selecciona un sistema operativo compatible con CUDA como ubuntu

adjuntar foto. (configuración máquina)

2. acceder o conectarse a la máquina virtual mediante SSH

Aquí se procede a instalar todas las dependencias necesarias, es decir los paquetes y herramientas que se requieren para el entrenamiento.

```
sudo apt update && sudo apt install -y git python3-pip

pip install transformers peft datasets accelerate bitsandbytes
```

Antes de comenzar el entrenamiento de un modelo, es importante verificar si tenemos acceso a una GPU NVIDIA con soporte para CUDA, esto sirve para acelerar significativamente el proceso. Para esto se hace uso del siguiente comando:

```
python3 -c "import torch; print(torch.cuda.is_available())"
```

En caso de que no se encuentre disponible se instala de la siguiente manera:

```
#Se descarga y se importa la clave del repositorio de NVIDIA

sudo apt-key adv --fetch-keys
https://developer.download.nvidia.com/compute/cuda/repos/ubuntu20
04/x86_64/3bf863cc.pub

sudo add-apt-repository "deb
https://developer.download.nvidia.com/compute/cuda/repos/ubuntu20
04/x86_64/ /"



# Se instala CUDA Toolkit

sudo apt update

sudo apt install -y cuda



# Se configuran las variables de entorno
```

```
echo 'export PATH=/usr/local/cuda/bin:$PATH' >> ~/.bashrc

echo 'export
LD_LIBRARY_PATH=/usr/local/cuda/lib64:$LD_LIBRARY_PATH' >>
~/.bashrc

source ~/.bashrc
```

Se carga el archivo `train_mistral.py` en el cual encontramos:

Primero se importa las librerías necesarias para trabajar con el modelo extraído de la plataforma hugging face, así como otras para el manejo del dataset.

```
from huggingface_hub import login

from transformers import AutoTokenizer, AutoModelForCausalLM,
Trainer, TrainingArguments, BitsAndBytesConfig

from datasets import Dataset, load_dataset

import torch

from peft import LoraConfig, get_peft_model,
prepare_model_for_kbit_training

import json

import os
```

En el archivo también se tiene una verificación de la versión de PyTorch (Biblioteca de aprendizaje automático y procesamiento de lenguaje natural) y la disponibilidad de CUDA

```
print("\n=== Verificación del Sistema ===")

print(f"PyTorch versión: {torch.__version__}")

print(f"CUDA disponible: {torch.cuda.is_available()}")

if torch.cuda.is_available():

    print(f"CUDA versión: {torch.version.cuda}")

    print(f"GPU disponible: {torch.cuda.get_device_name(0)}")
```

```
    print(f"Número de GPUs: {torch.cuda.device_count()}")

    print(f"Memoria GPU total:
{torch.cuda.get_device_properties(0).total_memory / 1024**3:.2f}
GB")

    print(f"Memoria GPU disponible:
{torch.cuda.memory_allocated(0) / 1024**3:.2f} GB usado")

print(f"BNB_CUDA_VERSION: {os.getenv('BNB_CUDA_VERSION')}")

print(f"LD_LIBRARY_PATH: {os.getenv('LD_LIBRARY_PATH')}")

print("==============================\n")
```

Se configura el token de autenticación de Hugging Face (Para acceder al modelo se requiere tener una cuenta en hugging face y generar un token) se especifica también el nombre del modelo.

```
HUGGING_FACE_TOKEN = "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"  #
(Aquí va el token generado desde hugging face para acceder al
modelo)

MODEL_NAME = "mistralai/Mistral-7B-Instruct-v0.2"

OUTPUT_DIR = "./results_mistral" #(Directorio donde se va a
guardar el modelo después del entrenamiento)



# Login en Hugging Face

print("Autenticando en Hugging Face...")

login(HUGGING_FACE_TOKEN)
```

Se realiza una Cuantización en 4 bits (técnica para reducir el tamaño del modelo) La configuración de bnb_config permite cargar el modelo en memoria reducida

```
bnb_config = BitsAndBytesConfig(

    load_in_4bit=True,

    bnb_4bit_quant_type="nf4",
```

```
    bnb_4bit_compute_dtype=torch.bfloat16,

    bnb_4bit_use_double_quant=True

)
```

bnb_4bit_compute_dtype=torch.bfloat16: Optimiza el rendimiento de los cálculos sin sacrificar la precisión.

load_in_4bit=True: Carga el modelo en formato de 4 bits para reducir el uso de memoria.

Se realiza el cargue del dataset (El dataset está construido con 1542 registros)

```
print("Cargando dataset...")

with open('dataset_historias_usuario.json', 'r',
encoding='utf-8') as f:

    data = json.load(f)



raw_dataset = Dataset.from_dict({

    'conversation': [item['conversation'] for item in data],

    'user_story': [item['user_story'] for item in data]

})

print(f"\nTamaño del dataset creado: {len(raw_dataset)}")

print("\nEjemplo del primer registro:")

print(f"Conversación: {raw_dataset[0]['conversation']}")

print(f"Historia de usuario: {raw_dataset[0]['user_story']}"
```

Se realiza el cargue del modelo y se configura las opciones de cuantización

```
print("Cargando tokenizer y modelo...")

tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME,
trust_remote_code=True)

tokenizer.pad_token = tokenizer.eos_token
```

```
tokenizer.padding_side = "right"

model = AutoModelForCausalLM.from_pretrained(

    MODEL_NAME,

    quantization_config=bnb_config,

    device_map="auto",

    trust_remote_code=True

)

model.config.pad_token_id = tokenizer.pad_token_id

print(f"Memoria GPU usada después de cargar modelo:
{torch.cuda.memory_allocated(0) / 1024**3:.2f} GB")
```

Uso y configuración de Lora para ajustar algunas partes específicas del modelo y que sea más eficiente y menos costoso computacionalmente para el ajuste fino.

```
lora_config = LoraConfig(

    r=16,  # rango de la descomposición de bajo rango

    lora_alpha=32,  # parámetro de escalado para ajustar la tasa
de aprendizaje

    target_modules=["q_proj", "k_proj", "v_proj", "o_proj"],  #
módulos objetivo para la actualización de bajo rango

    lora_dropout=0.05,  # probabilidad de dropout para
regularización

    bias="none",  # tipo de sesgo a aplicar (aquí no se aplica
sesgo)

    task_type="CAUSAL_LM"  # tipo de tarea, en este caso, un
modelo de lenguaje causal

)

print("Preparando modelo para entrenamiento...")

model = prepare_model_for_kbit_training(model)
```

```
model = get_peft_model(model, lora_config)
```

Explicación detallada

`r=16`: Cuánto se va a actualizar sin tener que actualizar todo el modelo.

`lora_alpha=32`:cuán fuerte son los cambios.

`target_modules=["q_proj", "k_proj", "v_proj", "o_proj"]`: Estas son las "piezas" específicas del modelo que vas a actualizar.

`lora_dropout=0.05`: en donde se va a omitir la actualización o no se va a cambiar para evitar hacer cambios innecesarios.

`bias="none"`: Significa que no estás añadiendo ningún sesgo extra en tus cambios.

`task_type="CAUSAL_LM"`: Indica que el modelo se va a usar para una tarea de generación de texto.

Se define cómo preprocesar los datos del dataset para que sean compatibles con el modelo.

```
def preprocess_function(examples):
    inputs = [
        f"<s>[INST] Basado en esta conversación, genera una historia de usuario:\n{conv} [/INST]"
        for conv in examples["conversation"]
    ]
    targets = [f"{story}</s>" for story in examples["user_story"]]
    concatenated = [inp + target for inp, target in zip(inputs, targets)]
    model_inputs = tokenizer(
        concatenated,
        max_length=512,
```

```
        padding="max_length",
        truncation=True,
        return_tensors="pt"
    )

    model_inputs["labels"] = model_inputs["input_ids"].clone()
    return model_inputs
```

Se realiza una división del dataset en datos de entrenamiento y de prueba. el 80% del dataset se usa para el entrenamiento y el 20% restante para pruebas y validación.

```
print("Preprocesando dataset...")
tokenized_dataset = raw_dataset.map(
    preprocess_function,
    batched=True,
    remove_columns=raw_dataset.column_names
)
tokenized_dataset = tokenized_dataset.shuffle(seed=42)

train_size = int(0.8 * len(tokenized_dataset))
train_data = tokenized_dataset.select(range(train_size))
eval_data = tokenized_dataset.select(range(train_size,
len(tokenized_dataset)))
```

Configuración de los argumentos o parámetros de entrenamiento.

```
training_args = TrainingArguments(
    output_dir=OUTPUT_DIR,
```

```
    evaluation_strategy="steps",
    eval_steps=25,
    learning_rate=1e-4,
    per_device_train_batch_size=2,
    per_device_eval_batch_size=2,
    num_train_epochs=5,
    weight_decay=0.01,
    save_total_limit=3,
    logging_dir="./logs",
    logging_steps=10,
    save_strategy="steps",
    save_steps=25,
    load_best_model_at_end=True,
    metric_for_best_model="eval_loss",
    warmup_steps=100,
    gradient_accumulation_steps=8,
    fp16=True,
    gradient_checkpointing=True,
)


trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_data,
    eval_dataset=eval_data,
```

```
)
```

Se inicia el entrenamiento y se guarda el modelo entrenado.

```
print("Iniciando entrenamiento...")

trainer.train()



print("Guardando modelo...")

model_path = "./mistral_user_story_generator"

model.save_pretrained(model_path)

tokenizer.save_pretrained(model_path)

print(f"Modelo guardado en {model_path}")
```

Dentro del script se incluye una función de prueba.

```
def generate_user_story(conversation_text, model, tokenizer):

    prompt = f"<s>[INST] Basado en esta conversación, genera una
historia de usuario:\n{conversation_text} [/INST]"

    inputs = tokenizer(prompt, return_tensors="pt",
max_length=512, truncation=True)

    inputs = inputs.to(model.device)



    with torch.no_grad():

        outputs = model.generate(

            inputs.input_ids,

            max_length=512,

            temperature=0.7,

            top_p=0.9,
```

```
            do_sample=True,
            num_return_sequences=1,
            pad_token_id=tokenizer.pad_token_id,
            eos_token_id=tokenizer.eos_token_id
        )
    return tokenizer.decode(outputs[0], skip_special_tokens=True)
print("\nRealizando pruebas finales...")
test_inputs = [
    "Sistema de búsqueda avanzada",
    "Panel de administración",
    "Exportación de reportes"
]

for test_input in test_inputs:
    try:
        generated_story = generate_user_story(test_input, model,
tokenizer)
        print(f"\nInput: {test_input}")
        print(f"Generated: {generated_story}")
    except Exception as e:
        print(f"Error generando historia para '{test_input}':
{str(e)}")

print("\n¡Proceso completado!")
```

Se ejecuta el script de entrenamiento

```
python3 train_mistral.py
```

Después de ejecutado el script de entrenamiento se debe verificar el entrenamiento del modelo esto lo puedes hacer revisando los logs y métricas del entrenamiento o exportar el modelo y probarlo para realizar una validación de las historias de usuario generadas.

Para el uso del modelo se brinda el siguiente script

```
# Onlinimport tkinter as tk

from tkinter import scrolledtext, Menu

from transformers import AutoModelForCausalLM, AutoTokenizer

import torch



# Cargar el modelo

model_path = "./mistral_user_story_generator"

tokenizer = AutoTokenizer.from_pretrained(model_path)

model = AutoModelForCausalLM.from_pretrained(model_path,
device_map="auto")



def generar_respuesta(conversacion):

    prompt = f"[INST] {conversacion} [/INST]"

    inputs = tokenizer(prompt, return_tensors="pt", max_length=512,
truncation=True, padding=True).to(model.device)



    with torch.no_grad():

        outputs = model.generate(

            input_ids=inputs['input_ids'],

            attention_mask=inputs['attention_mask'],
```

```
            max_length=512,
            temperature=0.7,
            top_p=0.9,
            do_sample=True,
            num_return_sequences=1,
            pad_token_id=tokenizer.pad_token_id,
            eos_token_id=tokenizer.eos_token_id
        )

    response = tokenizer.decode(outputs[0], skip_special_tokens=True)
    if "[INST]" in response:
        response = response.split("[/INST]")[-1].strip()
    return response
def conversacion_interactiva():
    conversacion = []
    funcionalidad = input("¿Qué funcionalidad necesitas? ")
    conversacion.append(f"User: {funcionalidad}")
    print("System: ¿Cuál es tu rol?")
    rol = input("User: ")
    conversacion.extend(["System: ¿Cuál es tu rol?", f"User: {rol}"])
    print("System: ¿Qué características específicas necesitas?")
    caracteristicas = input("User: ")
    conversacion.extend(["System: ¿Qué características específicas
necesitas?", f"User: {caracteristicas}"])
    print("System: ¿Algún requisito adicional?")
```

```
    requisitos = input("User: ")

    conversacion.extend(["System: ¿Algún requisito adicional?", f"User: {requisitos}"])



    conversacion_completa = "\n".join(conversacion)

    prompt_final = f"{conversacion_completa}\nSystem: Genera una historia de usuario en formato Como/Quiero/Para basada en esta conversación."



    return generar_respuesta(prompt_final)



# Configuración de la ventana en Tkinter

ventana = tk.Tk()

ventana.title("Chatbot - Generador de Historias")

ventana.geometry("500x600")

ventana.configure(bg="#f4f4f4")



# Menú principal

menu_bar = Menu(ventana)

ventana.config(menu=menu_bar)



file_menu = Menu(menu_bar, tearoff=0)

file_menu.add_command(label="Nueva conversación", command=lambda: chat_log.config(state=tk.NORMAL) or chat_log.delete("1.0", tk.END) or chat_log.config(state=tk.DISABLED))

file_menu.add_separator()

file_menu.add_command(label="Salir", command=ventana.quit)
```

```
menu_bar.add_cascade(label="Archivo", menu=file_menu)


# Contenedor del chat
frame_chat = tk.Frame(ventana, bg="white", padx=10, pady=10)
frame_chat.pack(pady=10, padx=10, fill=tk.BOTH, expand=True)


chat_log = scrolledtext.ScrolledText(frame_chat, wrap=tk.WORD,
width=60, height=25, state=tk.DISABLED, font=("Arial", 12))
chat_log.pack(padx=10, pady=10, fill=tk.BOTH, expand=True)


chat_log.tag_config("user", foreground="#007BFF", font=("Arial", 12,
"bold"))
chat_log.tag_config("bot", foreground="#333", font=("Arial", 12))


def enviar_mensaje():
    user_input = entrada_usuario.get()
    if user_input.strip():
        chat_log.config(state=tk.NORMAL)
        chat_log.insert(tk.END, f"Usuario: {user_input}\n", "user")
        chat_log.config(state=tk.DISABLED)
        entrada_usuario.delete(0, tk.END)


        respuesta = generar_respuesta(user_input)
        chat_log.config(state=tk.NORMAL)
        chat_log.insert(tk.END, f"Asistente: {respuesta}\n\n", "bot")
        chat_log.config(state=tk.DISABLED)
```

```
        chat_log.yview(tk.END)



frame_input = tk.Frame(ventana, bg="#f4f4f4")

frame_input.pack(pady=10, padx=10, fill=tk.X)



entrada_usuario = tk.Entry(frame_input, width=40, font=("Arial", 12))

entrada_usuario.pack(side=tk.LEFT, padx=5, pady=5, fill=tk.X,
expand=True)



boton_enviar = tk.Button(frame_input, text="Send",
command=enviar_mensaje, bg="#007BFF", fg="white", font=("Arial", 12,
"bold"))

boton_enviar.pack(side=tk.RIGHT, padx=5, pady=5)



ventana.mainloop()
```

Conclusiones

Al hacer uso de servicios en la nube como Google Cloud para el entrenamiento del modelo se debe monitorear el uso de la instancia o máquina virtual para evitar gastos innecesarios,Para optimizar gastos, se recomienda detener o eliminar la instancia cuando no esté en uso.

Adicionalmente si se hace uso de una instancia de menor capacidad por temas de accesibilidad o de costos es muy posible que se reduzca el rendimiento del modelo y que los tiempos de ajuste fino sean mucho más largos, lo que incurrirá en gastos por el tiempo de uso de la instancia. Es por esto que se recomienda evaluar previamente los recursos computacionales con los que se cuenta y verificar que si se cumple con los requerimientos mínimos del modelo. También se puede hacer una revisión mayor de técnicas de cuantización y entrenar solo unas capas del modelo para que no resulte tan pesado y optimizar el uso de memoria del modelo. También es importante evaluar si hay

actualizaciones en las bibliotecas y dependencias instaladas como transformers y torch, para garantizar compatibilidad y mejoras en el rendimiento.

Este manual proporciona una guía para replicar el ajuste fino realizado al modelo Mistral 7B, para cualquier consulta adicional sobre la configuración o el código, se recomienda revisar la documentación oficial de Hugging Face y Google Cloud.