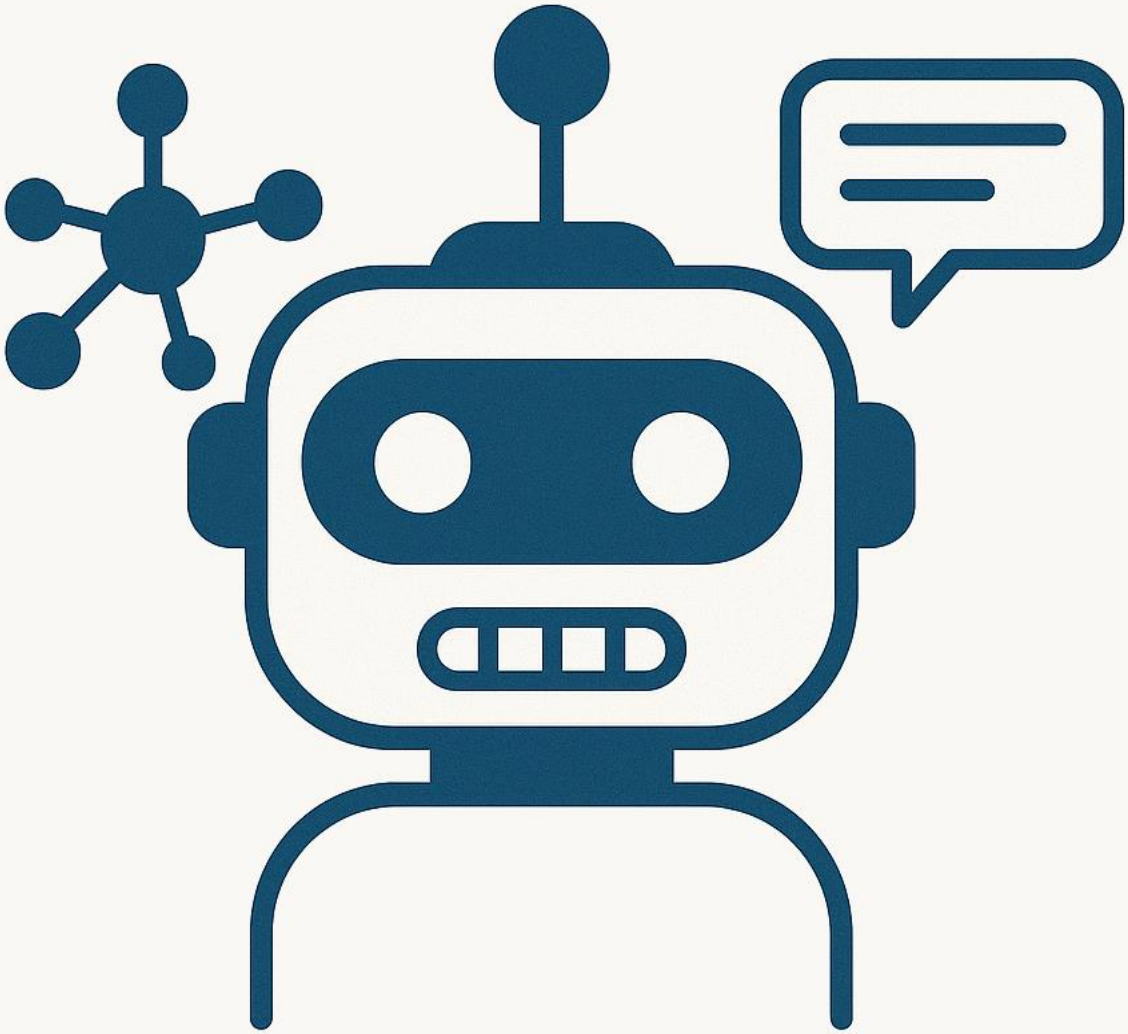
ESTRUCTURA Y RAZONAMIENTO
EN MODELOS DE LENGUAJE



Certex AI

Introducción

Los modelos de lenguaje se han convertido en herramientas fundamentales para procesar, generar y comprender texto de manera automática. Su estructura interna y capacidad de razonamiento permiten aplicaciones cada vez más sofisticadas, desde asistentes virtuales hasta sistemas de análisis inteligente.

Este taller, **"Estructura y Razonamiento en Modelos de Lenguaje"**, tiene como objetivo brindar una comprensión clara y práctica del funcionamiento de los LLMs. Exploraremos su arquitectura, el procesamiento de texto mediante tokens, técnicas de prompting y principios para diseñar interacciones efectivas.

A través de ejercicios interactivos y ejemplos reales, los participantes aprenderán cómo guiar el comportamiento de los modelos de lenguaje, utilizar herramientas externas y aplicar sus capacidades en escenarios concretos. Al finalizar, contarán con las bases necesarias para aprovechar el potencial de estos modelos en sus propios proyectos.

Temario

Recursos Necesarios

Conocimientos Básicos de:

- Windows.
- Editores de Texto.
- Python para la parte práctica (no necesario para la parte teórica).

1. Estructura y razonamiento en modelos de lenguaje

Un **modelo de lenguaje** es un sistema de inteligencia artificial entrenado para comprender y generar texto de forma coherente. Utiliza redes neuronales profundas para aprender patrones en grandes volúmenes de datos, lo que le permite captar el contexto, la estructura y el significado del lenguaje. Estos modelos se aplican en tareas como redacción, resumen, respuesta a preguntas o generación de código, adaptándose a distintos entornos según su implementación local o en línea.

1.1. Modelo de lenguaje

Un **modelo de lenguaje** es un sistema de inteligencia artificial diseñado para comprender y generar texto de forma similar a como lo haría un ser humano. Utiliza grandes volúmenes de datos y técnicas de **aprendizaje profundo** para aprender patrones y relaciones entre palabras.

Los **Large Language Models (LLMs)**, como ChatGPT, Claude o Gemini, funcionan a través de **redes neuronales profundas** que transforman el texto en representaciones numéricas (tokens). Estas representaciones permiten al modelo captar el contexto, la estructura y el significado del lenguaje natural, facilitando tareas como responder preguntas, redactar textos, traducir o resumir información.

En esencia, estos modelos no solo "leen" el texto, sino que **aprenden a predecir y razonar sobre él**, generando respuestas coherentes y relevantes en función del contexto.

1.1.1. Modelos Locales

Los **modelos de lenguaje locales** son modelos de inteligencia artificial que pueden ejecutarse directamente en dispositivos personales o servidores privados, sin necesidad de depender de servicios en la nube. Esto ofrece ventajas como mayor privacidad, menor latencia y mayor control sobre los recursos.

Con el auge de iniciativas de código abierto, modelos como **GPT-2**, **LLaMA**, **BERT**, **RoBERTa** y **Mistral** se han vuelto ampliamente accesibles. Cada uno ofrece diferentes enfoques y niveles de complejidad para tareas como generación automática de texto, análisis de contexto, o interacción con usuarios.

- **GPT-2**
  Desarrollado por: OpenAI

Tipo: Modelo autoregresivo basado en Transformers
Características:
- Uno de los primeros LLMs de código abierto.
- Puede generar texto coherente a partir de un prompt.
- Fácil de usar localmente para tareas de generación de lenguaje.

Licencia: Código abierto (permisivo)

- **LLaMA 2 / LLaMA 3**
  Desarrollado por: Meta (Facebook AI Research)
  Tipo: Modelos autoregresivos de tipo decoder-only
  Características:
  - Diseñados para ser eficientes y ejecutables localmente.
  - Entrenados en grandes corpus multilingües.
  - LLaMA 3 incluye mejoras significativas en comprensión y generación de texto.

  Licencia: Acceso restringido con uso local permitido (según condiciones de Meta)

- **BERT**
  Desarrollado por: Google AI
  Tipo: Modelo encoder-only, basado en transformers
  Características:
  - Procesa texto de forma bidireccional (izquierda y derecha al mismo tiempo).
  - Base para tareas como clasificación, reconocimiento de entidades y preguntas-respuestas.
  - Muy utilizado en modelos preentrenados para NLP.

  Licencia: Código abierto

- **RoBERTa**
  Desarrollado por: Facebook AI
  Tipo: Variante de BERT optimizada
  Características:
  - Entrenado con más datos y sin segmentación Next Sentence Prediction.
  - Mejor rendimiento que BERT en muchas tareas de NLP.
  - Compatible con tareas de clasificación y análisis semántico.

  Licencia: Código abierto

- **Mistral**
  Desarrollado por: Mistral AI
  Tipo: LLM ligero, decoder-only, arquitectura tipo transformer
  Características:

- Modelo pequeño pero potente, optimizado para ejecutarse en equipos locales.
- Alto rendimiento en generación de texto con menos recursos.
- Orientado a tareas eficientes en tiempo real o ambientes con restricciones de hardware.

Licencia: Código abierto

1.1.2. Modelos en Línea

Los **modelos de lenguaje en línea** son accesibles a través de plataformas web o APIs, y se ejecutan en la nube. Esto permite aprovechar una gran capacidad de cómputo, actualizaciones constantes y mejoras automáticas en el modelo. Son ideales para aplicaciones que requieren alto rendimiento, acceso a modelos de última generación y facilidad de integración.

- **GPT-3 / GPT-4 / GPT-4-turbo**
  Desarrollado por: OpenAI
  Tipo: Modelo autoregresivo, basado en Transformers
  Características:
  - Capaces de generar texto, razonar, traducir, programar, resumir, y más.
  - GPT-4-turbo es más rápido y económico que GPT-4 estándar.
  - Utilizados en múltiples entornos profesionales y educativos.

  Licencia: Licencia comercial (no open source)

- **Gemini**
  Desarrollado por: Google DeepMind
  Acceso: A través del asistente Gemini (antes Bard), en su versión web o desde productos integrados de Google.
  Tipo: Modelo multimodal autoregresivo, basado en Transformers
  Características:
  - Modelo multimodal (procesa texto, imagen, código).
  - Enfocado en respuestas detalladas y razonamiento complejo.
  - Integrado en herramientas como Google Workspace (Docs, Sheets).

  Licencia: Licencia comercial (no open source)

- **Claude 2 / Claude 3**
  Desarrollado por: Anthropic
  Acceso: A través de su sitio oficial claude.ai o vía API
  Tipo: Modelo autoregresivo, basado en Transformers
  Características:
  - Diseñado con énfasis en seguridad, transparencia y alineación ética.

- Claude 3 ofrece mejoras en razonamiento, velocidad y comprensión.
- Especialmente valorado en contextos donde se requiere control en el comportamiento del modelo.

Licencia: Licencia comercial (no open source)

1.1.3. Multimodales

Los modelos multimodales pueden analizar distintos tipos de datos, como:

- **Imágenes** (por ejemplo, interpretar una fotografía o leer texto en una captura de pantalla)
- **Documentos** (PDFs, presentaciones, facturas)
- **Tablas y gráficos**
- **Videos** (en modelos más avanzados)
- **Audio** (como transcribir o analizar voz)

2. Estructura interna de un modelo de lenguaje

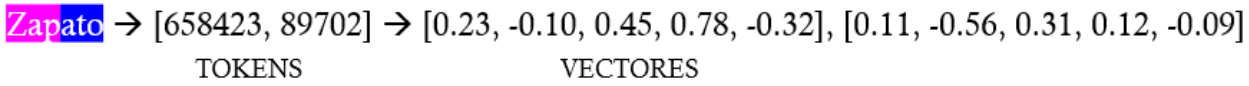
Al interactuar con modelos de lenguaje, es fundamental entender cómo procesan el texto. Este procesamiento implica dividir las frases en unidades mínimas llamadas ***tokens***, que son transformadas en **vectores numéricos** mediante un ***tokenizador***. Estos vectores son analizados por **redes neuronales**, como las **RNN** o **Transformers**, que permiten al modelo identificar patrones, establecer relaciones entre palabras y generar respuestas coherentes. Elementos como el **contexto**, lo que el modelo "recuerda" de una conversación o documento, y **parámetros ajustables**, que controlan su comportamiento, también influyen directamente en la calidad y precisión de sus salidas. Comprender estos mecanismos permite aprovechar al máximo las capacidades de los LLMs.

2.1. Tokens

Un **token** es una **unidad básica de texto** que un modelo de lenguaje procesa. Puede ser una palabra, una parte de palabra, o incluso un carácter, dependiendo de cómo esté diseñado el modelo.

Los modelos de lenguaje no trabajan con texto, sino que las palabras las divide en tokens (**Tokenizador**) y esos tokens se convierten en vectores numéricos para procesarlos (**Modelo**).

Ejemplo de vectores:



2.2. Tokenizador

El **tokenizador** es el componente que convierte el texto en una secuencia de tokens (unidades mínimas de significado). Por ejemplo:

"**Hola**" → [15496]

El **modelo** procesa esos tokens, convierte en vectores numéricos. Usa esos vectores para razonar, predecir o clasificar. Puede realizar operaciones como:

- **Atención**: encuentra qué partes del texto son importantes.
- **Transformaciones lineales**: aplica pesos aprendidos.
- **Funciones de activación**: introduce no linealidad.
- **Predicción**: genera una salida (por ejemplo, el siguiente token).

Notas importantes:

- ***Cada modelo está entrenado con un tokenizador específico***. Si usas otro, el modelo no entenderá bien el prompt.
  - GPT-2 usa un tokenizador diferente al de GPT-3.
  - GPT-4 (especialmente GPT-4-128k) usa un tokenizador más avanzado llamado **tiktoken.**
- El **token** no es el texto, sino un número entero único asignado a cada unidad.
- El **tokenizador** también maneja espacios, puntuación, y acentos como tokens separados si es necesario.

2.2.1. Ejercicio 01 - Tokens.ipynb

Para acceder al ejercicio, es necesario acceder al archivo **Tokens.ipynb**, ubicado en la carpeta: **1.Estructura Interna**.

Para el ejercicio se utilizará la librería llamada tiktoken que permite:

- **Codificar** texto en tokens compatibles con modelos de OpenAI como GPT-2, GPT-3, GPT-4, etc.
- **Decodificar** tokens de vuelta a texto.
- **Contar tokens**, útil para saber cuántos tokens estás usando y evitar exceder los límites del modelo.
- **Usar diferentes codificadores** según el modelo (por ejemplo: cl100k_base para GPT-4 y p50k_base para GPT-3.5).

Para instalarlo e importarlo se utiliza lo siguiente:

```
%pip install tiktoken
import tiktoken
```

En esta ocasión se realizará un ejercicio relacionado con el conteo de tokens:

1. Primero se elige el modelo a utilizar, para el ejemplo usaremos **gpt-3.5-turbo:**

```
encoding = tiktoken.encoding_for_model("gpt-3.5-turbo")
```

2. Se define la función que analizará el texto deseado y realizará el conteo:

```
def analizar_texto(texto):
    tokens = encoding.encode(texto)
    print(f"\nTexto original: {texto}")
    print(f"Número de tokens: {len(tokens)}")
    print("Tokens individuales:")
    for i, token in enumerate(tokens):
        decoded = encoding.decode([token])
        print(f"{i+1:02d} → Token ID: {token} → Texto: '{decoded}'")
```

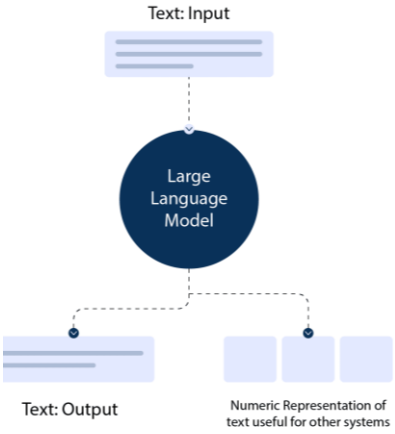
3. Por último, se añadirá un texto/frase a analizar y se llamará la función que se creó en el punto anterior:

```
texto_usuario = input("Escribe una frase para analizar: ")
analizar_texto(texto_usuario)
```

Para continuar con los ejercicios relacionados con este tema: ***Consultar el ejercicio práctico en el archivo Python: Tokens.ipynb***

2.3. ¿Cómo procesan el texto los LLM?

Los modelos de lenguaje (LLM) procesan el texto transformándolo en representaciones numéricas que capturan el significado y contexto de las palabras. Para lograrlo, utilizan arquitecturas de redes neuronales diseñadas específicamente para manejar secuencias, como las **Redes Neuronales Recurrentes (RNN)** y, más recientemente, los **Transformers.**



2.3.1. Redes Neuronales Recurrentes (RNN)

Las **RNN** son un tipo de **red neuronal** diseñada para trabajar con datos secuenciales, como texto o audio. A diferencia de las redes tradicionales, las RNN tienen conexiones recurrentes que les permiten recordar información anterior, lo que las hace especialmente útiles para procesar lenguaje natural.

Procesamiento secuencial:
Las RNN procesan el texto **palabra por palabra**, respetando el orden original de la secuencia. Esta estructura permite capturar dependencias temporales en los datos.

Memoria:
Mantienen un **estado oculto** (hidden state) que se actualiza en cada paso de la secuencia. Este estado actúa como una memoria interna que conserva información del contexto anterior, lo que ayuda al modelo a interpretar mejor cada nueva palabra en función de lo ya leído.

Ejemplo de uso:

“El cliente compró una laptop y un…”

Gracias a su memoria, una RNN puede inferir que lo siguiente podría ser **“mouse”**, **“cargador”** o **“estuche”**, considerando el contexto previo de la frase.

Aplicaciones comunes de las RNN:

- Completado de texto y predicción de la siguiente palabra.
- Generación de texto.
- Traducción automática.
- Reconocimiento de voz.

- Modelado del tiempo en series temporales (predicción de precios, clima, etc.).

Limitaciones

Las RNN simples tienen problemas para recordar información lejana en secuencias largas debido a lo que se conoce como el **problema del desvanecimiento o explosión del gradiente**. Para mitigar esto, se desarrollaron arquitecturas más potentes y eficientes como los **Transformers**.

2.3.2. Transformers

Los **Transformers** son una **arquitectura de red neuronal** avanzada diseñada específicamente para procesar secuencias de datos (como el lenguaje natural o texto), pero de forma más eficiente y poderosa que las redes neuronales recurrentes (RNN).

A diferencia de las RNN, los **Transformers** no procesan el texto secuencialmente, sino que analizan toda la secuencia al mismo tiempo. Esto les permite aprender relaciones entre palabras sin importar la distancia entre ellas, mejorando tanto la velocidad como la calidad del análisis.

Su principal innovación es el uso del mecanismo de **autoatención (self-attention),** que permite al modelo enfocarse en las partes más relevantes del texto para cada palabra que analiza. Por ejemplo, en una oración compleja, el modelo puede detectar automáticamente qué términos se relacionan entre sí, aunque estén separados por varias palabras.

En esta **arquitectura de red neuronal** los tres componentes clave son:

1. **Embedding**: Convierte el texto de entrada en vectores numéricos que capturan el significado de las palabras o subpalabras.
2. **Bloque Transformer** (que incluye Mecanismo de Atención + Capa MLP): **Es el núcleo del modelo**. Se encarga de procesar los vectores y entender el contexto entre las palabras.
    1. Mecanismo de Atención: conecta un token con todos los demás.
    2. MLP: ajusta y refina la representación de cada token.
3. **Probabilidades de salida**: Transforma las representaciones procesadas en predicciones de cuál será el siguiente token. Esto se hace usando capas finales lineales y softmax.

2.3.3. Self-Attention

El mecanismo de **self-attention** (autoatención) es una de las claves del funcionamiento de los **Transformers**. Es un mecanismo que permite al modelo asignar diferentes niveles de importancia a cada palabra del texto en función de las demás, es decir, permite que el modelo determine **qué palabras del texto son más relevantes** para comprender el significado de cada parte de la oración.

A diferencia de modelos tradicionales que tratan todas las palabras con igual peso o solo consideran las más cercanas, la autoatención **asigna distintos niveles de importancia** a cada palabra dependiendo del contexto completo. Es decir, el modelo **se "mira a sí mismo"** para decidir qué partes del texto deben influir más en la interpretación de una palabra específica.

Por ejemplo, en la frase:

"La computadora que compró Juan ayer llegó rota"

Gracias a **self-attention**, el modelo puede entender que ***"que compró Juan"*** se refiere a ***"computadora"*** y no a ***"ayer"*** o ***"llegó"***, estableciendo relaciones correctas entre palabras incluso si están separadas por otras.

Este mecanismo es fundamental para que los modelos modernos comprendan textos complejos, identifiquen relaciones semánticas y generen respuestas más precisas y coherentes.

2.4. Contexto

El **contexto** es uno de los elementos más importantes para que un modelo de lenguaje genere respuestas precisas, coherentes y relevantes. Se refiere a toda la información que el modelo utiliza para comprender y generar respuestas o realizar tareas.

El contexto como la "memoria de trabajo" del LLM, incluye:

- La Instrucción o Pregunta Inicial (Prompt)
- Información Anterior en la Conversación
- Datos Proporcionados Adicionalmente (documentos, fragmentos de código o textos largos)
- Ejemplos (Few-shot learning)

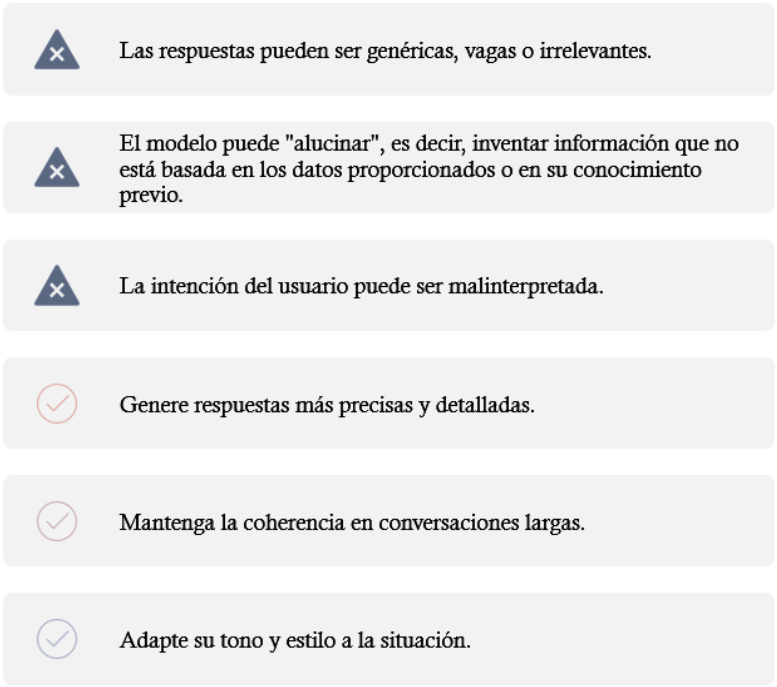
Un aspecto fundamental relacionado con el contexto es el **“context size”**, que se refiere a la **cantidad máxima de tokens** (fragmentos de texto) que el modelo puede procesar de una sola vez. Este límite determina:

- Cuánto contenido se puede incluir en una sola consulta o respuesta.
- Cuán bien puede el modelo mantener coherencia en conversaciones largas o cuando se trabaja con documentos extensos.

Un contexto bien construido y ajustado a los límites del modelo mejora significativamente la calidad de las respuestas y el razonamiento.

Modelo	Context Size
Gemini 1.5 Pro	2 millones de tokens
Gemini 2.0 Flash y Gemini 1.5 Flash	1 millón de tokens
GPT-4o	128,000 tokens
GPT-3.5	16,385 tokens

La importancia del contexto en los modelos de lenguaje es fundamental para que puedan generar respuestas coherentes, precisas y relevantes. Sus principales aportes son:



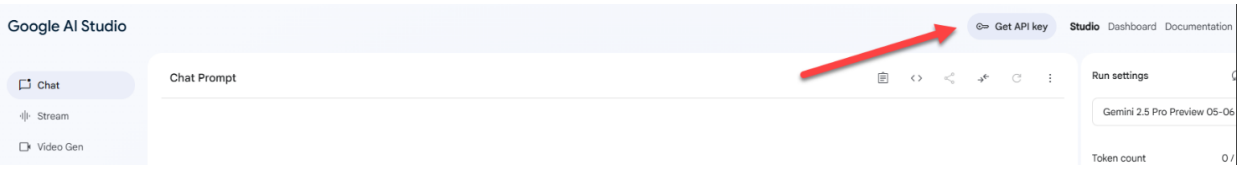
2.4.1. Ejercicio 02 – Contexto.ipynb

Para la realización del ejercicio es necesario acceder al archivo **Contexto.ipynb**, ubicado en la carpeta **1.Estructura Interna**.

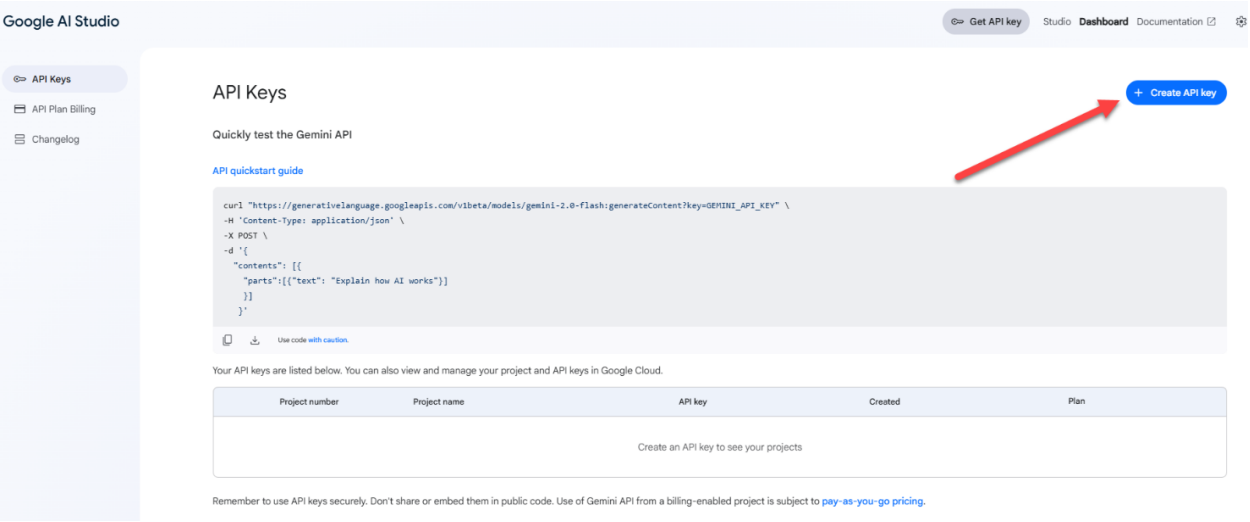
Antes de iniciar, es necesario obtener la **Gemini key** accediendo a la página: https://aistudio.google.com/.

Para obtener la Gemini key:

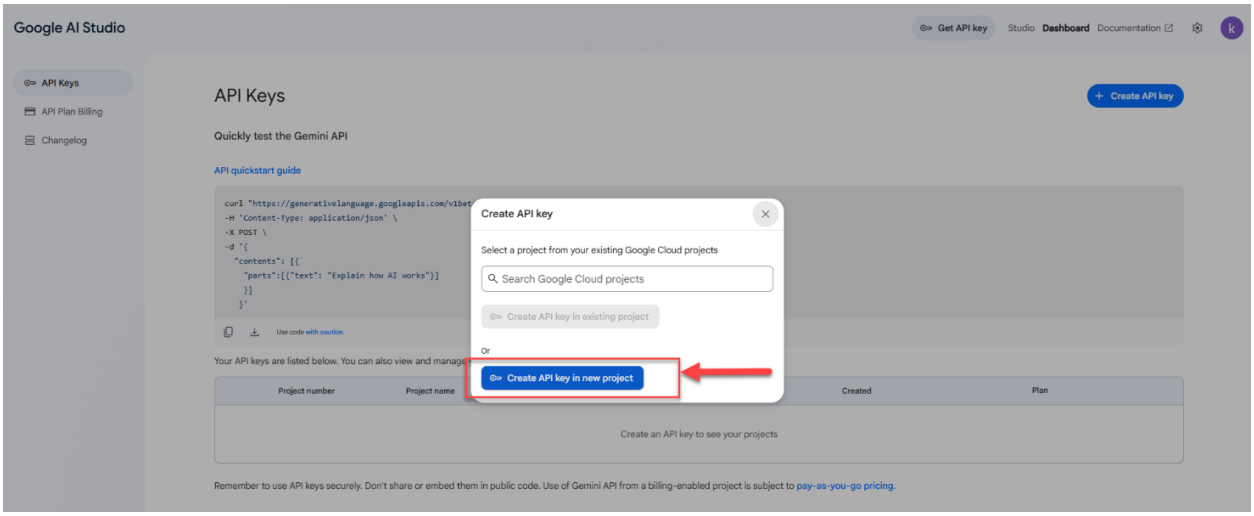
1. Dar clic en **"Get Api Key"**.



2. Dar clic en **"Create Api Key"**.



3. Click en **"Ceate API Key in New Project".**



Una vez creada la Gemini Key, se procederá a instalar las librerías requeridas para realizar el ejemplo para después acceder al dataset de productos que se generó en el taller de **"Web Scraping"** llamada **"productos.csv"** y se obtienen los productos del archivo. ***Para más detalles del código consultar el notebook de esta sección.***

Después pasamos a hacer un conteo de tokens que aproximadamente son de la lista de artículos para pasarlo a Gemini:

- Analiza si excede el número de tokens permitidos
- Cuantos tokens permite Gemini?

Para esto obtenemos el código:

```
encoding = tiktoken.encoding_for_model("gpt-4o")
tokenscount = encoding.encode(catalogo_texto)
print("Numero de tokens: ", len(tokenscount))
```

Ahora se creará el prompt para realizar el ejercicio donde se puede observar que se escribe dentro del contexto la información del dataset y a su vez realizando una pregunta acerca de la misma:

```
prompt = f"""
Este es un catálogo de productos:

{catalogo_texto}
```

```
Pregunta: ¿Qué productos tienen un precio menor a 800 MXN y son de la marca
Adidas?
"""
```

Para después contar de nuevo los tokens y enviar el prompt generado al modelo para recibir una respuesta de el:

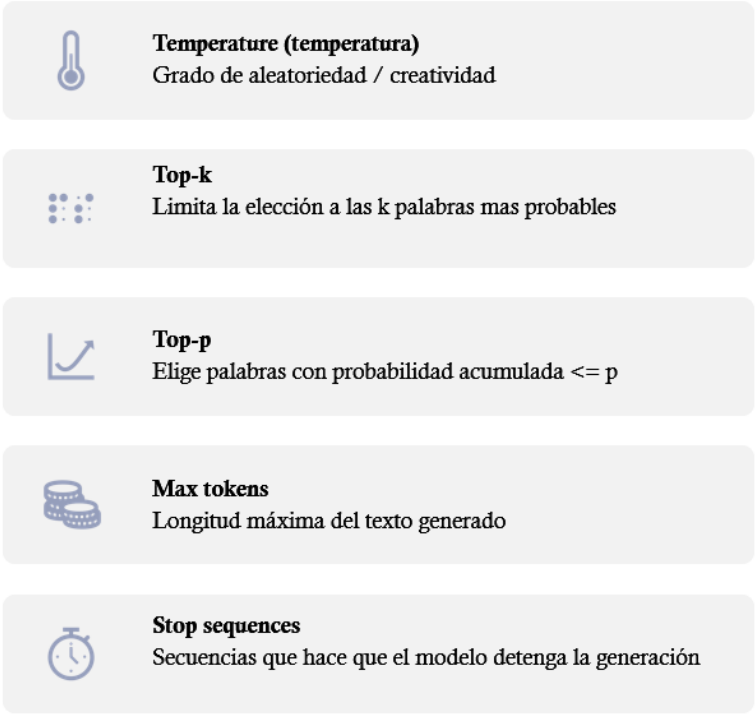
```
tokenscount = encoding.encode(prompt)
print("Numero de tokens: ", len(prompt))
```

```
response = model.generate_content(prompt)
print(response.text)
```

Para continuar con los ejercicios relacionados con este tema: ***Consultar el ejercicio práctico en el archivo Python: Contexto.ipynb***

2.5. Control del comportamiento del modelo: Parámetros ajustables

Los modelos de lenguaje permiten ajustar su comportamiento mediante ciertos **parámetros configurables**, que influyen en **cómo generan sus respuestas**. Estos parámetros no modifican el conocimiento del modelo, pero sí **afectan el estilo, la creatividad y la precisión** de sus salidas. Los más comunes son:



2.5.1. Ejercicio 03 – Parametros.ipynb

Para realizar el ejercicio es necesario acceder al archivo **Parametros.ipynb**, ubicado en la carpeta **1.Estructura Interna**.

Para más detalles del código consultar el notebook de esta sección.

Tomando en cuenta el tema actual, en este ejercicio se enviaran los distintos parámetros ajustados al modelo de Gemini y se genera una descripción de salida:

```
response = model.generate_content(
            prompt_template,
            generation_config=genai.types.GenerationConfig(
                temperature=1,
                top_p=0.5,
                max_output_tokens=200,
                stop_sequences=["\n"]
            )
        )
print("\n Descripción generada:")
print(response.text.strip())
```

Para continuar con los ejercicios relacionados con este tema: ***Consultar el ejercicio práctico en el archivo Python:Parametros.ipynb***

3. Razonamiento en LLMS

El razonamiento en los modelos de lenguaje (LLMs) constituye una habilidad esencial que les permite interpretar, analizar y generar respuestas coherentes a partir de la información recibida. Esta capacidad se manifiesta en la manera en que los modelos pueden conectar ideas, identificar patrones y resolver problemas, adaptándose a distintos contextos y niveles de complejidad, lo que los hace herramientas poderosas para una amplia variedad de aplicaciones.

3.1. ¿Qué es el razonamiento en LLMS?

El **razonamiento en LLMs (Large Language Models)** se refiere a la capacidad de estos modelos para **analizar, inferir y tomar decisiones lógicas** basadas en el texto que reciben. No se trata solo de repetir información aprendida, sino de **usar patrones y estructuras del lenguaje** para resolver problemas, completar tareas complejas o responder preguntas que requieren varios pasos de pensamiento.

El razonamiento puede manifestarse de forma **explícita** (cuando se detallan los pasos) o **implícita** (cuando el modelo llega a una conclusión sin

mostrar el proceso). Esta capacidad es clave para tareas avanzadas como programación, análisis de texto, resolución de casos hipotéticos, generación de argumentos, y más.

3.1.1. Lógica Implícita

La lógica implícita en los LLMs es la capacidad del modelo para seguir patrones y relaciones aprendidas del lenguaje sin aplicar reglas formales de lógica. Aunque no razonan como un humano, pueden inferir respuestas coherentes basándose en grandes cantidades de texto visto durante su entrenamiento.

Ejemplo:
Si el texto dice: ***"Pedro dejó su paraguas porque estaba lloviendo"***, el modelo puede deducir que *"estaba lloviendo"* explica por qué *"dejó su paraguas"*, aunque nunca se le haya enseñado explícitamente esa relación causa-efecto.

3.1.2. Lógica Explícita

La lógica explícita se refiere a la capacidad de razonar aplicando reglas formales y definidas, siguiendo principios claros sin ambigüedad.

Ejemplo:

Si tenemos la regla **"Si está lloviendo, entonces necesito un paraguas"** y la afirmación **"Está lloviendo"**, el modelo puede deducir explícitamente que **"necesito un paraguas"** aplicando esa regla lógica, aunque no haya visto ese caso específico antes, porque sigue un patrón formal de inferencia basado en la condición y la consecuencia.

3.2. Técnicas de prompting

Las técnicas de prompting son métodos y estrategias usadas para interactuar con modelos de lenguaje como los LLMs, diseñando cuidadosamente las entradas (prompts) que se les dan para obtener respuestas más precisas, relevantes o creativas. Básicamente, consiste en formular preguntas, instrucciones o ejemplos de manera específica para guiar al modelo hacia el tipo de salida deseada, aprovechando mejor su capacidad de comprensión y generación de texto. Estas técnicas pueden incluir desde prompts simples hasta cadenas de instrucciones complejas o contextos detallados que facilitan que el modelo entienda mejor la tarea y produzca resultados óptimos.

3.2.1. Zero-shot Prompting

Zero-shot prompting es una técnica en la que se le da al modelo una instrucción o pregunta sin proporcionarle ejemplos previos sobre cómo debe responder. El modelo debe comprender y resolver la tarea únicamente con base en la formulación del prompt y su conocimiento previo aprendido durante el entrenamiento. Esta técnica es útil cuando se busca evaluar la capacidad general del modelo para realizar una tarea sin entrenamiento específico o guía directa.

Prompt:

Clasifica este producto en una categoría:

“Sandalias deportivas Nike para mujer con suela de goma antideslizante y diseño ergonómico.”

Respuesta esperada:

Calzado Deportivo

3.2.2. Few-shot Prompting

Few-shot prompting es una técnica en la que se proporcionan al modelo uno o varios ejemplos antes de hacer la solicitud principal, con el fin de guiarlo sobre el formato, estilo o tipo de respuesta esperada. Estos ejemplos ayudan al modelo a entender mejor la tarea, sin necesidad de un entrenamiento adicional. Es especialmente útil en tareas complejas o con instrucciones ambiguas.

Prompt:

Ejemplo 1:

Producto: Camiseta adidas manga corta para hombre

Categoría: Ropa deportiva

Ejemplo 2:

Producto: Laptop HP con procesador i5 y 8GB RAM

Categoría: Electrónica

Ejemplo 3:

Producto: Zapatillas Nike Running Hombre

Categoría: Calzado deportivo

Entrada Producto:

Zapatillas Puma Running Mujer

Respuesta esperada:

Calzado Deportivo

3.2.3. Chain-of-Thought Prompting

Chain-of-thought prompting es una técnica que guía al modelo a generar no solo una respuesta final, sino también el razonamiento paso a paso que lleva a esa respuesta. Al incluir cadenas de pensamiento en el prompt o al pedir explícitamente que el modelo "piense en voz alta", se mejora su desempeño en tareas que requieren lógica, cálculo o múltiples etapas de análisis.

Prompt:

Tengo un presupuesto de $1000.

La camiseta cuesta $300 y los tenis cuestan $850.

¿Puedo comprar ambos? Explica paso a paso.

Respuesta esperada:

Paso 1: Sumo el precio de la camiseta y los tenis → $300 + $850 = $1150

Paso 2: Comparo el total con el presupuesto → $1150 > $1000

Conclusión: No puedo comprar ambos productos con ese presupuesto.

3.2.4. Meta Prompting

Meta prompting es una técnica avanzada en la que el modelo no solo responde a una tarea, sino que se le pide reflexionar sobre cómo debería abordar esa tarea o incluso generar prompts por sí mismo. Es decir, se le da una instrucción sobre cómo formular o mejorar otras instrucciones, convirtiéndolo en un generador o evaluador de prompts. Esta técnica es útil para afinar el rendimiento del modelo, mejorar la calidad de las respuestas o diseñar estrategias más efectivas de interacción.

Prompt:

Escribe un prompt para un modelo de lenguaje que le pida clasificar productos de un catálogo de e-commerce en categorías como “Ropa”, “Calzado”, “Electrónica” o “Otro”.

Respuesta esperada:

Clasifica los siguientes productos en una de las categorías: Ropa, Calzado, Electrónica u Otro. Producto:

3.2.5. Ejercicio 04 – Razonamiento.ipynb

Para realizar el ejercicio es necesario acceder al archivo **Razonamiento.ipynb**, ubicado en la carpeta **2.Razonamiento Interno**.

En este ejercicio se ejecutarán algunos ejemplos de las técnicas de prompt vistas en los puntos anteriores. ***Para más detalles del código consultar el notebook de esta sección.***

Antes de realizar los ejemplos es necesario cargar el modelo que se estará utilizando, en este caso sería **gemini-1.5-flash**.

Ejemplo Zero-shot Prompting

```
prompt = "Clasifica este producto: 'Zapatillas Nike Air Max para correr' en una categoría."

response = model.generate_content(prompt)
print("Zero-shot:", response.text.strip())
```

Ejemplo Few-shot Prompting

```
prompt = """
Ejemplo 1:
Producto: Camiseta adidas para hombre
Categoría: Ropa deportiva

Ejemplo 2:
Producto: Laptop Dell Inspiron 15
Categoría: Electrónica

Ahora tú:
Producto: Zapatillas Puma Nitro para correr
```

```
Categoría:
"""

response = model.generate_content(prompt)
print("Few-shot:", response.text.strip())
```

Ejemplo Chain-of-Thought Prompting

```
prompt = """
Tengo un presupuesto de $1000.
El producto A cuesta $350 y el producto B cuesta $750.
¿Puedo comprar ambos productos? Explica paso a paso.
"""

response = model.generate_content(prompt)
print("Chain-of-Thought:", response.text.strip())
```

Ejemplo Meta Prompting

```
prompt = "Crea un prompt para un modelo de lenguaje que clasifique productos de
una tienda en línea según su categoría."

response = model.generate_content(prompt)
print("Meta Prompting:", response.text.strip())
```

4. Principios de diseño de prompts

Los **principios de diseño de prompts** son pautas que ayudan a formular mejores instrucciones para interactuar con los modelos de lenguaje, facilitando una comunicación más efectiva y precisa. Estas estrategias permiten organizar y estructurar las indicaciones de forma que el modelo pueda comprender mejor lo que se espera, mejorando así la calidad de las respuestas generadas. A lo largo de esta sección se explorarán algunas de las técnicas más útiles para optimizar la interacción con estos sistemas.

4.1. Definición de principios del diseño de prompts

Los **principios de diseño de prompts** son estrategias prácticas que ayudan a formular mejores instrucciones para interactuar eficazmente con modelos de lenguaje como los LLMs. Su objetivo es guiar al modelo de manera clara, precisa

y estructurada para obtener respuestas útiles, coherentes y alineadas con la intención del usuario. A continuación se explican algunos de los principios más comunes.

4.1.1. Delimitadores

Los **delimitadores** son símbolos o marcas utilizados para separar claramente las instrucciones del contenido en un prompt. Ayudan al modelo a distinguir qué debe procesar como texto de entrada y qué como instrucción, reduciendo ambigüedades. Se pueden usar comillas, paréntesis, líneas, o bloques de código como ``` para marcar fragmentos específicos.

Ejemplo:

Describe este producto: **“Sandalias Skechers Uno Summer Stand2 Mujer”**

4.1.2. Instrucciones detalladas

Las **instrucciones detalladas** consisten en formular comandos claros, específicos y completos que indiquen exactamente qué se espera del modelo. Esto incluye el tipo de tarea, el formato deseado de la respuesta, el estilo, el idioma, e incluso el rol que debe asumir el modelo. Cuanta más precisión tenga la instrucción, mayor será la calidad y relevancia de la salida generada.

Ejemplo:

Genera una descripción en tono juvenil, de 2-3 párrafos, que destaque estilo y comodidad.

4.1.3. Estructura RTF (Role-Task-Format)

La **estructura RTF (Role-Task-Format)** es un principio de diseño de prompts que organiza la instrucción en tres componentes clave: el **rol** que debe asumir el modelo, la **tarea** que se le pide realizar y el **formato** en que debe entregar la respuesta. Esta estructura ayuda a establecer un contexto claro, una acción precisa y una expectativa concreta sobre cómo debe presentarse la salida.

Ejemplo:

Eres un experto en marketing. Clasifica el siguiente producto por temporada y cliente. Devuelve el resultado en JSON: **`Sandalias Onboard Dore Mujer`.**

4.1.4. Pasos secuenciales

El principio de **pasos secuenciales** consiste en dividir una tarea compleja en instrucciones paso a paso, lo que facilita que el modelo procese y responda con mayor precisión. Guiar al modelo mediante una secuencia ordenada de acciones reduce errores y mejora el razonamiento, especialmente en tareas que requieren lógica, múltiples etapas o procesos detallados.

Ejemplo:

Paso 1: Lee el nombre del producto. Paso 2: Detecta si es calzado o accesorio. Paso 3: Clasifica precio como bajo/medio/alto. Producto: Sandalias Skechers Uno Summer Stand2 Mujer

4.1.5. Tiempo para “pensar”

Dar **"tiempo para pensar"** es una técnica que consiste en invitar al modelo a razonar antes de dar una respuesta final. Esto se logra incluyendo frases como *“piensa paso a paso”* o *“razona antes de responder”*, lo que fomenta un proceso más reflexivo y estructurado. Esta estrategia es especialmente útil en problemas matemáticos, preguntas lógicas o situaciones que requieren inferencias complejas.

Ejemplo:

Piénsalo paso a paso. ¿Para qué temporada es más adecuada la Sandalia Onboard Dore Mujer?

4.1.6. Evitar instrucciones vagas

Es fundamental que las instrucciones sean claras y específicas, evitando términos o indicaciones ambiguas que puedan generar confusión o respuestas imprecisas. Las instrucciones vagas dificultan que el modelo entienda exactamente qué se espera, lo que puede derivar en salidas poco útiles o fuera de contexto. Evitar palabras como 'breve' o 'claro'; usar criterios medibles (e.g. número de párrafos).

Ejemplo:

✗ 'Haz una descripción clara.’

✓ 'Haz una descripción de 100 a 150 palabras.

4.1.7. Usar lenguaje positivo

Usar lenguaje positivo en los prompts implica formular las instrucciones de manera clara y constructiva, enfocándose en lo que se desea que el modelo haga en lugar de lo que no debe hacer. Este enfoque facilita que el modelo entienda mejor las expectativas y genera respuestas más efectivas y alineadas con la intención del usuario.

Ejemplo:

✗ 'No menciones marcas.'

✓ 'Describe el producto sin referirte a marcas específicas.

4.1.8. Dar pistas de formato

Dar pistas de formato consiste en indicar explícitamente cómo debe estructurarse la respuesta del modelo, especificando elementos como listas, párrafos, tablas, viñetas, o longitud del texto. Esto ayuda a que la salida sea más clara, organizada y útil según el contexto o la aplicación deseada.

Ejemplo:

Incluye la palabra 'import' para indicar que el código debe ser en Python, o 'SELECT' para que sea SQL.

4.1.9. Ejercicio 05 – Prompting.ipynb

Para realizar el ejercicio es necesario acceder al archivo **Prompting.ipynb**, ubicado en la carpeta **3.Principios de diseño de prompts**.

En este ejercicio se ejecutarán algunos ejemplos de prompts utiliazando diferentes formatos:

- **Prompt 1:** Etiqueta de marketing breve
- **Prompt 2:** Generación de SQL
- **Prompt 3:** Análisis de inconsistencias
- **Prompt 4:** Descripción informal para e-commerce
- **Prompt 5:** Clasificación de cliente y temporada

Para más detalles del código consultar el notebook de esta sección.

5. Formato de conversación en modelos de lenguaje

El **formato de conversación** se ha convertido en una estructura fundamental para interactuar con los modelos de lenguaje, ya que permite organizar el diálogo de manera clara y ordenada, simulando una charla natural entre los participantes. Esta modalidad no solo facilita que el modelo mantenga coherencia y contexto a lo largo de múltiples intercambios, sino que también mejora la calidad y efectividad de la comunicación, haciendo que la interacción con la inteligencia artificial sea más intuitiva y funcional para diversas aplicaciones.

5.1. ¿Qué es el formato de conversación?

El **formato de conversación** es una manera de estructurar la interacción con un modelo de lenguaje en forma de diálogo, simulando una charla natural entre dos o más participantes. Este formato organiza los intercambios en turnos de preguntas y respuestas, lo que facilita que el modelo mantenga el contexto y genere respuestas coherentes y relevantes a lo largo de la conversación. Es especialmente útil para tareas donde la interacción continua y el seguimiento del hilo son importantes.

Importancia:

- Mejora la calidad de la interacción gracias a la comprensión del contexto previo.
- Facilita tareas complejas que requieren múltiples intercambios.
- Hace que la IA sea más accesible y fácil de usar.

5.2. Componentes clave

Los **componentes clave** en el formato de conversación son las distintas partes que conforman el intercambio entre el usuario y el modelo.

- **Mensaje del Sistema**
  - Define **las reglas, el tono, el conocimiento y las limitaciones** del modelo.
  - Guía internamente el comportamiento del modelo.
- **Rol del Usuario**
  - Representa lo que escribe la persona que interactúa con el modelo
  - Puede hacer preguntas, dar instrucciones o mantener un diálogo
- **Rol del Asistente (modelo)**

- Es el que genera las respuestas a partir del sistema y los mensajes del usuario

5.3. Mensaje del sistema (System message)

El **Mensaje del sistema (System message)** es una instrucción inicial que establece el contexto y las directrices bajo las cuales el modelo debe operar durante la conversación. Define aspectos como el tono, el estilo, el conocimiento que debe usar y las limitaciones a respetar. Este mensaje actúa como una guía interna para el modelo, orientando su comportamiento y asegurando que las respuestas sean coherentes con el propósito y las expectativas del usuario.

Propósito:

- Guiar el comportamiento general del modelo.
- Definir restricciones o tareas específicas.
- Establecer un "personaje" o rol para el modelo.

Impacto: Un mensaje de sistema bien definido puede mejorar drásticamente la relevancia y la calidad de las respuestas del modelo.

Ejemplos: {

"role": "system",

"content": "**Eres un asistente de atención al cliente especializado en calzado deportivo. Respondes de forma clara, empática y rápida, ayudando con dudas sobre tallas, modelos, cambios y devoluciones.**"

}

{

"role": "system",

"content": "**Eres un experto en calzado deportivo. Ayudas a los vendedores a entender las diferencias técnicas entre modelos, materiales, tipos de pisada y usos recomendados.**"

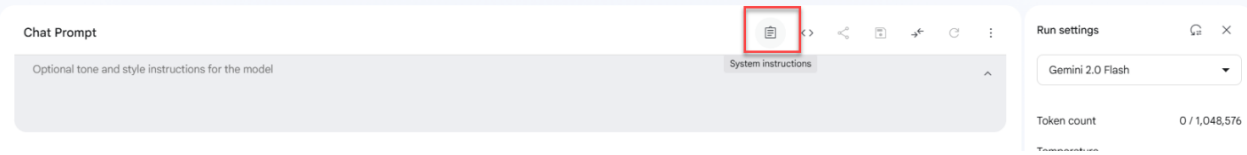
}

5.4. Ejercicio 06 – Conversacion.ipynb

Para realizar el ejercicio es necesario acceder al archivo **Prompting.ipynb**, ubicado en la carpeta **4.Formato Conversacion**.

Se requiere ir a la página: https://aistudio.google.com/ en donde se seguirán las siguientes instrucciones:

1. Seleccionar Modelo Gemini 2.0 Flash
2. Click en Systems instructions
3. Se debe pegar el texto en system instructions: **"Eres un asistente de atención al cliente especializado en calzado deportivo. Respondes de forma clara, empática y rápida, ayudando con dudas sobre tallas, modelos, cambios y devoluciones."**



4. Pegar en Start typing a prompt: **"Hola, me llegaron unas zapatillas que pedí online, pero no me quedaron. ¿Puedo cambiarlas?"**
5. Ejecuta el prompt y analiza el resultado

Y después seguir las instrucciones dentro del código para realizar los ejercicios. ***Para más detalles del código consultar el notebook de esta sección.***

6. Razonamiento con Herramientas Externas

Los modelos de lenguaje, por sí solos, tienen un conocimiento limitado al momento de responder preguntas o resolver problemas. Para superar esas limitaciones, se introduce el concepto de **Tool Calling**: la capacidad de un modelo de lenguaje para **invocar herramientas externas** (como funciones, APIs, bases de datos o sistemas empresariales) y así resolver tareas complejas o que requieren información actualizada.

Este enfoque es fundamental para integrar los modelos en entornos reales de trabajo, donde necesitan **interactuar con datos dinámicos o ejecutar acciones específicas**, como consultar inventarios, aplicar políticas empresariales o automatizar procesos administrativos.

6.1 Tool Calling

Es la capacidad de un modelo de lenguaje para **apoyarse en herramientas externas** (como calculadoras, buscadores web, bases de datos o APIs) para **resolver tareas que exceden su conocimiento interno** o requieren pasos complejos.

¿Por qué es importante?
Aunque los modelos de lenguaje son potentes, no lo saben todo (por ejemplo, no pueden hacer cálculos complejos o acceder a información actualizada por sí mismos). Al usar herramientas externas, **extienden sus capacidades**.

Ejemplos:

- Detectar si un documento es una factura y conectarse a SAP para capturarla automáticamente
- Un asistente puede acceder a documentos de políticas de devolución para seguir el procedimiento

6.1.1 Ejercicio 07 – Herramientas.ipynb

Para realizar el ejercicio es necesario acceder al archivo **Herramientas.ipynb**, ubicado en la carpeta **5.Herramientas**.

Este ejercicio tiene como objetivo introducir el uso de herramientas (tool calling) con el modelo Gemini para realizar tareas específicas simuladas de negocio.

En este se puede encontrar funciones para:

- Revisar inventario de sucursales
- Revisar promociones
- Revisar política de devoluciones
- Asociar nombres a funciones
- Crear modelo con herramientas habilitadas

Para más detalles del código consultar el notebook de esta sección.

6.2 Agentes

Un **agente con LLM (Large Language Model)** es un sistema autónomo basado en inteligencia artificial que utiliza un modelo de lenguaje para **comprender instrucciones en lenguaje natural**, **razonar sobre ellas** y **ejecutar acciones específicas** mediante funciones, APIs o herramientas externas.

Este tipo de agente no se limita a generar texto, sino que opera como un componente activo dentro de un entorno digital, capaz de tomar decisiones, interactuar con sistemas y resolver tareas complejas de forma flexible.

Características clave:

- **Comprensión contextual**: interpreta la intención del usuario y extrae los datos relevantes de un enunciado.
- **Capacidad de razonamiento**: analiza condiciones, aplica reglas y determina el mejor curso de acción.
- **Acción directa**: invoca herramientas programadas, como consultas a bases de datos, envío de notificaciones, cálculos o generación de documentos.

6.2.1 Ejercicio 08 – Agente.ipynb

Para realizar el ejercicio es necesario acceder al archivo **Agente.ipynb**, ubicado en la carpeta **5.Herramientas**.

Este ejercicio ejemplifica cómo un agente LLM puede ser **extendido con funciones personalizadas**, convirtiéndose en un asistente corporativo eficaz, flexible y capaz de tomar decisiones basadas en reglas del negocio.

Dentro de los ejercicios que se incluyen están:

- Validación y aprobación de descuentos con lógica de negocio
- Comunicación con equipos internos según el tipo de evento
- Programación de reabastecimientos automáticos
- Generación de borradores de comunicados empresariales

Para más detalles del código consultar el notebook de esta sección.

7. Gemini

En un mundo donde la interacción con la tecnología se vuelve cada vez más natural y multidimensional, Gemini surge como una innovación que integra distintas formas de comunicación (desde texto hasta audio, imágenes y video) para ofrecer experiencias más completas y efectivas. Esta evolución impulsa nuevas maneras de resolver problemas, generar contenido y automatizar tareas, transformando la forma en que nos relacionamos con la información.

7.1. ¿Qué es Gemini?

Gemini es una familia avanzada de modelos de lenguaje desarrollada por Google DeepMind que combina técnicas de inteligencia artificial y aprendizaje profundo para entender y generar texto de manera natural y precisa. Estos modelos están diseñados para ofrecer capacidades mejoradas en comprensión, razonamiento y generación de lenguaje, permitiendo aplicaciones más sofisticadas en áreas como asistencia virtual, generación de contenido, análisis de texto y más. Gemini representa un avance significativo en la evolución de los grandes modelos de lenguaje, integrando innovaciones que buscan una interacción más inteligente y contextualizada con los usuarios.

- Texto (como preguntas o artículos)
- Audio (como voz o sonidos)
- Imágenes (fotos, gráficos)
- Video (escenas en movimiento)
- Código (lenguajes de programación)

7.2. Razonamiento avanzado

El razonamiento avanzado en Gemini se refiere a su capacidad para abordar tareas que van más allá de la simple generación de texto, involucrando habilidades como la lógica, la planificación y el pensamiento abstracto. Este modelo está diseñado para resolver problemas matemáticos complejos, analizar código de programación y comprender documentos extensos con contextos complejos, lo que le permite ofrecer respuestas más precisas y útiles en escenarios que demandan un entendimiento profundo y multifacético.

7.3. Ejercicio 09 - Gemini.ipynb

Para realizar el ejercicio es necesario acceder al archivo **Gemini.ipynb**, ubicado en la carpeta **6.Gemini**.

Es necesario acceder al siguiente link: https://aistudio.google.com/ para poder realizar los ejercicios y seguir las instrucciones que se mencionan en el ejemplo.

Para más detalles del código consultar el notebook de esta sección.

8. Modelos Locales, Hugging Face y Bert

Modelos de Lenguaje y BERT

Desde la publicación del artículo **Attention is All You Need**, se sentaron las bases para una nueva generación de modelos de lenguaje basados en la arquitectura **Transformer**. A diferencia de los modelos generativos actuales como GPT, estos primeros modelos fueron diseñados para tareas de procesamiento de lenguaje natural (NLP) más específicas y no necesariamente para generar texto.

Uno de los modelos más influyentes en esta etapa fue **BERT (Bidirectional Encoder Representations from Transformers),** desarrollado por Google. **BERT** es un modelo no generativo que se ha convertido en la base de múltiples aplicaciones de NLP debido a su capacidad para entender el contexto de manera bidireccional. A pesar de haber sido lanzado en 2018, sus variantes siguen siendo ampliamente utilizadas por su eficiencia, precisión y facilidad de implementación en entornos locales (on-premise).

Los modelos de lenguaje pueden aplicarse a una gran variedad de tareas.
Algunas de las más comunes son:

1. **Reconocimiento de Entidades (NER)**
   Permite identificar y clasificar entidades dentro de un texto, tales como nombres de personas, organizaciones, ubicaciones, fechas, entre otros.
2. **Análisis de Sentimiento**
   Evalúa el tono emocional de un texto (positivo, negativo o neutral). Es ampliamente usado en aplicaciones como monitoreo de redes sociales o análisis de reseñas.
3. **Preguntas y Respuestas**
   Modelos diseñados para responder preguntas con base en un contexto dado.
4. **Traducción de Idiomas**
   Convierte texto de un idioma a otro usando modelos entrenados para mantener el significado semántico.
5. **Resumen Automático**
   Extrae la idea principal de un texto largo, generando un resumen coherente y compacto.
6. **Generación de Embeddings**
   Transforma palabras, frases o documentos en vectores numéricos que representan su significado semántico, útiles para tareas de búsqueda, similitud y clustering.

Hugging Face

Hugging Face se ha consolidado como una plataforma fundamental para el trabajo práctico con modelos de lenguaje, tanto generativos como no generativos. Ofrece herramientas, APIs y modelos preentrenados que facilitan el desarrollo de soluciones NLP, incluso en entornos locales.

Para utilizar modelos desde **Hugging Face**, es necesario:

1. Crear una cuenta.
2. Generar un token personal de acceso.
3. Usar este token en nuestro entorno (por ejemplo, en notebooks o scripts de Python) para autenticar las descargas de modelos.

Los modelos de **Hugging Face** nos permiten abordar distintas tareas de procesamiento de lenguaje natural. A continuación, veremos algunos de los usos más comunes:

- **Generación de Texto**
  Es la tarea con la que más se ha popularizado el uso de modelos de lenguaje. Implica generar texto nuevo a partir de una entrada inicial. Modelos como GPT y similares se destacan en esta tarea. Sin embargo, debido a su tamaño, son difíciles de ejecutar localmente sin hardware especializado.
- **Reconocimiento de Entidades (NER)**
  Hugging Face proporciona modelos como dbmdz/bert-large-cased-finetuned-conll03-english que permiten realizar NER de manera sencilla con el pipeline de transformers.
- **Análisis de Sentimiento**
  Tarea común que puede resolverse fácilmente con modelos preentrenados como distilbert-base-uncased-finetuned-sst-2-english. Ideal para clasificación binaria o multiclase de emociones.
- **Máscara de Lenguaje (Masked Language Modeling)**
  El modelado de lenguaje enmascarado (MLM) consiste en ocultar palabras del texto y predecirlas. Modelos como BERT fueron entrenados con esta tarea utilizando el token especial [MASK].
  Este tipo de modelos no son autorregresivos (a diferencia de GPT), lo que significa que predicen palabras faltantes con base en el contexto completo, y no solo en lo anterior.

Salida y Top-K

Cuando se realiza una tarea de predicción de máscara, el modelo devuelve las **probabilidades de todos los tokens** del vocabulario para esa posición en particular. Es nuestra responsabilidad aplicar un criterio de selección adecuado para elegir el mejor candidato.

Técnicas comunes de selección incluyen:

- **Top-K Sampling:** seleccionar entre los K tokens más probables.
- **Top-P (Nucleus Sampling):** seleccionar entre los tokens cuya suma de probabilidad alcanza un umbral P.
- **Temperatura:** ajusta la distribución de probabilidad para hacerla más o menos “creativa”.

Este proceso es similar al que usan los modelos generativos como Gemini, pero con la diferencia de que los modelos como BERT **pueden ver todo el contexto** de la oración, mientras que modelos generativos solo ven lo que está antes del token actual.

Bibliografía y Recursos Adicionales

1. **Ejemplo interactivo de Transformers:** https://poloclub.github.io/transformer-explainer/
2. **Google AI Studio:** https://aistudio.google.com/
3. **Online Tokenizer:** https://platform.openai.com/tokenizer
4. **Tesis Attention is all you need:** https://arxiv.org/abs/1706.03762
5. **BERT:** https://huggingface.co/blog/bert-101
6. **Documentación del taller:** http://dptaller.certexai.com/