

AI Engineering

Clase 5 - Chain of Thought



Chain of Thought y One-Shot vs Many-Shots en Generative AI

Duración: 45 minutos

Contexto:

En esta clase teórica, exploraremos en profundidad dos conceptos avanzados de la ingeniería de prompts para Inteligencia Artificial Generativa (GenAI): Chain of Thought (CoT) y One-Shot vs Many-Shots. Estos enfoques son esenciales para mejorar la calidad, coherencia y precisión de las respuestas generadas por modelos de lenguaje como GPT-4. Además, integraremos estos conceptos con las clases de Python previamente desarrolladas (**ModeloGenerativo** y **ModeloGPT**) para ilustrar cómo aplicar estas técnicas en proyectos prácticos de GenAI.

1. Chain of Thought (CoT)

Definición y Concepto

Chain of Thought (CoT) es una técnica que guía al modelo de IA a generar una secuencia lógica de pensamientos o pasos intermedios antes de proporcionar una respuesta final. Este enfoque mejora la capacidad del modelo para resolver problemas complejos que requieren múltiples etapas de razonamiento.

Importancia de CoT

- Mejora el Razonamiento: Facilita que el modelo desglose tareas complejas en pasos más manejables.
- Aumenta la Coherencia: Las respuestas generadas son más lógicas y estructuradas.
- Reduce Errores: Minimiza la probabilidad de respuestas incorrectas al seguir un proceso paso a paso.

Ejemplo de Chain of Thought (básico y no tan 'real')

Contexto: Eres un asistente de IA que ayuda a resolver problemas matemáticos paso a paso.

Tarea: Calcula la suma de los primeros 10 números naturales.

Chain of Thought:

Identificar los primeros 10 números naturales: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10.

Aplicar la fórmula de la suma de una serie aritmética: $S = n(n + 1)/2$. Sustituir $n = 10$: $S = 10(10 + 1)/2 = 55$.

Concluir que la suma es 55.

Respuesta: La suma de los primeros 10 números naturales es 55.

Ejemplo de Chain of Thought (un poco más real)

Contexto: Eres un asistente de IA que explica procesos de edición de imágenes paso a paso.

Tarea: Convertir una fotografía de retrato en una caricatura digital.

Chain of Thought:

1. **Analizar la imagen original:**
 - Identificar los rasgos más característicos del rostro (ojos, nariz, boca)
 - Observar las proporciones naturales
 - Notar elementos distintivos como lunares, cicatrices o accesorios
2. **Simplificar formas:**
 - Reducir los detalles complejos a formas básicas
 - Mantener solo las líneas esenciales
 - Eliminar texturas y detalles innecesarios
3. **Exagerar rasgos característicos:**
 - Aumentar el tamaño de los elementos más distintivos (si tiene ojos grandes, hacerlos aún más grandes)
 - Reducir elementos menos prominentes
 - Mantener el balance para que siga siendo reconocible
4. **Aplicar estilo de caricatura:**
 - Definir un estilo de línea (grueso, fino, variable)
 - Elegir una paleta de colores simplificada
 - Añadir sombras básicas para dar profundidad
5. **Refinar detalles:**
 - Ajustar expresiones para enfatizar personalidad
 - Añadir elementos estilísticos (brillos, texturas simples)
 - Verificar que mantiene el parecido con el original

Respuesta: La fotografía se ha convertido exitosamente en una caricatura manteniendo el parecido con el original pero exagerando sus características más distintivas y simplificando los detalles menos importantes.



Implementación en Prompts con **ModeloGenerativo**

Para implementar CoT, estructuramos el prompt de manera que el modelo genere los pasos intermedios antes de la respuesta final. Utilizaremos la clase **ModeloGenerativo** previamente definida para facilitar esta implementación.

```

class ModeloGenerativo:
    def __init__(self, nombre, version):
        self._nombre = nombre
        self._version = version

    @property
    def nombre(self):
        return self._nombre

    @nombre.setter
    def nombre(self, valor):
        self._nombre = valor

    @property
    def version(self):
        return self._version

    @version.setter
    def version(self, valor):
        self._version = valor


    def cargar_modelo(self):
        # Código para cargar el modelo
        pass

    def generar_texto(self, prompt):
        # Código para generar texto basado en el prompt
        pass

    def limpiar_prompt(self, prompt):
        prompt = prompt.strip()
        prompt = prompt.capitalize()
        return prompt

    def formatear_prompt(self, plantilla, variables):
        try:
            prompt_formateado = plantilla.format(**variables)
            return prompt_formateado
        except KeyError as e:
            print(f"Error: Falta la variable {e} en el diccionario de variables.")
            return plantilla

```



Extendemos la clase **ModeloGenerativo** para incluir métodos que permitan la incorporación de Chain of Thought en los prompts.


```

import requests
from database import BaseDatos

class ModeloGPT(ModeloGenerativo):
    def __init__(self, nombre, version, api_key, db: BaseDatos):
        super().__init__(nombre, version)
        self.api_key = api_key
        self.url = "https://api.openai.com/v1/completions"
        self.headers = {
            "Authorization": f"Bearer {self.api_key}",
            "Content-Type": "application/json"
        }
        self.db = db

```

```

def generar_texto_con_cot(self, prompt, cot_steps):
    prompt_cot = f"""
    {prompt}

    Chain of Thought:
    {cot_steps}

    Respuesta:
    """
    prompt_limpio = self.limpiar_prompt(prompt_cot)
    payload = {
        "model": "text-davinci-003",
        "prompt": prompt_limpio,
        "max_tokens": 150,
        "temperature": 0.7
    }
    response = requests.post(self.url, headers=self.headers,
                             json=payload)
    if response.status_code == 200:
        texto_generado = response.json()["choices"][0]["text"].strip()
        self.db.guardar_interaccion(prompt_limpio, texto_generado)
        return texto_generado
    else:
        print(f"Error: {response.status_code} - {response.text}")
        return None

```



Beneficios y Limitaciones

Beneficios:

- Mejora la precisión en tareas complejas.
- Facilita el seguimiento del razonamiento del modelo.
- Permite detectar y corregir errores en etapas intermedias.

Limitaciones:


- Incrementa el número de tokens utilizados.
- Puede ralentizar la generación de respuestas.
- Requiere una estructuración cuidadosa del prompt.

2. One-Shot vs Many-Shots

Definición y Concepto

One-Shot Prompts: Proporcionan un único ejemplo antes de la solicitud principal. Ayudan al modelo a entender el formato y la expectativa de la respuesta con un solo caso de referencia.

Many-Shots Prompts: Incluyen múltiples ejemplos antes de la solicitud principal. Ofrecen al modelo una variedad de casos que guían de manera más completa la generación de respuestas.



```
class ModeloGenerativo:
    # ... (código anterior)

    def crear_one_shot_prompt(self, tarea, ejemplo):
        return f"Ejemplo:\nTarea: {ejemplo['tarea']}\nRespuesta: {ejemplo['respuesta']}\n\nTarea: {tarea}\nRespuesta:"

    def crear_many_shots_prompt(self, tarea, ejemplos):
        prompt = ""
        for i, ejemplo in enumerate(ejemplos, 1):
            prompt += f"Ejemplo {i}:\nTarea: {ejemplo['tarea']}\nRespuesta: {ejemplo['respuesta']}\n\n"
        prompt += f"Tarea: {tarea}\nRespuesta:"
        return prompt
```

Podemos combinar CoT con Many-Shots para proporcionar al modelo no solo ejemplos variados sino también una estructura lógica de razonamiento.

```
class ModeloGPT(ModeloGenerativo):  
    # ... (código anterior)  
  
    def generar_texto_con_cot_many_shots(self, tarea, ejemplos, cot_steps_list):  
        prompt = "Contexto: Eres un asistente de IA que ayuda a resolver problemas complejos paso  
a paso.\n\n"  
        for i, (ejemplo, cot_steps) in enumerate(zip(ejemplos, cot_steps_list), 1):  
            prompt += f"Ejemplo {i}:\n"  
            prompt += f"Tarea: {ejemplo['tarea']}\n"  
            prompt += f"Chain of Thought:\n{cot_steps}\nRespuesta: {ejemplo['respuesta']}\n\n"  
        prompt += f"Tarea: {tarea}\nChain of Thought:\n"  
        return self.generar_texto(prompt)
```

```

if __name__ == "__main__":
    api_key = "tu_api_key_aquí" # Reemplaza con tu clave API de OpenAI
    db = BaseDatos()
    modelo_gpt = ModeloGPT("GPT-4", "v1.0", api_key, db)

    tarea = "Determina el tiempo de encuentro de dos vehículos que viajan en direcciones opuestas."
    ejemplos = [
        {
            "tarea": "Determina el tiempo de encuentro de dos vehículos que viajan en direcciones opuestas.",
            "cot_steps": "1. Identificar las velocidades de ambos vehículos.\n2. Calcular la distancia total entre los vehículos.\n3. Establecer la ecuación de movimiento: tiempo = distancia / (velocidad1 + velocidad2).\n4. Resolver la ecuación para encontrar el tiempo de encuentro.",
            "respuesta": "El tiempo de encuentro es X horas."
        },
        {
            "tarea": "Calcula el tiempo necesario para que dos trenes se encuentren si uno viaja a 80 km/h y el otro a 60 km/h, estando separados por 300 km.",
            "cot_steps": "1. Sumar las velocidades de ambos trenes: 80 + 60 = 140 km/h.\n2. Calcular el tiempo: 300 / 140 ≈ 2.14 horas.\n3. Concluir que los trenes se encontrarán en aproximadamente 2.14 horas.",
            "respuesta": "Los trenes se encontrarán en aproximadamente 2.14 horas."
        }
    ]
    cot_steps_list = [
        "1. Identificar las velocidades de ambos vehículos.\n2. Calcular la distancia total entre los vehículos.\n3. Establecer la ecuación de movimiento: tiempo = distancia / (velocidad1 + velocidad2).\n4. Resolver la ecuación para encontrar el tiempo de encuentro.",
        "1. Sumar las velocidades de ambos trenes: 80 + 60 = 140 km/h.\n2. Calcular el tiempo: 300 / 140 ≈ 2.14 horas.\n3. Concluir que los trenes se encontrarán en aproximadamente 2.14 horas."
    ]

    texto_generado = modelo_gpt.generar_texto_con_cot_many_shots(tarea, ejemplos, cot_steps_list)
    print(texto_generado)

```

One-Shot Prompts:

Ventajas:

- Menor consumo de tokens.
- Más rápido de diseñar.

Desventajas:

- Menos robusto ante variaciones en la tarea.
- Mayor riesgo de respuestas inconsistentes.

Many-Shots Prompts:

Ventajas:

- Mejora la coherencia y precisión.
- Mayor flexibilidad para manejar variaciones.

Desventajas:

- Mayor consumo de tokens.
- Requiere más tiempo para diseñar y seleccionar ejemplos adecuados.

3. Técnicas Avanzadas de Prompting

Combinar Chain of Thought con Many-Shots

Integrar CoT con Many-Shots permite que el modelo no solo siga una secuencia lógica de pensamiento, sino que también se beneficie de múltiples ejemplos para mejorar la calidad de las respuestas.


```
contexto = "Contexto: Eres un asistente de IA que ayuda a resolver problemas complejos paso a paso."  
tarea = "Determina el tiempo de encuentro de dos vehículos que viajan en direcciones opuestas."
```

```
prompt_combinado = f"""  
{contexto}
```

Ejemplo 1:

Tarea: Determina el tiempo de encuentro de dos vehículos que viajan en direcciones opuestas.

Chain of Thought:

1. Identificar las velocidades de ambos vehículos.
2. Calcular la distancia total entre los vehículos.
3. Establecer la ecuación de movimiento: $\text{tiempo} = \text{distancia} / (\text{velocidad1} + \text{velocidad2})$.
4. Resolver la ecuación para encontrar el tiempo de encuentro.

Respuesta:

El tiempo de encuentro es X horas.

Ejemplo 2:

Tarea: Calcula el tiempo necesario para que dos trenes se encuentren si uno viaja a 80 km/h y el otro a 60 km/h, estando separados por 300 km.

Chain of Thought:

1. Sumar las velocidades de ambos trenes: $80 + 60 = 140$ km/h.
2. Calcular el tiempo: $300 / 140 \approx 2.14$ horas.
3. Concluir que los trenes se encontrarán en aproximadamente 2.14 horas.

Respuesta:

Los trenes se encontrarán en aproximadamente 2.14 horas.

```
Tarea: {tarea}
```


Chain of Thought:

```
"""
```

Optimización de Prompts

- Claridad y Precisión: Asegúrate de que las instrucciones sean claras y específicas.
- Uso de Plantillas: Establece estructuras predefinidas para mantener la consistencia.
- Evitar Ambigüedades: Utiliza lenguaje directo para minimizar interpretaciones erróneas.

```
def formatear_prompt(contexto, ejemplos, tarea):  
    prompt = f"{contexto}\n\n"  
    for i, ejemplo in enumerate(ejemplos, 1):  
        prompt += f"Ejemplo {i}:\n"  
        prompt += f"Tarea: {ejemplo['tarea']}\n"  
        prompt += f"Chain of Thought:\n{ejemplo['cot']}\nRespuesta: {ejemplo['respuesta']}\n\n"  
    prompt += f"Tarea: {tarea}\nChain of Thought:"  
    return prompt
```



```
class ModeloGPT(ModeloGenerativo):
    # ... (código anterior)

    def generar_texto_con_cot_many_shots(self, tarea, ejemplos, cot_steps_list):
        prompt = "Contexto: Eres un asistente de IA que ayuda a resolver problemas complejos paso a paso.\n\n"

        for i, (ejemplo, cot_steps) in enumerate(zip(ejemplos, cot_steps_list), 1):
            prompt += f"Ejemplo {i}:\n"
            prompt += f"Tarea: {ejemplo['tarea']}\n"
            prompt += f"Chain of Thought:\n{cot_steps}\nRespuesta: {ejemplo['respuesta']}\n\n"
        prompt += f"Tarea: {tarea}\nChain of Thought:\n"
        return self.generar_texto(prompt)
```

```

if __name__ == "__main__":
    api_key = "tu_api_key_aquí" # Reemplaza con tu clave API de OpenAI
    db = BaseDatos()
    modelo_gpt = ModeloGPT("GPT-4", "v1.0", api_key, db)

    tarea = "Determina el tiempo de encuentro de dos vehículos que viajan en direcciones opuestas."
    ejemplos = [
        {
            "tarea": "Determina el tiempo de encuentro de dos vehículos que viajan en direcciones opuestas.",
            "cot_steps": "1. Identificar las velocidades de ambos vehículos.\n2. Calcular la distancia total entre los vehículos.\n3. Establecer la ecuación de movimiento: tiempo = distancia / (velocidad1 + velocidad2).\n4. Resolver la ecuación para encontrar el tiempo de encuentro.",
            "respuesta": "El tiempo de encuentro es X horas."
        },
        {
            "tarea": "Calcula el tiempo necesario para que dos trenes se encuentren si uno viaja a 80 km/h y el otro a 60 km/h, estando separados por 300 km.",
            "cot_steps": "1. Sumar las velocidades de ambos trenes: 80 + 60 = 140 km/h.\n2. Calcular el tiempo: 300 / 140 ≈ 2.14 horas.\n3. Concluir que los trenes se encontrarán en aproximadamente 2.14 horas.",
            "respuesta": "Los trenes se encontrarán en aproximadamente 2.14 horas."
        }
    ]
    cot_steps_list = [
        "1. Identificar las velocidades de ambos vehículos.\n2. Calcular la distancia total entre los vehículos.\n3. Establecer la ecuación de movimiento: tiempo = distancia / (velocidad1 + velocidad2).\n4. Resolver la ecuación para encontrar el tiempo de encuentro.",
        "1. Sumar las velocidades de ambos trenes: 80 + 60 = 140 km/h.\n2. Calcular el tiempo: 300 / 140 ≈ 2.14 horas.\n3. Concluir que los trenes se encontrarán en aproximadamente 2.14 horas."
    ]

    texto_generado = modelo_gpt.generar_texto_con_cot_many_shots(tarea, ejemplos, cot_steps_list)
    print(texto_generado)

```

4. Integración con Clases de Python (ver Github)

5. Beneficios y Limitaciones de las Técnicas Avanzadas

Beneficios:

Chain of Thought (CoT):

- Mayor Precisión: Al desglosar el problema en pasos, se reduce la probabilidad de errores.
- Transparencia: Facilita la comprensión del razonamiento del modelo.
- Flexibilidad: Se puede aplicar a una amplia gama de tareas complejas.

One-Shot vs Many-Shots:

One-Shot:

- Rapidez: Menor tiempo de diseño.
- Eficiencia: Consume menos tokens.

Many-Shots:

- Robustez: Mejor manejo de variaciones en las tareas.
- Calidad: Respuestas más coherentes y precisas.

Limitaciones:

Chain of Thought (CoT):

- Consumo de Tokens: Los pasos intermedios aumentan la cantidad de tokens utilizados.
- Complejidad: Requiere una estructuración cuidadosa del prompt.

One-Shot vs Many-Shots:

One-Shot:

- Menor Robustez: Puede no manejar bien variaciones en la tarea.
- Consistencia: Mayor riesgo de respuestas inconsistentes.

Many-Shots:

- Mayor Consumo de Tokens: Necesita más ejemplos, lo que aumenta el uso de tokens.
- Tiempo de Diseño: Requiere más tiempo para seleccionar y diseñar ejemplos adecuados.

PREGUNTAS

