Inversiones IA generativa

Clase 01 - Introducción a agentes y automatización con LLM's

Sebastián Egaña Santibáñez

Enlaces del profesor

https://segana.netlify.app

https://github.com/sebaegana

https://www.linkedin.com/in/sebastian-egana-santibanez/

Clase 01

Conceptos iniciales

Automatization, AI automation and AI agent

- Automations (automatizaciones)

Uso de tecnología para ejecutar tareas sin intervención humana. Siempre tienen un disparador (trigger) y una acción.

Ejemplo: "Al enviar un formulario, guardar los datos en Airtable."

- AI automations (automatizaciones con IA)

Igual que lo anterior + un LLM como "cerebro" (ChatGPT, Claude, etc.) que transforma o decide sobre los datos (resumir, clasificar, analizar sentimiento).

Ejemplo: "Formulario → LLM hace análisis de sentimiento → guardar la etiqueta en Google Sheets.

- AI agents (agentes de IA)

Igual que la AI automation, pero el LLM además tiene herramientas (puede "llamar" APIs/servicios/sub-workflows).

Ejemplo: Por Telegram le pides "resume mis 5 últimos correos" → el agente usa la herramienta Gmail → responde; o "agenda una reunión" → usa Calendar, etc.

Pueden encadenar sub-agentes (arquitectura tipo "CEO y equipos"), cambiar de modelo, hablar con otros LLMs, navegar, usar Python, etc.

En resumen:

Concepto	¿Tiene LLM?	¿Usa herramientas?	Ejemplo rápido
Automation	No	No	Form → Airtable
AI Automation	Sí	No (solo LLM)	Form → LLM (sentimiento) → Sheets
AI Agent	Sí	Sí (APIs/tools)	Telegram → Agente → Gmail/Calendar

API (Application Programming Interface)

Una API (Application Programming Interface) es una forma estandarizada de comunicación entre programas. Permite que una aplicación "hable" con otra sin que el usuario intervenga directamente.

En simple: es un puente que conecta sistemas distintos para intercambiar datos o ejecutar acciones.

Ejemplos:

- Una web usa la API de Google Maps para mostrar un mapa.
- n8n usa la API de Airtable para guardar datos automáticamente.

Estructura básica:

- Request (solicitud): se envía información (por ejemplo, "guarda este registro").
- Response (respuesta): la API devuelve un resultado ("registro guardado con éxito").

Qué es la API de OpenAI

La API de OpenAI permite conectar tus aplicaciones o automatizaciones con modelos como GPT-4, GPT-4-mini o GPT-4-Omni, usando una clave (API key). El cobro se realiza por cantidad de tokens procesados (entrada y salida de texto).

- Cómo acceder
  - Entra a OpenAI Playground: https://platform.openai.com/
  - Inicia sesión o crea una cuenta.
  - En el panel superior derecho abre tu perfil → Billing → agrega una tarjeta de crédito.
    - Puedes empezar con un crédito de prueba (~USD 5).
  - Crea un proyecto → API keys → Create new secret key.
    - Esa clave te permite conectar OpenAI con n8n, Python, u otras apps.
    - Guárdala en un lugar seguro (no se vuelve a mostrar).
    - Puedes revocarla o rotarla cuando quieras.

- Costos aproximados

Modelo	Entrada	Salida	Comentario
GPT-4 mini	$0.15 USD	$0.60 USD	Recomendado: barato y rápido
GPT-4 Omni	$2.50 USD	$10 USD	Más preciso y versátil
o3-mini	$1.10 USD	$4.40 USD	Buen equilibrio para código o análisis
GPT-4.5 / o1	Mucho más caro	—	No recomendado para agentes

1 millón de tokens 750 000 palabras, por lo tanto los costos reales suelen ser de pocos dólares.

- Qué se puede hacer
  - Probar modelos en el Playground ajustando:
    - Modelo (GPT-4 mini, o3-mini, etc.)
    - Temperature (precisión vs creatividad)
    - Máx tokens
    - System message (“You are a helpful assistant…”)
  - Crear proyectos separados y compartirlos con otros usuarios.
  - Definir límites de gasto (por ejemplo, alerta a $10 o tope de $40 mensuales).
  - Revisar consumo diario en la pestaña Usage.

Qué es el Function Calling

El Function Calling (llamado a funciones) permite que un modelo de lenguaje (LLM) use otras herramientas o programas externos para realizar tareas que por sí solo no puede hacer bien.

Idea principal: "Un LLM se comporta como un sistema operativo". Esto quiere decir que:

- Tiene memoria temporal (su context window, como la RAM).
- Es muy bueno con texto, pero no con tareas específicas (por ejemplo, cálculos o imágenes).
- Gracias al function calling, puede "pedir ayuda" a otros programas especializados.
- Ejemplos
  - El LLM llama a una calculadora → resuelve operaciones matemáticas.
  - Llama a un modelo de difusión → genera imágenes o audio.
  - Llama al navegador → busca información reciente.
  - Llama a Gmail o Calendar → envía correos o crea eventos (como hacen los agentes de IA).
- Cómo funciona

El function calling se implementa mediante API requests: el LLM envía una instrucción a otra aplicación, recibe el resultado y lo integra en su respuesta. Modelos como GPT-4, o3-mini o Llama 3 ya traen esta capacidad incorporada.

- Relación con los Agentes de IA
  - Un AI Agent usa function calling para comunicarse con sus herramientas (correo, calendario, búsqueda web, etc.).
  - Puede combinar varios LLMs que actúan como "departamentos" (CEO, CTO, CFO…), cada uno con funciones especializadas.

Veamos un ejemplo simple:

1. Ve a OpenAI Playground: https://platform.openai.com/playground?mode=chat
2. En el menú derecho, activa la pestaña "Functions".
3. Añade una función con este JSON:

```
{
  "name": "calculator",
  "description": "Performs arithmetic operations",
  "parameters": {
    "type": "object",
    "properties": {
      "a": {"type": "number"},
      "b": {"type": "number"},
      "operation": {
        "type": "string",
        "enum": ["sum", "subtract", "multiply", "divide"]
      }
    },
    "required": ["a", "b", "operation"]
  }
}
```

Otro ejemplo:

```
{
  "name": "sentiment_classifier",
  "description": "Clasifica el sentimiento de un texto corto.",
  "strict": false,
  "parameters": {
    "type": "object",
    "properties": {
      "text": {
        "type": "string",
        "description": "Texto a analizar."
      },
      "language": {
        "type": "string",
        "description": "Idioma del texto (por ejemplo: 'es', 'en').",
        "enum": [
          "es",
          "en"
        ]
      }
    },
    "required": [
      "text"
    ]
```

```
  }
}
```

4. En el prompt escribe: “What is 25 multiplied by 4?”

Cómo dar conocimiento adicional a un LLM

Para que un modelo (como ChatGPT o un agente de IA) use conocimiento externo, existen dos métodos principales:

1. In-context learning: agregar contexto directamente en el prompt (limitado por la ventana de contexto).
2. Direct technology / RAG (Retrieval-Augmented Generation): almacenar la información en una base de datos vectorial que el modelo consulta solo cuando la necesita.

Qué es “Direct Technology” o RAG

Es una técnica que permite a un LLM buscar información externa sin saturar su memoria.

Funciona así:

1. Subes tu conocimiento (PDF, texto, Markdown, etc.).
2. Un modelo de embeddings convierte cada fragmento de texto en un conjunto de números (vectores) que representan su significado.
3. Esos vectores se guardan en una base de datos vectorial (como Pinecone, Chroma o FAISS).
4. Cuando haces una pregunta, el modelo busca solo los fragmentos más relevantes dentro del espacio vectorial.
5. El LLM combina esa información con su propio razonamiento para responder.

Qué son los embeddings y la base vectorial

- Embeddings: transforman texto en vectores numéricos (por ejemplo [0.12, 0.85, -0.44]).
- Palabras o frases con significados similares quedan cercanas entre sí en este espacio tridimensional.

Ejemplo: “apple” y “banana” estarán cerca; “cat” y “dog” también.

- Vector database: almacena estos vectores y permite hacer búsquedas por similitud semántica (no exactitud textual).
- Por qué es útil

- Permite subir múltiples documentos sin limitar la ventana de tokens.
- Se puede usar con cualquier LLM, incluso modelos open-source.
- Es más rápido y barato que un fine-tuning.
- Ideal para agentes de IA especializados (por ejemplo, un agente que consulta tus PDFs técnicos antes de responder).

Ejemplo de aplicación con agentes

Imagina una organización de agentes:

- El CEO (LLM principal) coordina las tareas.
- Un agente de conocimiento tiene tu base vectorial (PDFs de fitness, finanzas, o bienes raíces).
- Otro agente redacta artículos o informes.
- Otro publica o guarda resultados.

El flujo sería:

1. El CEO pide un artículo sobre fitness.
2. El primer agente busca datos en su vector database.
3. El segundo escribe el texto con esa información.
4. El tercero lo publica o guarda localmente.

Sobre los embeddings

Entrenamiento previo de los embeddings

Los embeddings que usamos (por ejemplo, con OpenAI o modelos open source) no se crean desde cero cada vez que los aplicamos. En realidad, provienen de un proceso de entrenamiento previo (pre-training) realizado sobre enormes cantidades de texto.

¿Qué aprende el modelo?

Durante el entrenamiento, el modelo analiza millones o miles de millones de frases y aprende relaciones estadísticas entre palabras. El objetivo es que, al ver una secuencia de texto, el modelo pueda predecir la siguiente palabra o reconocer palabras que encajan en el mismo contexto.

Texto de entrenamiento	Tarea implícita	Lo que aprende
"El **perro** ladra en el jardín."	Predecir "perro" a partir de su contexto ("El … ladra…")	Que "perro" aparece en contextos parecidos a "gato", "animal", "mascota".

Texto de entrenamiento	Tarea implícita	Lo que aprende
"El **avión** aterriza en la pista."	Predecir "avión"	Que "avión" se asocia a "vuelo", "piloto", "aeropuerto".

Así, el modelo descubre automáticamente los significados y las similitudes semánticas sin reglas escritas por humanos.

¿Qué produce este aprendizaje?

El modelo transforma cada palabra (o token) en un vector numérico que resume su significado aprendido. Estos vectores viven en un espacio de alta dimensión (por ejemplo, 768 o 1536 dimensiones). En ese espacio, las palabras que comparten contexto quedan cerca entre sí, y las que no tienen relación quedan lejos.

Ejemplo esquemático (espacio 2D simplificado):

Imaginemos que tenemos solo dos dimensiones (en realidad hay miles). Cada palabra se convierte en un punto:

Palabra	Eje X (animalidad)	Eje Y (domesticidad)
gato	0.9	0.95
perro	0.85	0.97
lobo	0.88	0.30
avión	0.10	0.05

En el gráfico:

- gato y perro están muy cerca, porque comparten significado.
- lobo está un poco más lejos: sigue siendo animal, pero no doméstico.
- avión está totalmente fuera del grupo

Un ejemplo en python:

Instalación de librerías:

```
pip install openai numpy faiss-cpu
```

Generación de embeds

```
from openai import OpenAI
import numpy as np
```

```
import faiss  # base de datos vectorial
client = OpenAI()

# Textos a convertir en embeddings
sentences = [
    "The dog is barking.",
    "A cat is sleeping on the couch.",
    "A car is driving down the street.",
    "I love my pet animal."
]

# Crear embeddings usando el modelo de OpenAI
response = client.embeddings.create(
    model="text-embedding-3-small",
    input=sentences
)

# Extraer los vectores
vectors = np.array([data.embedding for data in response.data])
print("Dimensión de los embeddings:", vectors.shape)
```

Salida esperada:

```
Dimensión de los embeddings: (4, 1536)
```

```
# Crear un índice FAISS
index = faiss.IndexFlatL2(vectors.shape[1])
index.add(vectors)

# Consultar algo nuevo
query = "My puppy is barking loudly."
query_vec = client.embeddings.create(
    model="text-embedding-3-small",
    input=query
).data[0].embedding

# Buscar el texto más parecido
D, I = index.search(np.array([query_vec]), k=2)
print("Resultados similares:")
for idx in I[0]:
    print("-", sentences[idx])
```

Resultados similares:

- The dog is barking.
- I love my pet animal.