



# Introducción a los agentes IA

Javier Rodriguez (@emb0scad0)

Github: <https://github.com/pinguytaz>

Web: [www.pinguytaz.net](http://www.pinguytaz.net)

16 Enero 2026

# Biografia

Inicios con un sencillo ZX81 y un Spectrum, vinculado al desarrollo cercano al hardware y tecnologías emergentes, gran interés en los sistemas embebidos.

Publicación de artículos en revistas especializadas: ZX, TodoSpectrum, RPP, JavaMagazine.

Autor del libro “De 0 a 100 con Arduino y ESP32”

Datos personales:

Nombre: Fco. Javier Rodriguez

Telegram: @emb0scad0

Github: <https://github.com/pinguytaz>

Web: <https://www.pinguytaz.net>



Fco. Javier Rodriguez Navarro

De 0 a 100  
con Arduino  
y ESP32



# Objetivos

- Programación de agentes IA sin frameworks.
- Conceptos y su ciclo básico.
- Modelos en la plataforma HuggingFace.
- Definición de rol, contexto y objetivos para agentesIA.
- Orquestación llamadas agentes o servicios externos.
- Ideas de aplicación de agentes de IA. s



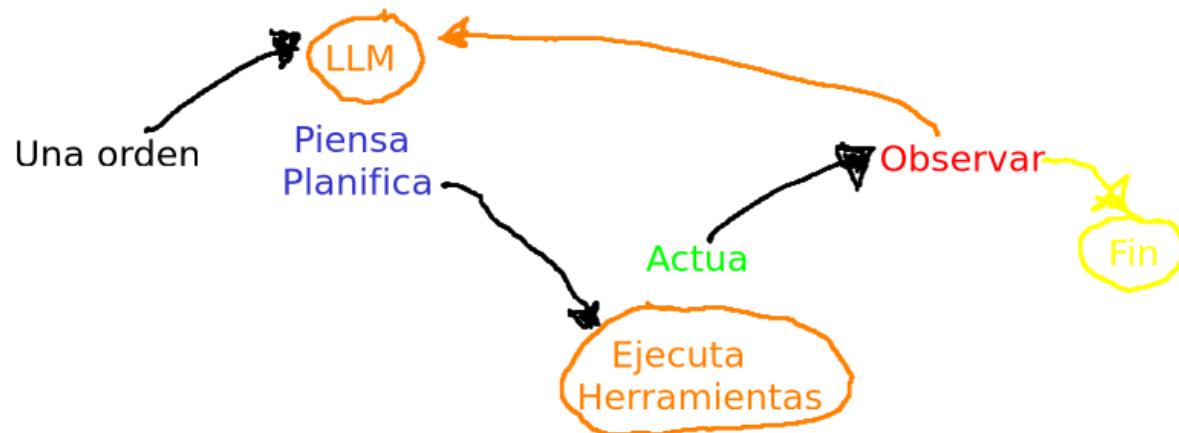


# Recordando conceptos

- **IA-Predictiva:** Localizamos un patrón
- **IA-Generativa:** IA que crea contenidos (texto, imágenes, video)
- **LLM (Large Language Models):** diseñados para procesar y generar lenguaje natural. Generación predictiva mediante probabilidades.
  - **Temperatura:** Baja más determinista, alta más creativa
  - **Top-k:** Número de TOKENs más probables (bajo deter.)
  - **Top-p:** Selección de TOKEN hasta llegar a la probabilidad
  - **Rep-Penalty:** Penaliza TOKENs repetidos. (Frecuencia o presencia)
  - **Max\_Token:** Límite de TOKENs generados
- **Ventana Contexto:** TOKENs que un modelo puede procesar.
- **Alucinaciones:** Respuesta generada con información falsa o engañosa.
- **RAG (Retival-Augmented Generation)** técnica que nos permite mejorar los modelos LLM, integramos información externa, a nuestras necesidades.



# ¿Que es un agente?



“Capacidad de razonamiento, planificación” interactúa con el entorno para lograr su objetivo.

- 1) Recoge una orden en lenguaje natural, pasa por el modelo.
- 2) El agente analizara la salida del LLM y determina las herramientas a utilizar.
- 3) La vuelta de las herramientas se envía también al LLM, realizando un bucle hasta que logra su objetivo.

**Ejemplos de agentes:** asistentes virtuales, chatbot de servicios a cliente, personajes de videojuegos



# Componentes agente



- Componentes
  - **Cerebro:** Decide cual sera el siguiente paso y que acciones realizar.

Más común un LLM, Entrada-Texto y Salida-Texto

- **Cuerpo:** acciona las capacidades/herramientas, que interactúan con el exterior.

Herramienta	Descripción
Búsqueda Web	Permite al agente obtener información actualizada de internet.
Generación de Imágenes	Crea imágenes basadas en descripciones textuales.
Recuperación	Recupera información de una fuente externa.
Interfaz de API	Interactúa con una API externa (GitHub, YouTube, Spotify, etc.).

- **Observación:** El modelo reflexiona sobre la respuesta.



# Cerebro

- Lo más común es texto de entrada, pero también se pueden usar modelos con entrada de audio. Proporciona la base para entender el lenguaje humano.
- Un LLM esta entrenado con gran cantidad de datos.

Modelo	Empresa
GPT4	Open AI
Deepseek-R1	Deepseek
Llama3	Meta (Facebook)
Gemma	Google
Mistral	Mistral
QWEN	AliBaBa

Buscar modelos **Instruct / Coder** afinados para seguir instrucciones.

**Qwen/Qwen2.5-Coder-1.5B-Instruct (Disco 2,9GB) 8GB RAM**





# Hugging Face

## Plataforma open source

- **Modelos** de lenguaje. (Afinadas por la comunidad )
  - Uso en local o por inferencia mediante API en remoto
  - DataSets
  - Infraestructura para entrenar y desplegar proyectos IA, librerías





# Hugging Face

Hugging Face

Search models, datasets, users...

Models

Datasets

Spaces

Main Tasks 1 Libraries Languages Licenses  
Other

Tasks

Text Generation Any-to-Any  
Image-Text-to-Text Image-to-Text  
Image-to-Image Text-to-Image  
Text-to-Video Text-to-Speech + 44

Parameters

< 1 3B 6B 12B 32B 128B > 500B

Models 1,136

instruct coder

Full-text search

Qwen/Qwen2.5-Coder-1.5B-Instruct-GGUF

Text Generation · 2B · Updated Nov 12, 2024 · 19.8k · 32

Qwen/Qwen2.5-Coder-0.5B-Instruct

Text Generation · 0.5B · Updated Nov 18, 2024 · 2.3M · 60

Instruct multiple

Qwen/Qwen2.5-Coder-1.5B-Instruct

Text Generation · 2B · Updated Jan 12, 2025 · 94.7k · 97

Qwen/Qwen2.5-Coder-0.5B-Instruct-GGUF

Text Generation · 0.6B · Updated Nov 11, 2024 · 5.34k · 13





# Hugging Face

Owen / Owen2.5-Coder-1.5B-Instruct

Text Generation Transformers Safetensors English qwen2 code codeqwen chat qwen qwen-coder conversational text-generation-inference

arxiv:2409.12186 arxiv:2407.10671 License: apache-2.0

Model card Files and versions Community 4 Edit model card

Downloads last month  
94,654

Safetensors Model size: 2B params Tensor type: BF16 Chat template

Inference Providers NEW Featherless AI Examples

Input a message to start chatting with Qwen/Qwen2.5-Coder-1.5B-Instruct.

**Qwen2.5-Coder-1.5B-Instruct**

**Qwen Chat**

### Introduction

Qwen2.5-Coder is the latest series of Code-Specific Qwen large language models (formerly known as CodeQwen). As of now, Qwen2.5-Coder has covered six mainstream model sizes, 0.5, 1.5, 3, 7, 14, 32 billion parameters, to meet the needs of different developers. Qwen2.5-Coder brings the following improvements upon CodeQwen1.5:

- Parámetros: : xB (x\*1000millones de parámetros. (Billones americanos))
- Tipo de datos: **BF16**(1-Signo, 8-exp., 7-mantisa) FP32, FP16, INT8, 4, 3, 2

Qwen2.5-1.5B: GPU float16: 3.5 GB VRAM CPU float16: 7.9 GB RAM



# Métricas Recursos

## Costes de operación

$$M = N * P * (1 + ovh(\%))$$

M → Memoria, N → Parámetros del modelo, P → Bytes del modelo

ovh → Overhead (GPU/VRAM \*1,2 CPU/RAM \*1.5 + 4GB)

## Rendimiento

Tokens por segundo (**t/s**): Velocidad de generación

10-15 t/s es aproximadamente leer, menos de 5t/s LENTO



# SinRed.py

```
↳ ./SinRed.py
Modelo cargado
Pregunta: Dame la temperatura en Madrid y Logroño, asi como la hora y tambien dime si llovera en Alemania
*** Rendimiento ***
    Velocidad: 1.77 tokens/s
    Latencia total: 17.50 segundos
----- Inicia Salidas procesando -----
Salida del agente: FUNC(elTiempo:Madrid)
FUNC(elTiempo:Logroño)
FUNC(laFecha)
FUNC(elTiempo:Alemania)
Acciones obtenidas [{"accion": "elTiempo", "param": "Madrid"}, {"accion": "elTiempo", "param": "Logroño"}, {"accion": "laFecha", "param": ""}, {"accion": "elTiempo", "param": "Alemania"}]
    Ejecutamos la accion elTiempo(Madrid)
    Ejecutamos la accion elTiempo(Logroño)
    Ejecutamos la accion laFecha()
    Ejecutamos la accion elTiempo(Alemania)
----- FIN de las salidas procesadas -----
**** El resultado es:
Madrid: 4.0°C, sensación termica 2.0, humedad 93%, lluvias 0.0, Partly cloudy

Logroño: 6.0°C, sensación termica 6.0, humedad 81%, lluvias 0.0, Partly cloudy

Fecha y Hora: 14-01-2026 22:53
Alemania: 6.0°C, sensación termica 5.0, humedad 93%, lluvias 0.0, Partly cloudy
```



# Esquema de Agente

- Cargar el modelo
- Definir Prompt
- Realizar pregunta “CONSULTA”
- Ejecución Herramientas “ACCION”
  - Analizar herramientas necesarias
  - Llamadas a las herramientas
- ¿Observación?



# Carga Modelo

Descargamos el modelo Descarga.py

```
from huggingface_hub import snapshot_download
```

```
snapshot_download(  
    repo_id=MODELO, → Modelo QWEN..  
    local_dir=cache_dir, → Directorio descarga  
)
```

Modelo.py

- 1) Tokenizador para el modelo "AutoTokenizer.from\_pretrained"
- 2) Cargamos el modelo "AutoModelForCausalLM.from\_pretrained" o "pipeline"

```
AutoModelForCausalLM.from_pretrained(  
    cache_dir, → Directorio modelo  
    dtype=torch.float16, → Tipo de dato de pesos  
    device_map="auto", → Como se ejecuta CPU, CUDA  
    local_files_only=True → Solo archivos LOCALES  
)
```



# Consulta Agente.py

- PROMPT

```
messages = [  
    {"role": "system", "content": SYSTEM_PROMPT},  
    {"role": "user", "content": pregunta},  
]
```

## ROLES

- **System:** Define la personalidad y comportamiento del modelo.
- **User:** Nosotros realizando la pregunta
- **Assistant:** Respuesta generada por el modelo.



# Consulta

- `tokenizer.apply_chat_template`: Convierte mensaje al formato del modelo

```
IDsEntradas = tokenizer.apply_chat_template(  
    messages,      # Mensaje con roles.  
    tokenize=True, # convertir a ID_TOKENs  
    add_generation_prompt=True, # Añade terminador (Token especial)  
    return_tensors="pt", #Generar un objeto pytorch, modelos locales.  
    padding=True,     # Rellenar mensajes a longitud maxima Batch  
    return_dict=True # Pone diccionario no Token  
).to(model.device)
```



# Consulta

- `model.generate`: Toma el mensaje y genera otro con la respuesta

```
model.generate(  
    **IDsEntradas,      mensaje  
    max_new_tokens=50,  #Límite de Tokens nuevos  
    temperature=0.25,   # Aleatoriedad  
    top_p = 0.9,        # grupo de palabras  
    top_k = 3,          # num. palabras más probables  
    do_sample=True,    # Activa la temperatura.  
    #pad_token_id=tokenizer.pad_token_id # Evita el warning  
    repetition_penalty=1.1, # Evita repeticiones.  
    pad_token_id=tokenizer.eos_token_id, # define TOKEN padding  
    eos_token_id=tokenizer.eos_token_id # Para al encontrar EOS  
)  
QWEN
```

```
tokenizer.decode(salidas[0][IDsEntradas["input_ids"].shape[1]:], skip_special_tokens=True)
```



# Acción

Las Herramientas son las capacidades de que dispone nuestro agente.

Herramienta	Descripción
Búsqueda Web	Permite al agente obtener información actualizada de internet.
Generación de Imágenes	Crea imágenes basadas en descripciones textuales.
Recuperación	Recupera información de una fuente externa.
Interfaz de API	Interactúa con una API externa (GitHub, YouTube, Spotify, etc.).

- 1) Análisis el mensaje para ver que acciones tenemos que lanzar. **obtieneAccion(salidaAgente)**

JSON, Texto ejemplo FUNC(nombre:parámetros), XML

- 2) Selección de la herramienta a ejecutar y ejecución con los datos necesarios.

- Llamadas a funciones
- Generación de código y ejecución
- Ejecución de comandos

- 3) Recogida y ensamblado de la información.

Verificamos que hemos cumplido “Observación”



# FrameWoks

Proporcionan flexibilidad en el flujo de trabajo para resolver una tarea especifica. Generan las cadenas de prompts, si es sencilla podemos usar el código plano sin necesidad de framework, pero si si estamos llamando a funciones y usamos multiples agentes se recomienda

- **Smoleagents**

<https://huggingface.co/learn/agents-course/es/unit2/smolagents/introduction>

- **Llamaindex**

<https://huggingface.co/learn/agents-course/es/unit2/llama-index/introduction>

- **LangGraph**

- <https://huggingface.co/learn/agents-course/es/unit2/langgraph/introduction>



# GRACIAS

