



2026

PROYECTO PERSONAL DE IMPLANTACIÓN DE IA EN LOCAL

IZAN RODRIGUEZ GARCIA

Datos personales

Nombre	Izan Rodriguez Garcia
Correo:	izanro06@gmail.com
Linkedin:	www.linkedin.com/in/izan-rodriguez-a35652295

Introducción

"Introducción y Propósito del Proyecto"

En la actualidad, la soberanía de los datos y la privacidad se han convertido en pilares fundamentales del desarrollo tecnológico. Este proyecto nace de la necesidad de desplegar un ecosistema de Inteligencia Artificial de alto rendimiento que funcione de manera 100% local, eliminando la dependencia de servicios en la nube y garantizando la seguridad de la información.

*A lo largo de esta documentación, se detalla el proceso de configuración de un entorno híbrido basado en Linux, utilizando contenedores **Docker** para la gestión de interfaces (**Open WebUI**), **Ollama** como motor de modelos de lenguaje (LLM) y **ComfyUI** para la generación avanzada de imágenes mediante el modelo **Flux.1**. El objetivo principal es demostrar que, con hardware de consumo (RTX 4060 Ti 16GB), es posible alcanzar flujos de trabajo profesionales en IA con latencia mínima y control total sobre los modelos."*

Hardware usado:

CPU:	Intel i5-13600KF
RAM:	2 x 8GB Kingston Fury 3200Mhz
GPU:	Zotac Nvidia 4060TI 16gb VRAM
Disco duro:	1TB NVME M.2 Kingston

Antes de empezar:

Antes de empezar: Consideraciones y Requisitos

Antes de proceder con la instalación, es fundamental asegurarse de que el entorno cumple con los requisitos técnicos y de configuración para garantizar un rendimiento óptimo de la IA en local.

1. Requisitos de Hardware (Especificaciones del Proyecto)

Este despliegue ha sido testado y optimizado para la siguiente configuración:

- **GPU:** NVIDIA GeForce RTX 4060 Ti (16GB VRAM). *La memoria de vídeo es el factor crítico para cargar modelos como Mistral 24B o Flux.1 sin recurrir a la memoria RAM del sistema, lo que ralentizaría el proceso.*
- **SO:** Linux (Ubuntu/Debian recomendado) con drivers de NVIDIA actualizados y soporte para CUDA.
- **Almacenamiento:** Se recomienda un SSD con al menos 40GB de espacio libre para el almacenamiento de pesos de modelos y contenedores.

2. Dependencias de Software

Es necesario contar con las siguientes herramientas instaladas o permisos para instalarlas:

- **Docker:** Para el aislamiento de la interfaz de usuario.
- **Python 3.10+:** Necesario para los entornos virtuales de ComfyUI.
- **Privilegios Sudo:** Para la gestión de servicios del sistema (Systemd) y permisos de Docker.

Datos personales	2
Introducción	2
Hardware usado:	2
Antes de empezar:	3
Antes de empezar: Consideraciones y Requisitos	3
1. Requisitos de Hardware (Especificaciones del Proyecto)	3
2. Dependencias de Software	3
Instalación de Ollama, verificación del servicio, Descarga del modelo.	5
Instalación del motor de inferencia (Ollama)	5
Descarga de modelos (LLM)	6
Instalación de docker, Despliegue de Open WebUI, Configuración inicial	7
Instalación del Entorno de contenedores	7
Despliegue de Open WebUI	8
Configuración Inicial y primer chat	9
Preparación del Entorno de Generación de Imágenes	13
Configuración del Entorno Virtual y Motores gráficos	14
Descarga de modelos Optimizados (Flux GGUF)	16
Instalación de soporte para GGUF (Custom nodes)	17
Configuración ComfyUI	18
Instalación de Codificadores de Texto (CLIP/T5) y VAE	19
Exposición de Ollama a la Red	23
Integración Final en OpenWebUI	24

Instalación de Ollama, verificación del servicio, Descarga del modelo.

Instalación del motor de inferencia (Ollama)

Para comenzar, instalamos **Ollama**, que actuará como el backend o "cerebro" encargado de ejecutar los modelos de Inteligencia Artificial en local.

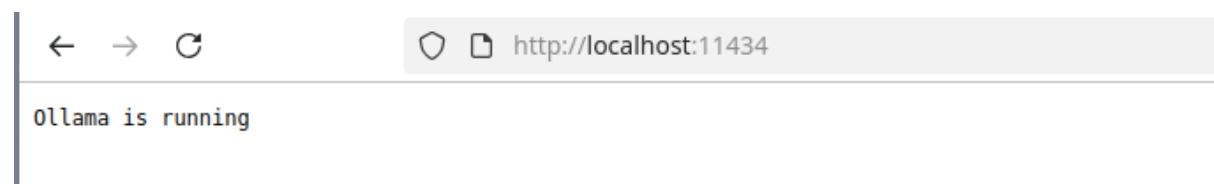
Comando usado: `curl -fsSL https://ollama.com/install.sh | sh`

Ejecutamos el script de instalación automática oficial. Este comando descarga los binarios necesarios, crea el usuario de sistema **ollama** y configura los servicios para que se inicien automáticamente con el sistema. Como se observa en el log, el instalador detecta correctamente el hardware gráfico (**NVIDIA GPU installed**), lo cual es crucial para la aceleración por hardware.

Una vez finalizada la instalación, verificamos que el servicio está activo y escuchando peticiones. Para ello, accedemos desde el navegador a la dirección local del puerto por defecto de la API de Ollama.

<http://localhost:11434> o <http://127.0.0.1:11434>

si aparece el siguiente mensaje la instalación se ha hecho correctamente



Descarga de modelos (LLM)

Con el servicio activo, procedemos a descargar nuestro primer modelo de lenguaje. Utilizamos el comando `ollama pull [nombre_modelo]` (Se recomienda hacer primero una búsqueda de qué modelo queremos o que podemos ejecutar, cada uno tiene sus características)

En la primera captura se observa un error al intentar descargar una versión inexistente o mal escrita (`dolphin-mistral-nemo`), por lo que el sistema devuelve "manifest file does not exist".

```
tzan@tzan:~$ ollama pull dolphin-mistral-nemo
pulling manifest
Error: pull model manifest: file does not exist
tzan@tzan:~$ █
```

Posteriormente, se ejecuta la descarga correcta del modelo **Mistral Nemo** (un modelo eficiente de 12B parámetros). El sistema descarga las diferentes capas (sha256 digest) y verifica la integridad del archivo manifiesto para finalizar la instalación en la librería local.

```
tzan@tzan:~$ ollama pull mistral-nemo
pulling manifest
pulling b559938ab7a0: 100% 7.1 GB
pulling 438402ddac75: 100% 683 B
pulling 43070e2d4e53: 100% 11 KB
pulling ed11eda7790d: 100% 30 B
pulling 65d37de20e59: 100% 486 B
verifying sha256 digest
writing manifest
success
tzan@tzan:~$ █
```

Instalación de docker, Despliegue de Open WebUI, Configuración inicial

Instalación del Entorno de contenedores

Para ejecutar la interfaz gráfica de forma aislada y segura, utilizamos **Docker**.

Procedemos a actualizar los repositorios e instalar el paquete [docker.io](#).

Actualizamos la lista de paquetes del sistema y procedemos a la instalación del motor de contenedores Docker desde los repositorios oficiales de Ubuntu.

Comando usado: *sudo apt update && sudo apt install docker.io -y*

```
izan@izan:~$ sudo apt update
sudo apt install docker.io -y
[sudo] contraseña para izan:
Obj:1 http://security.ubuntu.com/ubuntu noble-security InRelease
Obj:2 https://ppa.launchpadcontent.net/danielrichter2007/grub-customizer/ubuntu noble InRelease
Obj:3 http://archive.ubuntu.com/ubuntu noble InRelease
Obj:4 http://archive.ubuntu.com/ubuntu noble-updates InRelease
Obj:5 http://archive.ubuntu.com/ubuntu noble-backports InRelease
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
```

Una vez instalado, es necesario añadir nuestro usuario actual al grupo de seguridad **docker**. Esto permite ejecutar contenedores sin necesidad de utilizar **sudo** constantemente, mejorando la seguridad y la comodidad del flujo de trabajo. Aplicamos los cambios de grupo inmediatamente con el comando **newgrp**.

Comando usado: *sudo usermod -aG docker \$USER* y *newgrp docker*

```
izan@izan:~$ sudo usermod -aG docker $USER
izan@izan:~$ 
izan@izan:~$ newgrp docker
izan@izan:~$ 
```

Despliegue de Open WebUI

Desplegamos la interfaz **Open WebUI** utilizando un contenedor Docker. Esta interfaz se conectará automáticamente a nuestro servicio local de Ollama.

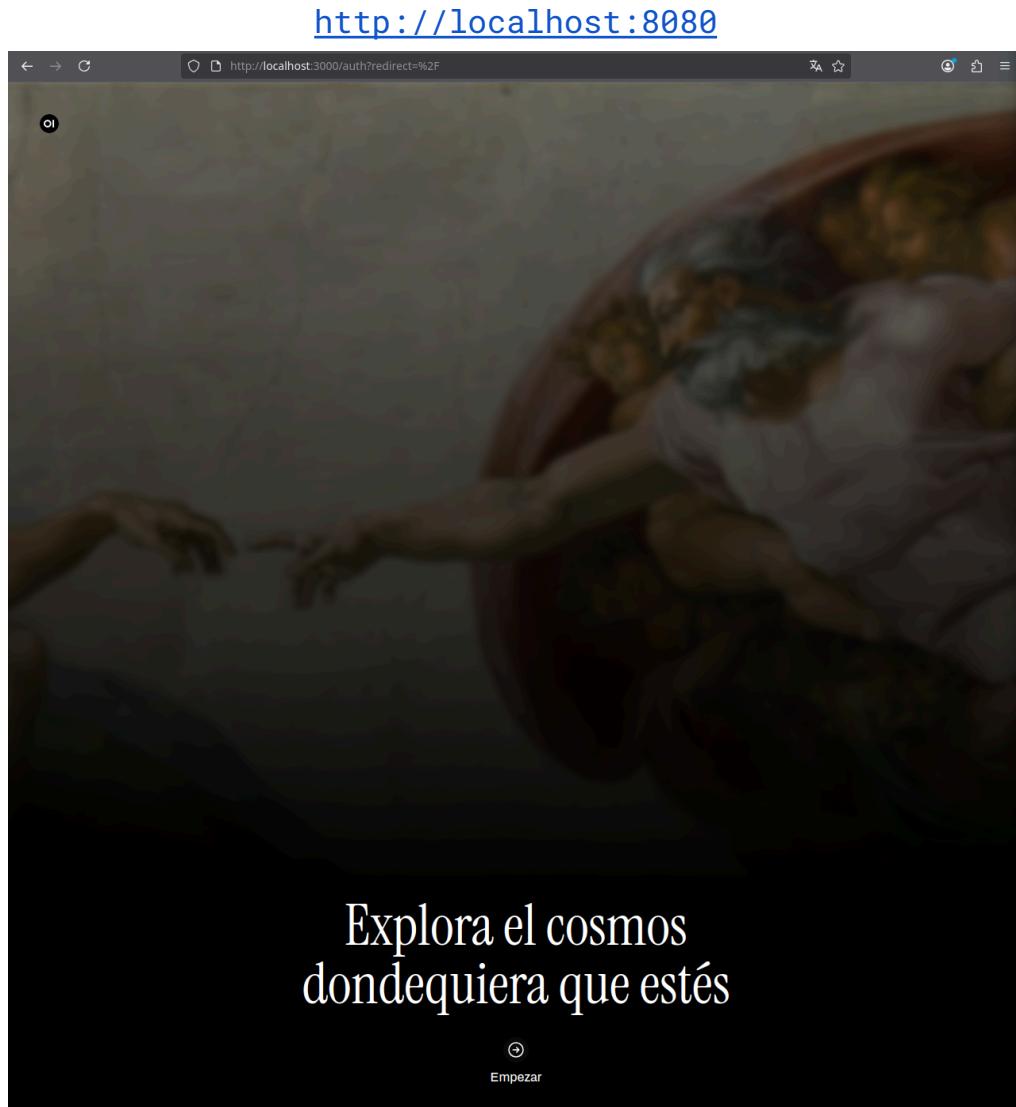
Ejecutamos el contenedor con la configuración de red en modo **host** (para que pueda ver a Ollama en el puerto 11434 local) y configuramos un volumen persistente para que nuestros chats y configuraciones no se borren al reiniciar. comando usado: `docker run -d --network host -v open-webui:/app/backend/data --name open-webui --restart always ghcr.io/open-webui/open-webui:main`

Explicación de flags:

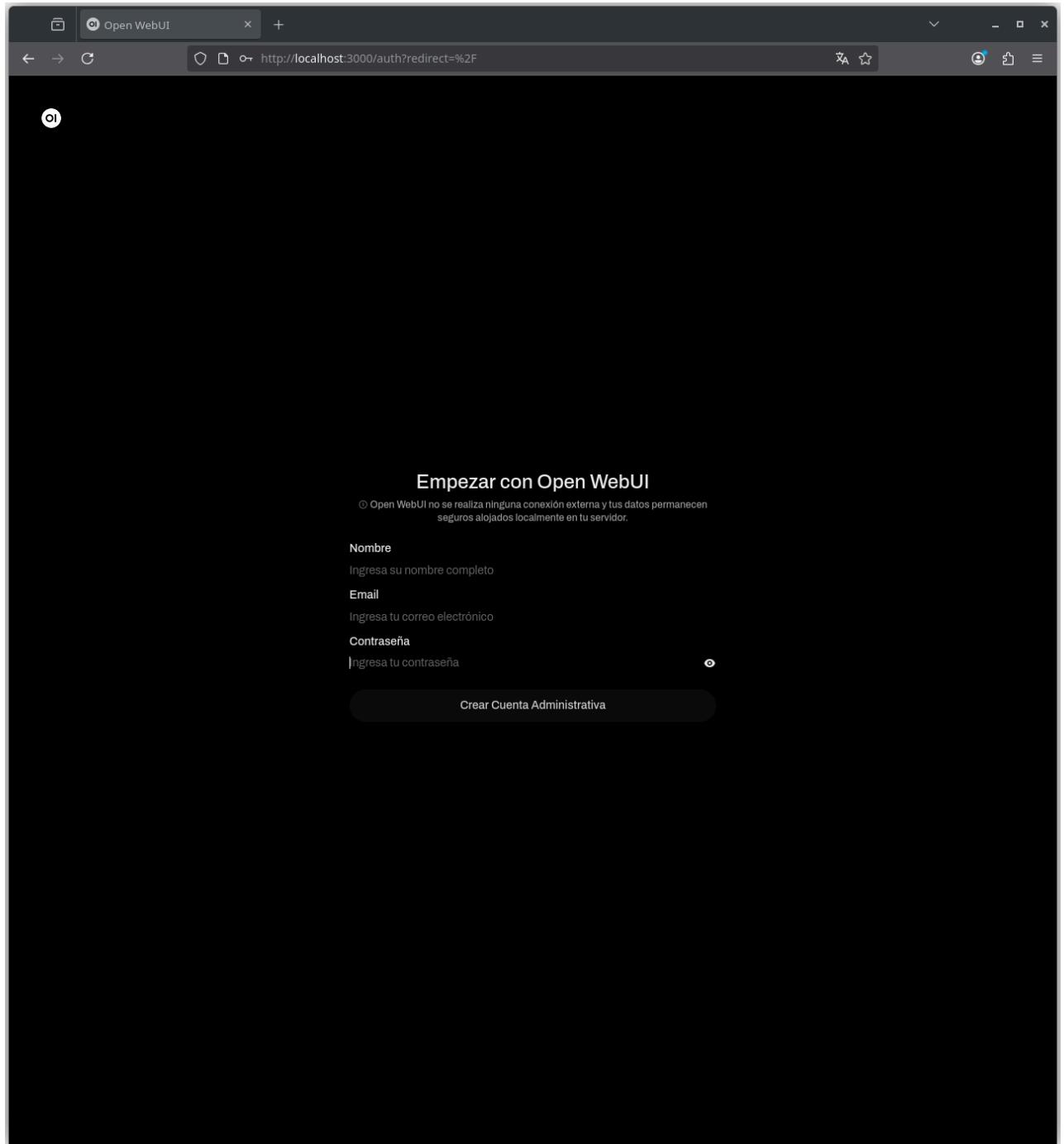
- **--network host:** Comparte la red con el PC (vital para conectar con Ollama).
- **--restart always:** Si el PC se reinicia, el chat arranca solo.
- **-v open-webui:....:** Guarda tus datos en un disco virtual seguro.

```
open-webui
izan@izan:~$ docker run -d --network host -v open-webui:/app/backend/data --name open-webui --restart always ghcr.io/open-webui/open-webui:main
4ad617fd8b515b21dc0f7e62585544a622dc062ca3a286bfdad7a80d365db388
izan@izan:~$
```

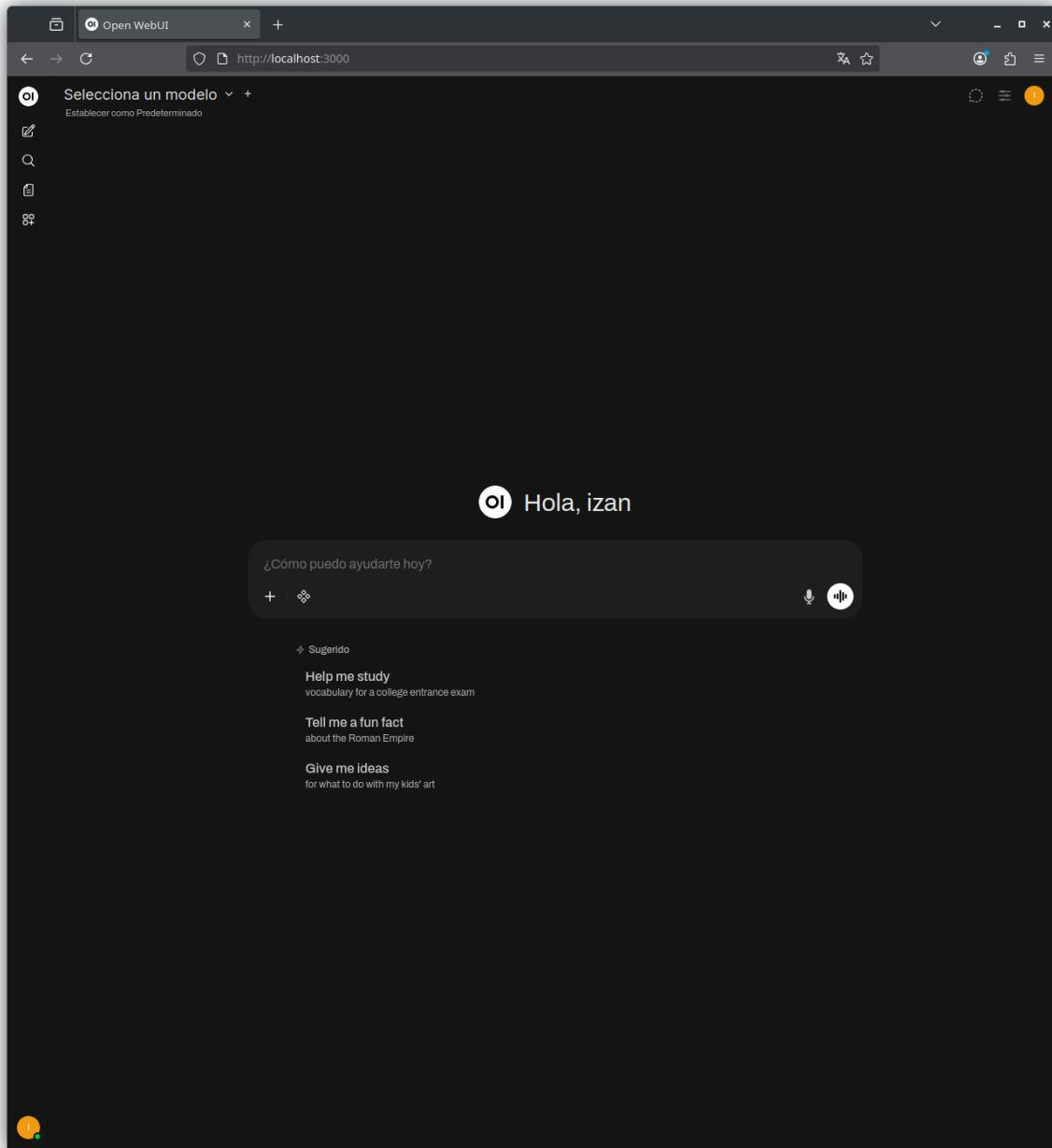
Configuración Inicial y primer chat



Accedemos a la interfaz web a través del navegador en <http://localhost:8080>. Al ser la primera vez, el sistema solicita crear una cuenta de administrador. *Nota: Estos datos son totalmente locales; no se envían a ningún servidor externo.*



Una vez dentro, la interfaz detecta automáticamente los modelos descargados en Ollama (como Mistral). Seleccionamos el modelo en el menú superior y realizamos una prueba de generación de texto. El sistema responde correctamente, confirmando la integración exitosa entre Docker (Frontend) y Ollama (Backend).



Preparación del Entorno de Generación de Imágenes

Para la generación de imágenes, utilizaremos **ComfyUI**, una interfaz basada en nodos altamente modular. Antes de instalarla, preparamos el sistema con las dependencias de Python necesarias y creamos una estructura de carpetas organizada.

Comenzamos instalando `git`, `python3-venv` y `python3-pip` desde los repositorios de Ubuntu. A continuación, creamos un directorio dedicado (`~/IA`) para mantener ordenados nuestros proyectos y clonamos el repositorio oficial de ComfyUI desde GitHub.

Comandos usados:

```
sudo apt install git python3-venv python3-pip -y  
mkdir -p ~/IA  
cd ~/IA  
git clone https://github.com/comfyanonymous/ComfyUI.git
```

```
Izan@izan:~$ sudo apt install git python3-venv python3-pip -y  
Leyendo lista de paquetes... Hecho  
Creando árbol de dependencias... Hecho  
Leyendo la información de estado... Hecho  
git ya está en su versión más reciente (1:2.43.0-1ubuntu7.3).  
python3-venv ya está en su versión más reciente (3.12.3-0ubuntu2.1).  
python3-pip ya está en su versión más reciente (24.0+dfsg-1ubuntu1.3).  
0 actualizados, 0 nuevos se instalarán, 0 para eliminar y 0 no actualizados  
Izan@izan:~$ 
```

```
Izan@izan:~$ mkdir -p ~/IA  
cd ~/IA  
git clone https://github.com/comfyanonymous/ComfyUI.git  
cd ComfyUI  
Clonando en 'ComfyUI'...  
remote: Enumerating objects: 29567, done.  
remote: Counting objects: 100% (7/7), done.  
remote: Compressing objects: 100% (5/5), done.  
remote: Total 29567 (delta 2), reused 4 (delta 2), pack-reused 29560 (from 1)  
Recibiendo objetos: 100% (29567/29567), 75.90 MiB | 32.85 MiB/s, listo.  
Resolviendo deltas: 100% (20046/20046), listo.  
Izan@izan:~/IA/ComfyUI$ 
```

Configuración del Entorno Virtual y Motores gráficos

Es una buena práctica aislar las librerías de Python para no interferir con el sistema. Creamos un entorno virtual (`venv`) y procedemos a instalar **PyTorch** con soporte para CUDA (NVIDIA), lo cual es vital para que la tarjeta gráfica realice los cálculos.

Generamos y activamos un entorno virtual dentro de la carpeta de ComfyUI. Una vez dentro (indicado por el prefijo (`venv`) en la terminal), instalamos las librerías de Torch optimizadas para nuestra versión de CUDA.

Comandos usados: `python3 -m venv venv`

```
izan@izan:~/IA/ComfyUI$ python3 -m venv venv
source venv/bin/activate
(venv) izan@izan:~/IA/ComfyUI$
```

Comandos usados:

Comando de instalación de dependencias: `pip install torch torchvision torchaudio --extra-index-url https://download.pytorch.org/whl/cu121`

Finalmente, instalamos el resto de requerimientos básicos del proyecto con: `pip install -r requirements.txt`

Nota: Este proceso puede tardar unos minutos

```
(venv) izan@izan:~/IA/ComfyUI$ pip install torch torchvision torchaudio --extra-index-url https://download.pytorch.org/whl/cu121
pip install -r requirements.txt
Looking in indexes: https://pypi.org/simple, https://download.pytorch.org/whl/cu121
Collecting torch
  Downloading torch-2.9.1-cp312-cp312-manylinux_2_28_x86_64.whl.metadata (30 kB)
Collecting torchvision
  Downloading torchvision-0.24.1-cp312-cp312-manylinux_2_28_x86_64.whl.metadata (5.9 kB)
Collecting torchaudio
  Downloading torchaudio-2.9.1-cp312-cp312-manylinux_2_28_x86_64.whl.metadata (6.9 kB)
Collecting filelock (from torch)
  Downloading filelock-3.20.3-py3-none-any.whl.metadata (2.1 kB)
Collecting typing-extensions>=4.10.0 (from torch)
  Downloading https://download.pytorch.org/whl/typing_extensions-4.15.0-py3-none-any.whl.metadata (3.3 kB)
Collecting setuptools (from torch)
  Downloading setuptools-80.9.0-py3-none-any.whl.metadata (6.6 kB)
Collecting sympy>=1.13.3 (from torch)
  Downloading sympy-1.14.0-py3-none-any.whl.metadata (12 kB)

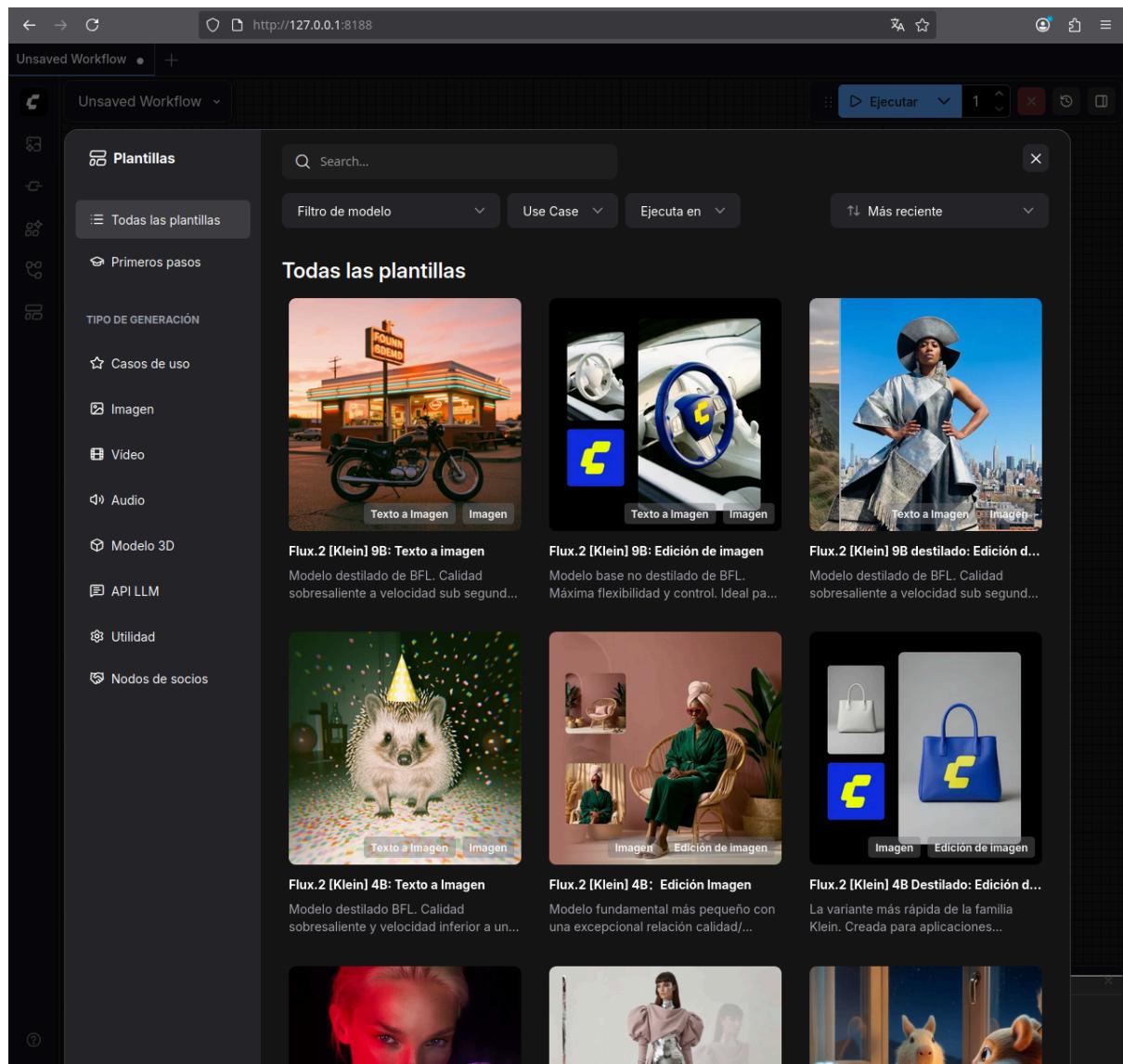
  Downloading torch-2.9.1-cp312-cp312-manylinux_2_28_x86_64.whl (899.7 MB)
  899.7/899.7 MB 12.8 MB/s eta 0:00:00
  Downloading nvidia_cublas_cu12-12.8.4.1-py3-none-manylinux_2_27_x86_64.whl (594.3 MB)
  594.3/594.3 MB 17.5 MB/s eta 0:00:00
  Downloading nvidia_cuda_cupti_cu12-12.8.90-py3-none-manylinux2014_x86_64_manylinux_2_17_x86_64.whl (10.2 MB)
  10.2/10.2 MB 74.4 MB/s eta 0:00:00
  Downloading nvidia_cuda_nvrtc_cu12-12.8.93-py3-none-manylinux2019_x86_64_manylinux_2_12_x86_64.whl (88.0 MB)
  88.0/88.0 MB 56.4 MB/s eta 0:00:00
  Downloading nvidia_cuda_runtime_cu12-12.8.96-py3-none-manylinux2014_x86_64_manylinux_2_17_x86_64.whl (954 kB)
  954.8/954.8 MB 84.1 MB/s eta 0:00:00
  Downloading https://download.pytorch.org/whl/nvidia_cudnn_cu12-9.10.2.21-py3-none-manylinux_2_27_x86_64.whl (706.8 MB)
  706.8/706.8 MB 14.8 MB/s eta 0:00:00
  Downloading nvidia_cufft_cu12-11.3.3-83-py3-none-manylinux_2014_x86_64_manylinux_2_17_x86_64.whl (193.1 MB)
  193.1/193.1 MB 48.3 MB/s eta 0:00:00
  Downloading nvidia_cufile_cu12-1.13.1.3-py3-none-manylinux_2_27_x86_64_manylinux_2_17_x86_64.whl (1.2 MB)
  1.2/1.2 MB 79.7 MB/s eta 0:00:00
  Downloading nvidia_curand_cu12-10.3.9.90-py3-none-manylinux_2_27_x86_64.whl (63.6 MB)
  63.6/63.6 MB 55.3 MB/s eta 0:00:00
  Downloading nvidia_cusolver_cu12-11.7.3.90-py3-none-manylinux_2_27_x86_64.whl (267.5 MB)
  267.5/267.5 MB 29.9 MB/s eta 0:00:00
  Downloading nvidia_cusparse_cu12-12.5.8.93-py3-none-manylinux2014_x86_64_manylinux_2_17_x86_64.whl (288.2 MB)
  288.2/288.2 MB 28.5 MB/s eta 0:00:00
  Downloading https://download.pytorch.org/whl/nvidia_cusparse_cu12-0.7.1-py3-none-manylinux2014_x86_64.whl (287.2 MB)
  287.2/287.2 MB 28.6 MB/s eta 0:00:00
  Downloading https://download.pytorch.org/whl/nvidia_nccl_cu12-2.27.5-py3-none-manylinux2014_x86_64_manylinux_2_17_x86_64.whl (322.3 MB)
  322.3/322.3 MB 26.8 MB/s eta 0:00:00
  Downloading nvidia_nvjitlink_cu12-12.8.93-py3-none-manylinux2010_x86_64_manylinux_2_12_x86_64.whl (39.3 MB)
  39.3/39.3 MB 54.3 MB/s eta 0:00:00
  Downloading https://download.pytorch.org/whl/nvidia_nvshmem_cu12-3.3.20-py3-none-manylinux2014_x86_64_manylinux_2_17_x86_64.whl (124.7 MB)
  124.7/124.7 MB 48.3 MB/s eta 0:00:00
  Downloading nvidia_nvtx_cu12-12.8.90-py3-none-manylinux2014_x86_64_manylinux_2_17_x86_64.whl (89 kB)
  89.0/89.0 kB 33.7 MB/s eta 0:00:00
  Downloading https://download.pytorch.org/whl/triton-3.5.1-cp312-cp312-manylinux_2_27_x86_64_manylinux_2_28_x86_64.whl (170.5 MB)
  170.5/170.5 MB 37.0 MB/s eta 0:00:00
  Downloading torchvision-0.24.1-cp312-cp312-manylinux_2_28_x86_64.whl (8.0 MB)
  8.0/8.0 MB 56.4 MB/s eta 0:00:00
  Downloading torchaudio-2.9.1-cp312-cp312-manylinux_2_28_x86_64.whl (2.1 MB)
  2.1/2.1 MB 86.1 MB/s eta 0:00:00
  Downloading fsspec-2026.1.0-py3-none-any.whl (201 kB)
  201.8/201.8 kB 56.6 MB/s eta 0:00:00
  Downloading networkx-3.6.1-py3-none-any.whl (2.1 MB)
  2.1/2.1 MB 76.8 MB/s eta 0:00:00
  Downloading pillow-12.1.0-cp312-cp312-manylinux_2_27_x86_64_manylinux_2_28_x86_64.whl (7.0 MB)
  7.0/7.0 MB 76.1 MB/s eta 0:00:00
  Downloading requests-3.1.0-py3-none-any.whl (6.2 MB)
```

Realizamos una primera ejecución de prueba para verificar que el sistema arranca correctamente. El log muestra que se ha detectado la **NVIDIA GeForce RTX 4060 Ti** y que el servidor está escuchando en el puerto 8188.

Comandos usados: *python main.py*

```
(venv) izan@izan:~/IA/ComfyUI$ python main.py
Checkpoint files will always be loaded safely.
Total VRAM 15939 MB, total RAM 15824 MB
pytorch version: 2.9.1+cu128
Set vram state to: NORMAL_VRAM
Device: cuda:0 NVIDIA GeForce RTX 4060 Ti : cudaMallocAsync
Using async weight offloading with 2 streams
Enabled pinned memory 15032.0
working around nvidia conv3d memory bug.
WARNING: You need pytorch with cu130 or higher to use optimized CUDA operations.
Found comfy_kitchen backend triton: {'available': True, 'disabled': True, 'unavailable_reason': None, 'capabilities': ['apply_rope', 'apply_rope1', 'dequantize_nvfp4', 'dequantize_per_tensor_fp8', 'quantize_nvfp4', 'quantize_per_tensor_fp8']}
```

Accedemos con <http://127.0.0.1:8188>



Descarga de modelos Optimizados (Flux GGUF)

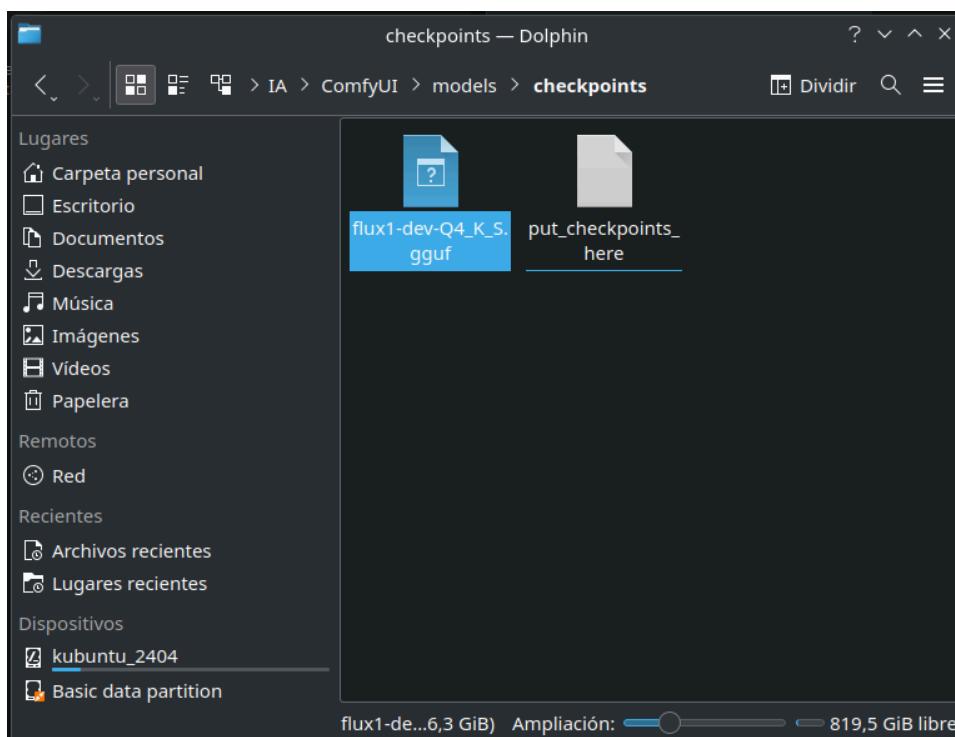
El modelo **Flux.1 Dev** original es demasiado grande para caber cómodamente en muchas configuraciones junto con otros procesos. Por ello, optamos por una versión **cuantizada (GGUF)**, que reduce el consumo de memoria sin apenas perder calidad.

Accedemos al repositorio de HuggingFace ([Link Repositorio](#)) de **city96/FLUX.1-dev-gguf** y descargamos la versión **Q4_K_S** (Cuantización a 4 bits). Este archivo es el "cerebro" que generará las imágenes.

flux1-dev-F16.gguf	Safe	23.8 GB		Upload flux1-dev-F16.gguf with huggingfa...	over 1 year ago
flux1-dev-Q2_K.gguf	Safe	4.03 GB		Upload flux1-dev-Q2_K.gguf with huggingf...	over 1 year ago
flux1-dev-Q3_K_S.gguf	Safe	5.23 GB		Upload flux1-dev-Q3_K_S.gguf with huggin...	over 1 year ago
flux1-dev-Q4_0.gguf	Safe	6.79 GB		Upload flux1-dev-Q4_0.gguf with huggin...	over 1 year ago
flux1-dev-Q4_1.gguf	Safe	7.53 GB		Upload flux1-dev-Q4_1.gguf with huggin...	over 1 year ago
flux1-dev-Q4_K_S.gguf	Safe	6.81 GB		Upload flux1-dev-Q4_K_S.gguf with huggin...	over 1 year ago
flux1-dev-Q5_0.gguf	Safe	8.27 GB		Upload flux1-dev-Q5_0.gguf with huggin...	over 1 year ago
flux1-dev-Q5_1.gguf	Safe	9.01 GB		Upload flux1-dev-Q5_1.gguf with huggin...	over 1 year ago
flux1-dev-Q5_K_S.gguf	Safe	8.29 GB		Upload flux1-dev-Q5_K_S.gguf with huggin...	over 1 year ago

Una vez descargado, movemos el archivo **flux1-dev-Q4_K_S.gguf** a la carpeta de modelos de ComfyUI para que el programa pueda localizarlo.

Ruta de destino: `~/IA/ComfyUI/models/checkpoints`



Instalación de soporte para GGUF (Custom nodes)

Por defecto, ComfyUI no lee archivos GGUF. Necesitamos instalar un "nodo personalizado" que le enseñe a hacerlo.

Con el entorno virtual activado, navegamos a la carpeta de nodos personalizados y clonamos el proyecto **ComfyUI-GGUF**. Posteriormente, instalamos sus requisitos específicos (`gguf`, `protobuf`, etc.).

comandos usados: `source venv/bin/activate`

```
izan@izan:~/IA/ComfyUI$ source venv/bin/activate
(venv) izan@izan:~/IA/ComfyUI$
```

comandos usados:

```
cd ~/IA/ComfyUI/custom_nodes
git clone https://github.com/city96/ComfyUI-GGUF.git
cd ..
pip install -r custom_nodes/ComfyUI-GGUF/requirements.txt
```

```
(venv) izan@izan:~/IA/ComfyUI/custom_nodes$ git clone https://github.com/city96/ComfyUI-GGUF.git
Clonando en 'ComfyUI-GGUF'...
remote: Enumerating objects: 814, done.
remote: Counting objects: 100% (508/508), done.
remote: Compressing objects: 100% (194/194), done.
remote: Total 814 (delta 457), reused 314 (delta 314), pack-reused 306 (from 2)
Recibiendo objetos: 100% (814/814), 190.00 KiB | 1.98 MiB/s, listo.
Resolviendo deltas: 100% (542/542), listo.
(venv) izan@izan:~/IA/ComfyUI/custom_nodes$
```

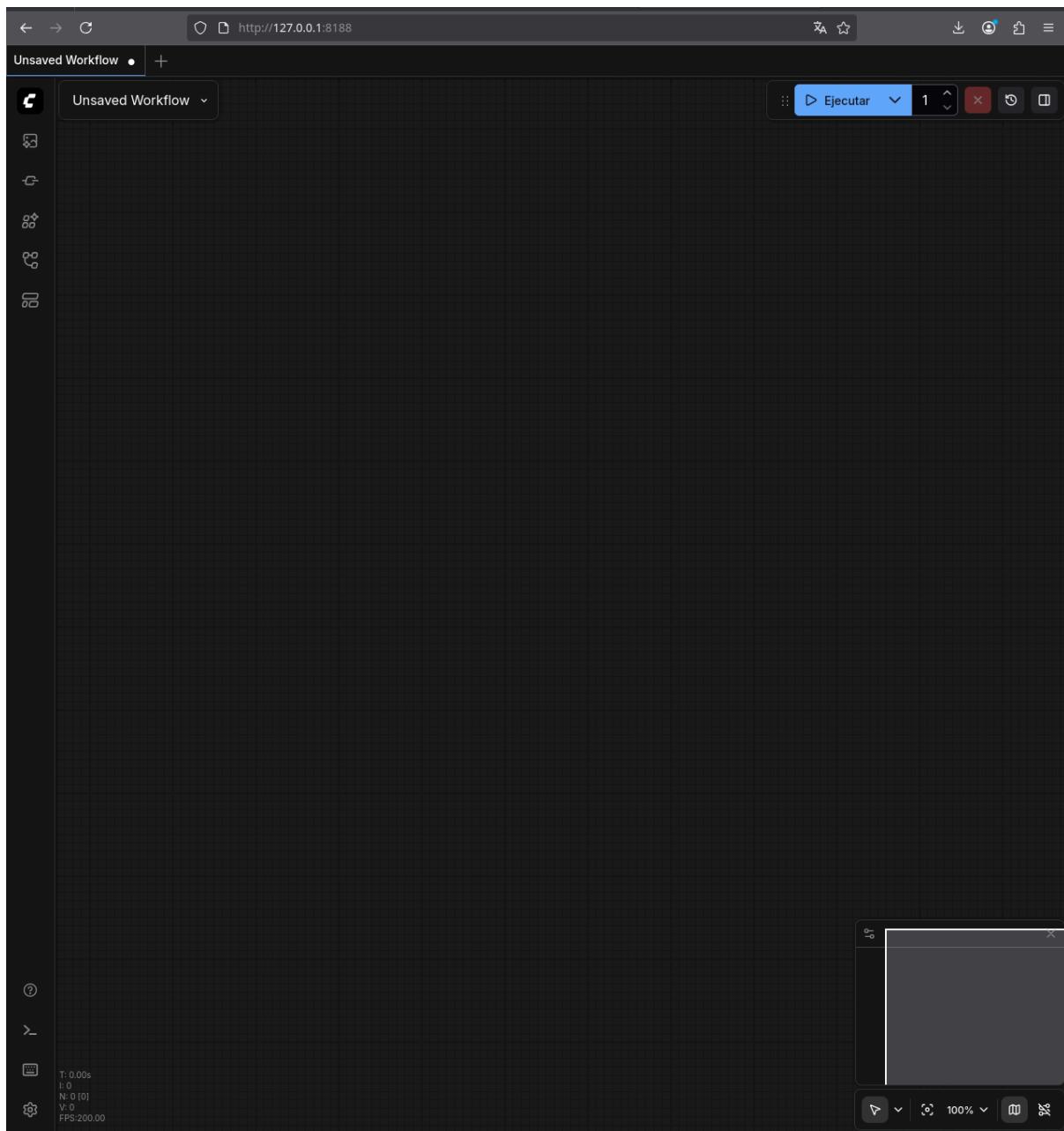
```
(venv) izan@izan:~/IA/ComfyUI$ pip install -r custom_nodes/ComfyUI-GGUF/requirements.txt
Collecting gguf>=0.13.0 (from -r custom_nodes/ComfyUI-GGUF/requirements.txt (line 2))
  Downloading gguf-0.17.1-py3-none-any.whl.metadata (4.3 kB)
Requirement already satisfied: sentencepiece in ./venv/lib/python3.12/site-packages (from -r custom_nodes/ComfyUI-GGUF/requirements.txt (line 4)) (0.2.1)
Collecting protobuf (from -r custom_nodes/ComfyUI-GGUF/requirements.txt (line 5))
  Downloading protobuf-6.33.4-cp39-abi3-manylinux2014_x86_64.whl.metadata (593 bytes)
Requirement already satisfied: numpy>=1.17 in ./venv/lib/python3.12/site-packages (from gguf>=0.13.0->-r custom_nodes/ComfyUI-GGUF/requirements.txt (line 2)) (2.4.1)
Requirement already satisfied: pyyaml>=5.1 in ./venv/lib/python3.12/site-packages (from gguf>=0.13.0->-r custom_nodes/ComfyUI-GGUF/requirements.txt (line 2)) (6.0.3)
Requirement already satisfied: tqdm>=4.27 in ./venv/lib/python3.12/site-packages (from gguf>=0.13.0->-r custom_nodes/ComfyUI-GGUF/requirements.txt (line 2)) (4.67.1)
Downloading gguf-0.17.1-py3-none-any.whl (96 kB)
  96.2/96.2 kB 4.3 MB/s eta 0:00:00
Downloading protobuf-6.33.4-cp39-abi3-manylinux2014_x86_64.whl (323 kB)
  323.3/323.3 kB 22.3 MB/s eta 0:00:00
Installing collected packages: protobuf, gguf
Successfully installed gguf-0.17.1 protobuf-6.33.4
```

Configuración ComfyUI

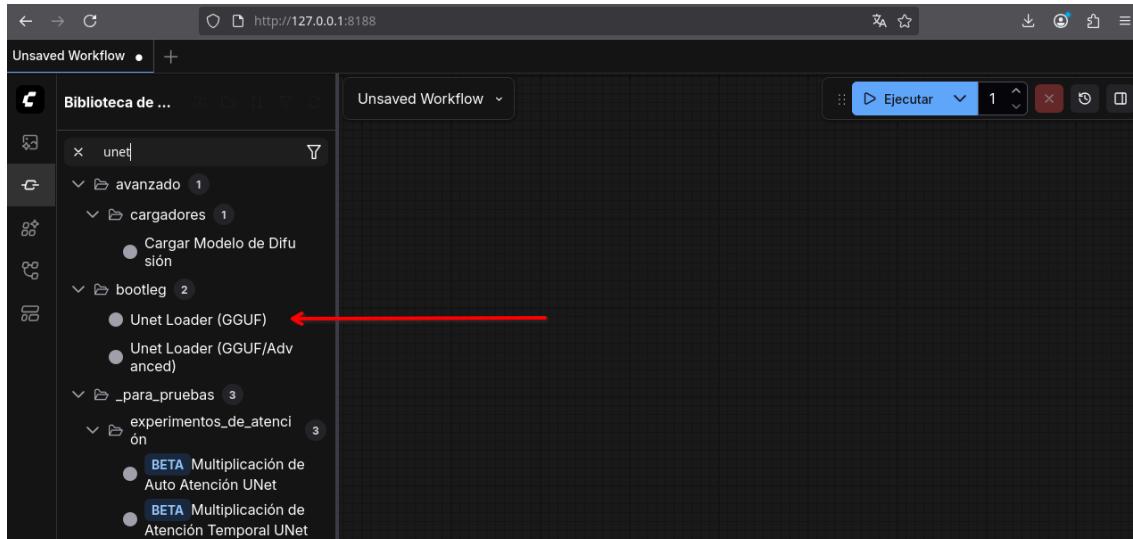
Instalación de Codificadores de Texto (CLIP/T5) y VAE

Accedemos a la url <http://127.0.0.1:8188> o <http://localhost:8188>

El modelo Flux es muy avanzado y requiere componentes separados para entender el texto (CLIP y T5) y para decodificar las imágenes (VAE). No vienen "dentro" del archivo principal, por lo que debemos descargarlos y colocarlos en sus carpetas correspondientes.

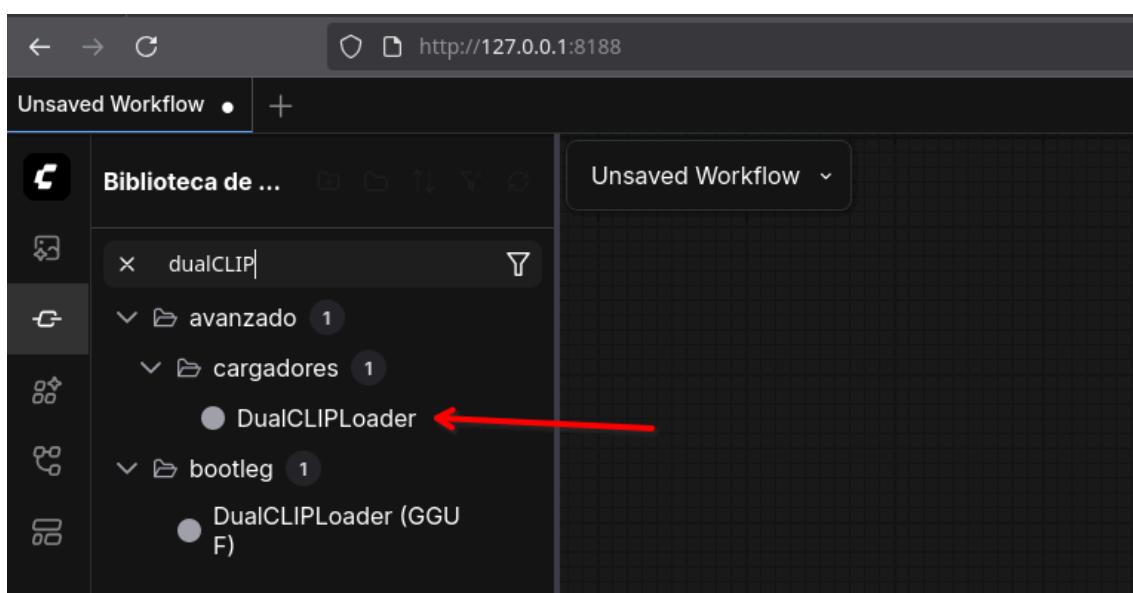


Unet Loader (GGUF): Seleccionamos `flux1-dev-Q4_K_S.gguf`



DualCLIPLoader: Cargamos `t5xxl_fp8...` y `clip_l...` en sus respectivas ranuras.

```
tzan@tzan:~/IA/ComfyUI/models/clip$ wget https://huggingface.co/comfyanonymous/flux_text_encoders/resolve/main/t5xxl_fp8_e4m3fn.safetensors
--2026-01-18 21:58:54-- https://huggingface.co/comfyanonymous/flux_text_encoders/resolve/main/t5xxl_fp8_e4m3fn.safetensors
Resolviendo huggingface.co (huggingface.co)... 18.67.250.31, 18.67.250.48, 18.67.250.73, ...
Conectando con huggingface.co (huggingface.co)[18.67.250.31]:443... conectado.
Petición HTTP enviada, esperando respuesta... 302 Found
Ubicación: https://cas-bridge.xethub.hf.co/xet-bridge-us/66ab96ab48b45330b7ef68a0/ca49b49b94ef106396a18f76600a52b6a07cd1322408c83337483ce5
6b803ba2X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Content-Sha256=UNSIGNED-PAYLOAD&X-Amz-Credential=ca%2F20260118%2Fus-east-1%2Faws4_request&X-Amz-Date=20260118T05854Z&X-Amz-Expires=3600&X-Amz-Signature=f09c513554c87cf40a90dbab581f7e3386e82bb1e56b9ff0cfa552ac5c1c2107&X-Amz-SignedHeaders=host&X-Xet-Cas-Uid=public&response-content-disposition=inline%3B+filename%3DUTF-8%27t5xxl_fp8_e4m3fn.safetensors%3B+filename%3B+fileext
```



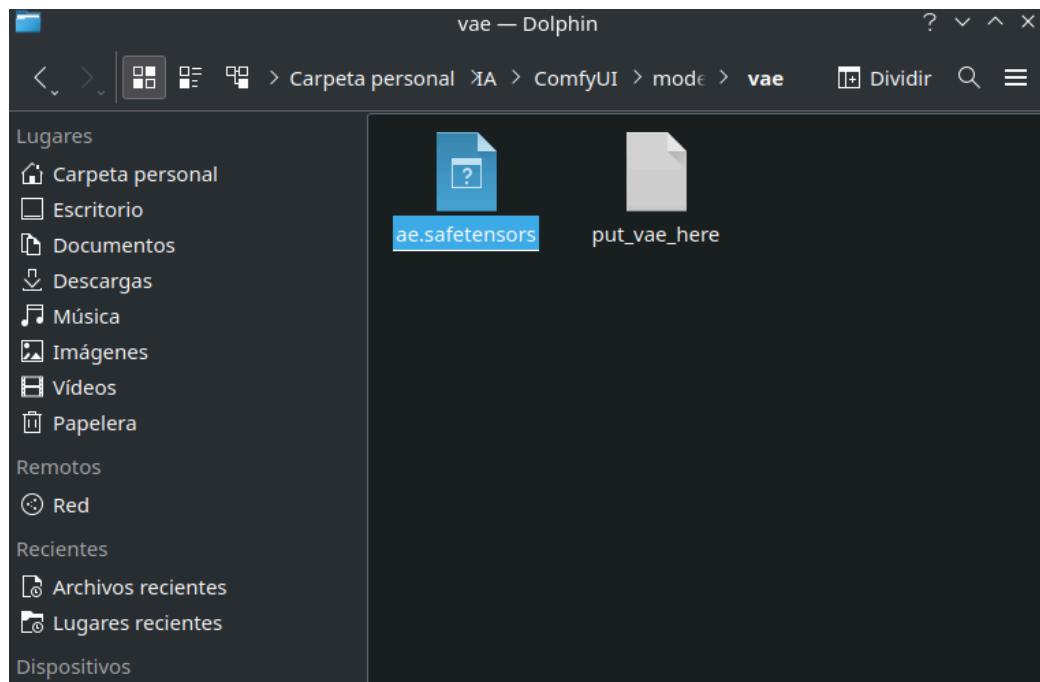
Movemos de directorio lo siguiente

```
(venv) izan@izan:~/IA/ComfyUI$ mv models/checkpoints/flux1-dev-Q4_K_S.gguf models/unet/
(venv) izan@izan:~/IA/ComfyUI$
```

Descargamos el siguiente vae

The screenshot shows the Hugging Face Model Card for 'FLUX.1-dev-ungated'. It displays the 'ae.safetensors' file, which is 335 MB in size and was uploaded 5 months ago. The file is stored with Xet and is too large to display. A red arrow points to the 'download' button.

Lo guardamos en esta ruta



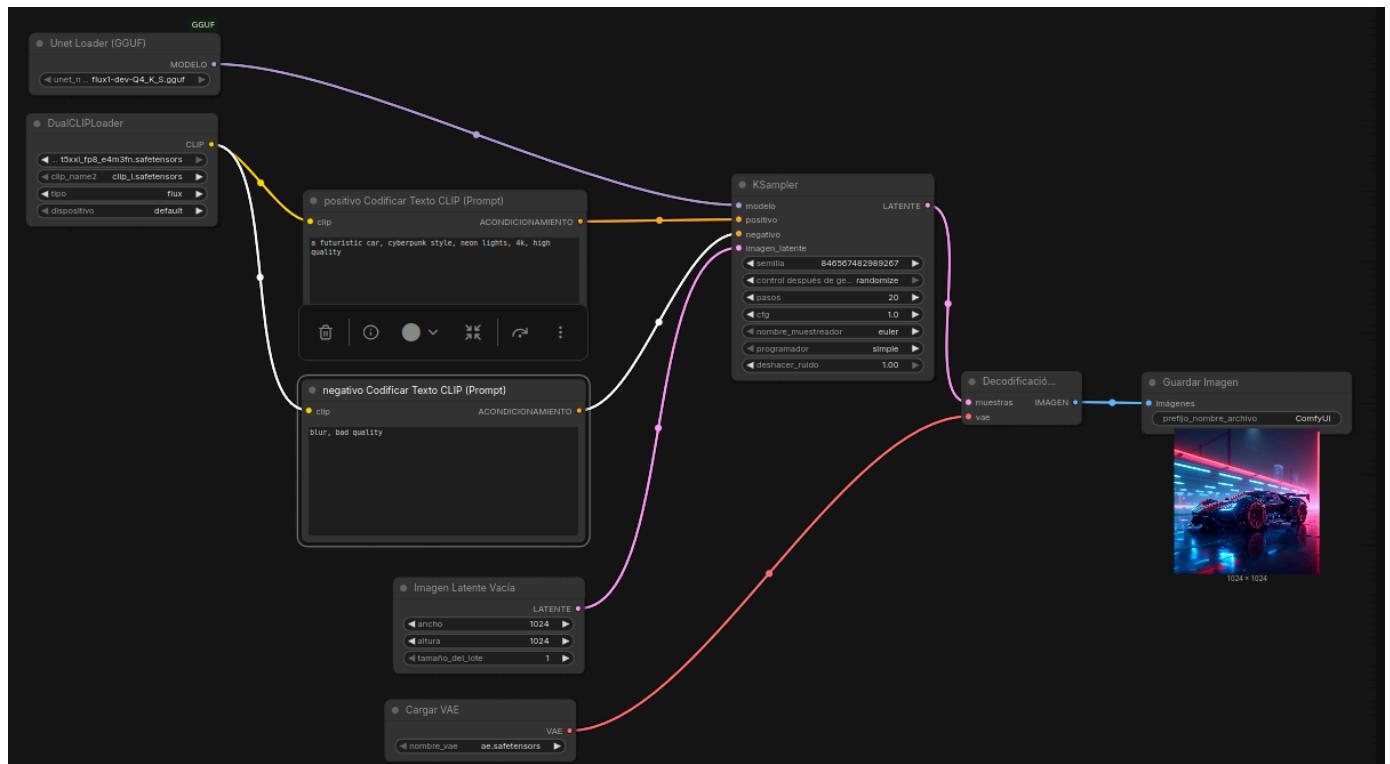
VAE Loader: Seleccionamos ae.safetensors.

```
tzan@tzan:~/IA/ComfyUI/models/clip$ wget https://huggingface.co/comfyanonymous/flux_text_encoders/resolve/main/clip_l.safetensors
--2026-01-18 21:57:43-- https://huggingface.co/comfyanonymous/flux_text_encoders/resolve/main/clip_l.safetensors
Resolviendo huggingface.co (huggingface.co)... 18.67.250.31, 18.67.250.48, 18.67.250.73, ...
Conectando con huggingface.co (huggingface.co)[18.67.250.31]:443... conectado.
Petición HTTP enviada, esperando respuesta... 302 Found
Ubicación: https://cas-bridge.xethub.hf.co/xet-bridge-us/66ab96ab48b45330b7ef68a0/645ba23540a1ce97d9c44332759ded7c2b5b8449914b8890eefed73d88
e6e02297X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Content-Sha256=UNSIGNED-PAYLOAD&X-Amz-Credential=ca8%2F20260118%2Fs-east-1%2Faws4_requ
est&X-Amz-Date=20260118T205645Z&X-Amz-Expires=36000&X-Amz-Signature=c19df621e8984643e30cda374ac1981f2d1ce77314d5d3497225f45fca6aff876X-Amz-S
ignedHeaders=host&X-Xet-Cas-Uid=public&response-content-disposition=inline%3B+filename%3DUIF-8%2F%2Fclip_l.safetensors%3B+filename%3D%22cl
ip_l.safetensors%22&X-Xet-Cas-Name=clip_l.safetensors
```

Ejecutamos el programa y accedemos a la url <http://127.0.0.1:8188> o <http://localhost:8188>

```
(venv) tzan@tzan:~/IA/ComfyUI$ python main.py
Checkpoint files will always be loaded safely.
Total VRAM 15939 MB, total RAM 15824 MB
pytorch version: 2.9.1+cu128
Set vram state to: NORMAL_VRAM
Device: cuda:0 NVIDIA GeForce RTX 4060 Ti : cudaMallocAsync
Using async weight offloading with 2 streams
Enabled pinned memory 15032.0
working around nvidia conv3d memory bug.
WARNING: You need pytorch with cu130 or higher to use optimized CUDA operations.
```

En el Workflow deberemos configurarlo para que quede de la siguiente forma, en la siguiente imagen::



El modelo entiende mejor el inglés así que si queremos ya podemos crear un prompt, tanto para el lado positivo como negativo de la caja Codificar Texto CLIP (Prompt). Si ejecutamos el Workflow con el botón superior derecho podremos nos generará una imagen y podremos seguir el proceso que lleva hacerlo (Tarda bastante tiempo, de media me tarda alrededor de 60seg x imagen)

Exposición de Ollama a la Red

Por defecto, Ollama solo escucha peticiones locales (`127.0.0.1`). Para que el contenedor Docker de Open WebUI pueda comunicarse con él, debemos permitir conexiones externas.

Editamos la configuración del servicio de sistema de Ollama para cambiar su dirección de escucha (Bind Address).

comandos usados: `sudo systemctl edit ollama.service`

```
izan@izan:~$ sudo systemctl edit ollama.service
[sudo] contraseña para izan:
izan@izan:~$ [REDACTED]
```

```
[Service] ←
# ExecStart=/usr/local/bin/ollama serve
# User=ollama
# Group=ollama
# Restart=always
# RestartSec=3
# Environment="PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/
Environment="OLLAMA_HOST=0.0.0.0" ←
[Install]
# WantedBy=default.target
```

Esto indica a Ollama que acepte conexiones desde cualquier interfaz de red, incluyendo la interfaz virtual de Docker.

comandos usados:

```
sudo systemctl daemon-reload
sudo systemctl restart ollama
```

```
izan@izan:~$ sudo systemctl daemon-reload
sudo systemctl restart ollama
izan@izan:~$ [REDACTED]
```

Integración Final en OpenWebUI

Finalmente, conectamos la interfaz de chat con el cerebro de IA.

Esto es necesario si queremos acceder a usar nuestra IA personal desde otro dispositivo (Incluso móvil) en nuestra misma red.

Accedemos a **Open WebUI** → **Panel de Administración** → **Ajustes** →

Conexiones. En el campo "Ollama Base URL", establecemos la dirección que apunta a nuestro host local.

URL de conexión: `http://host.docker.internal:11434` (*Alternativamente, si usamos red host: <http://127.0.0.1:11434>*)

Al guardar, verificamos la conexión. Ahora, desde el chat principal, podemos desplegar la lista de modelos y seleccionar **Mistral** o **Llama** para interactuar. El sistema está 100% operativo en local.

The screenshot shows the 'Connections' section of the OpenWebUI Admin Settings. The 'Ollama' subsection is selected. The 'Base URL' field is highlighted with a red arrow and contains the value 'http://127.0.0.1:11434'. Other connection types like 'API OpenAI' and 'API Ollama' are also listed with their respective URLs.

The screenshot shows the 'Connections' section of the OpenWebUI Admin Settings. The 'Ollama' subsection is selected. The 'Base URL' field is highlighted with a red arrow and contains the value 'http://host.docker.internal:11434'. Other connection types like 'API OpenAI' and 'API Ollama' are also listed with their respective URLs.