

# **AI Image Generation Pipeline with LangGraph and MCP**

**AIMUG**

Karim Lalani

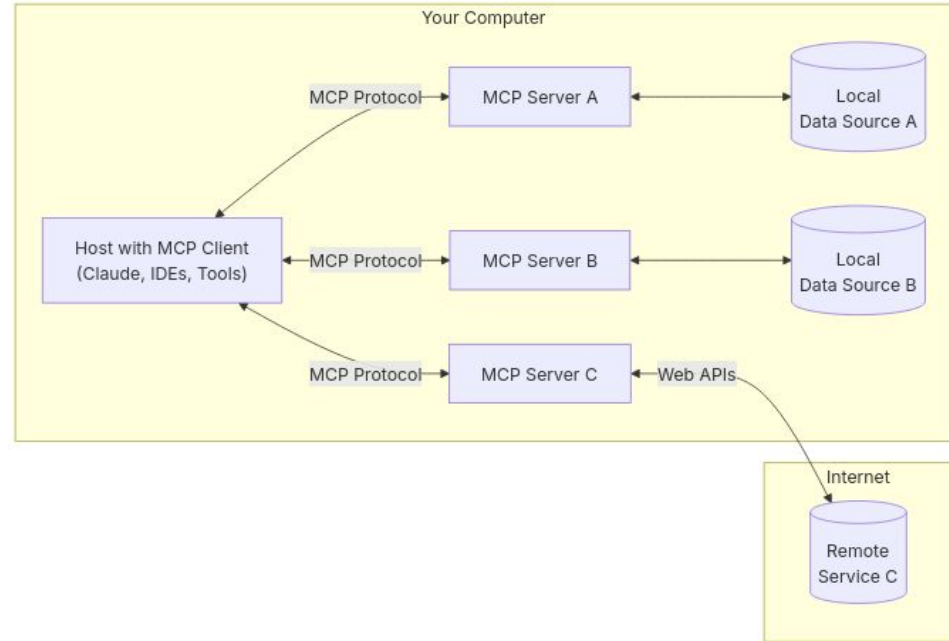
# About Me - Karim Lalani

- **Home:** Leander, TX
- **Work:** Software Engineer @ Office the Governor
- **Background:** Full Stack Engineer, Gen AI
- **FOSS:** Docker / Kubernetes, C#, Python, PHP, Rust
- **Using LangChain:** Experimentation, learning
- **Socials:**
  - LinkedIn <https://www.linkedin.com/in/-karim-lalani/>
  - Github <https://github.com/lalanikarim/>
  - Medium <https://medium.com/@klcoder>



# Model Context Protocol (MCP)

MCP is an open protocol that standardizes how applications provide context to LLMs. Think of MCP like a USB-C port for AI applications. Just as USB-C provides a standardized way to connect your devices to various peripherals and accessories, MCP provides a standardized way to connect AI models to different data sources and tools.



<https://modelcontextprotocol.io>

# Langgraph

## Graph API

At its core, LangGraph models agent workflows as graphs. You define the behavior of your agents using three key components:

1. [State](#)
2. [Nodes](#)
3. [Edges](#)

By composing [Nodes](#) and [Edges](#), you can create complex, looping workflows that evolve the [State](#) over time. The real power, though, comes from how LangGraph manages that [State](#).

## Functional API

The Functional API allows you to add LangGraph's key features -- persistence, memory, human-in-the-loop, and streaming — to your applications with minimal changes to your existing code.

# Langgraph Functional API vs. Graph API

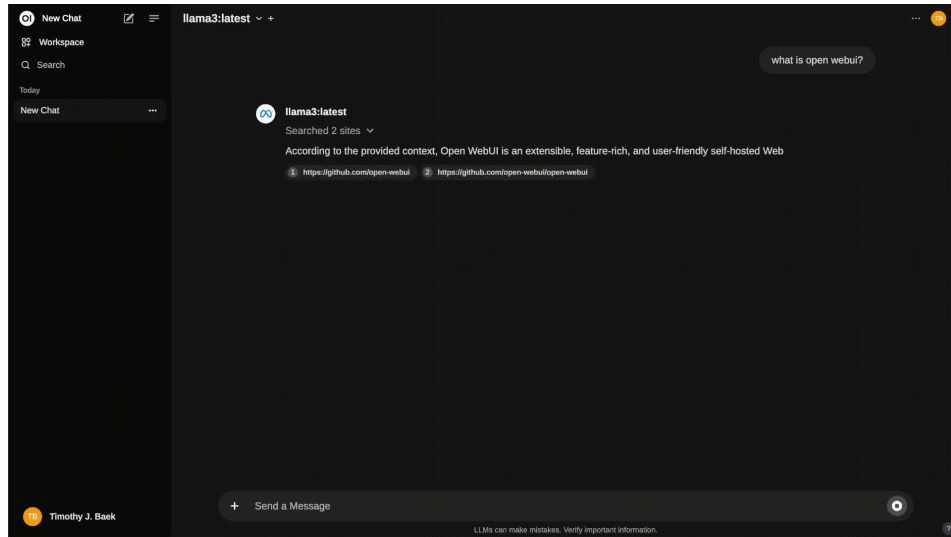
The **Functional API** and the [Graph APIs \(StateGraph\)](#) provide two different paradigms to create applications with LangGraph. Here are some key differences:

- **Control flow:** The Functional API does not require thinking about graph structure. You can use standard Python constructs to define workflows. This will usually trim the amount of code you need to write.
- **State management:** The **GraphAPI** requires declaring a [State](#) and may require defining [reducers](#) to manage updates to the graph state. [@entrypoint](#) and [@tasks](#) do not require explicit state management as their state is scoped to the function and is not shared across functions.
- **Checkpointing:** Both APIs generate and use checkpoints. In the **Graph API** a new checkpoint is generated after every [superstep](#). In the **Functional API**, when tasks are executed, their results are saved to an existing checkpoint associated with the given entrypoint instead of creating a new checkpoint.
- **Visualization:** The Graph API makes it easy to visualize the workflow as a graph which can be useful for debugging, understanding the workflow, and sharing with others. The Functional API does not support visualization as the graph is dynamically generated during runtime.

[https://langchain-ai.github.io/langgraph/concepts/functional\\_api/#functional-api-vs-graph-api](https://langchain-ai.github.io/langgraph/concepts/functional_api/#functional-api-vs-graph-api)

# Open WebUI

Open WebUI is an [extensible](#), feature-rich, and user-friendly self-hosted AI platform designed to operate entirely offline. It supports various LLM runners like Ollama and OpenAI-compatible APIs, with built-in inference engine for RAG, making it a powerful AI deployment solution.



<https://docs.openwebui.com/>

# Pipelines: UI-Agnostic OpenAI API Plugin Framework

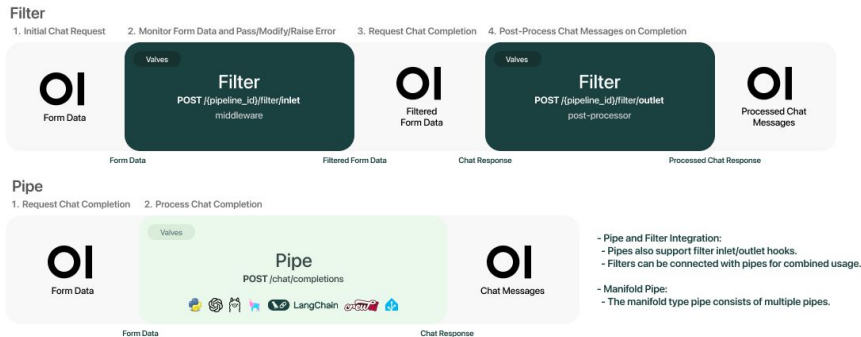
Pipelines bring modular, customizable workflows to any UI client supporting OpenAI API specs – and much more! Easily extend functionalities, integrate unique logic, and create dynamic workflows with just a few lines of code.



## Why Choose Pipelines?

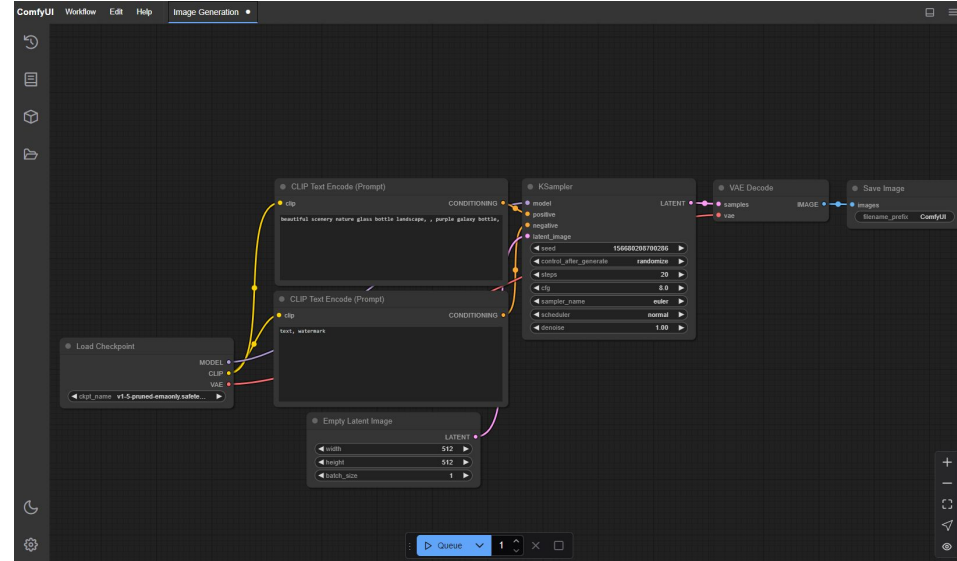
- **Limitless Possibilities:** Easily add custom logic and integrate Python libraries, from AI agents to home automation APIs.
- **Seamless Integration:** Compatible with any UI/client supporting OpenAI API specs. (Only pipe-type pipelines are supported; filter types require clients with Pipelines support.)
- **Custom Hooks:** Build and integrate custom pipelines.

<https://docs.openwebui.com/pipelines/>



# ComfyUI

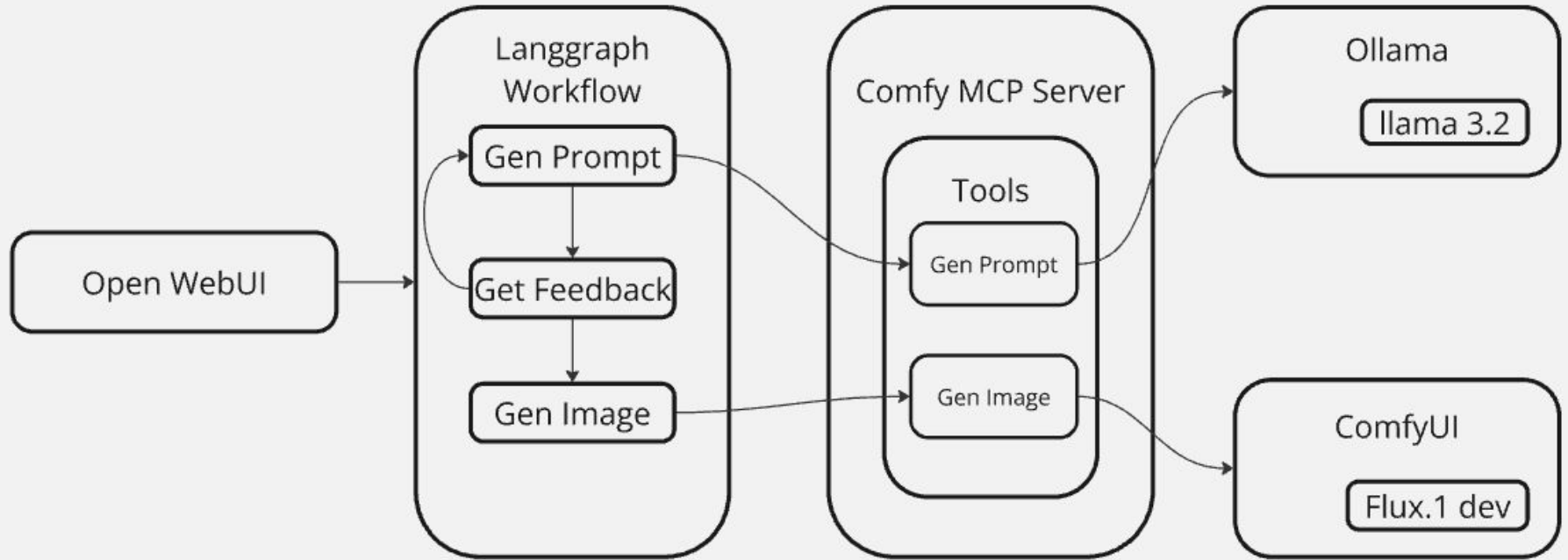
ComfyUI lets you design and execute advanced stable diffusion pipelines using a graph/nodes/flowchart based interface. Available on Windows, Linux, and macOS.



<https://github.com/comfyanonymous/ComfyUI>



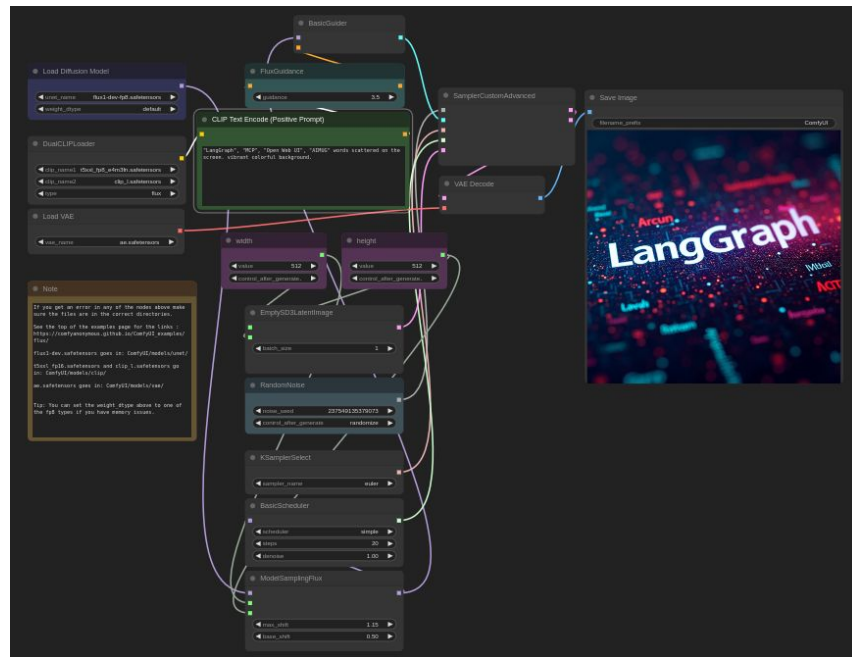
# Prompt and Image Generation Pipeline



# AI Image Generation Pipeline

## Image Generation Workflow

- Comfy UI API mode
- Flux.1 dev model
- workflow json



# Comfy MCP Server

## Overview

This script sets up a server using the FastMCP framework to generate images based on prompts using a specified workflow. It interacts with a remote Comfy server to submit prompts and retrieve generated images.

## Prerequisites

- [uv](#) package and project manager for Python.
- Workflow file exported from Comfy UI. This code includes a sample `Flux-Dev-ComfyUI-Workflow.json` which is only used here as reference. You will need to export from your workflow and set the environment variables accordingly.

## Usage

Comfy MCP Server can be launched by the following command: `uvx comfy-mcp-server`

<https://github.com/lalanikarim/comfy-mcp-server/>

# Comfy MCP Server

## Example Claude Desktop Config

```
{
  "mcpServers": {
    "Comfy MCP Server": {
      "command": "/path/to/uvx",
      "args": [
        "comfy-mcp-server"
      ],
      "env": {
        "COMFY_URL": "http://your-comfy-server-url:port",
        "COMFY_WORKFLOW_JSON_FILE": "/path/to/the/comfyui_workflow_export.json",
        "PROMPT_NODE_ID": "6",
        "OUTPUT_NODE_ID": "9",
        "OUTPUT_MODE": "file",
      }
    }
  }
}
```

# Comfy MCP Server

MCP Inspector v0.4.1

Transport Type

STDIO

Command

uv

Arguments

run --with mcp mcp run src/comf

> Environment Variables

▶ Connect

Connected

Error output from MCP server

[03/02/25 22:52:10] INFO  
Processing request of type  
server.py:432 ListToolsRequest

[03/02/25 22:52:31] INFO  
Processing request of type  
server.py:432 CallToolRequest

[03/02/25 22:52:33] INFO HTTP  
Request: POST \_client.py:1025  
http://aurora:11434/api/chat  
"HTTP/1.1 200 OK"

System

🔄

🗑️

🔍

Resources

Prompts

Tools

Ping

# Sampling

Roots

Tools

List Tools

Clear

generate\_prompt

Write an image generation prompt for a provided topic

generate\_image

Generate an image using ComfyUI workflow

generate\_prompt

Write an image generation prompt for a provided topic

topic

A cat holding a "AIMUG" sign

Run Tool

Tool Result: Success

"A sleek black cat is sitting on a velvet

{

"content": [

{

"type": "text",

"text": "\"A sleek black cat is sitting on a velvet couch, holding a small 'AIMUG' (Amateur Image Making User Group) sign in its mouth, with a mischievous grin on its face and a blurred cityscape background.\""

}

  ],

  "isError": false

}

Server Notifications

No notifications yet

# Comfy MCP Server

**MCP Inspector v0.4.1**

Transport Type

STDIO

Command

uv

Arguments

run --with mcp mcp run src/comf

> Environment Variables

▶ Connect

● Connected

Error output from MCP server

[03/02/25 23:00:08] INFO  
Processing request of type  
server.py:432 ListToolsRequest

[03/02/25 23:00:11] INFO  
Processing request of type  
server.py:432 CallToolRequest

INFO Warning: RuntimeWarning:  
coroutine server.py:495  
'ServerSession.send\_log\_message'  
was never awaited

System

🔍

🛠

📄

Resources Prompts Tools Ping # Sampling Roots

Tools

List Tools

Clear

generate\_prompt

Write an image generation prompt  
for a provided topic

generate\_image

Generate an image using ComfyUI  
workflow

generate\_image

Generate an image using ComfyUI workflow

prompt  
holding a small 'AIMUG' (Amateur Image Making User  
Group) sign in its mouth, with a mischievous grin on  
its face and a blurred cityscape background.

Run Tool

Tool Result: Success

lename=ComfyUI\_01733\_.png&subfolder=&type=c

Response:

```
{
  "content": [
    {
      "type": "text",
      "text": "http://aurora:8188/view?
filename=ComfyUI_01733_.png&subfolder=&ty
pe=output"
    }
  ],
  "isError": false
}
```

Server Notifications

No notifications yet

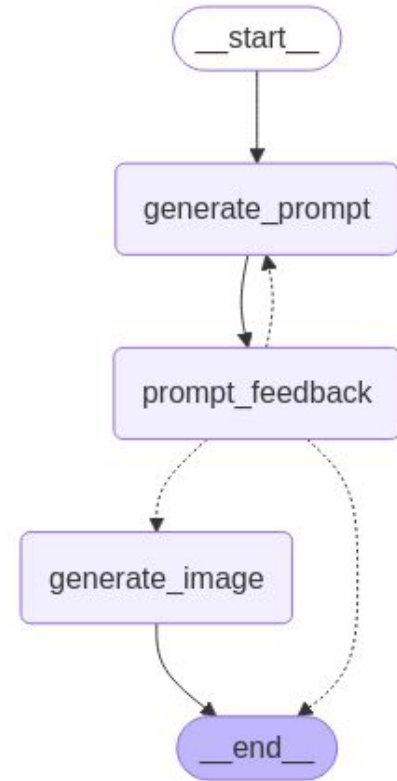
# Comfy MCP Server



# Langgraph + Comfy MCP Server (Graph API)

- Checkpoints stored in “checkpoints.sqlite” db
- Sessions persist across executions
- Generate prompt for topic by running  
uv run graph.py <thread\_id> --topic <topic>
- Provide feedback for prompt by running  
uv run graph.py <thread\_id> --feedback “y/n”

<https://github.com/lalanikarim/langgraph-mcp-pipeline/blob/main/graph.py>





# Langgraph + Comfy MCP Server (Functional API)

- Launch by running  
uv run app.py --topic <topic>
- Provide feedback on generated prompts “y/n”

<https://github.com/lalanikarim/langgraph-mcp-pipeline/blob/main/app.py>

```
output.sh
1 > uv run app.py --topic "A cat wearing a hat with 'AIMUG' on it"
2 Reading inline script metadata from `app.py`
3 thread_id='1934317a-abf4-40f3-a1d2-97b51ac79d16'
4 Step: generate_prompt
5 Step: __interrupt__
6 Prompt: "Generate an illustration of a curious cat sitting on a
7   velvet cushion, wearing a red fedora hat with white embroidery
8   reading 'AIMUG', its whiskers twitching with excitement."
9 Do you like this prompt (y/n)? n
10 Step: generate_prompt
11 Step: __interrupt__
12 Prompt: "Generate an illustration of a sleek black cat lounging
13   in a cozy armchair, wearing a bright red fedora hat adorned
14   with golden letters that spell out 'AIMUG' in bold, cursive
15   script."
16 Do you like this prompt (y/n)? y
17 Step: generate_image
18 Step: workflow
19 Image URL: http://aurora:8188/view?filename=ComfyUI_01735_.png&
20   subfolder=&type=output
21
```

# Comfy MCP Server



# Langgraph + Comfy MCP Server + Open WebUI Pipelines

- Simple Open WebUI pipeline integration
- Uses Langgraph Graph API
- Human in the Loop to collect user feedback on generated prompts
- Chat interface to interact with the workflow

<https://github.com/lalanikarim/langgraph-mcp-pipeline/blob/main/ai-image-gen-pipeline.py>

# Langgraph + Comfy MCP Server + Open WebUI Pipelines

Comfy MCP Langgraph HIL Pipeline ▾ +

...  

A cat wearing a hat with 'AIMUG' on it

**OI** Comfy MCP Langgraph HIL Pipeline

Prompt: "Illustrate a sophisticated, cartoon-style cat wearing a trendy black fedora hat adorned with silver letters 'AIMUG', sitting elegantly on a minimalist, modern desk."

Action: Do you like this prompt? (y/n)

n

**OI** Comfy MCP Langgraph HIL Pipeline

Prompt: "Create an illustration of a sleek black cat wearing a vibrant, colorful fedora hat with bold, white lettering that reads 'AIMUG', as if the cat is posing for a mysterious and playful mission briefing."

Action: Do you like this prompt? (y/n)

y



Send a Message



Web Search



Image



Code Interpreter



# Langgraph + Comfy MCP Server + Open WebUI Pipelines

Comfy MCP Langgraph HIL Pipeline ▾ +



OI Comfy MCP Langgraph HIL Pipeline



Send a Message



Web Search



Image



Code Interpreter



Questions?

Thank You