AI Image Generation Pipeline with LangGraph and MCP

AIMUG

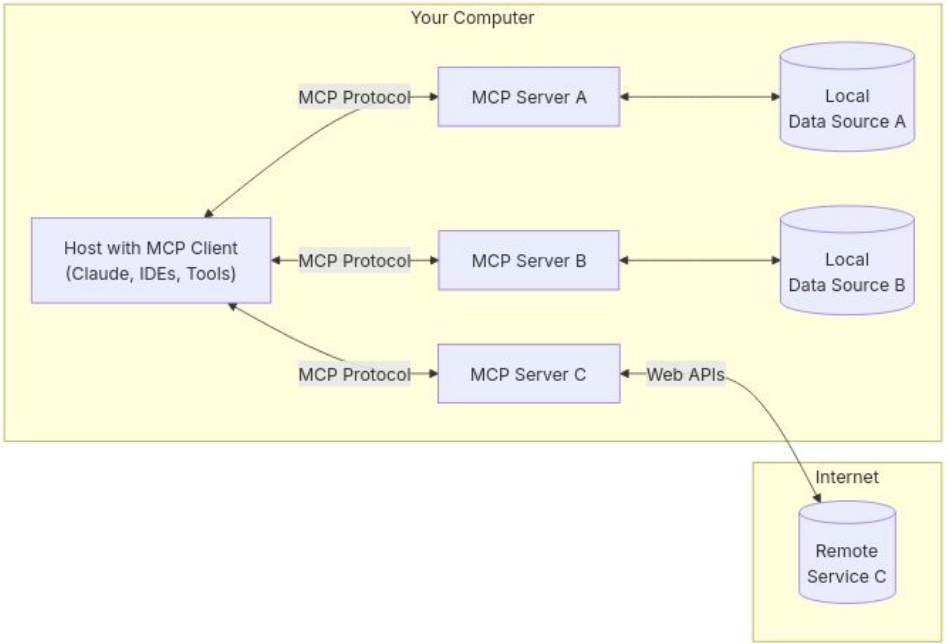
Karim Lalani

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

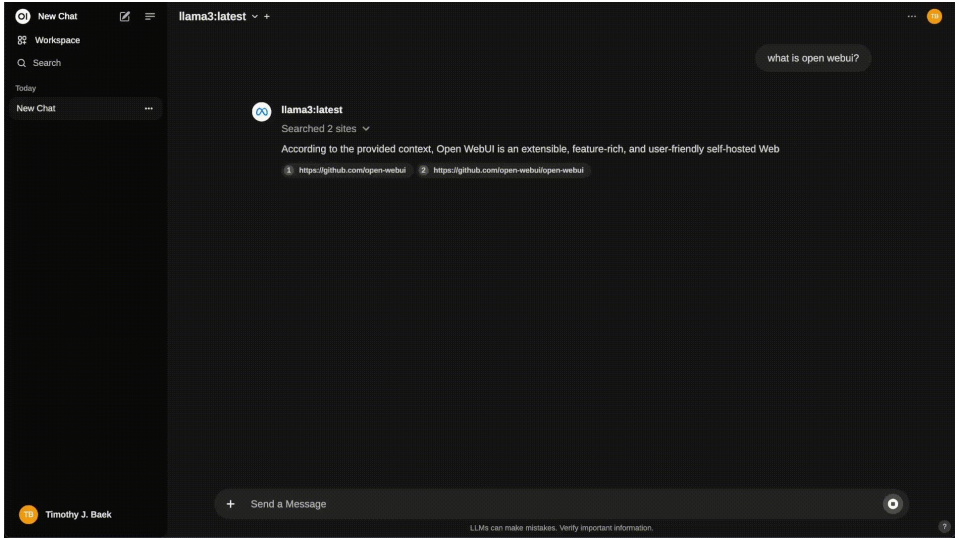
The **Functional API** and the Graph APIs (StateGraph) provide two different paradigms to create applications with LangGraph. Here are some key differences:

- **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

Open WebUI

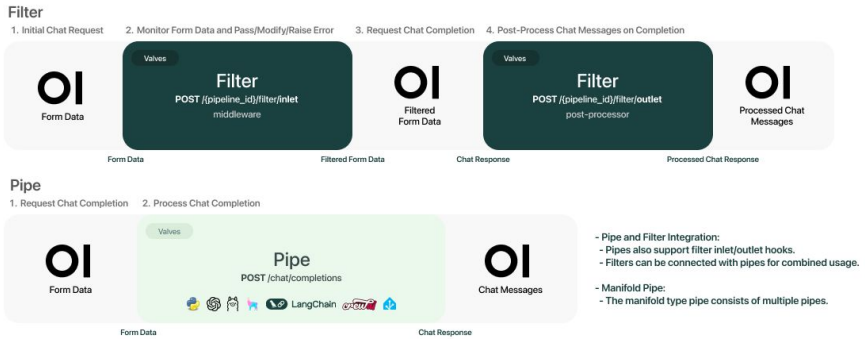
Open WebUI is an extensible, feature-rich, and user-friendly self-hosted AI platform designed to operate entirely offline. It supports various LLM runners like Ollama and OpenAI-compatible APIs, with built-in inference engine for RAG, making it a powerful AI deployment solution.



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.



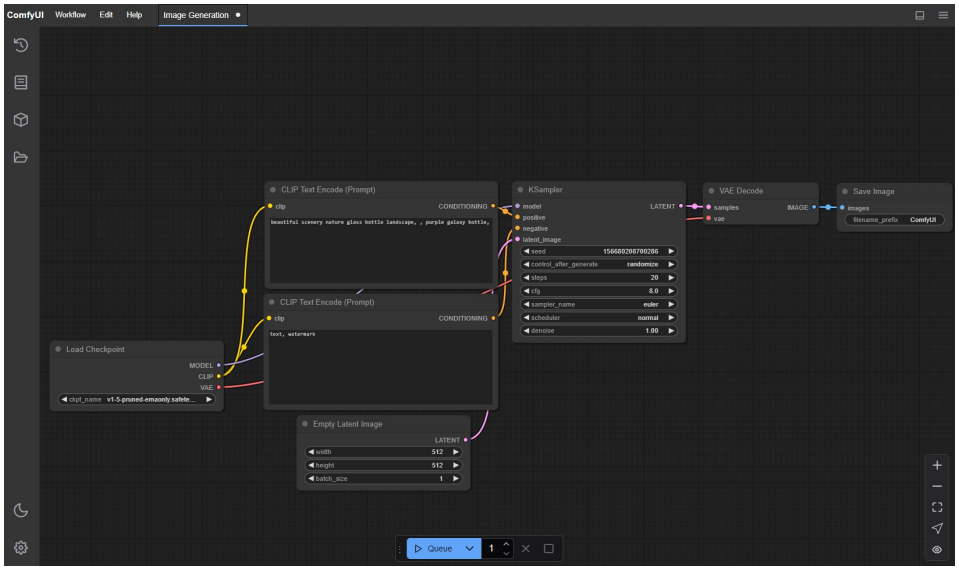
🚀 Why Choose Pipelines?

- **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/

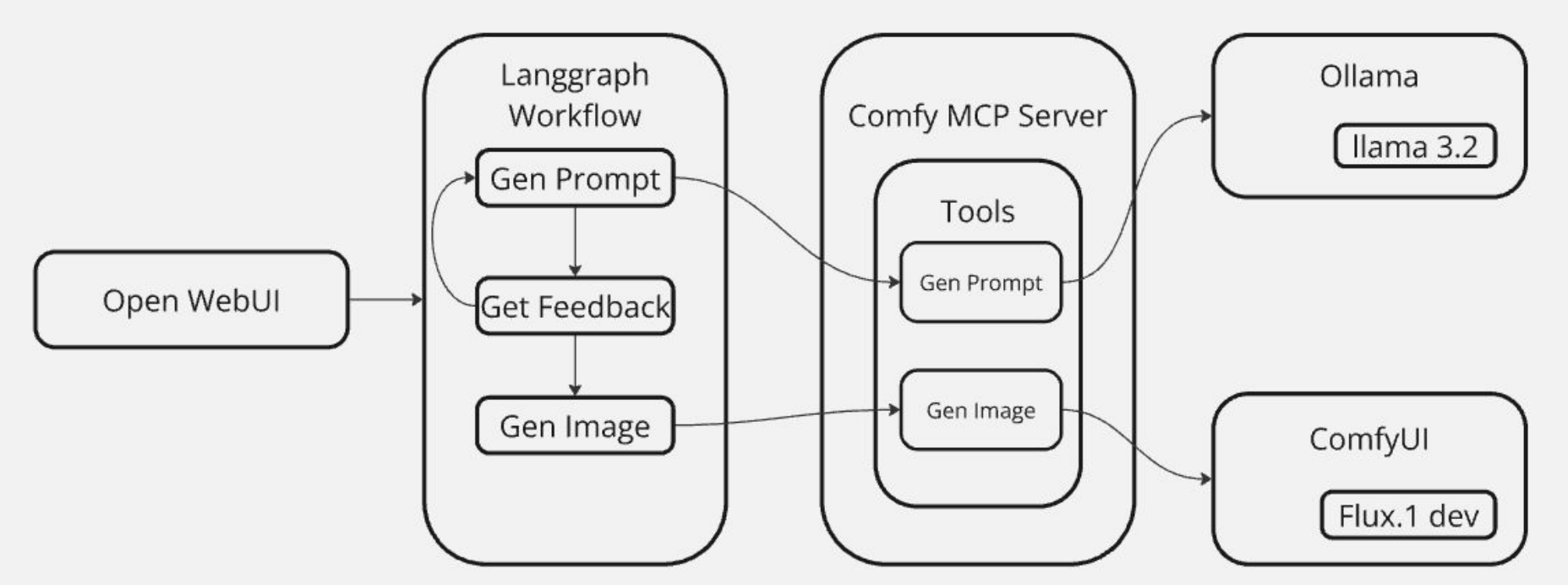
ComfyUI

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

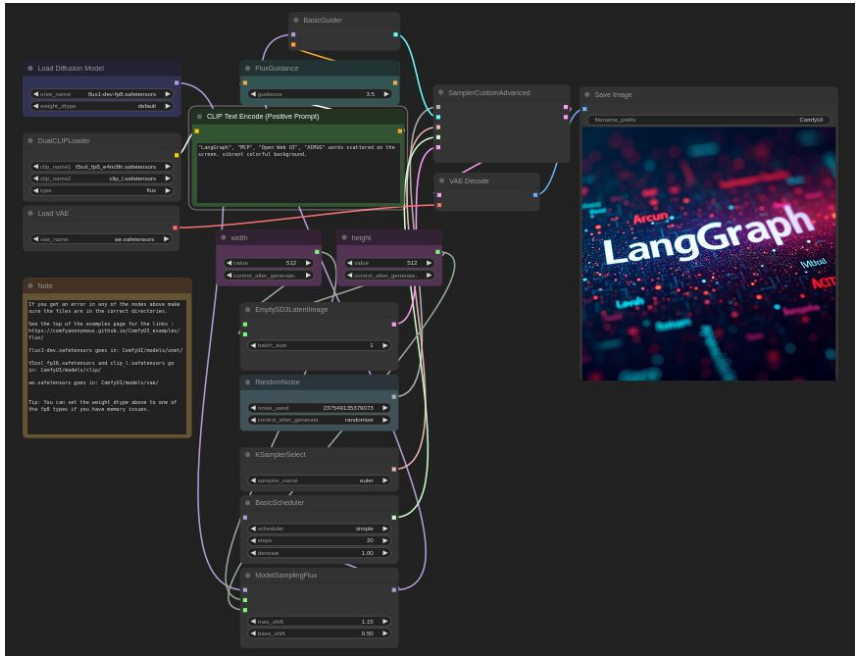
Prompt and Image Generation Pipeline



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

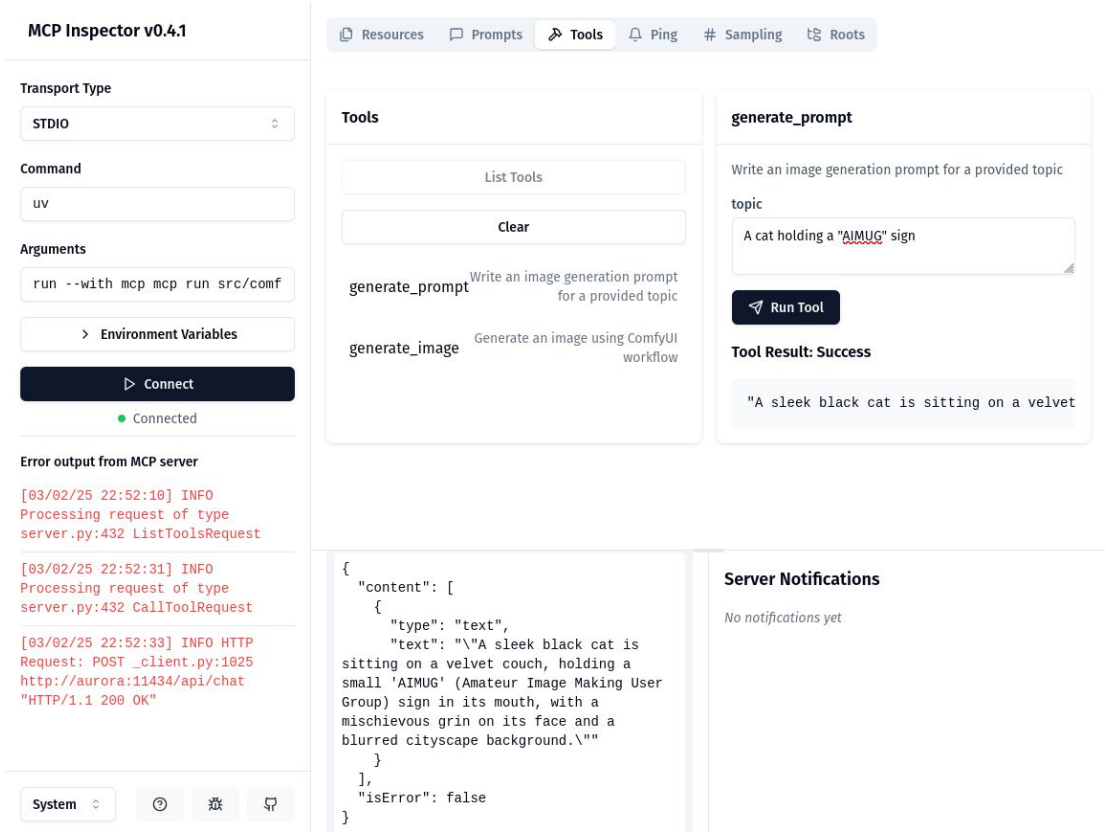
https://github.com/lalanikarim/comfy-mcp-server/

Comfy MCP Server

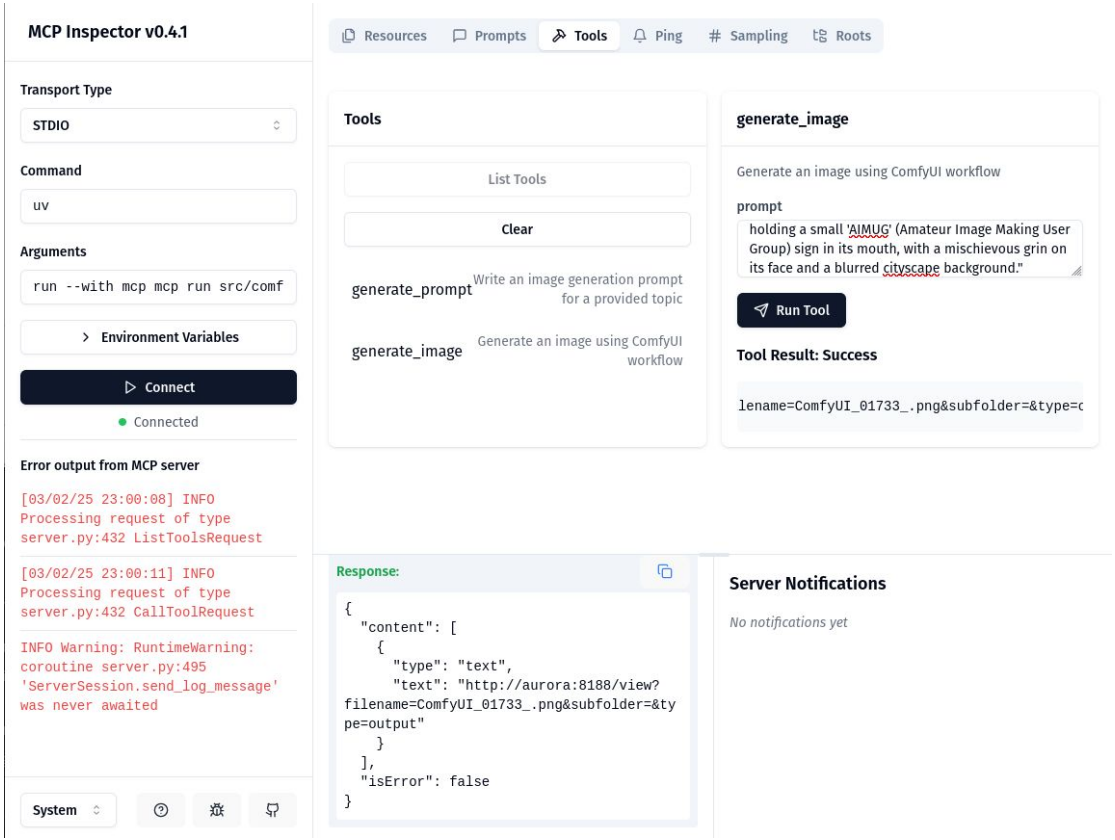
Example Claude Desktop Config

```
{
  "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



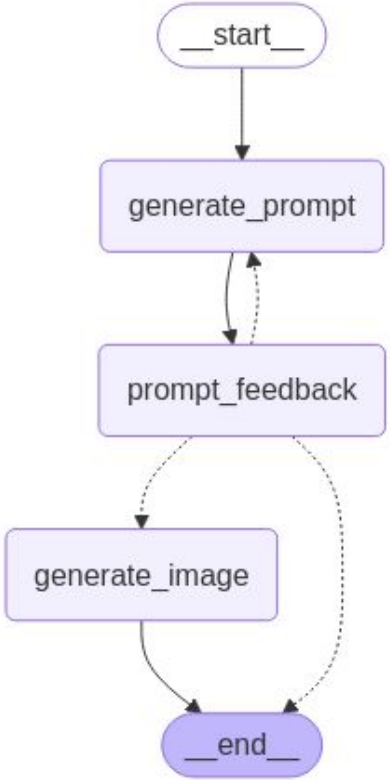
Comfy MCP Server



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

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

```

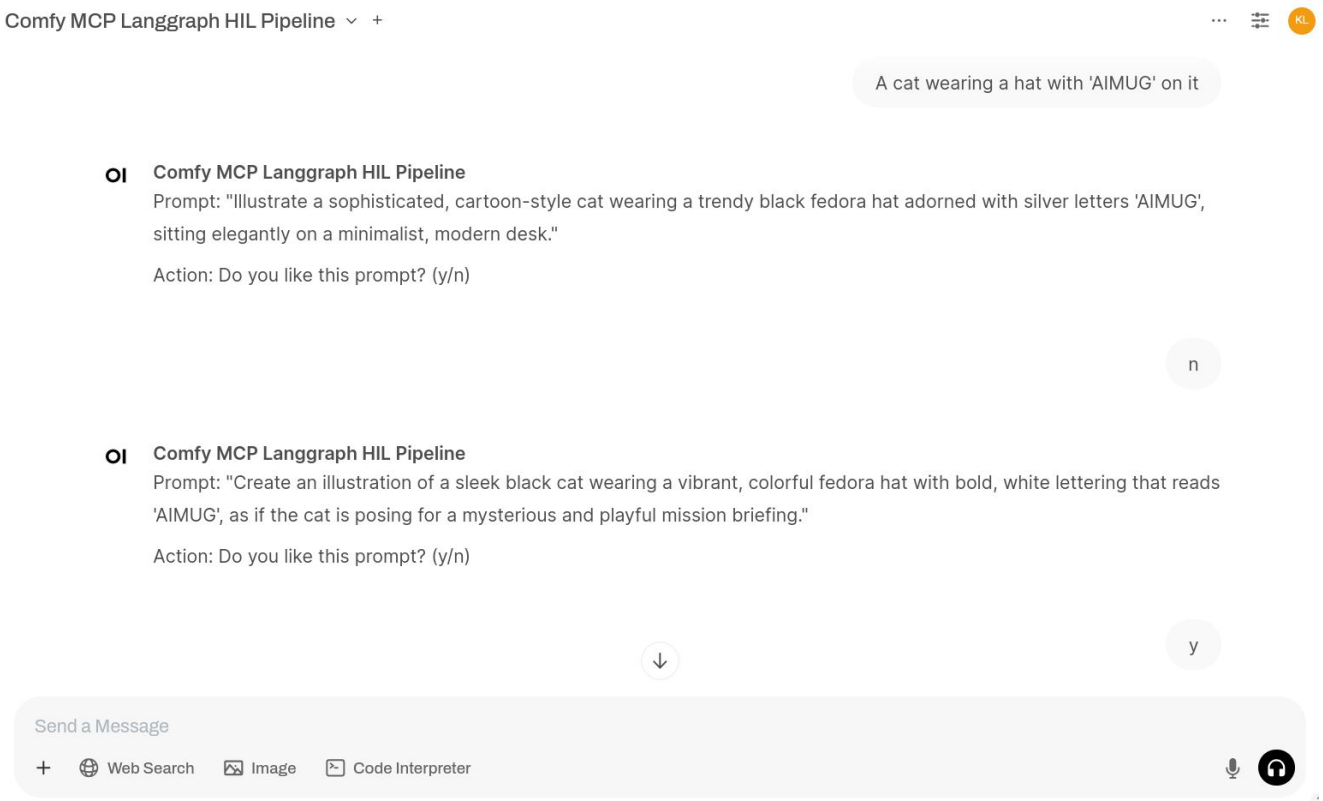
Comfy MCP Server

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



Langgraph + Comfy MCP Server + Open WebUI Pipelines

Questions?

Thank You