Embracing AI-Assisted Development with Jupyter AI and Nix Flakes

Technical Journal Entry Begins

No matter how much I want to keep things in Python Jupyter Notebooks to avoid Web development frontend work, all roads seem to lead to webdev. You can't just ask everyone to go into a Notebook and change values or interact with ipywidgets or use it like a dashboard. People expect easy self-serve web apps. What's more, when you do go into the code, people expect AI coding assistants.

The Rise of AI Coding Assistants

As of this writing, **Cursor AI** is all the craze. But if I didn't like VSCode before because of the implied Microsoft vendor lock-in and dependency it induces, then I really don't like Cursor AI because it's a fork of VSCode that is going to ask you for that money to use it right up front. Honestly, I do like Cursor AI. I like the way it can learn all your documents and have full context of your Github repo. But the official **Project Jupyter** who everyone is copying anyway is not to be outdone, and already has an official **Jupyter AI plugin** that works well with JupyterLab.

The Power of Jupyter AI with Nix Flakes

As of today's work, I can confirm that Jupyter AI works well with JupyterLab, even from a Nix Flake on NixOS or Apple Macbooks. What's more, it's a cinch to make it work with **Ollama** using the **Llama3.1 model**, making it a fully free and open source software solution to AI code assisted Notebooks. And you can do so with the deterministic reliability of the nix package management system. *The game has changed.*

A Balanced Approach to Tools

It might sound like I'm advocating staying away from the proprietary power tools that make you pay. I'm not. If they work for you, great. My priorities are different. I'm using the 80/20-rule solutions from the FOSS-world and making it very reliably share-able, without HAVING to be on the cloud or deployed to a server. It can run local on your own machine, yet it runs much like it would run on the cloud.

The Nix/NixOS Workflow: Zero-Cost Local Cloud Development

I would describe the Nix/NixOS workflow as a **zero-cost local cloud development methodology**. Although it runs on your local machine, most of the software resources required for development aren't provided by your host machine itself. Instead, your local machine serves as a host for binaries, libraries, and a shell environment, all managed and controlled by the Nix

package management system.

Benefits of the Nix Approach

- **Easy Deployment**: Your development environment can be easily deployed to the cloud or shared with coworkers.
- **Seamless Collaboration**: Facilitates smooth teamwork within a development team.
- **Standardized Environment**: Ensures consistent capabilities across different hardware without being tied to any specific cloud ecosystem or proprietary platform.

My Current Nix Flake Project

The Nix flake I'm working on sets up a robust development environment that includes:

1. **JupyterLab Notebook**: Ideal for quickly mocking up ideas with its dashboard-like UI.
2. **FastHTML Framework**: A new web development starting point based on the FastHTML framework, released in March 2024.

FastHTML Framework Overview

FastHTML, developed by Jeremy Howard—the creator of **nbdev** and **FastAI** (not FastAPI)—is a modern web framework built on several foundational technologies:

1. **Python**: The primary programming language.
2. **ASGI (Asynchronous Server Gateway Interface)**
3. **HTMX**: A lightweight JavaScript library for dynamic interactions.
4. **Starlette**: An ASGI framework for asynchronous capabilities.
5. **Uvicorn**: A high-performance ASGI server.

The Updated Nix Flake

```
{
  description = "Pipulate Development Environment with python-fasthtml and jupyter_ai";

  inputs = {
    nixpkgs.url = "github:NixOS/nixpkgs/nixos-24.05";
  };

  outputs = { self, nixpkgs }:
    let
      systems = [ "x86_64-linux" "aarch64-darwin" ];
      forAllSystems = f: builtins.listToAttrs (map (system: { name = system; value = f system; })
systems);
      localConfig = if builtins.pathExists ./local.nix then import ./local.nix else {};
      cudaSupport = if localConfig ? cudaSupport then localConfig.cudaSupport else false;
    in
    {
      devShells = forAllSystems (system: {
        default = let
```

```
  pkgs = import nixpkgs { inherit system; config.allowUnfree = true; };
  lib = pkgs.lib;
  cudaPackages = lib.optionals (cudaSupport && system == "x86_64-linux") (with pkgs; [
    pkgs.cudatoolkit
    pkgs.cudnn
    (pkgs.ollama.override { acceleration = "cuda"; })
  ]);
  ps = pkgs.python311Packages;
  pytorchPackage = if cudaSupport && system == "x86_64-linux" then
    ps.pytorch.override { cudaSupport = true; }
  else if system == "aarch64-darwin" then
    ps.pytorch-bin
  else
    ps.pytorch;
  pythonPackages = pkgs.python311.withPackages (ps: [
    ps.jupyterlab
    ps.pandas
    ps.requests
    ps.sqlitedict
    ps.numpy
    ps.matplotlib
    ps.nbdev
    ps.fastapi
    ps.simplenote
    pytorchPackage
    ps.pip
  ]);
  devTools = with pkgs; [
    git
  ];
in pkgs.mkShell {
  buildInputs = devTools ++ [ pythonPackages ] ++ cudaPackages ++ [
    pkgs.stdenv.cc.cc.lib
  ];

  shellHook = ''
    export NIXPKGS_ALLOW_UNFREE=1
    export LD_LIBRARY_PATH=${pkgs.stdenv.cc.cc.lib}/lib:$LD_LIBRARY_PATH
    echo "Welcome to the Pipulate development environment on ${system}!"
    ${if cudaSupport then "echo 'CUDA support enabled.'" else ""}

    if [ ! -d .venv ]; then
      ${pythonPackages.python.interpreter} -m venv .venv
    fi

    source .venv/bin/activate

    # Function to check and install a package
```

```
        check_and_install() {
          package=$1
          if ! python -c "import $package" 2>/dev/null; then
            echo "Installing $package..."
            pip install $package > /dev/null 2>&1
            if [ $? -eq 0 ]; then
              echo "$package installed successfully."
            else
              echo "Failed to install $package. Please check your internet connection and try
again."
            fi
          else
            echo "$package is already installed."
          fi
        }

        # Check and install nbdev
        check_and_install nbdev

        # Check and install python-fasthtml
        check_and_install python-fasthtml

        # Check and install jupyter_ai
        check_and_install jupyter_ai

        echo "Development environment is ready!"

        # === Start Jupyter Lab and Python Server ===

        echo "Starting Jupyter Lab..."
        jupyter lab &
        JUPYTER_PID=$!

        echo "Starting Python server..."
        python server.py &
        SERVER_PID=$!

        cleanup() {
          echo "Stopping Jupyter Lab..."
          kill $JUPYTER_PID

          echo "Stopping Python server..."
          kill $SERVER_PID
        }

        trap cleanup EXIT
      ";
    };
```

```
    });
  };
}
```

Key Features of the Updated Flake

- Support for various hardware platforms (x86 Linux, Windows under WSL, Mac)
- NVidia GPU acceleration using CUDA when available
- Full Jupyter Lab Data Science stack, including PyTorch
- Local LLM integrations via Ollama (good for Jupyter AI)
- FastHTML / HTML Web stack for Pythonic Web development
- Nbdev for creating formal Python packages from Jupyter Notebooks

Conclusion: A Versatile Development Environment

This Nix Flake provides a baseline set of capabilities necessary to bring a formal collaborative development environment to a group with varying skills:

- **Git Users**: Can clone the Pipulate Nix Flake repo and run nix develop.
- **Google Colab Users**: Can migrate work to a Colab instance when possible.
- **Web App Needs**: Potential for development under FastHTML.
- **AI Code Assistance**: JupyterLab distributed with AI of choice built-in (just provide the API key).

By embracing this approach, we create a flexible, powerful, and accessible development environment that caters to various needs while leveraging the benefits of open-source tools and AI assistance.