ParManus Setup Guide for Windows with VS Code

This guide provides step-by-step instructions for setting up ParManus on Windows using VS Code as your development environment.

Prerequisites

1. **Windows 10/11** (64-bit)
2. **Python 3.12** - Download and install from python.org
3. **Git** - Download and install from git-scm.com
4. **Visual Studio Code** - Download and install from code.visualstudio.com
5. **Visual Studio Build Tools** - Required for some Python packages
6. Download from visualstudio.microsoft.com
7. During installation, select "Desktop development with C++"

VS Code Extensions Setup

1. Open VS Code and install these recommended extensions:
2. **Python** (Microsoft) - For Python language support
3. **Pylance** (Microsoft) - For enhanced Python language features
4. **Python Debugger** (Microsoft) - For debugging Python code
5. **TOML Language Support** - For editing config.toml files

To install extensions, press `Ctrl+Shift+X`, search for each extension, and click "Install".

Basic Setup (Without GPU Support)

1. Clone the Repository

Open VS Code and:

1. Press `Ctrl+Shift+P` to open the command palette
2. Type "Git: Clone" and select it
3. Enter the repository URL: `https://github.com/mrarejimmyz/parmanus.git`
4. Select a folder location on your computer (use a short path like `C:\Projects\parmanus`)

5. When prompted, click "Open" to open the cloned repository

2. Set Up Python Environment in VS Code

1. Open the VS Code integrated terminal:

2. Press `` Ctrl+` `` or go to View > Terminal

3. Choose your preferred method:

Method 1: Using venv (Standard Python)

```
# Create virtual environment
python -m venv .venv

# Activate virtual environment
.\.venv\Scripts\activate

# Install dependencies
pip install -r requirements.txt

# Install Playwright browsers
playwright install chromium --with-deps
```

Method 2: Using uv (Recommended for faster installation)

```
# Install uv
curl -LsSf https://astral.sh/uv/install.ps1 | powershell

# Create virtual environment
uv venv --python 3.12

# Activate virtual environment
.\.venv\Scripts\activate

# Install dependencies
uv pip install -r requirements.txt

# Install Playwright browsers
playwright install chromium --with-deps
```

1. Select the Python interpreter in VS Code:
2. Press `Ctrl+Shift+P`
3. Type "Python: Select Interpreter"
4. Choose the interpreter from your `.venv` environment

3. Create Configuration File

1. In VS Code's Explorer panel (Ctrl+Shift+E), navigate to the `config` folder
2. Right-click on the folder and select "New File"
3. Name it `config.toml`
4. Paste the following content:

```toml
 # Global LLM configuration
[llm]
model = "gpt-4o"
base_url = "https://api.openai.com/v1"
api_key = "sk-..."  # Replace with your actual API key
max_tokens = 4096
temperature = 0.0

# Optional configuration for specific LLM models
[llm.vision]
model = "gpt-4o"
base_url = "https://api.openai.com/v1"
api_key = "sk-..."  # Replace with your actual API key
```

Alternatively, you can copy the example config using the terminal:

```
copy config\config.example.toml config\config.toml
```

Then edit `config\config.toml` in VS Code to add your API keys.

4. Run ParManus from VS Code

1. Open the main Python file you want to run (e.g., `main.py`)
2. Run the file using one of these methods:
3. Click the "Run" button (▶) in the top-right corner
4. Right-click in the editor and select "Run Python File in Terminal"
5. Press `F5` to run with debugging

For different versions: - For main version: Open and run `main.py` - For MCP tool version: Open and run `run_mcp.py` - For unstable multi-agent version: Open and run `run_flow.py`

Advanced Setup (With GPU Support)

For GPU acceleration with local models:

1. Install CUDA Toolkit

Download and install CUDA Toolkit 12.2 from NVIDIA's website.

2. Set Up Environment Variables in VS Code

1. Open the VS Code integrated terminal:
2. Press `Ctrl+`` or go to View > Terminal
3. Set environment variables:

```
# Set environment variables for current session
$env:CUDA_HOME = "C:\Program Files\NVIDIA GPU Computing
Toolkit\CUDA\v12.2"
$env:CUDACXX = "$env:CUDA_HOME\bin\nvcc.exe"
$env:PATH = "$env:CUDA_HOME\bin;$env:PATH"
```

1. To make these permanent in Windows:
2. Search for "Environment Variables" in Windows search
3. Click "Edit the system environment variables"
4. Click "Environment Variables"
5. Add the variables under "User variables"

3. Install GPU-Enabled llama-cpp-python

In the VS Code terminal:

```
# Activate your virtual environment first
.\.venv\Scripts\activate

# Uninstall regular version
pip uninstall -y llama-cpp-python

# Install CUDA-enabled version
$env:CMAKE_ARGS = "-DGGML_CUDA=on"
$env:FORCE_CMAKE = "1"
pip install llama-cpp-python
```

4. Download Models (Optional)

If you want to use local models instead of API-based ones:

```
 # Create models directory
mkdir -p models

# Download Llama model
curl -L --retry 3 --retry-delay 5 `
  "https://huggingface.co/grimjim/Llama-3.1-8B-Instruct-abliterated_via_adapter-GGUF/resolve/main/Llama-3.1-8B-Instruct-abliterated_via_adapter.Q5_K_M.gguf" `
  -o models\llama-jb.gguf

# Download LLaVA vision model
curl -L --retry 3 --retry-delay 5 `
  "https://huggingface.co/mys/ggml_llava-v1.5-7b/resolve/main/ggml-model-q4_k.gguf" `
  -o models\llava-vision.gguf
```

5. Update Configuration for Local Models

Edit your `config\config.toml` in VS Code:

```
 # Global LLM configuration
[llm]
model = "llama-jb"
model_path = "models/llama-jb.gguf"  # Use relative or absolute path
max_tokens = 2048
temperature = 0.0
n_gpu_layers = -1  # Use all available GPU layers

[llm.vision]
model = "llava-v1.5-7b"
model_path = "models/llava-vision.gguf"  # Use relative or absolute path
max_tokens = 2048
temperature = 0.0
n_gpu_layers = -1  # Use all available GPU layers
```

VS Code Debugging and Development

Setting Up Debugging

1. Create a launch configuration:
2. Go to the Run and Debug view (Ctrl+Shift+D)
3. Click "create a launch.json file"
4. Select "Python"

5. Choose "Python File"

6. VS Code will create a `.vscode/launch.json` file. Modify it to include:

```json
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "Python: Main",
      "type": "python",
      "request": "launch",
      "program": "${workspaceFolder}/main.py",
      "console": "integratedTerminal",
      "justMyCode": true
    },
    {
      "name": "Python: MCP",
      "type": "python",
      "request": "launch",
      "program": "${workspaceFolder}/run_mcp.py",
      "console": "integratedTerminal",
      "justMyCode": true
    },
    {
      "name": "Python: Flow",
      "type": "python",
      "request": "launch",
      "program": "${workspaceFolder}/run_flow.py",
      "console": "integratedTerminal",
      "justMyCode": true
    }
  ]
}
```

1. Now you can select which configuration to run from the dropdown in the Run and Debug view.

Recommended VS Code Settings

Create a `.vscode/settings.json` file with these recommended settings:

```json
{
  "python.linting.enabled": true,
  "python.linting.pylintEnabled": true,
  "python.formatting.provider": "black",
  "editor.formatOnSave": true,
  "python.analysis.extraPaths": [
    "${workspaceFolder}"
  ],
  "python.terminal.activateEnvironment": true,
  "terminal.integrated.env.windows": {
    "PYTHONPATH": "${workspaceFolder}"
  }
}
```

Troubleshooting in VS Code

Browser Automation Issues

If you encounter issues with browser automation:

```
 # Reinstall Playwright with all dependencies
playwright install --with-deps
```

GPU Support Verification

To verify GPU support is working, create a new file in VS Code called `verify_gpu.py`:

```
 from llama_cpp import Llama
print(f'CUDA available: {Llama.get_cuda_device_count() > 0}')
```

Run it with F5 or the Run button.

Python Package Installation Errors

If you encounter errors installing packages:

```
 # Update pip and setuptools
pip install --upgrade pip setuptools wheel

# Install Microsoft C++ Build Tools if needed
# Then retry the installation
pip install -r requirements.txt
```

VS Code-Specific Issues

1. **Terminal Not Showing Output**:
2. Go to View > Terminal to ensure the terminal is visible

3. Check Output panel (Ctrl+Shift+U) and select "Python" from the dropdown

4. **Python Extension Not Working**:

5. Reload VS Code (Ctrl+Shift+P, then "Developer: Reload Window")

6. Reinstall the Python extension

7. **Environment Not Activating**:

8. Manually activate with `.\.venv\Scripts\activate`
9. Check settings.json to ensure `python.terminal.activateEnvironment` is true

Windows-Specific Notes

1. **Path Length Limitations**: Windows has path length limitations. Clone to a short path (e.g., `C:\Projects\parmanus`) to avoid issues.

2. **PowerShell Execution Policy**: You might need to adjust PowerShell's execution policy: `powershell Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser`

3. **Antivirus Interference**: Some antivirus software may interfere with Python or browser automation. Consider adding exceptions if needed.

4. **File Permissions**: Ensure you have appropriate permissions for the directories you're working with.