AI-PC-Stack

#	Project	Essence (single-sentence North-Star)	Homepage	Run on PC (typical)	Integration recipe on the same PC

| 1 | **DeepSeek-R1** | Fully open reasoning LLM rivaling OpenAI-o1 on math & code via MIT weights & OpenAI API. | https://deepseek.com | 7–8 B 8-bit ≈ 8–10 GB VRAM; 70 B ≥ 48 GB VRAM | Expose `http://localhost:8000/v1` via `vllm serve DeepSeek-R1` or `ollama run deepseek-r1`; other tools point here | | 2 | **Ollama** | "Docker-for-LLMs" – pull, run, chat with any open model in one command. | https://ollama.ai | 7–13 B models in 8–16 GB RAM; CPU fallback | Universal endpoint: `http://localhost:11434/v1/chat/completions` | | 3 | **OpenManus** | No-code visual workbench chaining LLMs, tools & data into autonomous AI apps. | https://github.com/FoundationAgents/OpenManus | Docker-Compose; 4–8 GB RAM; GPU optional | UI connectors → `localhost:11434` or any `http://localhost:<port>` | | 4 | **LangChain** | Universal "LEGO kit" for LLM apps via modular prompts, memory, retrieval & agents. | https://langchain.com | Pure Python/JS; 2–4 GB RAM | Import `ChatOllama(base_url="http://localhost:11434")` | | 5 | **AutoGen (Microsoft)** | Multi-agent conversation framework spinning up LLM "teams" to negotiate tasks. | https://microsoft.github.io/autogen | `pip install pyautogen`; any OpenAI endpoint | `llm_config={"base_url":"http://localhost:11434/v1","api_key":"ollama"}` | | 6 | **OpenSora** | Open distributed training platform turning any cluster into a generative-model factory. | https://github.com/hpcaitech/Open-Sora | Linux + CUDA; dev 12 GB VRAM | Train → HF checkpoint → `ollama create mymodel -f Modelfile` | | 7 | **Haystack (deepset)** | End-to-end semantic search & RAG framework for production document workflows. | https://haystack.deepset.ai | CPU baseline; GPU optional | `OpenAIGenerator(api_base="http://localhost:11434/v1")`; share vector DB | | 8 | **Text-Generation-WebUI** | Browser dashboard to download, chat & serve open LLMs locally or via API. | https://github.com/oobabooga/text-generation-webui | 4 GB VRAM for 7 B; 12 GB for 30 B | Settings → API → `http://localhost:5000/v1` | | 9 | **Whisper** | Offline SOTA speech-to-text & translation in 99 languages. | https://github.com/openai/whisper | CPU real-time; GPU 10× faster | FastAPI wrapper → `http://localhost:9000/transcribe` | |10 | **Letta** *(ex-MemGPT)* | Persistent memory layer for LLM agents across sessions & frameworks. | https://letta.ai | 2–4 GB RAM; any local LLM endpoint | `letta server --model-endpoint http://localhost:11434/v1` | |11 | **Orpheus-TTS** *(new)* | Apache-2.0 multi-speaker, multilingual, zero-shot voice-cloning TTS (150 M→3 B tiers). | https://github.com/canopyai/orpheus-tts | 150 M CPU-only; 1 B ≈ 5 GB VRAM; 3 B ≈ 10 GB VRAM | `python -m orpheus.serve --host 0.0.0.0 --port 8001` → `http://localhost:8001/v1/audio/speech` |

I am unable to create and download a PDF version of the user manual directly. However, I can provide the complete user manual in Markdown format, which can be easily copied and saved as a text file. You can then use a Markdown editor or a free online service to convert it to a PDF.

Here is the user manual in Markdown format.

AI PC System User Manual

1. System Components and Overview

Your AI PC system is an integrated, local AI stack. The core components are standalone services that leverage a central language model server (**Ollama**) to provide various functionalities.

- **Ollama:** The central server for running large language models (LLMs) like Llama 3.1 and DeepSeek-R1. It acts as the backbone for other services.
- **Text-Generation-WebUI:** A user-friendly web interface for interacting with and managing the models running on Ollama.
- **Letta:** An AI agent framework that uses Ollama as its backend to create autonomous agents.
- **Whisper API:** An audio-to-text service that uses a FastAPI web server to transcribe audio files.
- **MeloTTS:** A text-to-speech service for generating audio from text via an API.
- **OpenManus:** An AI-powered agent orchestrator designed to run in a Docker container (if Docker is installed and enabled).

---

2. The PowerShell Scripts: Your Control Suite

This system is managed by a set of PowerShell scripts. Understanding the role of each script is critical for effective operation and troubleshooting.

2.1 `setup-ai-stack.ps1`

This is your primary deployment and uninstallation script. Its purpose is to automate the entire setup process.

- **Function:** Validates system requirements, installs prerequisites via `winget`, clones necessary repositories, sets up a Python virtual environment, installs packages, and launches all services in the background.
- **Usage:**
    - To install: `.\setup-ai-stack.ps1`
    - To uninstall: `.\setup-ai-stack.ps1 -Uninstall`

2.2 `check-services.ps1`

This is a quick-and-easy diagnostic tool for checking the health of your services.

- **Function:** Performs a basic health check by sending a web request to each service's URL to confirm it is online. It provides a color-coded output for a clear status report.
- **Usage:** `.\check-services.ps1`

2.3 `debug-services.ps1`

This script is a crucial debugging tool for identifying the root cause of service failures.

- **Function:** It launches each service in its own separate, visible PowerShell window. This allows you to observe the real-time startup logs and error messages, which are hidden during a normal background launch.
- **Usage:** `.\debug-services.ps1`

2.4 `verify-ai-stack.ps1`

This is the most comprehensive diagnostic and maintenance tool.

- **Function:** It performs a deep check of the entire stack, including system requirements, repository synchronization, Python virtual environment integrity, and service availability. It can also be configured to automatically fix issues.
- **Usage:**
    - To generate a detailed report: `.\verify-ai-stack.ps1`
    - To automatically fix issues: `.\verify-ai-stack.ps1 -AutoFix`

---

3. User Workflow: A Step-by-Step Guide

3.1 Startup Procedure

1. Open an elevated PowerShell terminal (Run as Administrator).
2. Navigate to your script directory: `cd C:\Users\sgins\OneDrive\Documents\GitHub\AI-PC-Stack`
3. Execute the setup script. This will install all components and launch the services in the background.
    - `.\setup-ai-stack.ps1`

3.2 Verification

After a few minutes, run the health check script to verify that all components are online.

- `.\check-services.ps1`
- **Expected Output:** All services except Ollama will be checked via HTTP requests, and their status will be reported.

3.3 Basic Operations

Once verified, you can begin using the system. All services are available at `http://localhost:<port>`.

Example 1: Using the Text-Generation-WebUI

1. Open your web browser and go to `http://localhost:5000`.
2. The interface will automatically connect to your Ollama server.
3. Navigate to the **Model** tab and select a model (e.g., `llama3.1:8b`).
4. Go to the **Chat** tab and start a conversation.

Example 2: Transcribing Audio with the Whisper API

You can use the Whisper API to programmatically transcribe an audio file.

```powershell
# Assuming you have an audio file named "speech.wav"
Invoke-WebRequest -Uri "http://localhost:9000/transcribe" -Method Post -InFile "C:\path\to\speech.wav"
```

Example 3: Generating Speech with the MeloTTS API

You can use the MeloTTS API to generate an audio file from text.

```powershell
# Create a JSON body with the text to be spoken
$body = @{ text = "Hello, this is a test of the text to speech service." } | ConvertTo-Json

# Send a request to the API
Invoke-WebRequest -Uri "http://localhost:8001/v1/audio/speech" -Method Post -Body $body -ContentType "application/json"
```