KAMSIOB.COM · A FREE GUIDE

Local AI on Linux.

The complete from-scratch setup for a private, subscription-free AI stack on Bazzite with AMD Strix Halo — chat, a full web interface, and image generation, running entirely on your own machine.

- own it, don't rent it
- keep it private
- keep it simple

Every command in this guide was proven on a real machine — including the failures. Where something breaks, you'll be told before you hit it, what the error looks like, and exactly how to fix it.

WHAT'S INSIDE

Ollama — local language models, GPU-accelerated · Open WebUI — a clean chat interface · ComfyUI — image generation · a full troubleshooting index of real errors

by Kam · kamsiob.com · youtube.com/@kamsiob

BEFORE ANYTHING ELSE

Is this guide for you?

This guide is deliberately specific. It was built and verified on one exact combination of hardware and software, and it's honest about that instead of pretending to be universal:

> **THE SETUP THIS COVERS**
> Bazzite Linux (or a similar immutable Fedora Atomic distro) + an AMD Strix Halo APU (Ryzen AI Max, gfx1151 graphics). That pairing is exactly why the usual install instructions fail, and exactly what this guide fixes.

If you're on a different distro or different graphics hardware, parts of this will still be useful as a map, but the specific fixes won't directly apply, and you shouldn't expect to follow it word for word.

What you'll have at the end

Three tools, each running as a proper background service that starts with your machine and needs no terminal in daily use:

Ollama	Runs large language models locally — the engine behind everything.
Open WebUI	A polished, private chat interface for those models, in your browser.
ComfyUI	Image generation on your own GPU, no cloud, no credits.

Nothing here requires an account, a subscription, or sending a single byte of your data anywhere. That's the point.

Two ways to use this guide

Track A is for people with access to an AI assistant like Claude — it gets you a personalized walkthrough in minutes. Track B is the full manual path, every command explained, for everyone else. Both end in the same place.

One habit to adopt right now, before either track: whenever this guide gives you a local address, it uses `127.0.0.1`, never the word “localhost.” Several browsers quietly mishandle the word “localhost” for plain local servers and show you a broken page for no visible reason. `127.0.0.1` is the same destination, minus the bug.

TRACK A

If you have an AI assistant

The fastest path by far. But there's a catch worth understanding: a generic AI assistant doesn't know this hardware's quirks unless you tell it. Left to its defaults, it will walk you into the same three walls this guide exists to avoid — an installer that silently half-fails on immutable filesystems, a GPU that gets ignored unless explicitly enabled, and a driver bug on this exact chip that crashes image generation.

So don't ask it to “install Ollama.” Hand it the full context up front, in one message. Copy this starter prompt exactly:

STARTER PROMPT — PASTE THIS TO YOUR ASSISTANT

```
I'm setting up a fully local AI stack and need step-by-step help.
My exact setup, all of which matters:

- Bazzite Linux (immutable Fedora Atomic base). /usr is read-only,
so installers that create users or write to /usr will silently
fail. Standard install scripts often need manual systemd service
creation instead.
- AMD Strix Halo APU (Ryzen AI Max, gfx1151 iGPU). Ollama ignores
integrated GPUs unless OLLAMA_IGPU_ENABLE=1 and OLLAMA_VULKAN=1
are set in its service environment.
- The standard PyTorch ROCm build segfaults on this chip
(crash in libamdhip64.so during memcpy_and_sync). The fix is
AMD's gfx1151-specific nightly build:
pip install --index-url https://rocm.nightlies.amd.com/v2/gfx1151/
torch torchaudio torchvision
- I use Podman (not Docker), and containers should be set up as
Quadlets so systemd manages them.
- For any local web address, always give me http://127.0.0.1:PORT,
never "localhost".

I want: Ollama as a systemd service using the GPU, Open WebUI as a
Podman Quadlet container connected to it, and ComfyUI with the
GGUF custom node, each verified working before moving to the next.
Explain each step briefly as we go. Ask me questions when needed.
```

IF IT GETS STUCK

If your assistant gets stuck or contradicts itself, point it at Track B of this guide — every working config file is printed there in full, and it can use them as ground truth instead of guessing.

TRACK B · PART 1 OF 3

Ollama — the engine

Ollama is what actually runs language models on your machine. Everything else connects to it. One thing to know before the first command: on Bazzite, the official installer partially fails by design — and that's fine, because the part that fails is the part we're about to do better ourselves.

Step 1 — Run the installer

```
curl -fsSL https://ollama.com/install.sh | sh
```

Watch the output. You will very likely see this line — it is expected, not a mistake you made:

```
useradd: cannot create directory /usr/share/ollama
```

That's Bazzite's immutable filesystem doing its job: /usr is read-only, the installer assumed it could write there, and as a result it silently skipped creating the background service. The Ollama program itself installed fine. We create the service properly ourselves in the next step.

Step 2 — Create the service by hand

A systemd service is just a small text file telling Linux “keep this program running in the background, restart it if it dies, start it at boot.” Open an editor for a new one:

```
sudo systemctl edit --force --full ollama.service
```

Paste exactly this, swapping `YOUR_USERNAME` for your actual username (run `whoami` if unsure):

OLLAMA.SERVICE — THE COMPLETE WORKING FILE

```
[Unit]
Description=Ollama Service
After=network-online.target

[Service]
ExecStart=/usr/local/bin/ollama serve
User=YOUR_USERNAME
Environment="OLLAMA_VULKAN=1"
Environment="OLLAMA_IGPU_ENABLE=1"
Environment="OLLAMA_HOST=0.0.0.0:11434"
Restart=always
RestartSec=3

[Install]
WantedBy=default.target
```

The two GPU lines are the whole reason this guide exists: without `OLLAMA_IGPU_ENABLE=1`, Ollama silently ignores integrated GPUs and runs everything on the CPU at a fraction of the speed — with no warning that it's doing so. `OLLAMA_HOST=0.0.0.0` lets the web interface we build in Part 2 reach it. (Small trade-off: this also makes Ollama reachable from other devices on your home network. Fine behind a home router; worth knowing it's happening.)

Step 3 — Start it and confirm

```
sudo systemctl enable --now ollama
systemctl status ollama
```

You're looking for active (running) in that output. Then prove it end to end with a small model:

```
ollama pull llama3.2:1b
ollama run llama3.2:1b "hello" # type /bye to exit the chat
```

SLOW ANSWERS?

If responses feel very slow, the GPU variables likely aren't being picked up — re-open the service file and check both Environment lines for typos, then `sudo systemctl restart ollama`.

TRACK B · PART 2 OF 3

Open WebUI — the interface

Open WebUI gives you a clean, private chat interface to those models in your browser. It runs in a container — an isolated box managed by Podman — and we'll register it as a Quadlet, which simply means systemd manages the container the same way it now manages Ollama: always on, restarts itself, starts at boot.

Step 1 — Pull the container image first, deliberately

```
podman pull ghcr.io/open-webui/open-webui:main
```

This download is several gigabytes. Doing it now, manually, avoids a trap in step 3: systemd gives a starting service only 90 seconds by default, gives up mid-download, and reports a confusing “timeout exceeded” failure that looks much worse than it is.

Step 2 — Create the Quadlet file

```
mkdir -p ~/.config/containers/systemd
nano ~/.config/containers/systemd/open-webui.container
```

Paste exactly this:

```
OPEN-WEBUI.CONTAINER — THE COMPLETE WORKING FILE


[Unit]
Description=Open WebUI Container
After=network-online.target

[Container]
Image=ghcr.io/open-webui/open-webui:main
ContainerName=open-webui
PublishPort=3000:8080
Volume=open-webui:/app/backend/data
AddHost=host.containers.internal:host-gateway
Environment=OLLAMA_BASE_URL=http://host.containers.internal:11434

[Service]
Restart=always
TimeoutStartSec=600

[Install]
WantedBy=default.target
```

THE SILENT-FAILURE TRAP

The Volume line reads `open-webui:` with no ".volume" suffix. Adding that suffix tells Quadlet to look for a separate definition file that doesn't exist — and instead of an error, systemd silently refuses to generate the service at all. You'd see only "Unit open-webui.service not found" with nothing to debug. This exact mistake cost real hours the first time.

Step 3 — Start it

```
systemctl --user daemon-reload
systemctl --user start open-webui
systemctl --user status open-webui
```

Note: no `sudo` on these — this service runs under your own user, not the system. Once status shows active, open `http://127.0.0.1:3000` in your browser. It will ask you to create an admin account — that account is stored entirely inside the container on your machine. Nothing leaves it.

TRACK B · PART 3 OF 3

ComfyUI — image generation

The most involved of the three, because this is where the chip-specific driver bug lives. Follow the order exactly — the PyTorch fix in step 2 is not optional on this hardware.

Step 1 — Install ComfyUI and the GGUF add-on

```
cd ~
git clone https://github.com/comfyanonymous/ComfyUI.git
cd ComfyUI
python3 -m venv venv
source venv/bin/activate
pip install -r requirements.txt
cd custom_nodes
git clone https://github.com/city96/ComfyUI-GGUF.git
pip install gguf
```

The venv is a self-contained Python environment inside the ComfyUI folder — it keeps everything this app needs isolated from your system. The ComfyUI-GGUF add-on teaches ComfyUI to load GGUF-format models; without it, the model we download below simply won't appear as loadable, with no explanation.

Step 2 — The critical fix: replace PyTorch

The standard PyTorch ROCm build crashes on this chip. The crash looks like a sudden “Segmentation fault” that kills ComfyUI mid-generation, pointing at `libamdhip64.so`. It is not your fault and not fixable by settings — the fix is AMD's chip-specific build:

```
THE GFX1151 FIX


# still inside the venv from step 1
pip uninstall -y torch torchaudio torchvision
pip install --index-url https://rocm.nightlies.amd.com/v2/gfx1151/ \
torch torchaudio torchvision
```

Step 3 — Download the models

Three files, three specific folders. The main one is ~22 GB — start it and walk away. Putting a file in the wrong folder makes it silently invisible to ComfyUI, so the destinations here matter:

```
cd ~/ComfyUI
mkdir -p models/diffusion_models models/text_encoders models/vae

# 1. The image model itself (~21.8 GB)
wget -P models/diffusion_models \
https://huggingface.co/city96/Qwen-Image-gguf/resolve/main/qwen-image-Q8_0.gguf

# 2. The text encoder (~8.8 GB)
wget -P models/text_encoders \
https://huggingface.co/Comfy-Org/Qwen-Image_ComfyUI/resolve/main/split_files/text_encoders/qwen_2.
5_vl_7b_fp8_scaled.safetensors

# 3. The VAE (~243 MB)
wget -P models/vae \
https://huggingface.co/Comfy-Org/Qwen-Image_ComfyUI/resolve/main/split_files/vae/qwen_image_vae.sa
fetensors
```

Step 4 — The launch script

This chip needs a few environment variables set every launch. Baking them into a script means never remembering them again. Create it once:

START_COMFYUI_ROCM.SH

```
cat > ~/ComfyUI/start_comfyui_rocm.sh << 'EOF'
#!/bin/bash
export HSA_ENABLE_SDMA=0
export HSA_USE_SVM=0
export HIP_VISIBLE_DEVICES=0
cd ~/ComfyUI
source venv/bin/activate
python main.py
EOF
chmod +x ~/ComfyUI/start_comfyui_rocm.sh
```

Launch with `~/ComfyUI/start_comfyui_rocm.sh`, then open `http://127.0.0.1:8188`. Running `python main.py` directly, without the script, skips those variables — and on this chip that's the difference between working and crashing.

WHEN SOMETHING BREAKS

Troubleshooting index

Every error below happened for real while building this the first time. Match yours against the exact text and skip the guesswork:

`useradd: cannot create directory /usr/share/ollama`

What it means: Bazzite's read-only /usr blocked the Ollama installer's user-creation step, so it silently skipped creating the background service.

Fix: Expected. The program itself installed — create the service manually (Part 1, step 2).

`Failed to start open-webui.service: Unit open-webui.service not found.`

What it means: The Quadlet file has a problem — most often the ".volume" suffix trap — and systemd silently skipped generating the service instead of showing an error.

Fix: Check the Volume line reads `open-webui:` with no suffix. See the real parse error with `journalctl --user -b | grep -i quadlet`.

`Job for open-webui.service failed because a timeout was exceeded.`

What it means: The first container download is several GB; systemd's default 90-second start window expired mid-download and killed it.

Fix: Pull manually first (`podman pull ghcr.io/open-webui/open-webui:main`) and keep `TimeoutStartSec=600` in the file.

`Segmentation fault (crash referencing libamdhip64.so)`

What it means: The standard PyTorch ROCm build has a memory bug on gfx1151 specifically. Nothing you configured caused it.

Fix: Install AMD's chip-specific nightly build (Part 3, step 2). This resolves it entirely.

`A downloaded model doesn't appear inside ComfyUI`

What it means: Either it's in the wrong models/ subfolder, or it's a GGUF file and the ComfyUI-GGUF add-on isn't installed — both fail with no visible error.

Fix: Check the folder against Part 3 step 3, confirm `custom_nodes/ComfyUI-GGUF` exists, then restart ComfyUI.

A local address shows a blank/broken page in the browser

What it means: The word “localhost” is being silently upgraded to https by the browser, which plain local servers can't answer.

Fix: Use `http://127.0.0.1:PORT` instead. Always. Ports: Open WebUI 3000, ComfyUI 8188, Ollama's API 11434.

YOU'RE DONE

Everything runs on your machine now.

Chat at `127.0.0.1:3000`. Images at `127.0.0.1:8188`. No accounts anywhere, no subscriptions, nothing phoning home. Technology that actually belongs to you.

I document this whole journey — what worked, what broke, and why, no fake enthusiasm — on the channel. And if this guide saved you the hours it cost me, the coffee link below is the entire business model.

THIS GUIDE COMES FROM AN APP — LOCAL AI HUB

• Ollama • Open WebUI • ComfyUI

One clean window for this whole stack: see what's running, start and stop each tool with a toggle, update or install models, get told plainly when something crashes — no terminal, ever again. Completely free, open source, zero tracking. Grab it at kamsiob.com

YOUTUBE	youtube.com/@kamsiob
GITHUB	github.com/kamsiob
WEBSITE	kamsiob.com
THE LAB (TELEGRAM)	t.me — invite link on the site
SUPPORT	buymeacoffee.com/kamsiob
SAY HI	hello@kamsiob.com

Thanks for stopping by. Genuinely. — Kam

this guide is free to share · no analytics were involved in the making of this pdf · it wouldn't be very me otherwise.