Set Up Your Own Private AI Server

A beginner's step-by-step guide: from a fresh Debian system to asking questions of your own documents — with no data leaving your machine.

Before you start

This guide picks up **after** Debian Linux is already installed. It won't walk you through installing Debian itself. You need:

- A computer with a fresh install of **Debian 12 (Bookworm)** or **Debian 13 (Trixie)**, connected to the internet.
- A **terminal** window. On a Debian desktop, open the applications menu and search for “Terminal”.
- A user account. If it can't use `sudo` yet (the command that runs things as administrator), Step 1 fixes that — it's a common surprise on fresh Debian, not a mistake you made.

How to use the command blocks. Each grey block is one step. Copy the whole block, paste it into the terminal with **Ctrl+Shift+V**, and press Enter. Wait for it to finish before moving on. A line ending in `\` is just one long command wrapped over two lines — pasting it whole still works.

How long will this take? About an hour, most of it waiting for downloads. The small test model is about 1.3 GB; later models are bigger, so a slow connection just means more waiting.

What you will do, step by step

There are nine steps, and each one builds on the last. Here's the whole journey before you start, so nothing comes as a surprise:

1. **Update the system and install basic tools.** Get the fresh system current, make sure `sudo` works, and install the small tools that keep turning up in setup guides.

2. **Install Docker and Docker Compose.** Docker runs OpenWebUI in a sealed box, so you never have to untangle its dependencies by hand.
3. **Install Ollama.** This is the engine that runs the AI models on your own machine.
4. **Learn the Ollama settings that matter.** Where it listens, where models live, and how to clean up — nothing to change yet, just things to know.
5. **Download a small model first.** A tiny test model proves the engine works before you spend time on anything big.
6. **Install OpenWebUI.** The friendly chat window you will actually talk to, running in a Docker container.
7. **Connect OpenWebUI to Ollama.** Point the chat window at the engine, then ask your first question through it.
8. **Start a workspace and upload your knowledge base.** Give the system your own documents to search, so answers come from your files instead of from general knowledge.
9. **Keep it private, updated and backed up.** The short habits that keep a private system private.

After that, a troubleshooting section covers the five problems beginners actually hit, and a final section explains graphics cards: what size model needs one, and what each option costs in Pakistan.

Step 1 — Update the system and install basic tools

Start by bringing the fresh system up to date. Then install the small tools every later step needs, plus a few more you will keep bumping into in other guides: `curl` and `wget` (both download files — different guides use one or the other), `nano` (a simple text editor), `git` (fetches projects other people publish), `unzip` (opens zip files), and `htop` (shows what is using your processor and memory while a model runs).

> **No `sudo` on your system?** A fresh Debian install often leaves it out. Log in as root and fix it once, then carry on with the rest of this guide as normal:

```
su -
apt update && apt install -y sudo
usermod -aG sudo your-username
exit
```

Replace `your-username` with your own account name, then log out and back in. If you stay logged in as root instead, type every command in this guide **without** the word `sudo`.

```
sudo apt update && sudo apt upgrade -y
sudo apt install -y curl wget nano git unzip htop ca-certificates gnupg
```

Check: no errors at the end, and you are back at the prompt. If you are asked “Do you want to continue?”, answer `y` and press Enter.

Step 2 — Install Docker and Docker Compose

OpenWebUI needs Docker to run, so install that first. This block adds Docker's own repository for Debian, then installs Docker and Docker Compose from it. It works on both Bookworm and Trixie, because it reads your version automatically instead of you having to type it.

```
sudo install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/debian/gpg \
  -o /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc
CODENAME=$(. /etc/os-release && echo "$VERSION_CODENAME")
echo "deb [arch=$(dpkg --print-architecture) \
signed-by=/etc/apt/keyrings/docker.asc] \
https://download.docker.com/linux/debian $CODENAME stable" \
| sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt update
sudo apt install -y docker-ce docker-ce-cli containerd.io \
  docker-buildx-plugin docker-compose-plugin
sudo systemctl enable docker
sudo systemctl start docker
```

Now let your own account run Docker without typing `sudo` every time:

```
sudo usermod -aG docker $USER
newgrp docker
```

The `newgrp docker` line applies the change immediately in this terminal. (On some systems you still need to log out and back in once for it to take effect everywhere.)

> **Check:** run this block. You should see version numbers and, at the end, Docker's own “Hello from Docker!” message.
>
> ```
> docker --version
> docker compose version
> docker run hello-world
> ```

Step 3 — Install Ollama

Ollama is the engine that runs the AI models on your own machine. One command installs it, and it sets itself up to start automatically every time the machine boots.

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

> **Check:** these two commands should answer without errors. The second one should print the words “Ollama is running”.
>
> ```
> ollama --version
> curl http://localhost:11434
> ```

If the second command says “connection refused”, Ollama isn't running. Start it with `sudo systemctl start ollama` and try again.

Step 4 — Ollama settings you should know

You don't need to change anything to get going — the defaults are right for a first setup. But four facts will save you confusion later:

- **Where it listens.** By default Ollama only accepts connections from the same machine, on address `localhost:11434`. That is exactly what OpenWebUI needs in Step 7, so leave it alone for now.
- **Where models live.** Downloaded models sit in your home folder under `.ollama`. If your disk fills up, this is the folder that grew.
- **It starts on boot.** Ollama runs as a system service, so it is already running after a restart. Check it any time with `sudo systemctl status ollama`.
- **Seeing and removing models.** `ollama list` shows what you have downloaded. `ollama rm model-name` deletes one and frees its disk space.

> **Optional — only if you want other computers on your home network to talk to Ollama directly.** Most readers should skip this: OpenWebUI (Step 6) is how you will use the system, and it doesn't need this. If you do need it, run `sudo systemctl edit ollama`, add the lines `[Service]` and `Environment="OLLAMA_HOST=0.0.0.0:11434"`, save, then `sudo systemctl restart ollama`. Only do this on a trusted home network — never open Ollama's port to the internet.

Step 5 — Download a small model first (your test case)

Don't start with a giant model. Download the smallest useful one first and prove the whole chain works. `llama3.2:1b` is a tiny model (about 1.3 GB) that runs comfortably on an ordinary processor with no graphics card.

```
ollama pull llama3.2:1b
```

This download takes a few minutes. When it finishes, ask it one question:

```
ollama run llama3.2:1b "Say hello and tell me what you are in one senten
ce."
```

> **Check:** you get a sensible one-sentence answer, and you are back at the prompt. Type `/bye` if it stays in chat mode. Your engine works. Everything after this is just nicer models and a nicer window to talk to them in.

As a second step, download a somewhat smarter model that still runs fine without a graphics card:

```
ollama pull llama3.2:3b
```

Bigger models than this exist (8 billion, 14 billion, 32 billion parameters and more), but they get slow on a plain processor. That is what the graphics-card section at the end is about — read it before downloading anything bigger than 3 billion parameters.

Step 6 — Install OpenWebUI

OpenWebUI is the friendly chat window for your local models. It runs in a Docker container, which is why Step 2 came first. Copy this whole block. One thing to change before you paste: replace `your-secret-key` with any long phrase you make up (it just protects the program's own data).

```
docker run -d -p 3000:8080 \
  --add-host=host.docker.internal:host-gateway \
  -v open-webui:/app/backend/data \
  -e WEBUI_SECRET_KEY=your-secret-key \
  --name open-webui --restart always \
  ghcr.io/open-webui/open-webui:main
```

What each part means, in plain words:

- `-d` runs it in the background, so you get your terminal back.
- `-p 3000:8080` publishes it on port 3000 of your machine — that is the number in the web address below.
- `--add-host=host.docker.internal:host-gateway` lets the container reach Ollama on the same machine. Step 7 depends on this line.
- `-v open-webui:/app/backend/data` gives it a permanent place to keep your chats and settings, so they survive a restart.

- `--restart always` starts it again automatically after a reboot.

Wait about a minute for the first start, then open it in a browser:

- On the server machine itself: `http://localhost:3000`
- From another computer on the same home network: `http://SERVER-IP:3000`, where SERVER-IP is the server's address on your network (find it with `hostname -I` in the terminal).

The first screen asks you to create an account. The first account you create becomes the administrator account, and it lives only on your machine — there is no cloud account and no signup with anyone.

> **Check:** the OpenWebUI page opens and lets you create an account. If the page doesn't load, wait another minute and refresh; the first start is the slowest. If it still doesn't load, see the troubleshooting section at the end.

Step 7 — Connect OpenWebUI to Ollama

OpenWebUI and Ollama are two separate programs, so you must point one at the other. This takes less than a minute:

1. In OpenWebUI, click your profile picture (bottom left) and open **Admin Panel**, then **Settings**, then **Connections**.
2. Find the **Ollama API** address field. Type exactly this and save:

   ```
   http://host.docker.internal:11434
   ```

3. Start a **New Chat**. At the top, open the model menu — you should now see `llama3.2:1b` and `llama3.2:3b`.
4. Pick `llama3.2:1b` and ask it anything. If it answers, the whole system works end to end.

> **If the model menu is empty:** check Ollama itself first with `ollama list` in the terminal — if your models are listed there, the problem is the address in step 2 (check it character by character). If `ollama list` is

> empty, go back to Step 5. Full troubleshooting list at the end of this guide.

Step 8 — Start a workspace and upload your knowledge base

This is the step that turns a chatbot into your private document assistant. A “knowledge base” is a collection of your own files that the system searches before answering, so answers come from your documents instead of from the model's general knowledge. One knowledge base can hold many documents — that is the whole point of it — and when you chat, you use the entire knowledge base, not one file at a time.

1. In OpenWebUI, open **Workspace** (left sidebar) and then **Knowledge**.
2. Create a new knowledge base. Give it a clear name for one topic or project, for example “Plant manuals” or “Bid documents 2026”. Keep one knowledge base per topic — it keeps answers clean.
3. Add your documents to it. You can upload individual files, or a whole folder at once: PDF, Word, plain text and Markdown files all work. Start with a handful of files, not a hundred, so you can tell whether it is working.
4. Wait for processing. OpenWebUI reads each file and turns it into searchable pieces (this is called “embedding”). A few files take a minute or two; a large library takes longer. Don't judge the system until processing shows as finished.
5. Start a **New Chat** and attach your knowledge base to the chat (look for the knowledge selector in the chat controls). You can attach more than one knowledge base to a chat — the system searches all of the ones you attach.
6. Ask a question whose answer you already know is in one of your files, then check the answer against the source file. A good answer quotes or closely follows the document. If it doesn't, your files may still be processing, or the question may need to be more specific.

Three habits that decide whether this works well:

- **File names matter.** “OEM-manual-pump-2021.pdf” is useful later; “scan001.pdf” is not.
- **Text must be real text.** A clean PDF or Word file works. A photo of a page, or a badly scanned document, gives the system almost nothing to search — if a file is scanned, it needs readable text (OCR) first.
- **Attach only the bases that hold the answer.** Attaching every knowledge base you own to every chat muddles the results. Keep bases per topic, and attach the ones relevant to the question you are asking.

That is the whole setup. From here, using the system is just: add documents to a knowledge base, ask questions, and check answers against the cited source before you rely on them for anything important.

Step 9 — Keep it private, updated and backed up

The point of this setup is that nothing leaves your machine. Keep it that way:

- **Don't forward ports on your router.** Never map ports 3000 or 11434 to the internet “so I can use it from outside”. That would publish your private documents to the world. If you need access away from home, that is a separate project (a private VPN), not a router setting.
- **Keep a strong login password** on the machine itself, since anyone who can log in can open OpenWebUI.
- **Update sometimes.** Every few weeks, run the system updates and refresh OpenWebUI:

```
sudo apt update && sudo apt upgrade -y
docker pull ghcr.io/open-webui/open-webui:main
docker stop open-webui && docker rm open-webui
docker run -d -p 3000:8080 \
  --add-host=host.docker.internal:host-gateway \
  -v open-webui:/app/backend/data \
  -e WEBUI_SECRET_KEY=your-secret-key \
  --name open-webui --restart always \
  ghcr.io/open-webui/open-webui:main
```

Your chats and knowledge bases live in the `open-webui` volume, so recreating the container this way keeps them. (Use the same secret key phrase you used in Step 6.)

- **Back up your documents folder.** The knowledge base is built from your files, so the files themselves are the backup. Keep a copy of them somewhere else, as you would for any important work.

Troubleshooting — the five problems beginners actually hit

“Permission denied” when running a docker command

Your user account isn't in the Docker group yet in this terminal. Run `newgrp docker`, or log out and back in once, then try again.

OpenWebUI opens, but the model menu is empty

Check Ollama first: `ollama list` should show your models. If it does, the Ollama address in Admin Panel → Settings → Connections is wrong — it must be exactly `http://host.docker.internal:11434`. If `ollama list` is empty, repeat Step 5.

OpenWebUI page doesn't load at all

Check whether the container is actually running: `docker ps` should list `open-webui`. If it is missing, see what it complained about with `docker logs open-webui` (the last lines usually name the problem). If it is running but the page won't load, wait a minute and refresh — the first start is slow.

“Port 3000 is already in use”

Something else on the machine already uses port 3000. Stop the old container with `docker stop open-webui && docker rm open-webui` and re-run Step 6, or change `-p 3000:8080` to `-p 3001:8080` and open port 3001 instead.

Model downloads are very slow, or answers are very slow

Downloads: models are gigabytes, so on a slow connection a download just takes a long time — let it finish. Answers: small models on a plain processor are not instant. If even the 1-billion model feels unusably slow, your machine is probably very old or very short of memory; if the 3-billion model is slow but usable, that is normal for processor-only running, and the graphics-card section below is your next step.

Going further — bigger models need a graphics card (GPU)

Everything above runs on an ordinary processor. That is enough for small models (1–3 billion parameters) and, with patience, for 8-billion ones. Larger models are a different story: on a plain processor they answer so slowly that they are painful to use, because the processor must grind through billions of calculations one batch at a time. A graphics card (GPU) does those calculations in parallel, which is what makes a big model feel responsive.

The number that matters is the card's memory (VRAM). As a rough rule, a model needs about **0.6 GB of VRAM per billion parameters** (at the usual 4-bit compression), plus a couple of gigabytes of headroom. So an 8-billion model needs roughly 8 GB, a 14-billion model roughly 12 GB, and a 32-billion model roughly 24 GB. Buy by VRAM first; the exact model of card second.

Model size	VRAM needed (approx.)	Example cards	Indicative price in Pakistan
1–3 billion	None — processor is fine	—	Rs 0
8 billion	~8 GB	RTX 4060 8GB	~Rs 90,000–123,000 new; used from ~Rs 72,000
14 billion	~12 GB	RTX 3060 12GB · RTX 4070 12GB · RTX 5070 12GB	3060: ~Rs 85,000–115,000 (used from ~Rs 70,000); 4070: ~Rs 190,000–240,000; 5070: ~Rs 225,000–290,000

32 billion	~24 GB	RTX 3090 24GB (mostly used) · RTX 4090 24GB	3090: ~Rs 210,000–280,000; 4090: ~Rs 600,000–700,000
70 billion	~48 GB	Two 24GB cards together	Beyond a first build — skip this unless you already know why you need it

Read the prices honestly. They are indicative, taken from Pakistani online computer shops in late October 2026, and they move with the exchange rate and stock. Many listings were already out of stock when checked, so treat every figure as a range to verify, not a quote. Used cards are much cheaper than new ones and are the sensible route for a first GPU — buy from a shop that offers a checking warranty, and check that your power supply can handle the card before you buy. A weak power supply will crash the machine under load; it won't just give you slow answers.

One more job when you add a GPU. Fitting the card is only half of it: Ollama also needs NVIDIA's drivers and the NVIDIA Container Toolkit before it can use the card. That is a separate afternoon's job with its own instructions in Ollama's official documentation — it is deliberately not part of this first setup, because this guide is about getting a working private system first, on the machine you already have.

Start small, prove it works, and only spend on a graphics card when you have a model in mind that your processor genuinely cannot run well. That order — working system first, hardware second — is the whole point.

Sources and further reading

- Ollama quickstart — running models locally, from the official documentation: https://docs.ollama.com/quickstart
- Open WebUI documentation — the self-hosted front end, its knowledge-base features, and connecting agents: https://docs.openwebui.com/
- Qdrant local quickstart — the vector database behind larger document setups, for when your library outgrows the basics: https://qdrant.tech/documentation/quickstart/

By Saad Hafeez · Companion to the blog post Private Local AI for Engineering Work