Guida Ollama OpenWebUI OpenTerminal Linux Mint 22 2

Ultimo aggiornamento: 2026-08-18

MANUALE OPERATIVO

Ollama, Open WebUI e OpenTerminal

Installazione, modelli multimodali, web search e creazione di progetti su Linux Mint 22.2

Ambiente realmente collaudato

Ollama 0.32.14 | Open WebUI 0.11.0 | OpenTerminal 0.11.35

Qwen 3.8 27B | Vision | Tools | Thinking | Web search

NVIDIA RTX 5060 Ti 16 GB | 32 GB RAM

Edizione di collaudo - 18 agosto 2026

Come usare questa guida

Obiettivo: Partire da un sistema Linux Mint 22.2 con Docker e driver NVIDIA funzionanti, installare Ollama come servizio, eseguire un modello con vision e tool calling, collegarlo a Open WebUI e aggiungere OpenTerminal come ambiente di lavoro isolato.

Indice operativo

Ambiente di collaudo

Risultato ottenuto

Chat locale con un modello multimodale capace di analizzare immagini.

Ricerca web agentica con search_web, fetch_url e collegamenti alle fonti.

Tool calling nativo verso OpenTerminal per creare, eseguire e verificare file.

Produzione di un progetto web completo e di uno ZIP scaricabile.

Controlli reali con curl, unzip -t e SHA-256.

1. Architettura e prerequisiti

Il browser parla con Open WebUI sulla porta 3000. Open WebUI raggiunge Ollama sul sistema host tramite host.docker.internal:11434 e raggiunge OpenTerminal, nella rete Docker dedicata, tramite il nome DNS open-terminal:8000. I modelli rimangono gestiti da Ollama sul sistema host; i file creati dall'agente restano nel volume Docker open-terminal.

1.1 Verifiche preliminari

uname -a lsb_release -a docker --version sudo docker info nvidia-smi free -h

Prerequisito GPU: L'immagine CUDA di Open WebUI accelera funzioni interne come Whisper ed embedding. L'inferenza del modello, in questa architettura, è eseguita da Ollama sul sistema host.

1.2 Spazio su disco

Un modello 27B quantizzato può occupare circa 18 GB, ai quali vanno aggiunti cache, immagini Docker, database di Open WebUI e workspace dell'agente. Prima del pull è prudente avere almeno 35-50 GB liberi.

df -h sudo docker system df ollama list

2. Installazione e verifica del servizio Ollama

2.1 Installazione o aggiornamento

La procedura ufficiale per Linux usa lo script di installazione Ollama [1].

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

ollama -v which ollama sudo systemctl enable --now ollama sudo systemctl status ollama --no-pager

2.2 Verifica API e log

curl -sS http://127.0.0.1:11434/api/version sudo journalctl -u ollama -n 100 --no-pager sudo journalctl -u ollama --no-pager --follow

2.3 Consentire l'accesso dal container Open WebUI

Con Open WebUI in bridge Docker, il servizio Ollama deve ascoltare su un indirizzo raggiungibile dal gateway Docker. Creare un override systemd [2]:

sudo systemctl edit ollama.service

[Service] Environment="OLLAMA_HOST=0.0.0.0:11434"

sudo systemctl daemon-reload sudo systemctl restart ollama sudo ss -ltnp | grep 11434 curl -sS http://127.0.0.1:11434/api/version

Sicurezza: OLLAMA_HOST=0.0.0.0 rende il servizio raggiungibile anche da altre interfacce. Non inoltrare la porta 11434 sul router e limita l'accesso con il firewall se la macchina è in una rete non fidata.

2.4 Errore 412: versione di Ollama troppo vecchia

Il pull iniziale del modello ha restituito: The model you are attempting to pull requires a newer version of Ollama. La versione installata era 0.32.11. La correzione è stata rieseguire lo script

ufficiale, riavviare il servizio e verificare la nuova versione 0.32.14.

curl -fsSL https://ollama.com/install.sh | sh sudo systemctl restart ollama ollama -v

3. Pull, esecuzione e tuning di qwen3.8:27b

3.1 Download e prima esecuzione

ollama pull qwen3.8:27b ollama list ollama run qwen3.8:27b

Per uscire dalla sessione interattiva usare /bye.

3.2 Verifica delle capacità native

ollama show qwen3.8:27b | sed -n '/Capabilities/,/^$/p'

Capabilities completion vision tools thinking

La presenza di vision conferma che il manifest Ollama contiene i componenti multimodali; tools abilita il function calling e thinking separa il ragionamento dalla risposta finale [4][5].

3.3 Test rapido da riga di comando

ollama run qwen3.8:27b ./immagine.png
"Descrivi l'immagine e trascrivi il testo visibile."

3.4 Contesto: parametro decisivo per web e agenti

Sotto 24 GiB di VRAM, Ollama usa normalmente 4K di contesto. La documentazione raccomanda almeno 64K per web search, agenti e strumenti di coding [3]. Nel collaudo, 16.384 token non sono stati sufficienti per completare in un solo turno una lunga sequenza OpenTerminal; impostando 60.000, Ollama ha allocato 60.384 e il modello ha completato ZIP e verifiche.

ollama ps

NAME SIZE PROCESSOR CONTEXT qwen3.8:27b 18 GB 35%/65% CPU/GPU 60384

Interpretazione: La colonna CONTEXT indica la capacità configurata, non l'occupazione corrente. L'offload 35% CPU / 65% GPU spiega la maggiore latenza: il modello da circa 18 GB non entra interamente nei 16 GB di VRAM.

3.5 Profilo Ollama opzionale tramite Modelfile

Se si desidera un alias dedicato agli agenti, creare un Modelfile [6]:

FROM qwen3.8:27b PARAMETER num_ctx 60000 PARAMETER num_predict 4096 PARAMETER temperature 0.2

ollama create qwen3.8-agent:27b -f Modelfile ollama run qwen3.8-agent:27b

Modelli Hugging Face: Un singolo GGUF non garantisce vision: possono mancare proiettore, template e metadati multimodali. Per il visioning, preferire un manifest ufficiale Ollama o una procedura di import esplicitamente supportata dall'architettura.

4. Installazione Docker di Open WebUI

4.1 Creazione della rete condivisa

sudo docker network create openwebui-agents sudo docker network inspect openwebui-agents

Se la rete esiste già, il messaggio di errore può essere ignorato dopo averne verificato il nome.

4.2 Secret persistente e container CUDA

WEBUI_SECRET_KEY="$(openssl rand -hex 32)" printf 'Salvare WEBUI_SECRET_KEY in un password manager: %s\n'
"$WEBUI_SECRET_KEY"

sudo docker run -d
--name open-webui
--restart unless-stopped
--network openwebui-agents
--gpus all
-p 3000:8080
--add-host=host.docker.internal:host-gateway
-e OLLAMA_BASE_URL=http://host.docker.internal:11434
-e WEBUI_SECRET_KEY="$WEBUI_SECRET_KEY"
-v open-webui:/app/backend/data
ghcr.io/open-webui/open-webui:cuda

La documentazione ufficiale indica Docker come percorso raccomandato e richiede --gpus all con l'immagine :cuda [7]. Per installazioni riproducibili è preferibile sostituire il tag mobile con una release fissata, per esempio v0.11.0-cuda.

4.3 Verifiche

sudo docker ps --format 'table {{.Names}}\t{{.Image}}\t{{.Ports}}' sudo docker logs --tail 100 open-webui sudo docker exec open-webui
curl -sS http://127.0.0.1:8080/api/version sudo docker exec open-webui
curl -sS http://host.docker.internal:11434/api/version

Aprire http://localhost:3000 e creare il primo account, che diventa amministratore.

Versione Open WebUI: Nel container 0.11.0, importlib.metadata può restituire PackageNotFoundError per open-webui. Usare l'endpoint /api/version, che nel collaudo ha risposto {"version":"0.11.0","deployment_id":""}.

4.4 Configurazione del modello

Aprire Admin Panel -> Settings -> AI -> Models e selezionare qwen3.8:27b.

Impostare Function Calling su Native, non Legacy.

Abilitare Vision, Builtin Tools e Web Search nelle capacità del modello.

Nei parametri avanzati impostare num_ctx a 60.000 per il profilo agente e max_tokens a 4.096.

Salvare e iniziare una nuova chat; una chat già lunga conserva la propria cronologia.

5. Configurazione e collaudo del visioning

5.1 Procedura

Selezionare qwen3.8:27b.

Premere + nella casella del messaggio e allegare vision-test.png.

Non chiedere a OpenTerminal di leggere il file: il modello deve osservare direttamente l'immagine.

Inviare il prompt di collaudo seguente.

Analizza esclusivamente l'immagine allegata. Dimmi il numero grande mostrato e descrivi, da sinistra verso destra, le tre figure indicando forma e colore.

5.2 Esito

Risposta corretta: 4827; cerchio rosso; quadrato/rettangolo blu; triangolo verde. Il modello ha riconosciuto anche l'intestazione COLLADUO VISION. Il test dimostra che l'immagine è stata effettivamente inviata al modello multimodale.

Figura 1 - Collaudo vision riuscito in Open WebUI.

6. Configurazione e collaudo della ricerca web

6.1 Configurazione globale

Aprire Admin Panel -> Settings -> Tools -> Web Search.

Abilitare Web Search e scegliere un motore supportato.

Inserire URL e/o credenziali richieste dal provider e salvare.

Nel modello abilitare Web Search come capability e, se desiderato, come Default Feature.

Verificare che Function Calling sia Native: search_web e fetch_url sono strumenti agentici [8].

6.2 Prompt di collaudo

Usa la ricerca web. Consulta una sola fonte ufficiale: la pagina GitHub Releases di Open WebUI. Rispondi solamente con versione più recente, data e URL diretto.

6.3 Esito e workaround

Il primo tentativo, con Web Search e OpenTerminal contemporaneamente attivi in una chat già articolata, ha eseguito search_web e fetch_url ma si è chiuso con no user query found in messages. In una nuova chat, con soltanto Web Search attivo, il test è riuscito: versione v0.11.0, data 27 luglio 2026 ore 09:30 e URL ufficiale GitHub.

Regola pratica: Per un test pulito abilita un solo gruppo di strumenti alla volta. Se il modello ha già prodotto molte chiamate tool, apri una nuova chat e ripeti una richiesta breve e vincolata a una fonte ufficiale.

Figura 2 - Web search agentica riuscita con search_web e fetch_url.

7. Installazione Docker e collegamento di OpenTerminal

7.1 Generazione della chiave

TERMINAL_API_KEY="$(openssl rand -hex 32)" printf 'Lunghezza chiave: %s caratteri\n' "${#TERMINAL_API_KEY}"

Il risultato atteso è 64 caratteri esadecimali. Conservare la chiave prima di chiudere la shell.

7.2 Avvio del container

sudo docker run -d
--name open-terminal
--restart unless-stopped

--network openwebui-agents
-v open-terminal:/home/user
-e OPEN_TERMINAL_API_KEY="$TERMINAL_API_KEY"
ghcr.io/open-webui/open-terminal

Il volume rende persistenti file, repository e ZIP. Non è necessario pubblicare la porta 8000 se Open WebUI e OpenTerminal condividono la rete Docker [9].

7.3 Collegamento in Open WebUI

Aprire Admin Panel -> Settings -> Integrations -> Open Terminal.

Aggiungere una connessione System, non External Tools.

URL: http://open-terminal:8000.

Auth Type: Bearer; API Key: la chiave generata.

Salvare, verificare e attivare la connessione.

Nella chat selezionare OpenTerminal dall'icona a forma di nuvola/terminale.

7.4 Verifiche di rete e servizio

sudo docker network inspect openwebui-agents
--format '{{range .Containers}}{{println .Name}}{{end}}'

sudo docker exec open-webui getent hosts open-terminal

sudo docker exec open-webui
curl -sS -w '\nHTTP %{http_code}\n'
http://open-terminal:8000/health

sudo docker exec open-terminal python -c
'import importlib.metadata as m; print(m.version("open-terminal"))'

7.5 Caso reale: DNS non aggiornato

Open WebUI era nato sulla rete bridge e OpenTerminal sulla rete openwebui-agents. Anche dopo aver collegato i container, il backend continuava a registrare Domain name not found per open-terminal. La soluzione è stata collegare entrambi alla stessa rete e riavviare Open WebUI.

sudo docker network connect openwebui-agents open-webui sudo docker restart open-webui

sudo docker exec open-webui getent hosts open-terminal sudo docker exec open-webui
curl -sS http://open-terminal:8000/health

Perché il riavvio conta: Il container risultava formalmente nella nuova rete, ma i processi già avviati in Open WebUI continuavano a usare lo stato DNS precedente. Il riavvio ha ricreato resolver e connessioni.

8. Collaudo progetto, ZIP, server HTTP e download

8.1 Prompt riproducibile

Usa OpenTerminal e crea /home/user/collaudo-open-terminal. Crea index.html, styles.css, script.js, README.md, sistema.txt e vision-test.png. La pagina deve mostrare informazioni di sistema e un

pulsante JavaScript. L'immagine deve contenere il numero 4827, un cerchio rosso, un quadrato blu e un triangolo verde.

Quindi:

1. verifica che tutti i file esistano;
2. avvia un server HTTP sulla porta 8080;
3. verifica la homepage con curl http://127.0.0.1:8080;
4. crea /home/user/collaudo-open-terminal.zip;
5. verifica lo ZIP con unzip -t;
6. calcola il SHA-256 dello ZIP;
7. mostrami index.html e vision-test.png tramite display_file. Non limitarti a scrivere il codice nella risposta: crea e verifica i file.

8.2 Risultati ottenuti

8.3 Verifica indipendente dall'host

sudo docker exec open-terminal
ls -lh /home/user/collaudo-open-terminal.zip

sudo docker exec open-terminal
unzip -t /home/user/collaudo-open-terminal.zip

sudo docker exec open-terminal
sha256sum /home/user/collaudo-open-terminal.zip

Per scaricare, aprire il pannello File, tornare alla radice visuale /home/user, aggiornare e usare l'icona download sullo ZIP. Il modello non inserisce necessariamente un link cliccabile nella risposta.

Figura 3 - Collaudo OpenTerminal: curl, ZIP, integrità, checksum e visualizzazione.

8.4 Server HTTP e pannello Ports

OpenTerminal dispone di rilevamento delle porte e proxy HTTP [10]. Nel collaudo specifico, il server Python rispondeva e il proxy funzionava, ma GET /ports restituiva una lista vuota; il pannello mostrava No servers detected.

Server diretto

curl http://127.0.0.1:8080/ # HTTP 200

API OpenTerminal

GET /ports # {"ports":[]}, HTTP 200 GET /proxy/8080/ # HTTP 200

Figura 4 - Anomalia osservata: il server era attivo ma Ports non lo elencava.

8.5 Tre modalità di anteprima

Metodo normale: avviare il server dal Terminal integrato e aprire la voce che compare in Ports.

Metodo stabile: ricreare OpenTerminal pubblicando -p 127.0.0.1:18080:8080 e chiedere al modello di usare sempre la porta 8080 con bind 0.0.0.0; aprire http://127.0.0.1:18080.

Metodo indipendente: scaricare lo ZIP, estrarlo in una directory temporanea e servirlo direttamente dall'host.

cd ~/Downloads PREVIEW_DIR="$(mktemp -d)" unzip -q collaudo-open-terminal.zip -d "$PREVIEW_DIR" python3 -m http.server 18080
--bind 127.0.0.1
--directory "$PREVIEW_DIR/collaudo-open-terminal"

Aprire nel browser:

http://127.0.0.1:18080/

9. Troubleshooting basato sui problemi incontrati

9.1 Diagnostica contesto e tool calling

ollama ps

sudo journalctl -u ollama --since "30 minutes ago" --no-pager | grep -Ei 'truncat|context|token|prompt'

sudo docker inspect open-webui
--format '{{range .Config.Env}}{{println .}}{{end}}' | grep -E '^CHAT_RESPONSE_MAX_TOOL_CALL_(ITERATIONS|RETRIES)='

In Open WebUI 0.11.0 il limite predefinito delle iterazioni tool è molto superiore alle tre chiamate osservate. Se la variabile non è impostata, il fermarsi dopo pochi strumenti è più spesso una decisione del modello o un problema di contesto, non un hard cap.

9.2 Diagnostica OpenTerminal

sudo docker logs --since 5m open-webui 2>&1 | grep -E 'terminals|files|ERROR|WARNING' | tail -n 100

sudo docker logs --since 5m open-terminal 2>&1 | grep -E 'files|401|403|ERROR' | tail -n 100

sudo docker exec open-webui
curl -sS http://open-terminal:8000/health

10. Manutenzione, backup e sicurezza

10.1 Volumi persistenti

sudo docker inspect open-webui
--format '{{range .Mounts}}{{println .Type .Source "->" .Destination}}{{end}}'

sudo docker inspect open-terminal
--format '{{range .Mounts}}{{println .Type .Source "->" .Destination}}{{end}}'

La rimozione del container non elimina i volumi nominati, purché non si esegua docker volume rm. Prima di aggiornamenti importanti effettuare un backup.

10.2 Backup essenziale

mkdir -p "$HOME/docker-backup"

sudo docker run --rm
-v open-webui:/data:ro
-v "$HOME/docker-backup":/backup
alpine sh -c 'tar czf /backup/open-webui-data.tgz -C /data .'

sudo docker run --rm
-v open-terminal:/data:ro
-v "$HOME/docker-backup":/backup
alpine sh -c 'tar czf /backup/open-terminal-home.tgz -C /data .'

10.3 Aggiornamenti

Per Ollama rieseguire lo script ufficiale. Per i container, effettuare il pull dell'immagine, rimuovere esclusivamente il container e ricrearlo con gli stessi volumi, rete, chiavi e parametri. Non rimuovere i volumi.

curl -fsSL https://ollama.com/install.sh | sh sudo systemctl restart ollama

sudo docker pull ghcr.io/open-webui/open-webui:cuda sudo docker pull ghcr.io/open-webui/open-terminal

10.4 Principi di sicurezza

Usare OpenTerminal in Docker, non direttamente sull'host, salvo necessità consapevoli.

Non montare /var/run/docker.sock: conferirebbe all'agente controllo equivalente a root sul sistema host [11].

Limitare OpenTerminal a utenti fidati e disabilitare la connessione quando non serve.

Ruotare periodicamente OPEN_TERMINAL_API_KEY e mantenere stabile WEBUI_SECRET_KEY.

Pubblicare le anteprime solo su 127.0.0.1, non su tutte le interfacce, se non è necessario l'accesso LAN.

Eseguire backup prima di aggiornare immagini e database.

11. Checklist finale di accettazione

Controllo rapido post-riavvio

systemctl is-active ollama curl -sS http://127.0.0.1:11434/api/version sudo docker ps --format 'table {{.Names}}\t{{.Status}}\t{{.Ports}}' sudo docker network inspect openwebui-agents --format '{{range .Containers}}{{println .Name}}{{end}}' sudo docker exec open-webui curl -sS http://open-terminal:8000/health

Fonti ufficiali

[1] Ollama - Installazione Linux. https://docs.ollama.com/linux

[2] Ollama - FAQ e variabili systemd. https://docs.ollama.com/faq

[3] Ollama - Context length. https://docs.ollama.com/context-length

[4] Ollama - Vision. https://docs.ollama.com/capabilities/vision

[5] Ollama - Tool calling. https://docs.ollama.com/capabilities/tool-calling

[6] Ollama - Modelfile reference. https://docs.ollama.com/modelfile

[7] Open WebUI - Quick Start Docker. https://docs.openwebui.com/getting-started/quick-start/

[8] Open WebUI - Agentic Web Search.
https://docs.openwebui.com/features/chat-conversations/web-search/agentic-search/

[9] Open WebUI - Installazione OpenTerminal.
https://docs.openwebui.com/features/open-terminal/setup/installation/

[10] Open WebUI - Build and Preview Websites.
https://docs.openwebui.com/features/open-terminal/use-cases/web-development/

[11] Open WebUI - Sicurezza OpenTerminal.
https://docs.openwebui.com/features/open-terminal/advanced/security/

[12] OpenTerminal - Port detection source.
https://github.com/open-webui/open-terminal/blob/main/open_terminal/utils/port.py

Nota conclusiva

La configurazione descritta è stata collaudata il 18 agosto 2026. Le interfacce e i tag delle immagini possono cambiare: prima di un'installazione futura verificare le release ufficiali, conservare i volumi e provare gli aggiornamenti in modo reversibile.

| Voce | Valore |

| --- | --- |

| Sistema | Linux Mint 22.2 |

| GPU | NVIDIA RTX 5060 Ti, 16 GB VRAM |

| RAM | 32 GB |

| Ollama | Aggiornato da 0.32.11 a 0.32.14 |

| Modello | qwen3.8:27b, circa 18 GB |

| Open WebUI | 0.11.0, immagine ghcr.io/open-webui/open-webui:cuda |

| OpenTerminal | 0.11.35, immagine ghcr.io/open-webui/open-terminal |

| Rete Docker | openwebui-agents |

| Voce | Valore |

| --- | --- |

| Browser -> Open WebUI | http://localhost:3000 |

| Open WebUI -> Ollama | http://host.docker.internal:11434 |

| Open WebUI -> OpenTerminal | http://open-terminal:8000 |

| Dati Open WebUI | Volume open-webui montato in /app/backend/data |

| Workspace agente | Volume open-terminal montato in /home/user |

| Voce | Valore |

| --- | --- |

| Chat e vision brevi | 16.384-32.768 token |

| Web e OpenTerminal | 60.000-64.000 token, se RAM e prestazioni lo consentono |

| Output | max_tokens/num_predict 4.096; aumentare a 8.192 per task lunghi |

| Effetto osservato | qwen3.8:27b: 35% CPU / 65% GPU, CONTEXT 60.384 |

| Test | Esito | Evidenza |

| --- | --- | --- |

| Creazione file | OK | 6 file più directory progetto |

| Server HTTP | OK | curl su 127.0.0.1:8080, HTTP 200 |

| ZIP | OK | 26 KB, /home/user/collaudo-open-terminal.zip |

| Integrità | OK | unzip -t: No errors detected |

| SHA-256 | OK | 1c0ce31000ea170bc10e9ee2f1dbe814586f95f75762dfae7adb6493f8bbcb47 |

| Download | OK | File browser Open WebUI |

| Sintomo | Causa probabile | Correzione |

| --- | --- | --- |

| Pull 412 | Ollama non abbastanza recente | Rieseguire install.sh, riavviare ollama, controllare ollama -v |

| Verify OpenTerminal HTTP 400 | URL, chiave o formato connessione | Usare integrazione Open Terminal, URL interno, Bearer e chiave identica |

| Health 200 ma UI non connette | DNS del container non aggiornato | Rete condivisa, getent hosts, quindi docker restart open-webui |

| PackageNotFoundError open-webui | Pacchetto non espone metadata Python | Usare GET /api/version |

| Il modello si ferma prima dello ZIP | Contesto insufficiente o ciclo tool interrotto | Nuova chat o prompt Continua; num_ctx 60K; max_tokens 4K-8K |

| Nessun link allo ZIP | Download non inserito inline | Usare il pannello File e l'icona download |

| No servers detected | Rilevamento /ports vuoto | Verificare server/proxy; usare port mapping o anteprima dallo ZIP |

| no user query found in messages | Interazione fra più tool o stato chat | Nuova chat, solo Web Search, prompt breve |

| Test | Esito | Evidenza |

| --- | --- | --- |

| Ollama systemd | OK | Servizio attivo, API 11434 raggiungibile |

| Pull ed esecuzione | OK | qwen3.8:27b, 18 GB |

| Capacità | OK | completion, vision, tools, thinking |

| Open WebUI | OK | 0.11.0 su http://localhost:3000 |

| Vision | OK | 4827 e forme/colori riconosciuti |

| Web Search | OK | search_web, fetch_url, fonti ufficiali |

| OpenTerminal | OK | 0.11.35, health 200, file browser e terminale |

| Progetto e ZIP | OK | 6 file, 26 KB, unzip -t e SHA-256 |

| HTTP | OK | Server diretto 200 e proxy OpenTerminal 200 |

| Ports automatico | NOTA | Lista vuota nel caso osservato; disponibili workaround |

Immagini importate

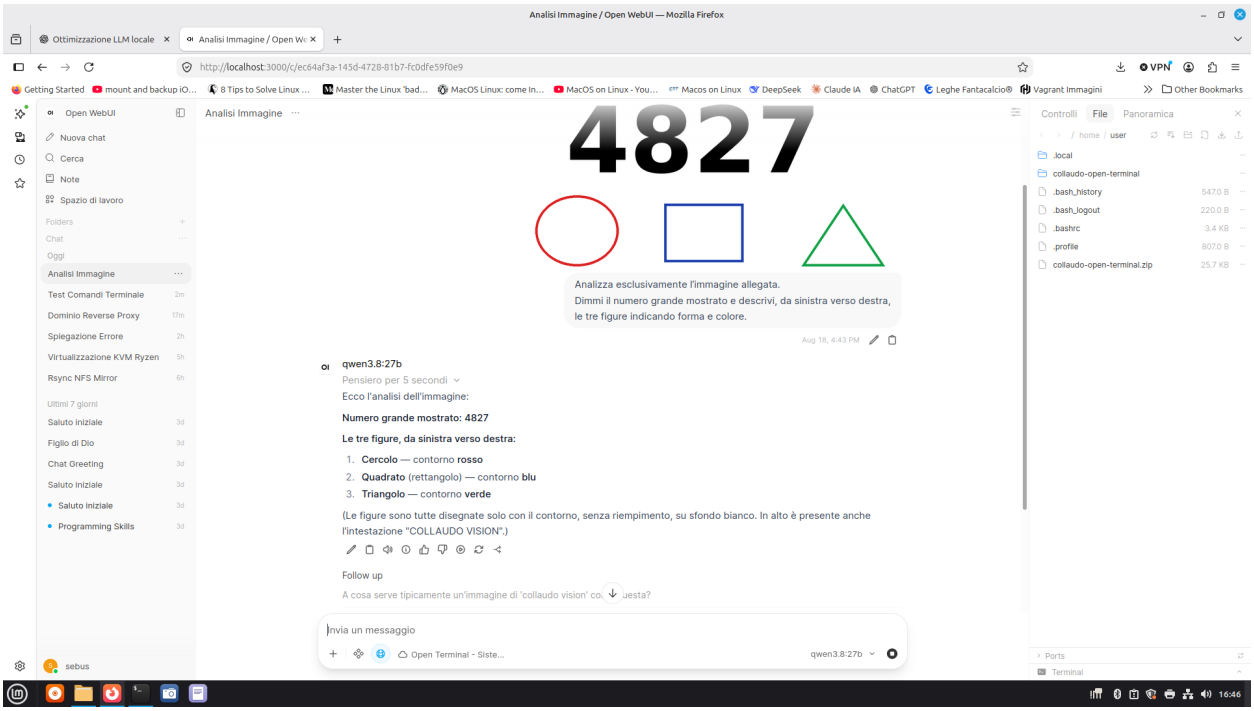

Immagine importata

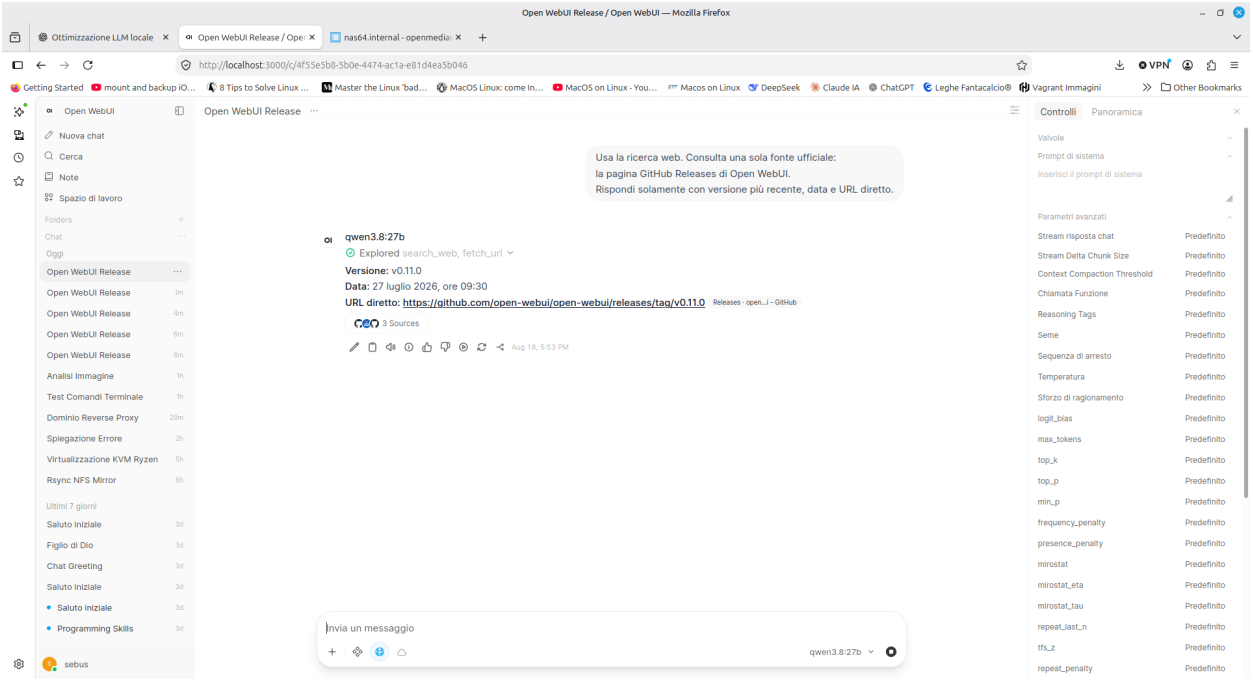

Immagine importata

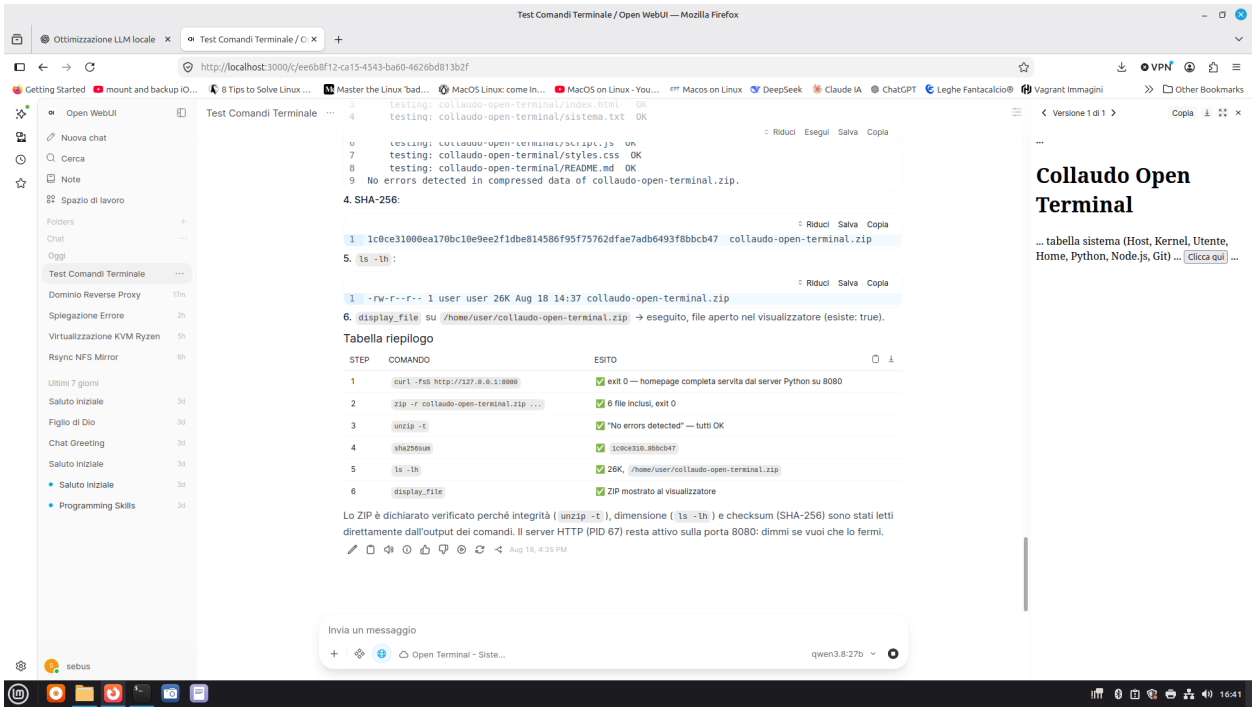

Immagine importata

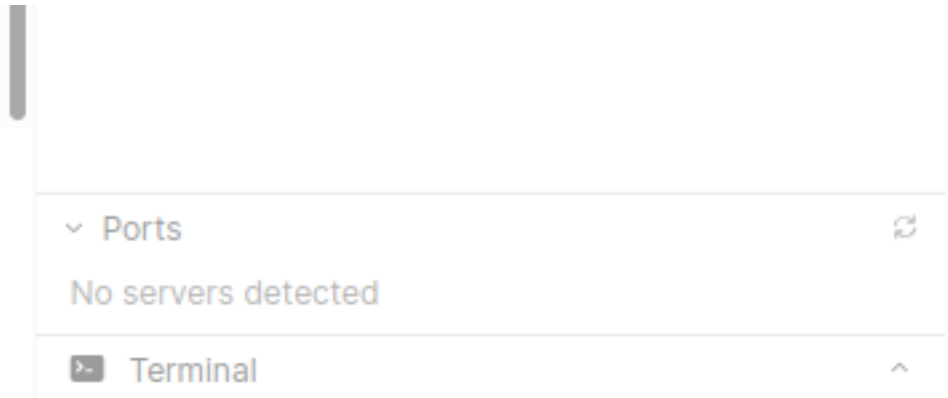

Immagine importata