Implementazione di un Large Language Model locale con strumenti open source su piattaforma Microsoft Windows

Lecture note tecnica basata sullo stack Ollama e Open WebUI

Versione 1.1, agosto 2026

Sommario

1. Introduzione e obiettivi didattici

Il presente documento costituisce una lecture note tecnica dedicata all'implementazione di un Large Language Model (LLM) eseguito interamente in locale su una postazione di lavoro Microsoft Windows, mediante software open source. Lo stack di riferimento è composto da Ollama, runtime di inference basato sul motore llama.cpp, e da Open WebUI, interfaccia web self-hosted distribuita come container Docker. Il percorso è organizzato in fasi progressive e verificabili, corredate da comandi riproducibili, schemi architetturali e indicazioni operative orientate a un pubblico di professionisti della sicurezza informatica.

1.1 Obiettivi formativi

Al termine del modulo il discente sarà in grado di:

- installare e configurare il runtime Ollama su Windows, gestendo modelli in formato GGUF e le relative quantization;
- effettuare il deployment di Open WebUI tramite Docker Desktop e amministrarne utenti e impostazioni;
- implementare una pipeline di Retrieval-Augmented Generation (RAG) su documenti locali;
- integrare il modello in script PowerShell e Python attraverso la REST API locale e l'endpoint OpenAI-compatible;
- valutare il threat model del deployment e applicare le misure di hardening appropriate;
- eseguire un fine-tuning parameter-efficient (LoRA/QLoRA) e distribuire il modello risultante in Ollama.

1.2 Motivazioni del deployment locale

Per i professionisti della cyber security l'esecuzione locale di un LLM presenta vantaggi che non si limitano alla riduzione dei costi:

- **Confidenzialità e data residency:** prompt, documenti e conversazioni non transitano verso provider esterni; è quindi possibile analizzare artefatti sensibili (log, report di incident response, evidenze forensi contenenti PII) senza alcuna uscita dal perimetro aziendale.
- **Operatività offline:** una volta scaricati i modelli, il sistema opera anche in ambienti air-gapped o in laboratori isolati.
- **Controllo e auditabilità:** versioni dei modelli, parametri e log restano sotto il pieno controllo dell'organizzazione, a beneficio di riproducibilità e compliance.
- **Assenza di vincoli contrattuali:** nessun rate limit e nessuna clausola di riutilizzo dei dati per finalità di training da parte di terzi.

1.3 Prerequisiti del discente

Si assumono familiarità con PowerShell e con la riga di comando, nozioni di base su container Docker e networking TCP/IP, nonché la disponibilità di diritti di amministratore locale sulla postazione di lavoro.

2. Concetti fondamentali

2.1 LLM e inference

Un LLM è una rete neurale di tipo transformer addestrata a predire il token successivo di una sequenza. La presente trattazione non riguarda il training, bensì la sola inference: il caricamento in memoria dei pesi di un modello pre-addestrato e la generazione autoregressiva del testo, token dopo token. Le prestazioni di inference si misurano tipicamente in token al secondo (t/s) e dipendono dalla dimensione del modello, dalla quantization adottata e dall'hardware disponibile.

2.2 Parametri, quantization e formato GGUF

I modelli si classificano per numero di parametri (7B, 8B, 14B, 70B, dove B indica miliardi). I pesi, nativamente in formato FP16/BF16, possono essere compressi mediante quantization, ossia la riduzione della precisione numerica (ad esempio a 4 bit), con un risparmio di memoria molto significativo a fronte di una perdita qualitativa contenuta. Il formato di riferimento per l'inference locale è GGUF, impiegato dal motore llama.cpp su cui Ollama è costruito. Le varianti di quantization sono identificate da sigle quali `q4_K_M`, `q5_K_M`, `q8_0`: come regola pratica, un modello in `q4_K_M` occupa circa 0,6 GB per miliardo di parametri, cui si aggiunge l'overhead della KV cache, proporzionale alla lunghezza del contesto.

2.3 Context window

La context window (parametro `num_ctx` in Ollama) definisce il numero massimo di token elaborabili in una singola conversazione, comprendendo prompt di sistema, documenti recuperati via RAG e risposta generata. Valori elevati incrementano il consumo di memoria: è buona prassi dimensionarla sul caso d'uso (4.096–8.192 token per impieghi generali; valori superiori per l'analisi documentale).

2.4 Architettura della soluzione

La Figura 1 illustra l'architettura adottata: Ollama è eseguito come servizio Windows nativo ed espone una REST API su `127.0.0.1:11434`; Open WebUI, in esecuzione come container Docker, funge da front-end multi-utente e raggiunge Ollama tramite l'alias `host.docker.internal`; l'utente interagisce via browser su `localhost:3000`. L'intero flusso dei dati rimane confinato alla postazione: la connettività Internet è richiesta esclusivamente per il download iniziale dei modelli.

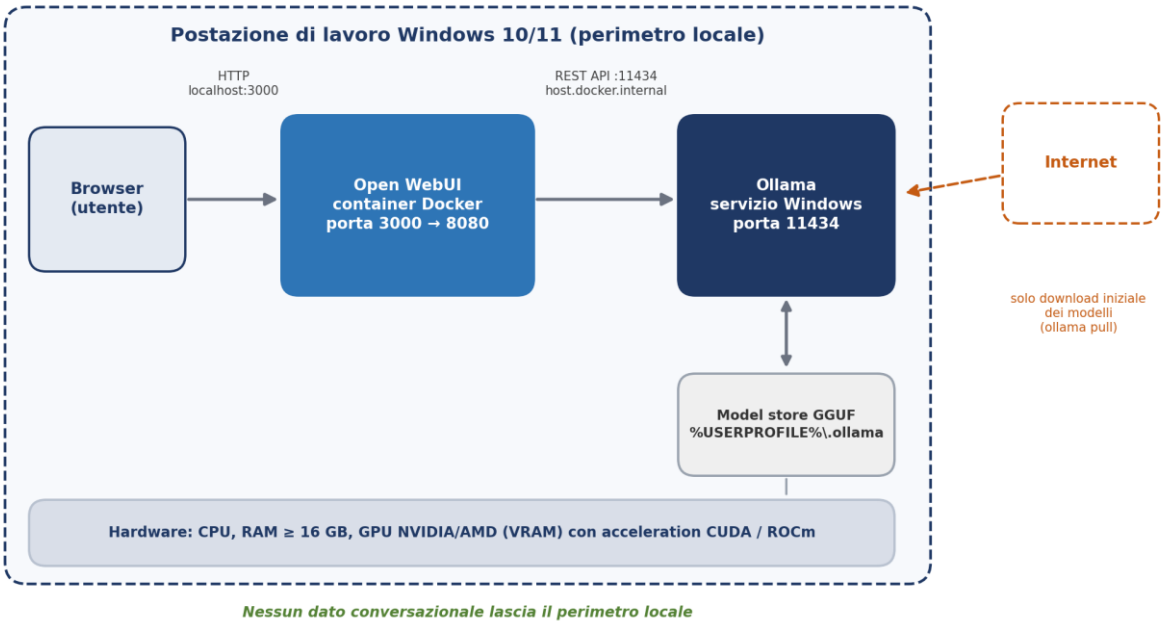

Figura 1. Architettura della soluzione: componenti, porte e perimetro locale.

3. Requisiti hardware e software

La tabella seguente fornisce un dimensionamento orientativo. In assenza di GPU l'inference avviene su CPU, con prestazioni ridotte ma adeguate a finalità didattiche; in presenza di una GPU NVIDIA (CUDA) o AMD supportata (ROCm), Ollama effettua automaticamente l'offload dei layer sulla VRAM disponibile.

Scenario	RAM / GPU (VRAM)	Modelli indicativi	Prestazioni attese
CPU-only	16 GB RAM, nessuna GPU	3B–8B in q4 (es. Llama 3.1 8B)	3–10 t/s: adeguate a test e didattica
GPU entry-level	16 GB RAM, 8 GB VRAM (es. RTX 4060)	7–8B in q4/q5	Fluide (30–60 t/s) sui modelli 8B
GPU mid-range	32 GB RAM, 12–16 GB VRAM (es. RTX 4070 Ti Super)	Fino a 14B; 27–32B in q4 con offload parziale	Fluide sui 14B; utilizzabili i 27–32B
GPU high-end	64 GB RAM, 24 GB VRAM (es. RTX 3090/4090)	27–32B in q4; 70B in q4 con offload su RAM	Idonee a un impiego professionale continuativo

Sul versante software sono richiesti:

- Windows 10 (22H2) o Windows 11 a 64 bit, con aggiornamenti correnti;
- driver GPU aggiornati (NVIDIA per CUDA; per AMD verificare il supporto ROCm della specifica scheda);
- virtualizzazione hardware abilitata nel firmware UEFI (Intel VT-x / AMD-V), necessaria per WSL2;
- Docker Desktop con backend WSL2;
- PowerShell 5.1 o 7.x;
- facoltativamente, per il fine-tuning (cap. 10): distribuzione Ubuntu su WSL2 e toolchain CUDA.

> **Nota:** Salvo diversa indicazione, tutti i comandi riportati nel documento si intendono da eseguire in una sessione PowerShell sull'host Windows; i comandi relativi al capitolo 10 vanno eseguiti nella shell Linux di WSL2.

4. Fase 1: installazione di Ollama

La Figura 2 riassume il percorso operativo complessivo, dalla verifica dei requisiti sino alle attività avanzate trattate nei capitoli successivi.

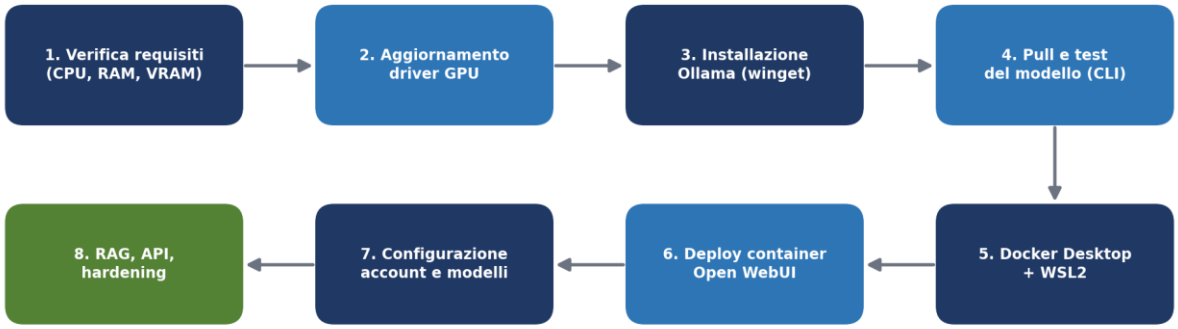

Figura 2. Flusso operativo complessivo dell'implementazione.

4.1 Procedura di installazione

1. Scaricare il pacchetto `OllamaSetup.exe` dal sito ufficiale (`https://ollama.com/download/windows`) ed eseguirlo; in alternativa, procedere tramite package manager:

```
winget install --id Ollama.Ollama
```

2. Completare la procedura guidata: l'installazione avviene a livello utente e registra Ollama come applicazione in background, con icona nell'area di notifica e avvio automatico all'accesso.
3. Al termine, aprire una nuova sessione PowerShell (necessaria per il refresh della variabile `PATH`) e verificare versione e stato del servizio:

```
ollama --version
Invoke-RestMethod http://localhost:11434/api/version
```

4. Verificare che il listener sia attivo esclusivamente sull'interfaccia di loopback:

```
netstat -ano | findstr 11434    # atteso: 127.0.0.1:11434 in LISTENING
```

4.2 Variabili d'ambiente rilevanti

Il comportamento del servizio è governato da variabili d'ambiente utente (Impostazioni → Sistema → Informazioni → Impostazioni di sistema avanzate → Variabili d'ambiente); dopo ogni modifica è necessario riavviare Ollama dall'icona nell'area di notifica.

Variabile	Funzione	Valore predefinito
`OLLAMA_HOST`	Interfaccia e porta di binding della REST API	127.0.0.1:11434
`OLLAMA_MODELS`	Percorso dello store locale dei modelli	%USERPROFILE%\.ollama\models
`OLLAMA_KEEP_ALIVE`	Permanenza del modello in memoria dopo l'ultima richiesta	5m
`OLLAMA_NUM_PARALLEL`	Richieste servite in parallelo per modello	automatico
`OLLAMA_MAX_LOADED_MODELS`	Numero massimo di modelli caricati simultaneamente	automatico
`OLLAMA_ORIGINS`	Origini aggiuntive consentite per richieste cross-origin (CORS)	localhost

> **Attenzione:** Non impostare `OLLAMA_HOST=0.0.0.0` per esporre l'API sulla rete senza avere prima applicato le misure descritte nel capitolo 9: l'API di Ollama non prevede autenticazione nativa.

5. Fase 2: gestione dei modelli

5.1 Comandi essenziali

Il ciclo di vita dei modelli è gestito interamente da CLI. Il download avviene dalla model library ufficiale (`ollama.com/library`); il tag specifica dimensione e quantization e, ove omesso, viene scaricata la variante predefinita (tipicamente `q4_K_M`).

```
ollama pull llama3.1:8b            # download del modello
ollama run llama3.1:8b --verbose   # sessione interattiva con statistiche (t/s)
ollama list                        # modelli presenti nello store locale
ollama ps                          # modelli in memoria e ripartizione CPU/GPU
ollama show llama3.1:8b            # metadati: parametri, quantization, licenza
ollama rm <modello>                # rimozione dallo store
```

5.2 Selezione del modello

La scelta dipende da hardware disponibile, lingua di lavoro e tipologia di task. A titolo orientativo, alla data di stesura risultano di interesse professionale:

Famiglia di modelli	Dimensioni tipiche	Ambito d'impiego
Llama 3.1 / 3.3 (Meta)	8B (solo 3.1) e 70B	Uso generale; buon equilibrio qualità/risorse
Qwen 2.5 / Qwen 3 (Alibaba)	7B–32B	Ottimo supporto multilingua e coding

Famiglia di modelli	Dimensioni tipiche	Ambito d'impiego
Mistral / Ministral	7B–8B	Compatti ed efficienti; varianti con licenza Apache 2.0
Phi-4 (Microsoft)	14B	Elevata qualità di ragionamento a parità di dimensioni
DeepSeek-R1 distill	7B–32B	Task di reasoning esteso
nomic-embed-text, bge-m3	< 1B	Embedding model per pipeline RAG (cap. 7)

> **Nota:** L'offerta di modelli evolve con cadenza mensile: le indicazioni riportate sono valide alla data di stesura; si raccomanda la consultazione della library ufficiale per lo stato corrente.

5.3 Personalizzazione tramite Modelfile

Il Modelfile è un descrittore testuale, concettualmente affine a un Dockerfile, che consente di derivare un modello personalizzato definendo system prompt e parametri di inference. L'esempio seguente predispone un assistente per attività di analisi SOC:

```
FROM llama3.1:8b
PARAMETER temperature 0.2
PARAMETER num_ctx 8192
SYSTEM """Sei un assistente tecnico per analisti di sicurezza.
Rispondi in italiano, in modo conciso e strutturato.
Se un'informazione non è verificabile, dichiaralo esplicitamente."""
```

```
ollama create soc-assistant -f .\Modelfile
ollama run soc-assistant
```

6. Fase 3: deployment di Open WebUI

Open WebUI fornisce un'interfaccia conversazionale multi-utente, la gestione delle knowledge base per il RAG, la cronologia delle conversazioni e un pannello di amministrazione con role-based access control (RBAC). La modalità di distribuzione raccomandata è il container Docker.

6.1 Installazione del container

1. Installare Docker Desktop per Windows selezionando il backend WSL2; al primo avvio confermare l'eventuale installazione o aggiornamento di WSL2 (`wsl --update`).
2. Verificare il corretto funzionamento del runtime:

```
docker version
docker run --rm hello-world
```

3. Avviare il container di Open WebUI:

```
docker run -d -p 127.0.0.1:3000:8080 `
```

```
  -e OLLAMA_BASE_URL=http://host.docker.internal:11434 `
  -v open-webui:/app/backend/data `
  --name open-webui --restart always `
  ghcr.io/open-webui/open-webui:main
```

4. Attendere il bootstrap iniziale (alcuni minuti al primo avvio) e collegarsi a `http://localhost:3000`.

Il significato delle opzioni impiegate è il seguente:

- `-p 127.0.0.1:3000:8080`: pubblica la porta interna del container esclusivamente sul loopback dell'host; in assenza dell'indirizzo esplicito, Docker esporrebbe la porta su tutte le interfacce di rete;
- `-e OLLAMA_BASE_URL=…`: indirizzo al quale il container raggiunge il servizio Ollama in esecuzione sull'host;
- `-v open-webui:/app/backend/data`: volume persistente contenente database applicativo, utenti, conversazioni e indici RAG;
- `--restart always`: riavvio automatico del container all'avvio di Docker Desktop.

6.2 Primo accesso e configurazione

1. Alla prima connessione creare l'account iniziale mediante “Sign up”: il primo utente registrato assume automaticamente il ruolo di administrator; le credenziali sono memorizzate esclusivamente nel volume locale.
2. Verificare in Admin Panel → Settings → Connections che l'endpoint di Ollama risulti configurato (`http://host.docker.internal:11434`) e che i modelli installati compaiano nel selettore.
3. Disabilitare le registrazioni spontanee (Admin Panel → Settings → General, opzione “Enable New Sign Ups”) oppure avviare il container con `-e ENABLE_SIGNUP=false`.
4. Eseguire un test funzionale selezionando un modello e inviando un prompt di prova.

> **Nota:** In alternativa a Docker, Open WebUI è installabile come pacchetto Python (`pip install open-webui`, quindi `open-webui serve`, con Python 3.11): opzione utile su postazioni prive di virtualizzazione, a fronte di un maggiore onere manutentivo.

7. Fase 4: RAG su documenti locali

7.1 Principio di funzionamento

Il Retrieval-Augmented Generation integra la conoscenza parametrica del modello con contenuti documentali aggiornati e verificabili. Come illustrato in Figura 3, in fase di indexing i documenti vengono segmentati in chunk, convertiti in vettori tramite un embedding model e memorizzati in un vector database (Open WebUI integra ChromaDB); a runtime la query dell'utente viene a sua volta vettorizzata, i chunk semanticamente più affini (top-k) vengono recuperati e concatenati al prompt, e il modello genera una risposta corredata di citazioni. Rispetto al fine-tuning, il RAG consente l'aggiornamento immediato della base di conoscenza e la tracciabilità delle fonti.

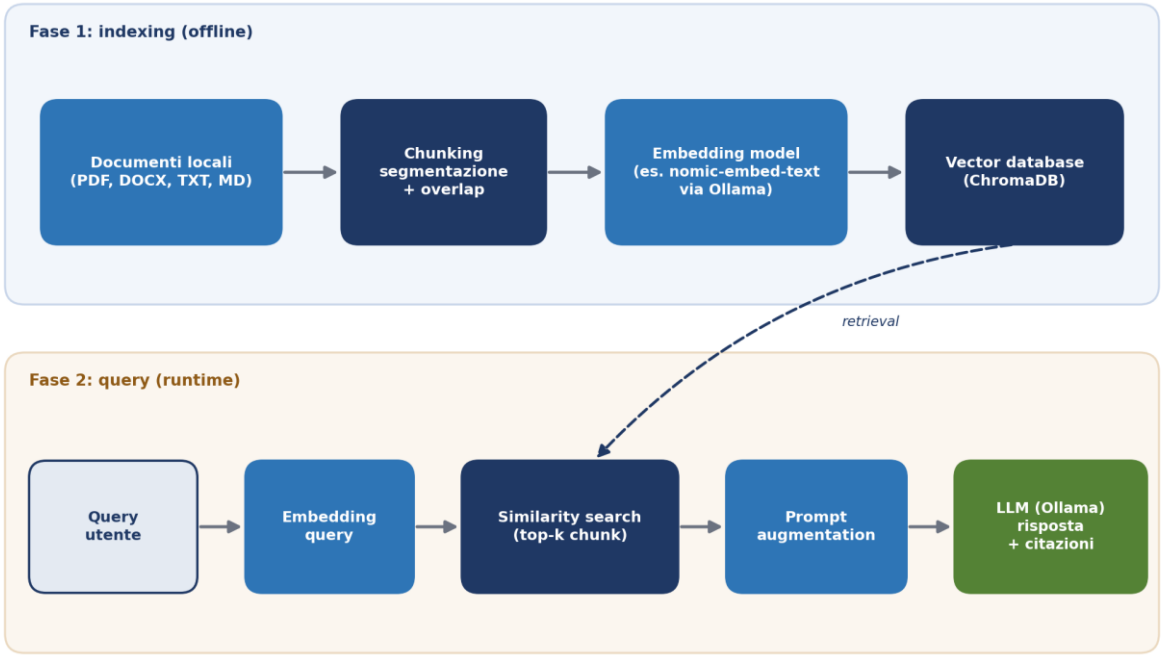

Figura 3. Pipeline RAG: fase di indexing e fase di query.

7.2 Configurazione dell'embedding model

1. Scaricare un embedding model dedicato:

```
ollama pull nomic-embed-text
```

2. In Admin Panel → Settings → Documents impostare “Ollama” come embedding engine e selezionare `nomic-embed-text`; adottare come punto di partenza i parametri predefiniti di chunking (chunk size 800–1.200 caratteri, overlap 10–15%, top-k 4–6).
3. Qualora l'embedding model venga sostituito in un momento successivo, procedere alla reindicizzazione delle knowledge base esistenti: i vettori prodotti da modelli differenti non sono tra loro compatibili.

7.3 Creazione e utilizzo della knowledge base

1. Da Workspace → Knowledge selezionare “+ Create a Knowledge Base”, assegnando nome e descrizione (ad esempio “Policy di sicurezza aziendali”).
2. Caricare i documenti (PDF, DOCX, TXT, Markdown): estrazione del testo, chunking e indicizzazione avvengono automaticamente.
3. In una nuova chat digitare il carattere `#` e selezionare la knowledge base per vincolare la risposta ai documenti caricati; in alternativa, allegare un singolo file alla conversazione.
4. Verificare che le risposte riportino le citazioni ai passaggi utilizzati e validarne la pertinenza.

7.4 Caso d'uso in ambito security

Un impiego ricorrente è l'interrogazione in linguaggio naturale di corpora interni: security policy, procedure di incident response, standard e linee guida, report di Cyber Threat Intelligence, documentazione post-

incident. La knowledge base rimane interamente on-premise ed è aggiornabile in ogni momento senza alcun riaddestramento del modello.

> **Attenzione:** I documenti indicizzati concorrono alla costruzione del prompt: contenuti ostili possono veicolare prompt injection (cfr. §9.5). Sottoporre a vetting le sorgenti prima dell'ingestione.

8. Fase 5: API locale e scripting

8.1 Endpoint disponibili

Ollama espone una REST API locale, priva di autenticazione, all'indirizzo `http://localhost:11434`. Gli endpoint principali sono `/api/generate` (completamento singolo), `/api/chat` (conversazione multi-turno), `/api/embed` (calcolo di embedding) e `/api/tags` (elenco dei modelli installati). È inoltre disponibile un endpoint OpenAI-compatible sul percorso `/v1`, che consente il riuso di librerie e strumenti esistenti mediante la semplice ridirezione verso l'host locale.

8.2 Esempio in PowerShell

```
$body = @{
    model  = "llama3.1:8b"
    prompt = "Elenca tre indicatori tipici di una campagna di phishing."
    stream = $false
} | ConvertTo-Json

Invoke-RestMethod -Uri "http://localhost:11434/api/generate" `
                  -Method Post -ContentType "application/json" `
                  -Body $body | Select-Object -ExpandProperty response
```

8.3 Esempio in Python (REST API nativa)

```
import requests

messages = [
    {"role": "system",
     "content": "Sei un analista SOC. Rispondi in modo conciso."},
    {"role": "user",
     "content": "Classifica questa riga di log come benigna o sospetta: <riga>"},
]

r = requests.post(
    "http://localhost:11434/api/chat",
    json={"model": "llama3.1:8b", "messages": messages, "stream": False},
    timeout=120,
)
print(r.json()["message"]["content"])
```

8.4 Esempio con SDK OpenAI-compatible

```
from openai import OpenAI
```

```
# La chiave è fittizia: l'endpoint locale non la verifica
client = OpenAI(base_url="http://localhost:11434/v1", api_key="ollama")
resp = client.chat.completions.create(
    model="llama3.1:8b",
    messages=[{"role": "user",
               "content": "Riassumi il principio del least privilege."}],
)
print(resp.choices[0].message.content)
```

8.5 Automazione di attività di supporto all'analisi

L'API locale abilita l'automazione di attività ripetitive: pre-classificazione (triage) di righe di log, sintesi di report, normalizzazione di indicatori di compromissione, generazione di bozze di documentazione. Il tutto senza che i dati lascino la postazione.

> **Attenzione:** L'output del modello è probabilistico e soggetto a hallucination: prevedere sempre la validazione umana e non collegare l'API ad azioni dispositive (blocchi, cancellazioni, risposte automatiche) prive di revisione.

9. Fase 6: security e privacy

9.1 Threat model del deployment

Un deployment locale riduce drasticamente la superficie di attacco legata a terze parti, ma non elimina i rischi. La Figura 4 sintetizza il threat model di riferimento: esposizione di rete dei servizi, supply chain dei modelli, prompt injection veicolata dai contenuti e protezione dei dati a riposo.

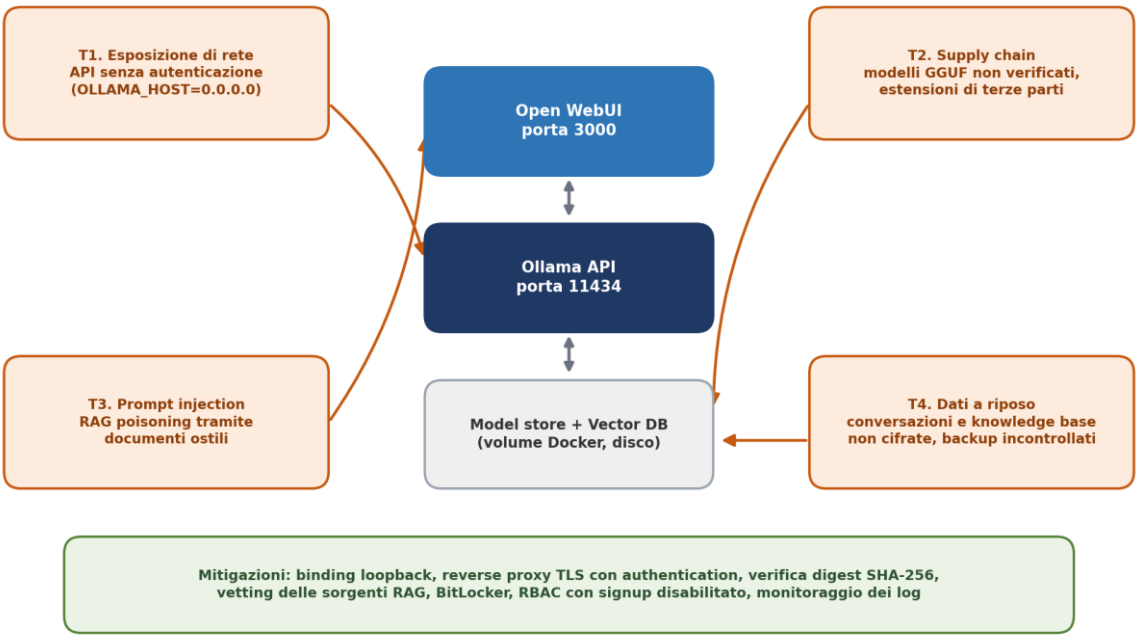

Figura 4. Threat model del deployment locale e relative mitigazioni.

9.2 Esposizione di rete

Nella configurazione predefinita Ollama è in ascolto esclusivamente su 127.0.0.1 e il container Open WebUI, avviato come indicato al §6.1, è anch'esso raggiungibile dal solo loopback. Ricerche condotte con motori quali Shodan hanno documentato migliaia di istanze Ollama esposte pubblicamente senza autenticazione, con la possibilità per chiunque di eseguire inference o manipolare i modelli: un errore di configurazione da evitare. Le misure raccomandate sono le seguenti:

- mantenere il binding predefinito su loopback e verificarlo periodicamente con `netstat`;
- ove sia necessario l'accesso da altri host, interporre un reverse proxy (nginx, Caddy, Traefik) con TLS e autenticazione, oppure una VPN (ad esempio WireGuard);
- definire regole di Windows Defender Firewall che blocchino le connessioni inbound verso le porte 11434 e 3000;
- ricordare che l'opzione Docker `-p` priva di indirizzo esplicito pubblica la porta su tutte le interfacce di rete dell'host.

9.3 Autenticazione e controllo degli accessi

L'API di Ollama non implementa autenticazione: qualsiasi processo locale può invocarla. Open WebUI applica invece autenticazione e ruoli (administrator, user, pending): dopo la creazione dell'account amministrativo è opportuno disabilitare le registrazioni spontanee, imporre password robuste e, in scenari multi-utente, valutare l'integrazione con SSO/OIDC aziendale.

9.4 Supply chain dei modelli e delle estensioni

I modelli sono artefatti binari e vanno trattati con la medesima diligenza riservata al software di terze parti:

- prediligere la library ufficiale di Ollama o repository Hugging Face riconducibili a publisher verificati;
- registrare i digest SHA-256 dei modelli approvati (esposti da `ollama show` e nei manifest dello store) e mantenerne un inventario interno;
- verificare le licenze d'uso (Llama Community License, Apache 2.0, Gemma Terms, ecc.) rispetto all'impiego previsto;
- considerare anche le licenze del software impiegato: Ollama è rilasciato con licenza MIT, mentre Open WebUI adotta dalla versione 0.6.6 (aprile 2025) una licenza di derivazione BSD-3 con clausola di protezione del branding, non approvata OSI e con esenzione dalla clausola per i deployment fino a 50 utenti; l'uso interno con branding invariato non ne risulta impattato, mentre la rimozione del branding o la redistribuzione richiedono una verifica di conformità;
- considerare che le estensioni di Open WebUI (tools, functions, pipelines) eseguono codice Python sul server: installarle esclusivamente da fonti fidate e previa code review.

9.5 Prompt injection e RAG poisoning

La prompt injection (voce LLM01 della OWASP Top 10 for LLM Applications) consiste nell'inserimento di istruzioni ostili nei contenuti elaborati dal modello: un documento caricato in knowledge base o un testo incollato in chat possono tentare di alterarne il comportamento (RAG poisoning). Le mitigazioni comprendono il vetting e il controllo di provenienza dei documenti, la separazione dei privilegi (il modello non deve disporre di capacità dispositive), la revisione sistematica dell'output e il principio di non considerare mai attendibili le istruzioni provenienti dai dati.

9.6 Protezione dei dati e conformità

Il trattamento interamente locale favorisce i principi di minimizzazione e di data residency rilevanti ai fini del GDPR; permangono tuttavia gli obblighi di protezione del dato a riposo: conversazioni, knowledge base e indici vettoriali risiedono nel volume Docker `open-webui`, i modelli nello store di Ollama. Le misure raccomandate includono la cifratura del disco con BitLocker, il controllo degli accessi alla postazione, una gestione di backup e retention coerente con le policy aziendali e la verifica delle impostazioni di telemetria degli strumenti di contorno (Ollama non trasmette dati di utilizzo; Docker Desktop dispone di telemetria configurabile).

9.7 Checklist di hardening

- binding dei servizi limitato a `127.0.0.1` e verificato con `netstat`;
- regole firewall inbound esplicite per le porte 11434 e 3000;
- signup disabilitato in Open WebUI e RBAC configurato;
- accesso remoto, ove richiesto, esclusivamente tramite reverse proxy TLS con autenticazione o VPN;
- inventario dei modelli con digest SHA-256 e verifica delle licenze;
- estensioni di terze parti sottoposte a code review;
- vetting delle sorgenti documentali destinate al RAG;
- BitLocker attivo e backup cifrati del volume `open-webui`;
- aggiornamento periodico di Ollama, Open WebUI e Docker Desktop.

10. Fase 7: fine-tuning con LoRA

10.1 Principio di funzionamento

Il full fine-tuning di un LLM richiede risorse fuori portata per una workstation; le tecniche di Parameter-Efficient Fine-Tuning (PEFT) rendono invece l'adattamento praticabile. LoRA (Low-Rank Adaptation) congela i pesi del modello base e addestra esclusivamente coppie di matrici a basso rango iniettate nelle proiezioni dei layer di attention: i parametri addestrabili si riducono a circa lo 0,1–1% del totale e l'esito è un adapter di pochi megabyte. QLoRA combina LoRA con il caricamento del modello base quantizzato a 4 bit, portando il requisito di VRAM per un modello 7–8B a circa 10–12 GB. Gli iperparametri principali sono il rank `r` (tipicamente 8–64), `alpha`, il learning rate e il numero di epoch.

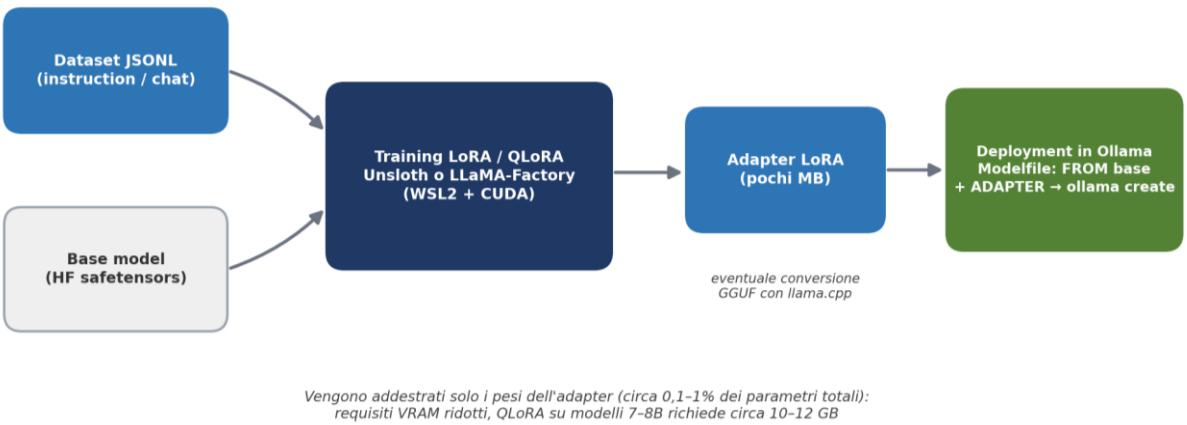

Figura 5. Workflow di fine-tuning LoRA e deployment in Ollama.

10.2 Preparazione del dataset

Il dataset assume la forma di un file JSONL in formato instruction o chat, ad esempio:

```
{"messages": [
  {"role": "user", "content": "Che cos'è una regola Sigma?"},
  {"role": "assistant", "content": "Una regola Sigma è ..."}
]}
```

La qualità prevale sulla quantità: per specializzazioni di stile, formato o tassonomia sono spesso sufficienti da alcune centinaia ad alcune migliaia di esempi curati. Trattandosi di contenuti che il modello può memorizzare, i dataset contenenti informazioni sensibili devono essere preventivamente sanitizzati.

10.3 Training in ambiente WSL2

Gli stack di training più diffusi (Unsloth, LLaMA-Factory, Axolotl) operano al meglio in ambiente Linux: su Windows la via raccomandata è WSL2 con GPU NVIDIA, che espone CUDA alle applicazioni Linux.

1. Installare la distribuzione: `wsl --install -d Ubuntu`; il supporto GPU è incluso nei driver NVIDIA recenti e si verifica eseguendo `nvidia-smi` dall'interno di WSL2.
2. Creare un ambiente Python dedicato e installare il framework prescelto: `pip install unsloth` (in alternativa, LLaMA-Factory offre una web UI di training).
3. Eseguire il training QLoRA sul modello base con il proprio dataset, monitorando la loss di validazione al fine di prevenire l'overfitting.
4. Esportare l'adapter LoRA (directory safetensors) oppure il modello merged in formato GGUF: Unsloth e llama.cpp forniscono utility dedicate di conversione e quantization.

10.4 Deployment del modello adattato in Ollama

L'integrazione avviene tramite Modelfile secondo due alternative: (a) referenziare il modello merged convertito in GGUF con la sola direttiva `FROM`; (b) mantenere separato l'adapter e applicarlo al modello base con la direttiva `ADAPTER`, soluzione più leggera che esige l'esatta corrispondenza tra il base model del training e quello referenziato.

```
FROM llama3.1:8b
ADAPTER ./lora-adapter
```

```
ollama create soc-tuned -f .\Modelfile
```

10.5 Criteri di scelta: fine-tuning o RAG

Il fine-tuning è indicato per consolidare stile, formato e comportamenti ricorrenti (ad esempio tassonomie di classificazione), mentre il RAG rimane preferibile per la conoscenza fattuale soggetta ad aggiornamento, grazie all'immediatezza dell'aggiornamento e alla tracciabilità delle fonti. I due approcci sono complementari e frequentemente combinati.

11. Troubleshooting

La tabella seguente raccoglie le anomalie riscontrate con maggiore frequenza in fase di laboratorio.

Sintomo	Causa probabile	Intervento
`ollama` non riconosciuto come comando	PATH non aggiornato nella sessione corrente	Aprire una nuova sessione PowerShell; verificare l'installazione
Inference lenta, GPU inutilizzata	Driver obsoleti o VRAM insufficiente (fallback su CPU)	Aggiornare i driver; verificare la ripartizione CPU/GPU con `ollama ps`; adottare una quantization inferiore
Errore “model requires more system memory”	RAM/VRAM insufficienti per modello e context	Scegliere un modello più piccolo; ridurre `num_ctx`
Open WebUI non elenca i modelli	Il container non raggiunge Ollama sull'host	Verificare `OLLAMA_BASE_URL` in Settings → Connections; riavviare il container
Docker Desktop non si avvia	Virtualizzazione disabilitata o WSL2 assente	Abilitare VT-x/AMD-V nel firmware UEFI; eseguire `wsl --update`
Porta 3000 già occupata	Conflitto con altro servizio locale	Rimappare la pubblicazione, ad esempio `-p 127.0.0.1:8081:8080` (interfaccia quindi raggiungibile su `localhost:8081`)

12. Riferimenti e note di versione

Documentazione di riferimento, verificata alla data di stesura:

- Ollama, documentazione e model library: `https://docs.ollama.com` e `https://ollama.com/library`
- Documentazione ufficiale di Open WebUI: `https://docs.openwebui.com`
- llama.cpp, motore di inference e utility GGUF: `https://github.com/ggml-org/llama.cpp`
- Unsloth: `https://github.com/unslothai/unsloth`; LLaMA-Factory: `https://github.com/hiyouga/LLaMA-Factory`
- OWASP Top 10 for LLM Applications: `https://owasp.org/www-project-top-10-for-large-language-model-applications/`
- NIST AI Risk Management Framework: `https://www.nist.gov/itl/ai-risk-management-framework`

Note di versione – 1.0, agosto 2026: prima emissione. 1.1, agosto 2026: precisazione sulla licenza di Open WebUI (§9.4), chiarimento sulle varianti Llama nella tabella dei modelli (§5.2), indicazione della porta rimappata nel troubleshooting (cap. 11).

> **Nota:** L'ecosistema open source degli LLM evolve con estrema rapidità: denominazioni dei modelli, comandi e percorsi di menu possono variare tra le versioni. Prima dell'erogazione in aula si raccomanda una verifica dei passaggi sull'ultima release degli strumenti.