CYBER SECURITY & AI

LLM locale con Ollama e Open WebUI

Implementazione di un Large Language Model eseguito interamente in locale su piattaforma Microsoft Windows, con strumenti open source

Versione 1.1 · agosto 2026

INTRODUZIONE

Obiettivi formativi

Installare Ollama

Configurare il runtime su Windows e gestire modelli GGUF e quantization

Deployare Open WebUI

Distribuire il front-end via Docker Desktop, amministrare utenti e impostazioni

Costruire una pipeline RAG

Indicizzare documenti locali e interrogarli con citazioni delle fonti

Integrare via API

Usare REST API locale ed endpoint OpenAI-compatible da PowerShell e Python

Valutare il threat model

Applicare le misure di hardening appropriate al deployment locale

Eseguire un fine-tuning

Adattare un modello con LoRA/QLoRA e distribuirlo in Ollama

Prerequisiti del discente: familiarità con PowerShell e riga di comando · nozioni di base su container Docker e networking TCP/IP · diritti di amministratore locale sulla postazione di lavoro

MOTIVAZIONI

Perché eseguire un LLM in locale

Per chi opera nella cyber security i vantaggi non si limitano alla riduzione dei costi.

Confidenzialità e data residency

Prompt, documenti e conversazioni non transitano verso provider esterni: si possono analizzare log, report di incident response ed evidenze forensi con PII senza uscite dal perimetro.

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: 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.

Sul fronte GDPR: minimizzazione e data residency by design — restano gli obblighi sul dato a riposo (→ Fase 6).

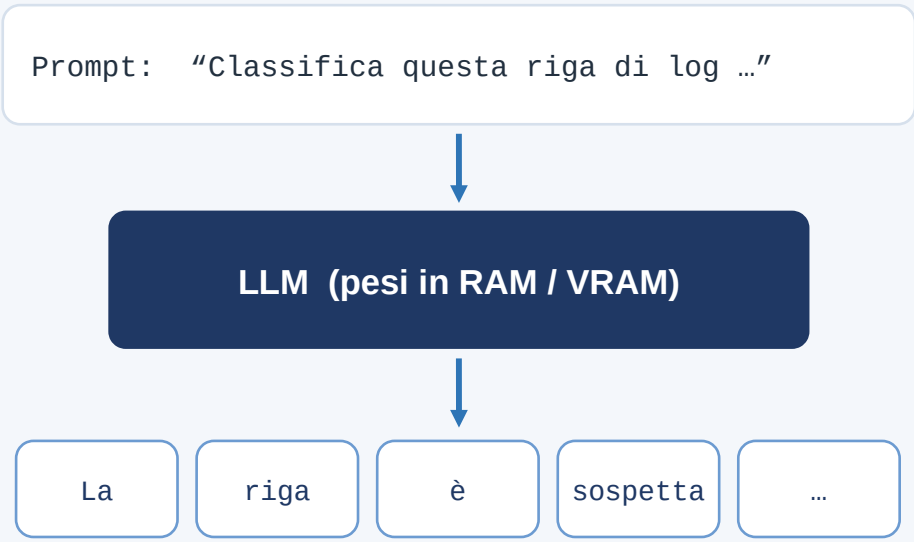
LLM e inference

- Un LLM è una rete neurale transformer addestrata a predire il token successivo di una sequenza.
- In questo modulo non trattiamo il training, ma la sola inference: caricamento in memoria dei pesi di un modello pre-addestrato.
- La generazione è autoregressiva: il testo viene prodotto token dopo token, ciascuno condizionato dai precedenti.

La metrica: token al secondo (t/s)

Le prestazioni di inference dipendono da tre fattori: dimensione del modello, quantization adottata, hardware disponibile.

GENERAZIONE AUTOREGRESSIVA



un token alla volta, ciascuno condizionato dai precedenti

3–60 t/s intervallo tipico su workstation, da CPU-only a GPU dedicata

Parametri, quantization e context window

QUANTIZATION E FORMATO GGUF

- I modelli si classificano per numero di parametri: 7B, 8B, 14B, 70B (B = miliardi).
- Pesi nativi in FP16/BF16, comprimibili via quantization (es. 4 bit): forte risparmio di memoria, perdita qualitativa contenuta.
- GGUF è il formato di riferimento per l'inference locale, usato da llama.cpp su cui Ollama è costruito.
- Varianti identificate da sigle: q4_K_M, q5_K_M, q8_0.

Regola pratica

≈ 0,6 GB per miliardo di parametri in q4_K_M

+ overhead della KV cache, proporzionale alla lunghezza del contesto

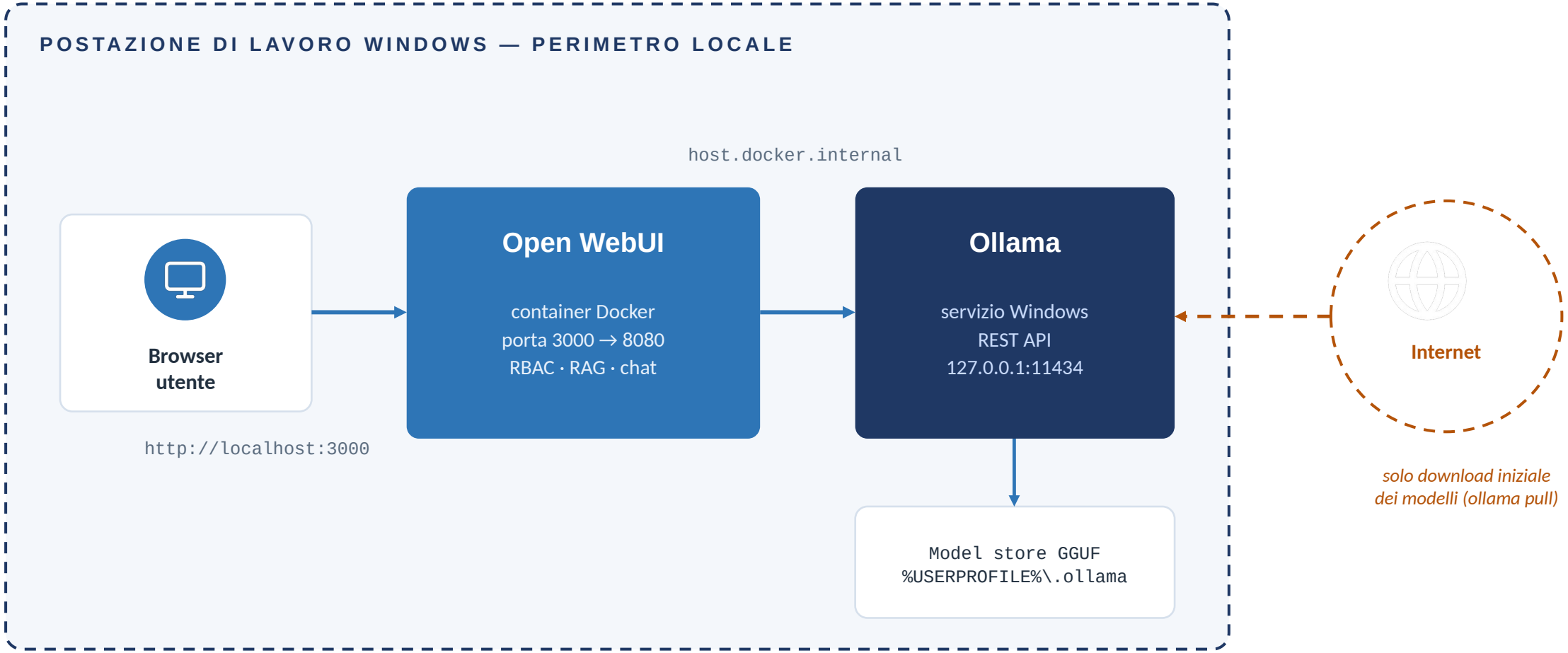
Context window (num_ctx)

Numero massimo di token elaborabili in una singola conversazione. Comprende:

- Prompt di sistema
- Documenti recuperati via RAG
- Risposta generata

Dimensionare sul caso d'uso
4.096–8.192 token per impieghi generali; valori superiori per l'analisi documentale. Valori elevati aumentano il consumo di memoria.

Architettura della soluzione



Nessun dato conversazionale lascia il perimetro locale

Requisiti hardware e software

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: test e didattica
GPU entry-level	16 GB RAM, 8 GB VRAM (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 (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 (RTX 3090/4090)	27–32B in q4; 70B in q4 con offload su RAM	Impiego professionale continuativo

In assenza di GPU l'inference avviene su CPU; con GPU NVIDIA (CUDA) o AMD supportata (ROCm), Ollama effettua automaticamente l'offload dei layer sulla VRAM.

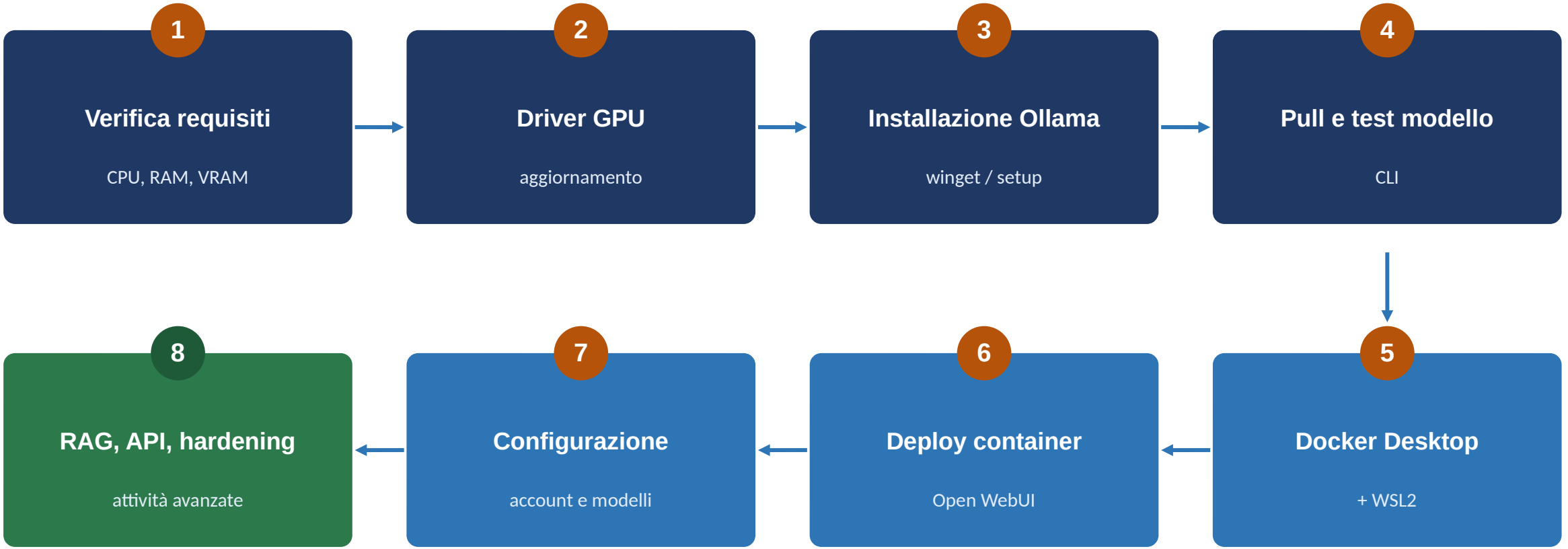
Software richiesto

- Windows 10 (22H2) o Windows 11 a 64 bit, aggiornati
- Driver GPU aggiornati (NVIDIA/CUDA; per AMD verificare il supporto ROCm)
- Virtualizzazione VT-x / AMD-V abilitata nel firmware UEFI (per WSL2)
- Docker Desktop con backend WSL2
- PowerShell 5.1 o 7.x
- Per il fine-tuning (Fase 7): Ubuntu su WSL2 + toolchain CUDA

PERCORSO OPERATIVO

Il flusso complessivo dell’implementazione

Fasi 1–3 del modulo



Fasi 4–7 del modulo

Installazione di Ollama

1 Download ed esecuzione

OllamaSetup.exe dal sito ufficiale (ollama.com/download/windows) oppure via package manager winget.

2 Procedura guidata

Installazione a livello utente: Ollama diventa applicazione in background con icona nell'area di notifica e avvio automatico.

3 Verifica del servizio

Aprire una nuova sessione PowerShell (refresh del PATH) e interrogare versione e API.

4 Verifica del binding

Il listener deve essere attivo esclusivamente sull'interfaccia di loopback.

```
# Installazione via package manager
winget install --id Ollama.Ollama

# Verifica (in una NUOVA sessione)
ollama --version
Invoke-RestMethod `
  http://localhost:11434/api/version

# Binding solo su loopback
netstat -ano | findstr 11434
# atteso: 127.0.0.1:11434 LISTENING
```

Perché conta il punto 4: la REST API di Ollama non prevede autenticazione. Verificare da subito che ascolti solo su 127.0.0.1 è la prima misura di sicurezza del deployment.

Variabili d’ambiente rilevanti

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 CORS	`localhost`

Impostazioni → Sistema → Informazioni → Impostazioni di sistema avanzate → Variabili d’ambiente · dopo ogni modifica riavviare Ollama dall’icona nell’area di notifica.

⚠ Non impostare OLLAMA_HOST=0.0.0.0 per esporre l’API sulla rete senza avere prima applicato le misure della Fase 6: l’API di Ollama non prevede autenticazione nativa.

Gestione dei modelli: 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: se omesso viene scaricata la variante predefinita, tipicamente q4_K_M.
- ollama ps mostra la ripartizione del modello tra CPU e GPU: utile per diagnosticare prestazioni scarse.

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

In laboratorio: eseguire il primo ollama run con --verbose e annotare i token/secondo ottenuti: serviranno per confrontare quantization e hardware.

Selezione del modello

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

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
Mistral / Ministral	7B–8B	Compatti ed efficienti; varianti 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 (→ Fase 4)

Nota: l'offerta di modelli evolve con cadenza mensile: le indicazioni sono valide alla data di stesura — consultare la library ufficiale per lo stato corrente.

Personalizzazione tramite Modelfile

- Descrittore testuale, concettualmente affine a un Dockerfile: deriva un modello personalizzato da uno esistente.
- FROM indica il modello base; PARAMETER fissa i parametri di inference (temperature, num_ctx, …).
- SYSTEM definisce il prompt di sistema: ruolo, lingua, stile di risposta.
- Il modello derivato compare in ollama list ed è selezionabile anche da Open WebUI.

```
# Modelfile
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 una
informazione non è verificabile,
dichiaralo esplicitamente."""
```

Caso d'uso: un assistente per analisti SOC con temperatura bassa (risposte più deterministiche) e contesto ampio per l'analisi di log.

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

FASE 3

Deployment di Open WebUI

FASE 3/7

Front-end conversazionale multi-utente con RBAC, knowledge base RAG e cronologia.
Distribuzione raccomandata: container Docker.

```
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
```

Prerequisito: Docker Desktop con backend WSL2 (verifica: docker version · docker run --rm hello-world).
Primo bootstrap: alcuni minuti, poi http://localhost:3000.

Alternativa senza Docker: pacchetto Python (pip install open-webui, poi open-webui serve, con Python 3.11) — utile senza virtualizzazione, a fronte di maggiore onere manutentivo.

`-p 127.0.0.1:3000:8080`

Pubblica la porta del container solo sul loopback: senza indirizzo esplicito Docker la esporrebbe su tutte le interfacce.

`-e OLLAMA_BASE_URL=…`

Indirizzo con cui il container raggiunge Ollama sull'host (alias host.docker.internal).

`-v open-webui:/app/backend/data`

Volume persistente: database applicativo, utenti, conversazioni e indici RAG.

`--restart always`

Riavvio automatico del container all'avvio di Docker Desktop.

Primo accesso e configurazione

1 Creare l’account iniziale

Con “Sign up”: il primo utente registrato assume automaticamente il ruolo di administrator. Credenziali memorizzate solo nel volume locale.

2 Verificare la connessione a Ollama

Admin Panel → Settings → Connections: endpoint http://host.docker.internal:11434 e modelli visibili nel selettore.

3 Disabilitare le registrazioni spontanee

Admin Panel → Settings → General, opzione “Enable New Sign Ups” — oppure avvio con -e ENABLE_SIGNUP=false.

4 Eseguire un test funzionale

Selezionare un modello e inviare un prompt di prova.

Ruoli disponibili (RBAC)

- administrator — gestione completa
- user — utilizzo dei modelli assegnati
- pending — in attesa di approvazione

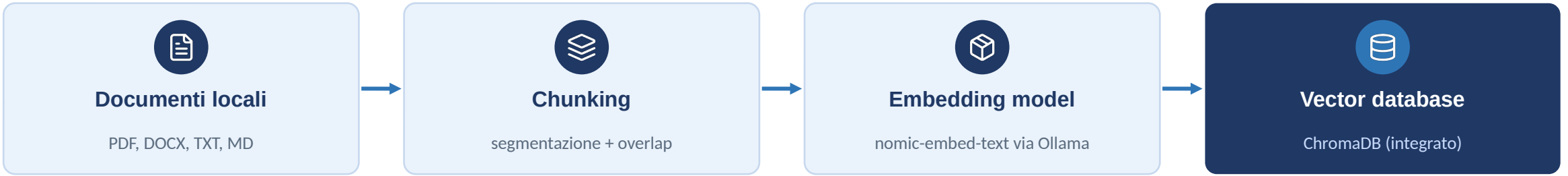
Checkpoint di fine fase

- Chat funzionante su localhost:3000
- Signup spontaneo disabilitato
- Modelli di Ollama visibili nel selettore

RAG su documenti locali: la pipeline

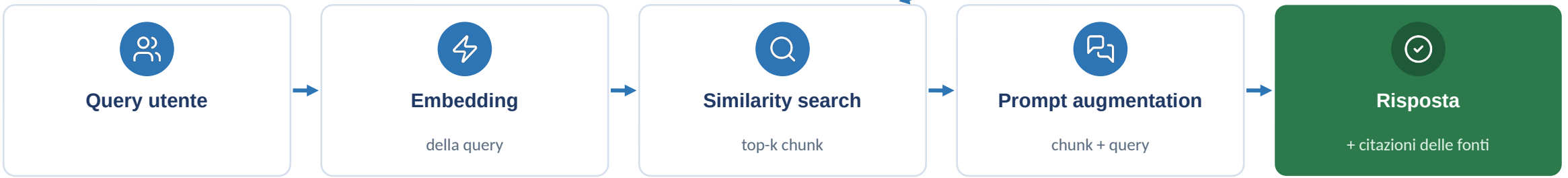
Il Retrieval-Augmented Generation integra la conoscenza parametrica del modello con contenuti documentali aggiornati e verificabili.

FASE DI INDEXING (offline)



retrieval

FASE DI QUERY (runtime)



Rispetto al fine-tuning: aggiornamento immediato della base di conoscenza e tracciabilità delle fonti — nessun riaddestramento del modello (→ confronto in Fase 7).

RAG in pratica: configurazione e caso d’uso

1 Embedding model dedicato

ollama pull nomic-embed-text; in Admin Panel → Settings → Documents impostare “Ollama” come embedding engine.

2 Parametri di partenza

Chunk size 800–1.200 caratteri, overlap 10–15%, top-k 4–6. Se si sostituisce l’embedding model: reindicizzare (vettori non compatibili).

3 Knowledge base

Workspace → Knowledge → “+ Create”, quindi caricare i documenti: estrazione, chunking e indicizzazione sono automatici.

4 Utilizzo in chat

Digitare # e selezionare la knowledge base per vincolare la risposta ai documenti; verificare le citazioni riportate.

Caso d’uso in ambito security

- Security policy e standard interni
- Procedure di incident response
- Report di Cyber Threat Intelligence
- Documentazione post-incident

Corpus interamente on-premise, aggiornabile in ogni momento.

I documenti indicizzati concorrono alla costruzione del prompt: contenuti ostili possono veicolare prompt injection (RAG poisoning). Sottoporre a vetting le sorgenti prima dell’ingestione — dettagli in Fase 6.

API locale: endpoint disponibili

Ollama espone una REST API locale, priva di autenticazione, su http://localhost:11434.

`/api/generate`	Completamento singolo (prompt → risposta)
`/api/chat`	Conversazione multi-turno con ruoli system / user / assistant
`/api/embed`	Calcolo di embedding vettoriali
`/api/tags`	Elenco dei modelli installati nello store

Endpoint OpenAI-compatible

Percorso **`/v1`**: consente il riuso di librerie e strumenti esistenti con la semplice ridirezione della base URL verso l'host locale.

Nessuna autenticazione nativa: qualsiasi processo locale può invocare l'API. È una scelta accettabile finché il binding resta su loopback — diventa un rischio critico se il servizio viene esposto in rete (→ Fase 6).

Esempi di scripting

PowerShell — REST API nativa

```
$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
```

Python — 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 least privilege."}])
print(resp.choices[0].message.content)
```

Disponibile anche la REST API nativa da Python (requests → /api/chat) con ruoli system e user — esempio completo nella lecture note, §8.3.

FASE 5

Automazione di attività di supporto all’analisi

FASE 5/7

Triage di righe di log

Pre-classificazione benigno / sospetto su grandi volumi, prima della revisione umana

Normalizzazione di IoC

Indicatori di compromissione ricondotti a formati e tassonomie uniformi

Sintesi di report

Riassunti operativi di documenti lunghi (CTI, post-incident)

Bozze di documentazione

Prime stesure di procedure e note tecniche da rifinire

Il tutto senza che i dati lascino la postazione.

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

Security e privacy: il threat model

Un deployment locale riduce drasticamente la superficie di attacco legata a terze parti, ma non elimina i rischi.

T1 · Esposizione di rete

API senza autenticazione raggiungibili dalla rete (OLLAMA_HOST=0.0.0.0, -p senza indirizzo esplicito).

T2 · Supply chain

Modelli GGUF non verificati; estensioni di terze parti che eseguono codice sul server.

T3 · Prompt injection / RAG poisoning

Istruzioni ostili veicolate da documenti in knowledge base o testi incollati in chat.

T4 · Dati a riposo

Conversazioni, knowledge base e modelli non cifrati; backup incontrollati del volume Docker.

Mitigazioni: binding su loopback · reverse proxy TLS con autenticazione · verifica digest SHA-256 · vetting delle sorgenti RAG · BitLocker · RBAC con signup disabilitato · monitoraggio dei log

T1 — Esposizione di rete

Migliaia

di istanze Ollama esposte pubblicamente senza autenticazione, documentate da ricerche con motori come Shodan: chiunque può eseguire inference o manipolare i modelli.

Un errore di configurazione, non un exploit:
basta OLLAMA_HOST=0.0.0.0 senza contromisure, o una -p di Docker priva di indirizzo esplicito, che pubblica la porta su tutte le interfacce dell'host.

Misure raccomandate

- Mantenere il binding predefinito su loopback e verificarlo periodicamente con netstat.
- Accesso da altri host solo tramite reverse proxy con TLS e autenticazione (nginx, Caddy, Traefik) oppure VPN (es. WireGuard).
- Regole di Windows Defender Firewall che blocchino le connessioni inbound verso le porte 11434 e 3000.
- Ricordare che -p senza indirizzo esplicito pubblica la porta su tutte le interfacce di rete.

T2 e T3 — Supply chain e prompt injection

Supply chain di modelli ed estensioni

- Trattare i modelli come software di terze parti: library ufficiale di Ollama o publisher verificati su Hugging Face.
- Registrare i digest SHA-256 dei modelli approvati (ollama show, manifest dello store) e mantenerne un inventario.
- Verificare le licenze: modelli (Llama Community License, Apache 2.0, Gemma Terms) e software — Ollama è MIT; Open WebUI dalla v0.6.6 usa una BSD-3 con clausola di branding, non approvata OSI (esenzione ≤ 50 utenti).
- Le estensioni di Open WebUI (tools, functions, pipelines) eseguono codice Python sul server: solo fonti fidate, previa code review.

Prompt injection e RAG poisoning

LLM01 — OWASP Top 10 for LLM Applications: inserimento di istruzioni ostili nei contenuti elaborati dal modello. Un documento in knowledge base o un testo incollato in chat possono tentare di alterarne il comportamento.

Mitigazioni

- Vetting e controllo di provenienza dei documenti
- Separazione dei privilegi: il modello non deve avere capacità dispositive
- Revisione sistematica dell'output
- Mai considerare attendibili le istruzioni provenienti dai dati

Checklist di hardening

- Binding dei servizi limitato a 127.0.0.1 e verificato con netstat
- Signup disabilitato in Open WebUI e RBAC configurato
- Inventario dei modelli con digest SHA-256 e verifica delle licenze
- Vetting delle sorgenti documentali destinate al RAG
- Aggiornamento periodico di Ollama, Open WebUI e Docker Desktop
- Regole firewall inbound esplicite per le porte 11434 e 3000
- Accesso remoto solo via reverse proxy TLS con autenticazione o VPN
- Estensioni di terze parti sottoposte a code review
- BitLocker attivo e backup cifrati del volume open-webui

Protezione dei dati e conformità (GDPR): il trattamento locale favorisce minimizzazione e data residency, ma restano gli obblighi sul dato a riposo: cifratura del disco, controllo degli accessi, backup e retention coerenti con le policy. Ollama non trasmette dati di utilizzo; la telemetria di Docker Desktop è configurabile.

FASE 7

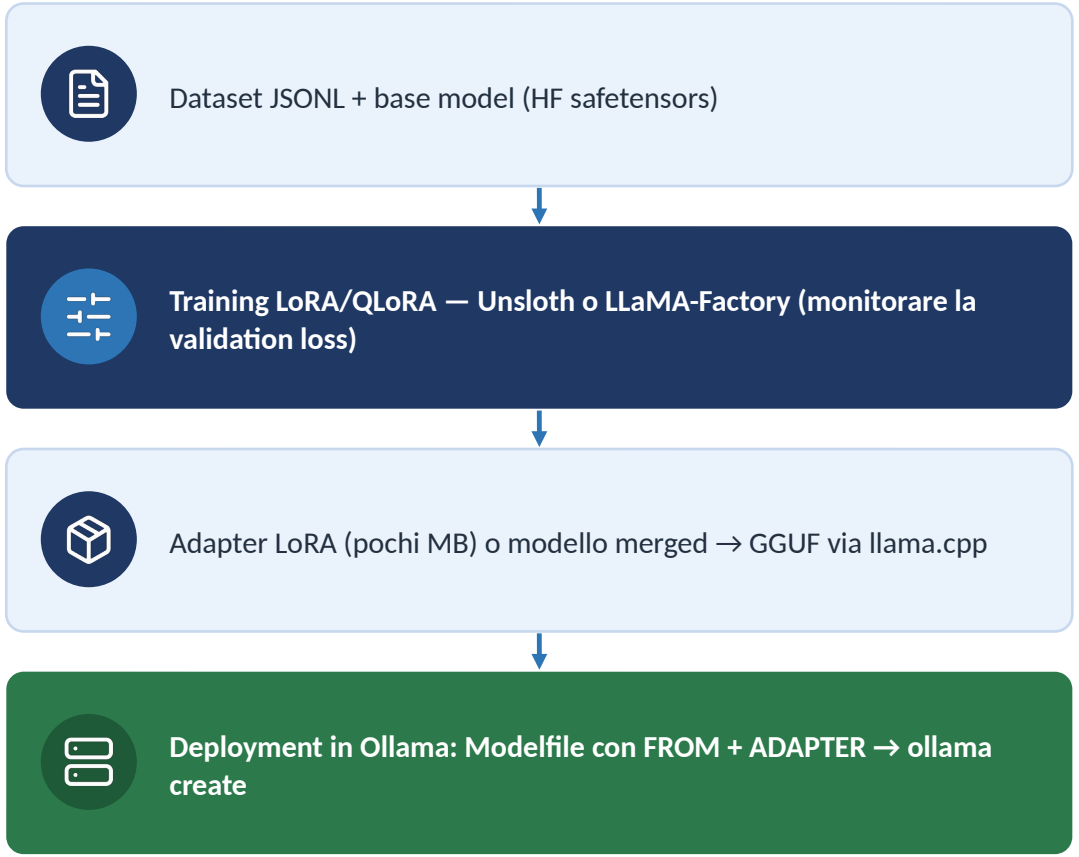
Fine-tuning parameter-efficient con LoRA

FASE 7/7

- Il full fine-tuning è fuori portata per una workstation: le tecniche PEFT rendono l'adattamento praticabile.
- LoRA congela i pesi del modello base e addestra solo coppie di matrici a basso rango nei layer di attention: parametri addestrabili ≈ 0,1–1% del totale, adapter di pochi MB.
- QLoRA carica il modello base quantizzato a 4 bit: VRAM richiesta ≈ 10–12 GB per un modello 7–8B.
- Iperparametri principali: rank r (8–64), alpha, learning rate, epoch.
- Dataset JSONL (formato instruction o chat): qualità > quantità; sanitizzare i dati sensibili — il modello può memorizzarli.

ADAPTER esige l'esatta corrispondenza tra il base model del training e quello referenziato nel Modelfile.

WORKFLOW (WSL2 + CUDA)



Criteri di scelta: fine-tuning o RAG?

Fine-tuning

- Consolida stile, formato e comportamenti ricorrenti
- Ideale per tassonomie di classificazione stabili
- Richiede dataset curato, training e validazione
- La conoscenza resta “congelata” nell’adapter

RAG

- Preferibile per conoscenza fattuale soggetta ad aggiornamento
- Aggiornamento immediato: basta caricare un documento
- Tracciabilità delle fonti tramite citazioni
- Nessun riaddestramento del modello

I due approcci sono complementari e frequentemente combinati: adapter per lo stile, knowledge base per i fatti.

Troubleshooting: anomalie frequenti in 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 con ollama ps; quantization inferiore
"model requires more system memory"	RAM/VRAM insufficienti per modello e contesto	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
Docker Desktop non si avvia	Virtualizzazione disabilitata o WSL2 assente	Abilitare VT-x/AMD-V nel firmware UEFI; wsl --update
Porta 3000 già occupata	Conflitto con altro servizio locale	Rimappare, es. -p 127.0.0.1:8081:8080 (interfaccia su localhost:8081)

Metodo: prima il sintomo, poi la verifica strumentale (netstat, ollama ps, docker logs), infine l'intervento — mai procedere per tentativi.

RIFERIMENTI E NOTE DI VERSIONE

Per approfondire

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

Nota finale — l'ecosistema open source degli LLM evolve con estrema rapidità: prima dell'erogazione in aula, verificare i passaggi sull'ultima release degli strumenti. · Materiale: lecture note v1.1, agosto 2026.