Profice

AI ACADEMY

Applicare l’Intelligenza Artificiale nello sviluppo software

AI ACADEMY

Docker & HF Spaces Deploy
01/07/2025

INTRODUZIONE DELL'ISTRUTTORE

Tamas Szakacs

Formazione

- Laureato come programmatore matematico
- MBA in management

Principali esperienze di lavoro

- Amministratore di sistemi UNIX
- Oracle DBA
- Sviluppatore di Java, Python e di Oracle PL/SQL
- Architetto (solution, enterprise, security, data)
- Ricercatore tecnologico e interdisciplinare di IA

Dedicato alla formazione continua

- Teorie, modelli, framework IA
- Ricerche IA
- Strategie aziendali
- Trasformazione digitale
- Formazione professionale

email: tamas.szakacs@proficegroup.it

MOTIVI E RIASSUNTO DEL CORSO

L'**Intelligenza Artificiale (AI)** è oggi il motore dell'innovazione in ogni settore, grazie alla sua capacità di analizzare dati, automatizzare processi e generare nuove soluzioni. Questo corso offre una panoramica completa e pratica sullo sviluppo di applicazioni AI moderne, guidando i partecipanti dall'ideazione al rilascio in produzione.

Attraverso una **combinazione di teoria chiara ed esercitazioni pratiche**, saranno affrontate le tecniche e gli strumenti più attuali: **machine learning, deep learning, reti neurali, Large Language Models (LLM), Transformers, Retrieval Augmented Generation (RAG)** e progettazione di agenti AI.
Le competenze acquisite saranno applicate in progetti concreti, dallo sviluppo di chatbot all'integrazione di modelli generativi, fino al deploy di soluzioni AI in ambienti reali e collaborativi.

Il percorso è pensato per chi vuole imparare a progettare, valutare e integrare sistemi AI di nuova generazione, con particolare attenzione alle best practice di programmazione, collaborazione in team, sicurezza, valutazione delle performance ed etica dell'AI.

DURATA: 17 GIORNI

OBIETTIVI

Il percorso formativo è progettato per **giovani consulenti junior**, con una conoscenza base di programmazione, che stanno iniziando un percorso professionale nel settore AI.

L’obiettivo centrale è fornire una panoramica pratica, completa e operativa sull’intelligenza artificiale moderna, guidando ogni partecipante attraverso tutte le fasi fondamentali.

OBIETTIVI

- Allineare conoscenze AI, ML, DL di tutti i partecipanti
- Saper usare e orchestrare modelli LLM (closed e open-weight)
- Costruire pipeline RAG complete (retrieval-augmented generation)
- Progettare agenti AI semplici con strumenti moderni (LangChain, tool calling)
- Capire principi di valutazione, robustezza e sicurezza dei sistemi GenA
- Migliorare la produttività come sviluppatori usando tool GenAI-driven
- Padroneggiare best practice di sviluppo, versioning e deploy AI
- Introdurre i fondamenti di Graph Data Science e Knowledge Graph
- Ottenere capacità di valutazione dei modelli e metriche
- Comprensione dell'etica e dei bias nei modelli di intelligenza artificiale
- Approfondire le normative di riferimento: AI Act, compliance e governance AI

Il corso è **estremamente pratico** (circa il 40% del tempo in esercitazioni hands-on, notebook, challenge e hackathon), con l'utilizzo di Google Colab, GitHub, e tutti gli strumenti necessari per lavorare su progetti reali e simulati.

STRUTTURA DELLE GIORNATE – PROGRAMMA BREVE

Tutte le giornate sono di 8 ore (9:00-17:00), con 1 ora di pausa suddivisa (mezz'ora pranzo, due pause da 15 min durante la mattina e il pomeriggio).

La progettazione sintetica delle giornate:

Giorno	Tema	Breve descrizione
1	Git & Python clean-code	Collaborazione su progetti reali, versionamento, codice pulito e testato
2	Machine Learning Supervised	Modelli supervisionati per predizione e classificazione
3	Machine Learning Unsupervised	Clustering, riduzione dimensionale, scoperta di pattern
4	Prompt Engineering avanzato	Scrivere e valutare prompt efficaci per modelli generativi
5	LLM via API (multi-vendor)	Uso pratico di modelli LLM via API, autenticazione, deployment
6	Come costruire un RAG	Pipeline end-to-end per Retrieval-Augmented Generation
7	Tool-calling & Agent design	Progettare agenti AI che usano strumenti esterni
8	Hackathon: Agentic RAG	Challenge pratica: chatbot agentico RAG in team

STRUTTURA DELLE GIORNATE – PROGRAMMA BREVE

Tutte le giornate sono di 8 ore (9:00-17:00), con 1 ora di pausa suddivisa (mezz'ora pranzo, due pause da 15 min durante la mattina e il pomeriggio).

La progettazione sintetica delle giornate:

Giorno	Tema	Breve descrizione
9	Hackathon: Rapid Prototyping	Da prototipo a web-app con Streamlit e GitHub
10	AI Productivity Tools	Workflow con IDE AI-powered, automazione e refactoring assistito
11	Docker & HF Spaces Deploy	Deployment di app GenAI containerizzate o su HuggingFace Spaces
12	AI Act & ISO 42001 Compliance	Fondamenti di compliance e governance AI
13	Knowledge Base & Graph Data Science	Introduzione a Knowledge Graph e query con Neo4j
14	Model evaluation & osservabilità	Metriche avanzate, explainability, strumenti di valutazione
15	AI bias, fairness ed etica applicata	Analisi dei rischi, metriche e mitigazione dei bias
16-17	Project Work & Challenge finale	Lavoro a gruppi, POC/POD, presentazione e votazione progetti

METODOLOGIA DEL CORSO

1. Approccio introduttivo ma avanzato

Il corso è introduttivo nei concetti base dell'AI applicata allo sviluppo, ma affronta anche tecnologie, modelli e soluzioni avanzate per garantire un apprendimento completo.

2. Linguaggio adattato

Il linguaggio utilizzato è chiaro e adattato agli studenti, con spiegazioni dettagliate dei termini tecnici per favorirne la comprensione e l'apprendimento graduale.

3. Esercizi pratici

Gli esercizi pratici sono interamente svolti online tramite piattaforme come Google Colab o notebook Python, eliminando la necessità di installare software sul proprio computer.

4. Supporto interattivo

È possibile porre domande in qualsiasi momento durante le lezioni o successivamente via email per garantire una piena comprensione del materiale trattato.

NOTA

Il corso segue un **approccio laboratoriale**: ogni giornata combina sessioni teoriche chiare e concrete con molte attività pratiche supervisionate, per sviluppare *competenze reali* immediatamente applicabili.

I partecipanti lavoreranno spesso in gruppo, useranno notebook in Colab e versioneranno codice su GitHub, vivendo una vera simulazione del lavoro in azienda AI.

Nessun prerequisito avanzato richiesto: si partirà dagli strumenti e flussi fondamentali, con una crescita graduale verso le tecniche più attuali e richieste dal mercato.

ORARIO TIPICO DELLE GIORNATE

Orario	Attività	Dettaglio
09:00 – 09:30	Teoria introduttiva	Concetti chiave, schema della giornata
09:30 – 10:30	Live coding + esercizio guidato	Esempio pratico, notebook Colab
10:30 – 10:45	*Pausa breve*	
10:45 – 11:30	Approfondimento teorico	Tecniche, best practice
11:30 – 12:30	Esercizio hands-on individuale	Sviluppo o completamento di codice
12:30 – 13:00	Discussione soluzioni + Q&A	Condivisione e correzione
13:00 – 14:00	*Pausa pranzo*	
13:30 – 14:15	Teoria avanzata / nuovi tools	Nuovi strumenti, pattern, demo
14:15 – 15:30	Esercizio a gruppi / challenge	Lavoro di squadra su task reale
15:30 – 15:45	*Pausa breve*	
15:45 – 16:30	Sommario teorico e pratico	
16:30 – 17:00	Discussioni, feedback	Riepilogo, best practice, domande aperte

DOMANDE?

Cominciamo!

OBIETTIVI DELLA GIORNATA

Obiettivi della giornata

- ripetere i concetti base della containerizzazione con Docker applicata a progetti GenAI.
- Distinguere tra base-image e multi-stage build per ottimizzare immagini Docker.
- Imparare a pubblicare un’applicazione GenAI su Hugging Face Spaces.
- Gestire in sicurezza le variabili segrete (secrets) durante il deploy.
- Analizzare i diversi hardware tiers e i limiti delle Spaces disponibili su HF.
- Monitorare rollout, esecuzione e debug logs di app GenAI distribuite.

SCOPO DEL PROGETTO

Creare il chatbot con agentic RAG usando rapid prototyping con Streamlit

Capacità o caratteristiche richieste

- ☐ Anonimizzazione dei dati sensitivi (min. nomi e IBAN) usando NER e regex
- ☐ Gestione di documenti con tecniche RAG nei prompt
- ☐ Uso di embedding, chunking, vettorizzazione dei documenti e prompt (similarity search su documenti)
- ☐ Autonomia / agente per gestire files e risposte (con creazione di files, per esempio risposta mail in un file)
- ☐ Chat e gestione di una singola sessione con la sequenza dei messaggi con Langchain

Visualizzazioni richieste con Streamlit

- ☐ Chat input e output in formato history (tipo ChatGPT)
- ☐ Bottone per caricare documenti (da gestire con autonomia dal modello)
- ☐ Elenco dei file caricati
- ☐ (Opzionale) Gestione di diverse sessioni
- ☐ (Opzionale) Gestione di memoria, a disposizione di ogni sessione

DELIVERABLES E DOCUMENTI MINIMI PER IL PROGETTO

Per consegnare i lavori bisogna preparare questi deliverable:

- **Risposta progettuale**
  Breve descrizione di come il team ha risolto il problema proposto, con riferimento alle scelte principali.
- **Schema architetturale**
  Schema essenziale (testuale o diagramma semplice) dell'architettura della soluzione e dei principali componenti (modello, agenti, pipeline, API, ecc.).
- **Codice sorgente**
  Tutto il codice sviluppato, con commenti chiari e autoesplicativi (documentazione del codice generata direttamente dai commenti).
- **Dataset di test**
  File di dati usati per le prove e le demo, rappresentativi dei casi d'uso.
- **Metodo di test**
  Breve descrizione del metodo di test applicato: quali casi, quali dati, come è stato valutato il funzionamento.
- **(Opzionale – da aggiungere dopo lezione su etica/EU AI Act)**
  Eventuali note su conformità etica, privacy e regole AI.

RIPASSO: CONTAINERIZZAZIONE E GENAI

Quando serve la containerizzazione in AI:

- Quando vuoi replicare facilmente un ambiente di sviluppo/produzione (stesso codice, stesse dipendenze, stessi risultati).
- Quando devi distribuire o pubblicare la tua app GenAI su cloud, server di clienti, piattaforme come HF Spaces o DockerHub.
- Quando il tuo progetto prevede librerie o strumenti non banali da installare (es. CUDA, modelli custom, pipeline complesse).

Perché è fondamentale per AI/ML:

- Garantisce **portabilità**: lo stesso container gira ovunque.
- Riduce problemi di **“funziona solo sul mio PC”**.
- Agevola **scalabilità** e gestione delle risorse (GPU, RAM, CPU).
- Permette aggiornamenti, rollback e versionamento del servizio.

In pratica:

- Simili agli ambienti virtuali, venv o conda, un container è come un ambiente “blindato”, pronto da spedire.
- Nel ciclo di lavoro GenAI: dal prototipo locale → al container → al deploy su cloud/spaces.

COSA È E COME SI USA UN CONTAINER

Un container è un ambiente isolato che racchiude applicazione, dipendenze e configurazioni in un unico “pacchetto” eseguibile su qualsiasi sistema compatibile con Docker (o analoghi).

Uso pratico di un container:

- Avvio rapido dell’applicazione GenAI, senza configurazioni aggiuntive sulla macchina ospite.
- Aggiornamenti e rollback facilitati grazie a versioni diverse dello stesso container.
- Supporto a deployment su server, cloud e piattaforme collaborative come Hugging Face Spaces.

Esternalizzazione di dati e gestione sicura dei segreti:

- I dati sensibili (database, file di log, modelli AI addestrati) vengono montati dall’esterno tramite volumi o variabili d’ambiente.
- I segreti (API key, token, password) **non devono essere inseriti nel container**: si utilizzano strumenti come Docker secrets, variabili d’ambiente o sistemi di gestione segreti integrati nelle piattaforme (es. Secret Manager di HF Spaces).
- Questa separazione migliora la sicurezza, semplifica il backup e permette di aggiornare dati o chiavi senza ricostruire il container.

DOCKERFILE: STRUTTURA E ISTRUZIONI ESSENZIALI

Il Dockerfile

Un semplice file di testo che definisce tutti i passaggi per creare una Docker image, elencando comandi ed istruzioni in modo sequenziale.

Struttura tipica:

- FROM – Specifica l'immagine base (es. python:3.10) o da capo (scratch)
- WORKDIR – Imposta la directory di lavoro interna al container
- COPY – Copia file e cartelle dal host al container
- RUN – Esegue comandi di installazione (es. dipendenze)
- ENV – Imposta variabili d'ambiente
- EXPOSE – Indica la porta esposta dal container
- CMD o ENTRYPOINT – Comando di default eseguito all'avvio

BASE IMAGE

- Docker non virtualizza l'hardware, ma isola processi in ambiente Linux (o Windows).
- Il container necessita di una struttura minima per eseguire qualsiasi comando: ad esempio, anche solo /bin/sh.
- Le immagini base più minimali sono tipo scratch (vuota) o alpine (essenziale Linux).

Opzioni di base image

FROM scratch

- L'immagine veramente vuota.
- Puoi usarla solo se aggiungi tu manualmente ogni file binario e tutte le librerie richieste dal tuo programma.
- Serve per programmi **compilati staticamente** (come un binario Go, Rust, ecc.).
- Non puoi lanciare script Python, Node, ecc. senza tutto l'ambiente Python/Node già incluso nel build.

FROM alpine

- Immagine Linux ultraleggera (~5MB), include package manager apk e strumenti minimi.
- Ideale per app Python o altre che richiedono comunque una shell o interprete.

Esempio di immagine Python ufficiale: python:3.10-slim, già pronto per script/app Python.

BASE IMAGE VS MULTI-STAGE BUILD

Base-image

- È l'immagine di partenza su cui si costruisce il container (ad es. python:3.10, ubuntu:22.04).
- Include solo il sistema operativo e, se necessario, l'interprete Python o altri componenti minimi.
- Semplice e veloce, ma può produrre immagini più "pesanti" se si installano molte dipendenze nel container.

Multi-stage build

- Consente di usare più fasi nella creazione di una Docker image.
- Si utilizzano immagini temporanee per compilare o installare dipendenze e una seconda fase "pulita" (più piccola) per eseguire solo l'applicazione finale.

Vantaggi:

- Riduce la dimensione dell'immagine definitiva (contiene solo ciò che serve in produzione).
- Migliora la sicurezza: non sono presenti strumenti di build o file temporanei nella fase finale.

Esempio tipico:

- Primo stage: installazione e build di librerie o modelli.
- Secondo stage: solo runtime e file necessari all'avvio dell'app.

DOCKER SU WINDOWS

Passaggi per installare Docker Command Line su Windows

1. Scarica Docker Desktop per Windows

- Vai sul sito ufficiale: https://www.docker.com/products/docker-desktop/
- Scarica la versione per Windows.

2. Installa Docker Desktop

- Esegui il file .exe scaricato.
- Segui la procedura guidata “Avanti”, e accetta le opzioni di default). **BISOGNA RIAVVIARE WINDOWS!**
- **Richiede Windows 10/11 Pro/Enterprise/Education** (o Home con WSL2).

3. (Opzionale) Abilita WSL2

- Docker Desktop, sulle versioni Home, usa **WSL2** (Windows Subsystem for Linux 2).
- **Se richiesto**, l’installer ti aiuterà a configurare WSL2.

4. Avvia Docker Desktop

- Trova “Docker Desktop” tra i programmi e avvialo.
- Dopo qualche secondo, l’icona balena deve diventare verde.

5. Usa Docker da Command Line

- Apri **Prompt dei comandi** (cmd) oppure **Windows PowerShell** o **Windows Terminal**.
- Puoi scrivere, ad esempio: `docker --version`

BEST PRACTICE PER IMMAGINI DOCKER LEGGERE

- **Usare base image “slim” o “alpine”**
  (es. python:3.11-slim, alpine:latest): immagini di dimensioni ridotte, solo l’essenziale.
- **Multi-stage build**
  Costruire in uno stage (con compilatori/tool), copiare solo i file necessari nello stage finale.
- **Rimuovere dipendenze non necessarie**
  Evitare di installare software o librerie superflue nel container.
- **Pulire la cache durante l’installazione**
  (es. `apt-get clean && rm -rf /var/lib/apt/lists/*`)
- **Copiare solo il codice/app necessario**
  Usare .dockerignore per escludere file inutili (es. log, dati temporanei).
- **Minimizzare layer**
  Unire comandi simili in un’unica istruzione RUN per ridurre il numero di layer.

Obiettivo:
Immagini più piccole → download più rapido, minori rischi di sicurezza, uso efficiente delle risorse.

ESERCIZIO: CREARE UN DOCKER IMAGE MINIMAL

Obiettivo:

Creare una **immagine Docker minimale** per eseguire uno script Python che stampa “Ciao dal container!”.

Istruzioni:

1. Crea una nuova cartella e al suo interno aggiungi un file main.py con questa riga:

```
print("Ciao dal container!")
```

2. Scrivi un Dockerfile che:
   - Usa una base image leggera (python:3.11-slim o simile)
   - Copia il file saluta.py nel container
   - Imposta come comando di default l’esecuzione dello script
3. Costruisci l’immagine ed eseguila, osservando il risultato.

Suggerimento:

- Cerca di mantenere il Dockerfile **il più semplice possibile**.

GESTIONE DELLE VARIABILI SEGRETE (SECRETS)

Perché gestire i secrets?

- Le chiavi API, password e dati sensibili **non devono mai essere hardcoded** nel codice o inclusi in immagini pubbliche.
- Evita rischi di sicurezza e leak accidentali.

Principali metodi:

- **File .env**: file di testo con coppie chiave=valore, escluso da versionamento (usa .gitignore).
- **Variabili d'ambiente**: passate al container all'avvio (docker run -e ... oppure tramite file .env).
- **Docker secrets**: (per Docker Swarm) gestione centralizzata, i secrets diventano file temporanei nel container.
- Gestione dei segreti su Kubernetes: utilizzo di Helm charts
- **Gestione cloud**: servizi di secrets manager su Azure, AWS, GCP, Hugging Face Spaces.

Best practice:

- **Mai committare i secrets su Git.**
- **Separare i file di configurazione dai dati sensibili.**
- **Aggiornare e ruotare periodicamente le chiavi.**
- **Usare volumi/variabili per montare i secrets al container solo all'avvio.**

ESERCIZIO: USARE SEGRETI ESTERNI

Obiettivo:

Creare una **immagine Docker minimale** per eseguire uno script Python che carica il contenuto di un file .env e poi stampa la chiave.

Istruzioni:

1. Crea una nuova cartella e al suo interno aggiungi un file main.py con questa riga:

```
print(f"Ciao! La tua chiave è: {api_key}")
```

2. Scrivi un Dockerfile che:
   - Usa una base image leggera (python:3.11-slim o simile)
   - Copia il file main.py nel container e la chiave
   - Imposta come comando di default l'esecuzione dello script
3. Costruisci l'immagine ed eseguila, osservando il risultato.

Suggerimento:

- Cerca di mantenere il Dockerfile **il più semplice possibile**.
- Per mettere a disposizione del container un file, basta copiarlo nel Dockerfile.

COMANDI UTILI DI DOCKER PER GESTIRE IMMAGINI

Gestione immagini Docker – comandi essenziali

- `docker images`
  Elenca tutte le immagini presenti sul sistema.
- `docker pull <nome>:<tag>`
  Scarica un'immagine dal registry (es: Docker Hub).
- `docker build -t <nome>:<tag> .`
  Costruisce una nuova immagine dal Dockerfile nella directory corrente.
- `docker tag <immagine>:<tag> <nuovo_nome>:<nuovo_tag>`
  Rinomina/tagga un'immagine già esistente.
- `docker rmi <nome>:<tag>`
  Rimuove una o più immagini dal sistema.
- `docker save -o <file.tar> <nome>:<tag>`
  Esporta un'immagine in un file .tar.
- `docker load -i <file.tar>`
  Importa un'immagine da un file .tar.
- `docker push <nome>:<tag>`
  Carica un'immagine su un registry remoto (dopo `docker login` adatto).
- `docker run -e <chiave>=<valore> <nome>:<tag>`
  Fornisce una variabile all'immagine

DOCKER IMAGE SU MICROSOFT AZURE

1. **Azure Container Instances (ACI)**

- Servizio **più semplice**: si avvia direttamente una Docker image (da Docker Hub, Azure Container Registry o altro).
- Si paga solo per il tempo di esecuzione.
- Nessuna infrastruttura da gestire.

2. **Azure App Service (Web App for Containers)**

- Piattaforma per applicazioni web/container.
- Si può caricare una Docker image come backend di una web app (API, Streamlit, FastAPI, etc).
- Offre scalabilità, monitoring, gestione SSL, etc.

3. **Azure Kubernetes Service (AKS)**

- Per lesigenze più complesse (scalabilità automatica, orchestrazione di molti container), puoi usare AKS.
- Carichare la Docker image in un cluster Kubernetes gestito.

4. **Azure Machine Learning / Azure OpenAI**

- Anche alcuni servizi di AI possono accettare container personalizzati (ma spesso solo con certi requisiti e formati).

CONTAINER RUNTIME PIÙ DIFFUSI IN AI SECOPS

containerd

Un runtime container open source, mantenuto dalla CNCF (Cloud Native Computing Foundation), nato come parte interna di Docker e ora usato da default in molte distribuzioni Kubernetes.
È il runtime predefinito in Kubernetes (da v1.24+ ha sostituito Docker come runtime), presente in quasi tutte le piattaforme cloud-native moderne.

Vantaggi:

Leggerezza, velocità, robustezza.
Integrazione diretta con orchestratori come Kubernetes.
Ampio supporto da parte di grandi cloud provider (AWS EKS, GCP GKE, Azure AKS, Red Hat OpenShift).

CRI-O

Runtime leggerissimo progettato per Kubernetes, sviluppato da Red Hat.

- Popolare in ambienti enterprise, soprattutto su **OpenShift** (la piattaforma Kubernetes di Red Hat).
- Usato dove la compliance e la sicurezza enterprise sono prioritari.

Vantaggi:

- Integrato perfettamente con Kubernetes tramite l'interfaccia CRI (Container Runtime Interface).
- Preferito in ambienti dove si vuole evitare la dipendenza da Docker.

DOMANDE?

PAUSA

INTRODUZIONE AD AI SECOPS

Definizione di SecOps

AI SecOps è l'integrazione di strumenti e pratiche di **Intelligenza Artificiale (AI)** all'interno delle attività di **Security Operations (SecOps)**, ossia l'insieme delle operazioni quotidiane di sicurezza informatica in azienda.

L'obiettivo è **aumentare l'efficacia, la velocità e l'automazione** nella rilevazione, risposta e prevenzione delle minacce, sfruttando algoritmi e sistemi AI.

Perché è importante?

Le minacce evolvono rapidamente e in modo sofisticato, spesso più velocemente della capacità di risposta manuale umana.

L'AI permette di:

- Analizzare enormi volumi di dati in tempo reale (log, eventi, traffico, alert).
- Identificare pattern anomali, potenziali attacchi e vulnerabilità.
- Automatizzare risposte e remediation.
- Ridurre il carico sugli analisti di sicurezza, che possono concentrarsi su casi critici e attività strategiche.

INTRODUZIONE AD AI SECOPS

Applicazioni principali

- **Threat detection**: identificazione automatica di minacce sconosciute.
- **Incident response**: automazione di processi di risposta agli attacchi.
- **Vulnerability management**: analisi predittiva e prioritarizzazione delle vulnerabilità.
- **User & Entity Behavior Analytics (UEBA)**: riconoscimento di comportamenti sospetti di utenti e sistemi.
- **Compliance & auditing**: controllo continuo e reporting automatizzato.

PRINCIPALI MINACCE AI-SPECIFICHE

- **Prompt Injection**
  Manipolazione del prompt fornito a un LLM per ottenere risposte non previste, estrarre informazioni riservate o aggirare restrizioni.
- **RAG Poisoning**
  Alterazione dei dati nel database esterno utilizzato in Retrieval-Augmented Generation, inserendo contenuti fuorvianti o malevoli che l'LLM recupera e usa nelle risposte.
- **Model Inversion**
  Tecnica per ricostruire o estrarre dati sensibili (es. dati personali, testi originali) dal modello AI interrogandolo ripetutamente.
- **Data Poisoning**
  Inserimento di dati manipolati nel dataset di training per influenzare il comportamento del modello e introdurre vulnerabilità.
- **Membership Inference**
  Attacco che permette di capire se un determinato dato era presente nel dataset di training, violando la privacy.

HUGGING FACE SPACES

Introduzione

E’ una piattaforma cloud per **pubblicare e condividere applicazioni di intelligenza artificiale** in modo rapido e semplice.

Permette di mostrare, testare e far utilizzare modelli AI tramite interfacce web, senza gestire infrastruttura.

Caratteristiche principali

- Supporta app **Python** (Streamlit, Gradio, FastAPI, Flask, ecc.), **R** e **Javascript**.
- Hosting gratuito con possibilità di upgrade per hardware accelerato (CPU, GPU, T4, A10G, H100…).
- Ogni Space ha una propria URL pubblica e può essere privato o pubblico.

Come funziona

- **Caricare il codice** e i file necessari in un repository Space.
- **Configurare le dipendenze** (requirements.txt).
- La piattaforma costruisce ed esegue l’app, visibile via browser.

Casi d’uso

- Demo interattive di modelli AI.
- Applicazioni di RAG, chatbot, image/audio generation, ecc.
- Collaborazione e condivisione progetti AI con la community o clienti.

HARDWARE TIERS E LIMITI DELLE HF SPACES

Gli hardware tiers di HF Spaces

HF Spaces offre diverse configurazioni hardware (“tiers”) per eseguire le app pubblicate, sia gratuite sia a pagamento.

La scelta del tier determina la potenza disponibile (CPU, RAM, GPU), le performance e i limiti di utilizzo.

Principali livelli hardware:

- **CPU Basic** (gratis): solo CPU, risorse limitate, adatto a demo/test/testi leggeri.
- **CPU Upgrade** (a pagamento): più CPU, più RAM, per workload più pesanti.
- **GPU** (a pagamento): accesso a una GPU (es. Nvidia T4/A10G), necessario per modelli di deep learning, visione, audio, ecc.
- **Private Spaces**: isolamento, risorse dedicate, ideali per progetti aziendali o con dati sensibili.

Limiti tipici:

- **Timeout inattività**: dopo 1h di inattività, lo Space gratuito viene “spento”.
- **Limiti di memoria e disco**: ogni tier ha una quota di RAM e storage (vedi documentazione aggiornata HF).
- **Utilizzo GPU limitato**: per tier gratuiti, risorse condivise e limitate; priorità agli utenti paganti.
- **Concurrency**: numero di utenti contemporanei gestibili varia in base al tier.

ESERCIZIO: IMPOSTARE SECRETS E VARIABILI D'AMBIENTE

Obiettivo:
Imparare a configurare variabili segrete (es. API key) e variabili d'ambiente per la tua app su Hugging Face Spaces.

1. **Creazione del secret**

- Vai nella pagina del tuo Space su Hugging Face.
- Clicca su **Settings** > **Secrets**, poi su **Advanced e Repository secrets**
- Inserisci una **nuova chiave** (es. OPENAI_API_KEY) e il suo valore.
- Salva: la variabile sarà disponibile solo nell'ambiente del tuo Space (non visibile agli altri utenti).

2. **Uso nel codice Python**

```python
import os
openai_key = os.environ.get("OPENAI_API_KEY")
print("La tua API Key:", openai_key)  # Per test: in produzione non stampare!
```

3. **Variabili d'ambiente pubbliche**

- Oltre ai secrets, puoi definire variabili d'ambiente pubbliche direttamente su **Settings** > **Environment Variables**.
- Queste sono visibili a chi accede al tuo Space (non usare qui dati sensibili).

ROLLOUT E GESTIONE DEI LOGS SU HF SPACES

Obiettivo:
Esercitare come pubblicare una nuova versione dell'app (rollout) su Hugging Face Spaces e monitorare i logs per il debug.

1. **Deploy/Rollout su HF Spaces**

- Ogni push su **main** (via Git oppure upload da interfaccia) esegue automaticamente il deploy della nuova versione.
- Il deployment è continuo: ogni modifica al codice/caricamento di file avvia un rebuild automatico.
- **Rollback:** Se la nuova versione ha problemi, puoi ripristinare una versione precedente tramite la cronologia dei commit (tab "Files and versions").

2. **Visualizzazione e Analisi dei Logs**

- Nella pagina del tuo Space, vai su **Settings** > **Logs**.
- Qui visualizzi in tempo reale stdout/stderr prodotti dal tuo script (print, errori, warning).
- Usa i logs per:
  - Debug di errori
  - Monitoraggio del comportamento dell'app
  - Analisi delle richieste utenti

ROLLOUT E GESTIONE DEI LOGS SU HF SPACES

3. Best Practice

- Scrivi messaggi chiari nei logs (print("Inizio elaborazione..."))
- Non loggare dati sensibili (es. chiavi API)
- In caso di errori, fornisci indicazioni utili per la correzione (stacktrace, descrizione problema)
- Ogni modifica = nuovo deploy.
- I logs sono accessibili da browser per analisi immediata e debug.

ESERCIZIO PERSONALE

Deploy end-to-end di un’app NER+RAG su Hugging Face Spaces e Azure OpenAI Services

Obiettivo:
Trasferire la vostra applicazione locale (NER + Streamlit, con backend GPT-4 o4 via Azure API) su Hugging Face Spaces, configurando l’uso sicuro di secrets/variabili d’ambiente e garantendo l’accesso via browser.

- Separare codice, dati e segreti
- Effettuare un deploy sicuro
- Integrare componenti locali e cloud su Hugging Face Spaces

Condividi il link della tua Space e un breve README operativo per l’uso!

Extra:
Logging (in modo sicuro) delle chiamate cloud per troubleshooting/debug, senza esporre dati sensibili.

CHECKLIST PER LA PUBBLICAZIONE DI UNA GENAI APP

1. **Codice funzionante**
   - Tutte le funzioni principali testate localmente
   - Gestione degli errori robusta
2. **Requisiti specificati**
   - File requirements.txt o environment.yml completo
   - Versioni compatibili dei pacchetti
3. **Gestione delle variabili segrete**
   - Nessuna chiave/API hardcoded nel codice
   - Variabili d'ambiente o secrets impostate
4. **File di configurazione**
   - Eventuale .env o file di settings incluso/escluso dal repo (usa .gitignore)
   - Parametri facilmente modificabili
5. **Documentazione minima**

   README.md chiaro con:
   - Descrizione app e uso
   - Esempi di input/output
   - Istruzioni per il deploy (se necessario)

CHECKLIST PER LA PUBBLICAZIONE DI UNA GENAI APP

6. **Test di compatibilità**
   - Prova su un ambiente pulito/simile alla destinazione (es. container, VM, Space)
   - Controllo dipendenze e permessi
7. **Logs**
   - Print di stato e log degli errori implementati
   - Niente dati sensibili nei logs
8. **Gestione delle risorse**
   - App ottimizzata per RAM, CPU, disco
   - Seleziona tier hardware adeguato al carico
9. **Privacy & Compliance**
   - Nessun dato sensibile salvato senza consenso
   - Rispetto policy GDPR/EU AI Act se necessario
10. **Pronto al deploy**
    - Push del codice (Git) o upload
    - Monitoraggio del deploy e logs
    - Primo test di funzionamento sul server/Space

DOMANDE?

Esercizi

GRAZIE PER L'ATTENZIONE