Profice

AI ACADEMY

Applicare l’Intelligenza Artificiale nello sviluppo software

AI ACADEMY

Come costruire un RAG
23/06/2025

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

- Valutazione degli esercizi con LLM.

- Cos’è il retrieval-augmented generation (RAG)
- Architettura tipica e varianti
- Integrazione di knowledge base e LLM (es. Azure, Hugging Face)
- Dataset e casi d’uso reali (domande su dati aziendali, ricerca in documenti, FAQ)
- Esercizio guidato: pipeline RAG con documenti aziendali (testo, PDF)
- Prompt engineering per RAG (query, contestualizzazione, estrazione risposta)
- Concetti per l’utilizzo di RAG
- Metodi di valutazione dell’output RAG

INTRODUZIONE A NATURAL LANGUAGE PROCESSING

Utilizzi tipici del Natural Language Processing (NLP)

Classificazione del Testo

Ricerca di Informazioni (Information Retrieval)

Riconoscimento di Entità Nominate (NER)

Correzione Grammaticale e Ortografica

Analisi del Sentiment

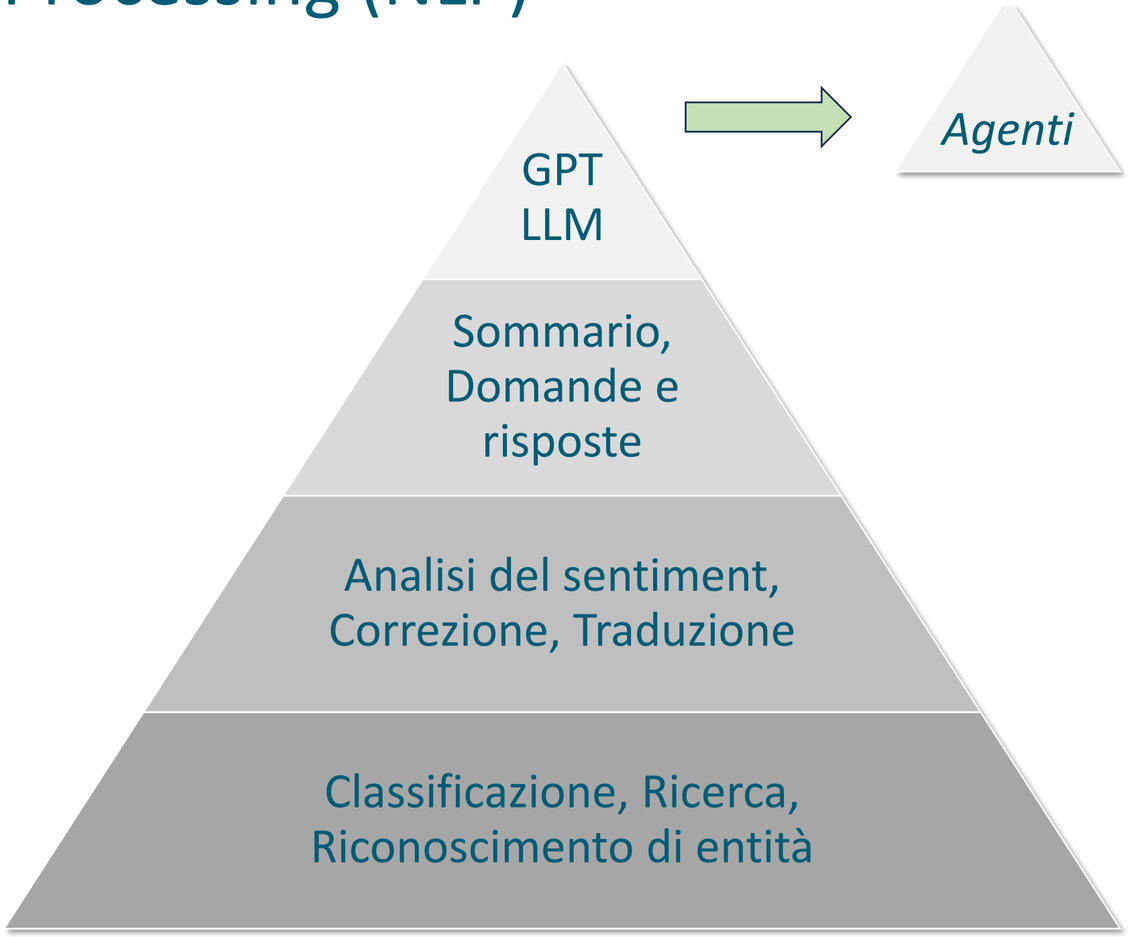
Sommario Automatico

Traduzione Automatica

Generazione del Linguaggio Naturale (NLG)

Domande e Risposte (Question Answering)

Chatbot e Assistenti Virtuali



ANALISI DOCUMENTALE E PROTEZIONE DATI

SmartDocs Srl – Analisi documentale e protezione dati

Scenario:
SmartDocs Srl, media azienda europea, deve gestire email e documenti contenenti dati sensibili di clienti (es: IBAN, codice fiscale, indirizzi, nomi, numeri di telefono).

L’azienda vuole automatizzare:

- Estrazione di entità e dati sensibili (NER, pattern matching)
- Riepilogo automatico e risposta alle richieste clienti
- Tutelare la privacy (alcuni dati NON devono mai lasciare il server locale)
- Minimizzare i costi cloud e garantire risposte rapide

ANALISI DOCUMENTALE E PROTEZIONE DATI

Useremo due modelli LLM per la soluzione aziendale

Architettura semplificata

1. **Modello locale open source (es: TinyLLaMA)**
    1. Viene eseguito localmente, direttamente sul vostro computer o su server aziendali.
    2. Si occupa delle operazioni più “sensibili”, come l’estrazione di dati personali e la protezione della privacy (Named Entity Recognition e masking dei dati).
    3. È veloce, economico, e mantiene i dati riservati all’interno dell’azienda.

2. **Modello cloud avanzato (es: Azure OpenAI GPT-3.5/4)**
    1. Viene utilizzato tramite API esterne, in cloud.
    2. Si occupa di attività più complesse come il riepilogo automatico, l’analisi semantica e la generazione di risposte ai clienti.
    3. Permette di gestire testi lunghi e offre maggiore potenza di calcolo e qualità delle risposte.

ANALISI DOCUMENTALE E PROTEZIONE DATI

L’obiettivo:

Sfruttare **i punti di forza di entrambi i modelli**:

- La privacy e la velocità del modello locale
- La potenza e la flessibilità del modello cloud

Garantire che **i dati più delicati non escano dall’azienda**, mentre sfruttiamo le migliori tecnologie disponibili per la produttività e l’automazione.

Distribuzione dei lavori

- Tutti lavorano sullo **stesso problema reale** ma con strumenti diversi.

Lavoro locale autonomo

- privacy, sicurezza, regex/NER, masking.

Lavoro cloud con l’aiuto dell’istruttore

- prompt, parametri, API, gestione delle risposte.

Altri componenti futuri per i giorni successivi

- RAG, contestualizzazione, configurazione, deployment ecc.
- La **collaborazione** tra i modelli con una architettura AI.

RUOLI DEI DUE MODELLI

1. Modello locale (TinyLLaMA o simili) — “Difensore/controllore”
Viene **eseguito localmente.**
Obiettivi:

- **NER (Named Entity Recognition)**: trova nomi, indirizzi, IBAN, codice fiscale, numeri, ecc.
- **Pattern Detection**: segnala se sono presenti dati sensibili o “red flag”.
- **RAG** (Retrieval Augmented Generation) opzionale: usa una knowledge base aziendale locale per arricchire le risposte.
- **Controlla l’output** prima che venga inviato al modello cloud, mascherando o anonimizzando i dati sensibili (es. sostituendo “Mario Rossi” con “[NOME]”).

2. Modello cloud (Azure GPT-3.5, GPT-4, ecc) — “Analista/colloquio”
Viene usato tramite API cloud, guidato passo-passo.
Obiettivi:

- Riceve documenti **già “ripuliti”** o parzialmente anonimizzati dal modello locale.
- Esegue:
  - Riepilogo (summary)
  - Analisi semantica
  - Generazione di risposte per i clienti
- Gestisce solo dati che non violano la privacy.

ESEMPIO DI PIPELINE COLLABORATIVA

Input:
L'utente carica un documento/email.

Passo 1 (locale):
Il modello locale fa NER, segnala dati sensibili e li anonimizza (es: "Mario Rossi" → "[NOME]", "IT60X0542811101000000123456" → "[IBAN]").

Passo 2 (controllo):
L'output ripulito viene controllato/validato (gli esperti possono anche vedere che i dati sono davvero rimossi).

Passo 3 (cloud):
Il testo anonimizzato viene inviato all'API Azure che fa il riepilogo, classifica la richiesta e prepara una risposta.

Passo 4 (output):
L'output finale può essere ricomposto localmente, reinserendo alcune entità dove permesso, oppure consegnato così.

TIPI GENERALI DI MODELLI

Categorie di modelli:

- **Text generation** (generazione testo)
- **NER** (Named Entity Recognition)
- **Summarization** (riepilogo)
- **Translation** (traduzione)
- **Classification** (classificazione)
- **Embeddings** (vettorizzazione)

Anche se i modelli grandi sono capaci a ssvolgere tutti questi compiti, molte volte i modelli dedicati per un singolo compito, sono addestrati meglio per ottenere miglior risultato.

Allo stesso tempo, i modelli più piccoli con un unico scopo richiedono meno risorse GPU, e così il loro utilizzo costa meno per l'azienda.

USO DI HUGGINGFACE

Autenticazione Hugging Face da command line

1. Prendi il tuo access token

Segui i passaggi già indicati:

Vai su https://huggingface.co/settings/tokens

Crea un nuovo token “Read” e copia il valore. Seleziona il modello per garantire accesso.

2. Installa il client Hugging Face

```
pip install huggingface_hub
```

3. Login dalla command line

```
huggingface-cli login
```

oppure

```
huggingface-hub login
```

4. Incolla il token e premi Invio, dopo potrai inviare programmi

5. Usare il modello dedicato per esercizi NER.

DOMANDE?

Discussione dei risultati

DOMANDE?

PAUSA

IL RETRIEVAL-AUGMENTED GENERATION (RAG)

Definizione

Il Retrieval-Augmented Generation (RAG) è un'architettura che combina la potenza dei modelli linguistici (LLM) con la capacità di recuperare informazioni da basi di dati esterne o documenti.

Come funziona:

- Quando riceve una domanda, il sistema RAG cerca ("retrieval") nei documenti o database rilevanti per trovare informazioni utili.
- Poi il modello LLM genera la risposta ("generation") utilizzando sia il prompt sia i contenuti recuperati.
- Questo permette di produrre risposte aggiornate, basate su dati aziendali, documentazione tecnica, o qualsiasi knowledge base personalizzata.

Perché è importante:

- Supera il limite della conoscenza interna degli LLM.
- Rende le risposte più affidabili, aggiornate e personalizzate.

ARCHITETTURA TIPICA E VARIANTI DI RAG

Input:
L'utente invia una domanda o richiesta.

Retrieval:
Un modulo di ricerca trova documenti o passaggi rilevanti da una knowledge base (database, file, web, ecc.).

Generazione:
Il modello LLM riceve sia la domanda sia le informazioni recuperate e genera una risposta combinata e contestualizzata.

Varianti di architettura:

- **RAG standard:** Retrieval + Generation, una sola iterazione.
- **Iterative RAG:** Il processo retrieval-generation può essere ripetuto per migliorare la risposta.
- **Hybrid RAG:** Combina retrieval semantico (basato su embedding) e retrieval classico (full-text).
- **Tool-augmented RAG:** Il LLM può chiamare strumenti esterni oltre a consultare la knowledge base (es. API, funzioni aziendali).

ESERCIZIO PRATICO: RAG SEMPLICE

Obiettivo:
Realizzare un prototipo semplicissimo di RAG usando i modelli di giorno 5 (Azure OpenAI – GPT o4).
Lo scopo business è di definire il tipo del documento fornito.

Istruzioni:

1. **Creati diversi tipi di documenti brevi** (mail, nota di credito, ordine di acquisto, contratto, ecc.)
2. **Usa il testo in Python** (in un file di testo).
3. **Formula una domanda** che richieda una risposta specifica basata solo su quel documento
   Esempio: determina la categoria aziendale del testo
4. **Prompt engineering:**
   - Istruzione generale
   - Indicazione per usare il documento
   - Incollare il testo
   - Restrizione per rispondere alla base del testo
5. **Testing con diverse domande e prompt**

Obiettivo finale:
Esercizio e validazione del retrieval diretto e come cambiano le risposte del modello secondo i tipi di documenti.

ANALISI DI RETRIEVAL-AUGMENTED GENERATION

Nella tecnica RAG un modello LLM viene “aiutato” con informazioni recuperate da una knowledge base esterna prima di generare la risposta.
Il retrieval può essere semplice (manuale) o avanzato (automatico con pipeline e strumenti).

RAG “manuale”

- Anche copiare e incollare un pezzo di testo direttamente nel prompt e chiedere al modello di usarlo per rispondere, è di fatto una forma base di RAG.
- In questo caso, la fase di “retrieval” è manuale.
- Il modello esegue la “generation”, e genera la risposta usando il testo fornito.
- È il modo più semplice per vedere subito i vantaggi di fornire contesto, senza bisogno di pipeline complesse.

RAG “automatica”

- In produzione o per grandi moli di dati, il retrieval viene automatizzato:
  1. Il sistema cerca automaticamente i documenti più rilevanti usando search, embedding, similarity, ecc.
  2. Il risultato (pezzi di testo) viene dato come input al LLM.
- **La logica di fondo è la stessa**:
  prima retrieval, poi generation.

MODI AVANZATI PER REALIZZARE RAG

Retrieval selettivo:

- Usi uno script, un sistema di search, o una libreria (es. embedding + similarity search, full-text search…)
- **Si recuperano SOLO le informazioni rilevanti** (ad es. solo 1-3 risultati più pertinenti alla domanda).
- Si mettono solo quei risultati nel prompt.
- Si fa la valutazione del RAG

Strumenti avanzati

- **LangChain, LlamaIndex**: automatizzano retrieval, chunking, costruzione prompt, ecc.
- **Database vectoriali** (es: FAISS, Pinecone, ChromaDB):
  Per gestire retrieval efficiente anche su database di milioni di record.

In sintesi

- Per un database piccolo: **si mette nel prompt**
- Per uno grande: **retrieval selettivo → solo i documenti più rilevanti** finiscono nel prompt

VETTORIZZAZIONE PER REALIZZARE RAG

1. **Vettorizzazione (Embedding)**
   - **Database esterno**:
     Ogni documento/frammento viene trasformato in un vettore numerico (embedding) usando un modello apposito.
   - **Richiesta utente**:
     Anche la domanda viene trasformata in embedding (con lo stesso metodo).
   - **Strumenti**:
     - SentenceTransformer
     - text-ada-002
     - MiniLM
2. **Similarity Search**
   - Si calcola la **distanza** (coseno, L2, ecc.) tra l'embedding della richiesta e quelli dei documenti.
   - Si selezionano i N più simili (tipicamente 1-5), cioè quelli **più rilevanti** rispetto alla domanda.
3. **Prompt Building**
   - Solo i documenti trovati con la similarity search vengono inclusi nel prompt finale per il modello LLM.

VETTORIZZAZIONE PER REALIZZARE RAG

	text-embedding-ada-002 (OpenAI)	MiniLM (es. all-MiniLM-L6-v2, SBERT)
Tipologia	Proprietario, cloud API (OpenAI)	Open-source, Hugging Face/SBERT
Lingue	Multi-lingua, ottimo su EN/IT/altre	Multi-lingua, ottimo su EN/IT
Performance	Eccellente, SoTA	Ottima (molto competitivo)
Costo	$ (a consumo API)	Gratis (locale, GPU/CPU)
Velocità	Veloce (API), batch ottimizzato	Molto veloce su CPU/GPU
Scalabilità	Altissima (API, cloud)	Buona (dipende da tua macchina)
Limiti	Richiede chiave, soggetto a policy	Nessun limite d'uso
Privacy	I dati passano su cloud OpenAI	Restano locali (privacy totale)
Dimensione embedding	1536	384 (all-MiniLM-L6-v2), 768/1024 in altri
Uso ideale	Produzione, sistemi di grandi dimensioni	Prototipi, didattica, deployment on-prem

VETTORIZZAZIONE PER REALIZZARE RAG

Perché è potente

Scalabilità:
Puoi avere migliaia o milioni di documenti.

Efficienza:
Solo pochi testi vengono “letti” dal LLM ogni volta.

Qualità:
La risposta sfrutta solo le fonti più pertinenti.

Strumenti tipici

- **LangChain**, **LlamaIndex**: per orchestrare tutta la pipeline.
- **Database vettoriali**: FAISS, Pinecone, ChromaDB, Weaviate, Milvus…

ESERCIZIO: CONFRONTA EMBEDDING DI FRASI

Obiettivo:

Osservare come gli embedding rappresentano la similarità semantica tra frasi.

Istruzioni:

1. Scegli un modello di embedding (es. MiniLM, SentenceTransformer, o altro).
2. Inserisci le seguenti 3 frasi ad esempio:
    - Luca ha comprato una macchina nuova.
    - Luca si è appena comprato una macchina nuova.
    - Oggi piove molto a Milano.
3. Calcola l'embedding vettoriale di ciascuna frase.
4. Calcola la similarità (es: cosine similarity) tra:

   La prima e la seconda frase (simili)

   La prima e la terza frase (diverse)

Osserva e discuti:

Le frasi simili hanno vettori più "vicini"?

Le frasi diverse sono distanti nello spazio degli embedding?

Conclusione:

La potenza di embedding semantico è trasformare il significato del testo in vettori.

DOMANDE?

PAUSA

METODI DI VALUTAZIONE DELL'OUTPUT RAG

Perché valutare l'output RAG?
È essenziale misurare la qualità sia del retrieval (ricerca documenti) sia della risposta generata dall'LLM, per garantire risposte affidabili e pertinenti.

Metriche chiave:

- **Precision@k:**
  Percentuale di casi in cui la risposta giusta si trova tra i primi *k* documenti recuperati dal sistema.
  Esempio: Precision@3 = 80% → 8 volte su 10 la risposta era tra i primi 3 risultati.
- **Answer correctness (correttezza della risposta):**
  Quanto la risposta finale del modello è effettivamente corretta, completa e supportata dai documenti forniti.
  Valutabile sia manualmente (umano controlla) che automaticamente (con test o dataset etichettato).
- **Faithfulness (fedeltà alle fonti):**
  La risposta è davvero giustificata dai testi recuperati, senza "hallucination".

Coverage (copertura):
Quante domande ricevono risposta grazie al retrieval, rispetto al totale delle richieste.

DOCUMENT LOADER & METADATA

Concetto:
Il document loader è il componente che importa dati esterni (PDF, Word, email, database, siti web…) e li rende disponibili per la pipeline RAG.

Metadata:
Informazioni aggiuntive su ogni documento (es. autore, data, fonte, categoria)
usate per filtrare, cercare e ordinare i risultati.

Esempio:
Caricare PDF aziendali, per ognuno creare la struttura:

```
{ "titolo": "...", "data": "...", "autore": "...", "testo": "..." }
```

CHUNKING STRATEGICO E MODEL COMPARISON

Concetto:
Il chunking suddivide i documenti in pezzi (“chunk”) di dimensione ottimale per il retrieval e il prompt (es. paragrafi, frasi, blocchi di 200 parole).
Strategia:

- Chunk troppo lunghi → info dispersa.
- Chunk troppo corti → risposte frammentate.
- LangChain offre strumenti e strategie per il *chunking* ideale.

Esempio:
Un regolamento di 10 pagine → 40 chunk da un paragrafo ciascuno.

Concetto:
Gli embedding models trasformano testi in vettori numerici.
La scelta del modello influisce su qualità, costo, performance.
Confronto:

- text-embedding-ada-002 (OpenAI): molto preciso, cloud, a pagamento.
- MiniLM (open-source): veloce, gratuito, usabile localmente, qualità spesso sufficiente.

Quando preferire uno o l’altro?
Dipende da privacy, budget, qualità richiesta.

VECTORSTORE (FAISS, QDRANT, PINECONE)

Concetto:
Un VectorStore salva i vettori embedding dei documenti e permette di cercare velocemente i più simili tramite similarity search.

Soluzioni comuni:

- **FAISS:** open-source, locale, molto veloce.
- **Qdrant, Pinecone:** cloud, scalabili, facili da integrare con pipeline moderne.

Esempio:

- 50.000 documenti aziendali → FAISS li indicizza e li cerca in pochi millisecondi.

SIMILARITY METRICS E HYBRID SEARCH

Concetto:
Le similarity metrics misurano “quanto” due vettori sono simili (quindi quanto due testi sono vicini come significato).

Le principali:

- **Cosine similarity:** misura l’angolo tra i vettori (più usata, valori da -1 a 1).
- **Dot product:** prodotto scalare, semplice e veloce.
- **L2 (euclidean):** distanza “classica” tra punti nello spazio.

Scelta:
Cosine quasi sempre preferita per il testo.

Concetto:
Hybrid search unisce la ricerca classica (keyword, es. BM25) con la ricerca semantica (embedding).
Prima si recuperano i risultati “grezzi” per keyword, poi si riordinano con la similarità vettoriale.
Vantaggio:
Migliora la pertinenza e la copertura delle risposte, utile con dati rumorosi o eterogenei.

CONVERSATIONAL MEMORY & TOKEN WINDOW

Concetto:

Conversational memory: il sistema tiene traccia della conversazione precedente per risposte coerenti.
Token window: ogni modello ha un limite di “finestra di contesto” (max token gestibili), oltre il quale “dimentica” il passato.

Implicazione:
Serve scegliere cosa ricordare, cosa scartare e come riassumere la memoria.

Attenzione:
Questi concetti sono simili, ma diversi dalla sessione.

CONTEXT INJECTION E LLM ANSWER FLOW

Concetto:
Il contesto recuperato (chunk/documenti) viene “iniettato” nel prompt che l’LLM userà per rispondere.

Flow tipico:

- Domanda utente
- Retrieval dei chunk
- Prompt costruito con i chunk
- LLM genera la risposta basata solo su quel contesto

ESERCIZIO: PROGETTA UNA PIPELINE RAG CON LANGCHAIN

Introduzione a LangChain

LangChain è una libreria Python che semplifica la creazione di pipeline intelligenti con LLM e retrieval.

Permette di collegare in pochi passaggi:

- Caricamento di documenti
- Suddivisione in chunk
- Creazione di embedding
- Ricerca e recupero (retrieval)
- Prompting verso il modello
- (e molto altro)

Con LangChain puoi costruire **velocemente sistemi RAG** su documenti aziendali, PDF, email, database, ecc.

GRAZIE PER L'ATTENZIONE