Profice

AI ACADEMY

Applicare l’Intelligenza Artificiale nello sviluppo software

AI ACADEMY

Knowledge Base & Graph
03/07/2025

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

- Comprendere quando e perché utilizzare una Knowledge Base o un Knowledge Graph nei progetti di AI.
- Capire la differenza tra Knowledge Graph, Vector DB e Database relazionali.
- Imparare i concetti base di RDF e Property Graph.
- Saper modellare dati in Neo4j: nodi, relazioni, attributi, constraint.
- Imparare a scrivere query Cypher per ricerca e analisi dati.
- Integrare un Knowledge Graph con modelli NER o GPT per potenziare l'analisi semantica.
- Valutare pro e contro di varie architetture.
- Sviluppare e interrogare un piccolo Knowledge Graph da zero.
- Esercitarsi su use case pratici (analisi relazioni, motori RAG avanzati, QA su KG).

RIPRENDIAMO L'ESERCIZIO: MODELLO SMS E ANALISI

Esercizio: Addestramento modello SMS e analisi della conformità EU AI Act

Contesto
I messaggi SMS vengono spesso usati per comunicazioni aziendali, supporto clienti e notifiche di servizio. L'analisi automatica degli SMS, ad esempio per classificare i messaggi in "spam" e "non spam", è un tipico caso d'uso di AI in azienda.

Obiettivo:
Addestrare un modello di classificazione su un dataset di SMS per distinguere tra messaggi "spam" e "ham" (non spam).

Utilizzare il dataset pubblico "SMS Spam Collection" disponibile su Kaggle.

ESERCIZIO: FINE-TUNING DEL MODELLO SMS

Esercizio: Fine-tuning del modello SMS

Compito
Analizza I messaggi SMS e usa un Foundation Model (GPT, Gemini ecc.) per generare altri esempi SMS per migliorare il riconoscimento dei spam. Quanto percento di nuovi dati pendi di creare?

Obiettivo:
Migliorare il risultato di riconoscimento dei messaggi “spam” e anche “ham” (non spam) se ci sono falsi negativi.

Aumentare il dataset e rifare l’addestramento.

ESERCIZIO: SVILUPPO DEL MODELLO SMS

Esercizio: Cambiare classificazione SMS da Spam/Ham a Sentiment Analysis

Compito
Analizza I messaggi SMS, cambia etichette spam a negativo, non spam (ham) estendi a sentimenti. Per esempio: neutro, positivo (informale, complimento). Usa un Foundation Model per i messaggi non spam soltanto e non per i spam!!

Obiettivo:
Riutilizzare modello e dataset per altro scopo. Usare transfer learning.

Domanda: indovina o trova nome del metodo di transfer learning, un modello grande istruisce un piccolo.

DOCUMENTI DA PREPARARE PER LA CONFORMITÀ

- **Valutazione del rischio**
  Analisi scritta sul livello di rischio associato al sistema AI.
- **Descrizione tecnica del sistema**
  Architettura, funzionamento, scopi e limiti del modello.
- **Dataset statement**
  Elenco, descrizione e provenienza dei dati utilizzati (inclusi eventuali bias e misure correttive).
- **Procedure di gestione del rischio**
  Politiche, misure tecniche e organizzative adottate.
- **Valutazione impatti (AI Impact Assessment/DPIA)**
  Analisi delle conseguenze su diritti, privacy, sicurezza, discriminazione, ecc.
- **Registro delle decisioni e log**
  Documentazione delle scelte, delle correzioni e dei test effettuati.
- **Manuale utente e user notice**
  Informazioni per l'utilizzatore finale sulla natura e i limiti del sistema.
- **Dichiarazione di conformità**
  Documento che attesta il rispetto dei requisiti dell'AI Act.
- **Fascicolo tecnico**
  Raccolta completa di tutti i documenti, disponibile per le autorità.

ESERCIZIO: ANALISI DI CONFORMITÀ SU UN CASO PRATICO

Scenario:
Avete sviluppato un'applicazione che combina un modello NER (Named Entity Recognition) locale e GPT-4 in cloud per analizzare e rispondere automaticamente alle richieste su documenti aziendali.

Compiti:
Scegliere un documento dall'elenco, pianificare e scrivere bozza del documento.

Da consegnare: (dopo di: Identificare a quale "tier" appartiene. Quali obblighi si applicano? Redigere una semplice scheda di valutazione rischi, adempimenti e comunicazione verso l'utente.)

Bozza del documento scelto.

DOMANDE?

PAUSA

KNOWLEDGE BASE (KB)

Definizione
Una **Knowledge Base** (KB) è una raccolta strutturata di informazioni, dati o regole relative a un determinato dominio o argomento.

- Può essere costituita da tabelle, elenchi, manuali, domande/risposte (FAQ), dati estratti da testi o documenti aziendali.
- Le KB possono essere statiche (aggiornate manualmente) o dinamiche (alimentate da sistemi automatici di estrazione e aggiornamento).

Scopo: rendere facilmente accessibili le informazioni, supportare processi decisionali e automatizzare risposte o attività.

Esempio: una KB aziendale contiene elenchi di prodotti, procedure interne, manuali per clienti, regole operative.

KNOWLEDGE GRAPH (KG)

Knowledge Graph (KG)

Un **Knowledge Graph (KG)** è una rappresentazione strutturata della conoscenza tramite nodi (entità) e archi (relazioni).

- Ogni **nodo** rappresenta un oggetto, concetto o evento (ad es. persona, prodotto, data).
- Ogni **arco** esprime una relazione tra entità (ad es. “lavora per”, “è parte di”, “possiede”).
- I KG collegano informazioni in modo esplicito, permettendo query complesse e inferenze automatiche.

Esempio: Un KG aziendale collega clienti, ordini, prodotti e fornitori tramite le rispettive relazioni.

[Mario Rossi] —lavora_in→ [SmartDocs Srl]

DIFFERENZA TRA KB, KG E VECTOR DB

Knowledge Base (KB):

- Archivio organizzato di dati e regole.
- Spesso formato da tabelle, documenti o database relazionali.
- Adatto per domande dirette e consultazione di dati noti.

Knowledge Graph (KG):

- Struttura a grafo di entità e relazioni.
- Permette navigazione e inferenza sulle connessioni tra dati.
- Ideale per rappresentare domini complessi e trovare pattern nascosti.

Vector Database (Vector DB):

- Archivio di vettori numerici (embedding) che rappresentano contenuti testuali, immagini, ecc.
- Ottimizzato per ricerche di similarità (es. “trova documenti simili”).
- Usato in applicazioni RAG, recommendation, clustering.

Sintesi:

KB: dati tabellari / strutturati.

KG: relazioni esplicite tra entità.

Vector DB: ricerca semantica tramite vettori.

QUANDO USARE KB, KG O VECTOR DB

Knowledge Base (KB):

- Quando servono risposte dirette da un archivio strutturato.
- Adatto per regole fisse, FAQ, glossari.
- Esempio: database delle policy aziendali.

Knowledge Graph (KG):

- Quando si devono mappare relazioni complesse tra entità.
- Ideale per domande che richiedono inferenza o traversing tra nodi.
- Esempio: analisi di reti sociali, mapping di supply chain.

Vector DB:

- Per ricerche di similarità su grandi quantità di dati non strutturati.
- Ideale per casi RAG, recommendation, semantic search.
- Esempio: “trova documenti simili”, clustering di contenuti.

GRAFO, NODI, ARCHI, PROPRIETÀ

Definizione:

Un **grafo** è una struttura dati composta da **nodi** (o vertici) e **archi** (o relazioni) che collegano i nodi tra loro.

- **Nodi:** rappresentano entità o oggetti (es. una persona, un documento, una città).
- **Archi:** rappresentano relazioni tra i nodi (es. “lavora con”, “è situato in”).
- **Proprietà:** sia nodi che archi possono avere attributi o valori associati (es. nome, data, tipo di relazione).

Esempio:

- Nodo: “Mario Rossi” (proprietà: ruolo=Manager)
- Nodo: “Progetto A” (proprietà: status=In corso)
- Arco: “Mario Rossi lavora su Progetto A” (proprietà: data_inizio=2024-05-01)

Applicazione:

Questa struttura permette di modellare e analizzare connessioni complesse, come reti sociali, percorsi logistici, collegamenti tra documenti, ecc.

ESEMPI DI GRAFI NEL MONDO REALE

Reti sociali:
Utenti come nodi, relazioni di “amicizia” o “follow” come archi.
Esempio: Facebook, LinkedIn, X (ex Twitter).

Knowledge Graph di Google:
Entità come persone, luoghi, eventi collegate da relazioni (es. “Niccolò Machiavelli – nato a – Firenze”).

Trasporti e logistica:
Stazioni, aeroporti, magazzini come nodi; tratte e spedizioni come archi.
Esempio: mappa della metropolitana, reti di spedizione pacchi.

Gestione documentale:
Documenti come nodi, riferimenti e citazioni come archi.
Esempio: gestione di contratti e collegamenti tra revisioni.

Biologia:
Proteine, geni, malattie come nodi; interazioni biologiche come archi.
Esempio: reti di interazione genica.

GRAFI VS TABELLE RELAZIONALI

Grafi

- **Modello dati:** Nodi (entità) e archi (relazioni), con proprietà.
- **Flessibilità:** Adatti a dati con molte relazioni, strutture irregolari e dinamiche.
- **Query:** Ricerca e traversing di collegamenti multipli e profondi, anche su dati non strutturati.
- **Performance:** Efficienti su query di relazione complessa (es. “distanza tra due nodi”).
- **Esempi d’uso:** Social network, raccomandazioni, reti di conoscenza, supply chain.

Tabelle relazionali

- **Modello dati:** Tabelle predefinite con righe (record) e colonne (attributi).
- **Flessibilità:** Struttura fissa, difficile aggiungere nuove relazioni senza modificare lo schema.
- **Query:** SQL, ottimo per transazioni, aggregazioni e dati tabellari ben strutturati.
- **Performance:** Eccellenti per join semplici e grandi volumi di dati uniformi.
- **Esempi d’uso:** Database clienti, fatture, contabilità, gestionali classici.

Quando usare i grafi?

- **Quando il fulcro è “come sono collegati” più che “cosa sono”.**
- **Quando le relazioni** cambiano spesso o sono molto numerose.

RDF – RESOURCE DESCRIPTION FRAMEWORK

Definizione

RDF (Resource Description Framework) è uno standard del W3C per rappresentare informazioni strutturate come “triplette” (soggetto, predicato, oggetto), facilitando la condivisione e l’integrazione di dati tra sistemi diversi.

Come funziona:

I dati sono espressi come **triplette**:

- **Soggetto:** l’entità di cui si parla (es: Contratto123)
- **Predicato:** la proprietà o relazione (es: firmatoDa)
- **Oggetto:** il valore o altra entità collegata (es: Mario Rossi)

Tutte le entità e proprietà sono identificate da URI univoci.

Esempio:

```
Contratto123   firmatoDa     MarioRossi
Fattura456     riferitaA     ProgettoX
Cliente789     haIndirizzo   ViaRoma10
```

Queste triplette possono essere memorizzate in database specializzati (triplestore) e interrogate tramite linguaggi come SPARQL.

Vantaggi principali:

- Facilita l’interoperabilità tra sistemi e fonti dati diverse.
- Ottimo per collegare informazioni eterogenee (open data, web semantico).
- È la base tecnica di molti knowledge graph moderni, anche su larga scala.

PROPERTY GRAPH MODEL

Definizione

Il Property Graph Model è un modello di dati per rappresentare grafi in cui sia i nodi che le relazioni (archi) possono avere proprietà (attributi chiave-valore).

Struttura:

- **Nodi (Vertices):** rappresentano entità (es: Cliente, Contratto, Fattura).
- **Relazioni (Edges):** collegano i nodi e possono avere una direzione (es: “haFirmato”, “riguarda”).
- **Proprietà:** sia nodi che relazioni hanno attributi associati (es: nome, data, importo).

Esempio:

```
(Cliente: Mario Rossi {email: "mario@azienda.it"})
   ──[haFirmato {data: "2024-06-01"}]──>
(Contratto: ABC123 {valore: 10_000})
   ──[riguarda]──>
(Progetto: SmartDocs {start: "2024-03-15"})
```

Vantaggi principali:

- Più espressivo dei semplici grafi; permette di modellare scenari aziendali reali con molti dettagli.
- Supportato da database moderni (Neo4j, TigerGraph).
- Ottimo per analisi di collegamenti complessi, ricerche flessibili e tracciabilità di relazioni.

DOMANDE?

Esercizi

ESTRAZIONE ENTITÀ DA DOCUMENTI AZIENDALI (NER)

Scenario:
Applichi NER su diversi documenti (email, fatture, contratti) per estrarre entità rilevanti:

- Nomi persone (clienti, fornitori, referenti)
- Date (scadenze, firma)
- Importi, codici fiscali, IBAN
- Nomi di aziende, prodotti

Tipo di grafo: Property Graph

Struttura di esempio:
Nodi: Documento, Entità (Cliente, Fornitore, Data, Importo, Prodotto)
Archi:
Documento “contiene” Entità
Entità “collega” più documenti (es. stesso cliente in più fatture)

Uso pratico:
Cerca tutte le fatture sopra una certa soglia, tutti i contratti con un certo cliente, ecc.

ESTRAZIONE ENTITÀ DA DOCUMENTI AZIENDALI (NER)

1. **Crea branch o copia del progetto NER per esercitare graph dati.**
2. **Normalizza e pulisci le entità:**
   Uniforma formati (es. date, nomi azienda, numeri). Rimuovi duplicati.
3. **Assegna tipi e relazioni:**
   Decidi che tipo di nodo sarà ogni entità (es. “Cliente”, “Fornitore”, “Data”, “Importo”) e quali relazioni collegano entità (es. “emesso da”, “inviato a”, “pagato il”).
4. **Costruisci la struttura del grafo:**
   Usa una libreria (es. **`networkx`**) per creare nodi (entità) e archi (relazioni) secondo il tuo schema.
5. **Verifica e arricchisci il grafo:**
   Puoi aggiungere nuove fonti di dati, linkare entità già esistenti, aggiornare proprietà.
6. **(Opzionale)**: Usa `nx.draw` ecc., per visualizzare il graph.

ESTRAZIONE ENTITÀ DA DOCUMENTI AZIENDALI (NER)

Esempio di entità estratte

Testo:

"Fattura n.123 emessa da Alfa Srl il 5 giugno 2024, inviata a Beta Srl per l'importo di 2000 euro."

NER (ipotetico):

- **Alfa Srl** → Azienda
- **Beta Srl** → Azienda
- **5 giugno 2024** → Data
- **2000 euro** → Importo
- **n.123** → NumeroFattura

Assegna i tipi ai nodi

Ogni entità avrà un tipo (**label**) specifico:

- "Alfa Srl" → **Cliente** (o Fornitore, a seconda del contesto)
- "Beta Srl" → **Cliente** (o Fornitore)
- "5 giugno 2024" → **Data**
- "2000 euro" → **Importo**
- "n.123" → **NumeroFattura**

Definisci le relazioni

Relazioni tipiche per documenti aziendali:

- **emesso da** (fattura → fornitore)
- **inviato a** (fattura → cliente)
- **pagato il** (fattura → data)
- **importo** (fattura → importo)

ESTRAZIONE ENTITÀ DA DOCUMENTI AZIENDALI (NER)

Crea un grafo dove ogni nodo ha un **tipo** (proprietà tipo) e ogni collegamento rappresenta una **relazione** reale del contesto aziendale.

```
import networkx as nx
G = nx.DiGraph()

# Nodi
G.add_node("n.123", tipo="NumeroFattura")
G.add_node("Alfa Srl", tipo="Fornitore")
G.add_node("Beta Srl", tipo="Cliente")
G.add_node("5 giugno 2024", tipo="Data")
G.add_node("2000 euro", tipo="Importo")

# Relazioni
G.add_edge("n.123", "Alfa Srl", relazione="emesso da")
G.add_edge("n.123", "Beta Srl", relazione="inviato a")
G.add_edge("n.123", "5 giugno 2024", relazione="pagato il")
G.add_edge("n.123", "2000 euro", relazione="importo")
```

DOMANDE?

PAUSA

NEO4J: IL DATABASE GRAFICO

Neo4j:

Database NoSQL orientato ai grafi, leader di mercato per la gestione, analisi e query di dati collegati tramite nodi e relazioni.

Principali caratteristiche:

- Architettura nativa a grafo (ogni dato è nodo, relazione o proprietà)
- Linguaggio di query: Cypher (intuitivo e potente)
- Visualizzazione e analisi dei grafi via interfaccia web
- Supporta sia property graph che import da RDF
- Ottimizzato per query su dati fortemente collegati (es. social network, knowledge graph aziendali)

Applicazioni tipiche:

- Knowledge graph aziendali e cataloghi
- Analisi di reti (clienti, fornitori, email, supply chain)
- Rilevamento frodi, raccomandazioni, ricerca semantica

Motivi per usarlo:

- Semplice da installare (anche locale, cloud e Docker)
- Community ampia e tanti tutorial disponibili
- Ottimo supporto Python tramite librerie dedicate (py2neo, neomodel, official driver)

CYPHER: LINGUAGGIO DI QUERY DEI GRAFI

Cos'è Cypher?

Linguaggio di query dichiarativo sviluppato per Neo4j.
Permette di creare, leggere, aggiornare ed eliminare dati nei grafi (CRUD).
Simile a SQL per i grafi, ma ottimizzato per nodi e relazioni.

Sintassi base: pattern matching

Trova nodi e relazioni con una sintassi leggibile:

```
MATCH (p:Persona)-[:LAVORA_PER]->(a:Azienda)
RETURN p.nome, a.nome
```

Creazione di nodi e relazioni

Aggiungi nuovi dati nel grafo:

```
CREATE (c:Cliente {nome: "Mario Rossi"})
CREATE (f:Fattura {numero: "INV2025"})
CREATE (c)-[:HA_RICEVUTO]->(f)
```

Filtri e condizioni

Restringi la ricerca con WHERE:

```
MATCH (f:Fattura)
WHERE f.importo > 1000
RETURN f.numero, f.importo
```

CYPHER: LINGUAGGIO DI QUERY DEI GRAFI

Aggiornamento e cancellazione
Modifica o elimina dati:

```
MATCH (c:Cliente {nome: "Mario Rossi"})
SET c.email = "mario.rossi@email.com"

MATCH (f:Fattura {numero: "INV2025"})
DELETE f
```

Query aggregate
Statistiche e aggregazioni:

```
MATCH (f:Fattura)
RETURN avg(f.importo) AS importo_medio
```

Esempio pratico:
Trovare tutti i clienti che hanno ricevuto fatture sopra i 5000€:

```
MATCH (c:Cliente)-[:HA_RICEVUTO]-
>(f:Fattura)
WHERE f.importo > 5000
RETURN c.nome, f.numero, f.importo
```

Vantaggi di Cypher

- Semplice, espressivo, progettato per grafi.
- Visualizzazione immediata delle relazioni.

ESERCIZIO: KG + LLM

Task:
Poni una domanda sul Knowledge Graph Neo4j (esempio:
“Quali clienti hanno almeno un contratto attivo nel 2024?”).

Scrivi una query Cypher per estrarre la risposta dal grafo.

Prendi la risposta della query (in formato tabellare o testo).
Passa la risposta a GPT/LLM, chiedendo di riformulare in linguaggio naturale chiaro per il business.

Obiettivo:
Sperimentare la combinazione tra ricerca strutturata (Cypher su Knowledge Graph) e generazione di testo (LLM/GPT) per ottenere risposte business-ready.

Extra:
- Cambia la domanda o la struttura dei dati.
- Valuta la qualità della risposta riformulata.

NEO4J: IL DATABASE GRAFICO

Installa Neo4j Desktop o avvia un database Neo4j (anche Docker va bene).
Scarica Neo4j Desktop https://neo4j.com/download/

Installa il driver Python: `pip install neo4j py2neo`

```python
from neo4j import GraphDatabase
uri = "bolt://localhost:7687"
driver = GraphDatabase.driver(uri, auth=("neo4j", "password"))
with driver.session() as session:
    result = session.run("MATCH (n) RETURN n LIMIT 5")
    for record in result:
        print(record)


from py2neo import Graph
uri = "bolt://localhost:7687"
graph = Graph(uri, auth=("neo4j", "password"))
results = graph.run("MATCH (p:Persona) RETURN p.nome LIMIT 5")
for record in results:
    print(record)
```

NEO4J: USARE NODI E RELAZIONI

```
from py2neo import Graph, Node, Relationship
graph = Graph("bolt://localhost:7687", auth=("neo4j", "tua_password"))

# Nodi
persona = Node("Persona", nome="Mario Rossi")
azienda = Node("Azienda", nome="SmartDocs Srl")
fattura = Node("Fattura", codice="Fattura123", data="2024-05-10")

# Aggiungi nodi (evita duplicati se necessario)
graph.merge(persona, "Persona", "nome")
graph.merge(azienda, "Azienda", "nome")
graph.merge(fattura, "Fattura", "codice")

# Relazioni
rel1 = Relationship(persona, "CLIENTE_DI", azienda)
rel2 = Relationship(fattura, "EMESSA_DA", azienda)
rel3 = Relationship(fattura, "DESTINATARIO", persona)

graph.merge(rel1)
graph.merge(rel2)
graph.merge(rel3)
```

DOMANDE?

Esercizio 1

ESERCIZIO: UN KNOWLEDGE GRAPH DA DATI AZIENDALI

Identifica i tipi di nodi

Esempi: Persona, Azienda, Documento, Progetto, Fattura, IBAN, Email, Data, Indirizzo.

Identifica le relazioni principali:

Esempi:

- HA_FIRMATO (Persona → Contratto)
- EMESSA_DA (Fattura → Azienda)
- PARTECIPA_A (Persona → Progetto)
- INVIATO_A (Email → Persona)
- USA_IBAN (Azienda → IBAN)
- DATA_FIRMA (Contratto → Data)

Stabilisci le proprietà utili per nodi e relazioni:

- Persona: nome, cognome, ruolo, P.IVA
- Azienda: nome, settore, partita IVA
- Documento: tipo, data, importo
- Relazioni: timestamp, stato, motivazione

ESERCIZIO: UN KNOWLEDGE GRAPH DA DATI AZIENDALI

Normalizza le entità:

- Unifica nomi simili (Mario Rossi ≠ M. Rossi), usa codici univoci se possibile.

Crea uno schema di grafo coerente:

- Prima di importare dati, disegna (anche solo su carta) uno schema: quali nodi? quali archi?
- Esempio: [Persona]-[HA_FIRMATO]->[Contratto]-[RIFERITO_A]->[Azienda]

Scegli il database grafo:

- Neo4j (più usato). (**Alternative, altri database grafici:** TigerGraph - analisi su grandi grafi, ArangoDB – multimodello, Amazon Neptune - cloud native)

Carica i dati via script Python:

- Usa py2neo, neomodel o driver ufficiale per Neo4j.
- Carica prima i nodi, poi le relazioni, assicurandoti che le chiavi siano univoche.

Valorizza la ricerca:

Prepara query per rispondere a domande aziendali (Chi ha firmato quali contratti? Quante fatture sopra 10k nel 2024?).

DOMANDE?

Esercizio 2

ESERCIZIO: KNOWLEDGE GRAPH PER L'ANONIMIZZAZIONE

Esercizio in gruppi: Creare un Knowledge Graph per l'Anonimizzazione e il Ripristino

Scenario

Realizzate una pipeline in cui:

1. Estrai entità dai documenti aziendali (es. nomi, IBAN, indirizzi).
2. Inserisci queste entità (con tutte le loro proprietà) in Neo4j come **nodi temporanei**.
3. Sostituisci nei testi originali i dati sensibili con un identificatore anonimo (es. [NOME_123]).
4. Quando necessario, per esempio pr la risposta di mail del modello GPT ecc., **ripristini** i dati originali usando il Knowledge Graph come lookup.

Obiettivo finale

- **Automatizzare** l'anonimizzazione e il ripristino sicuro dei dati, mantenendo la tracciabilità e la privacy tramite il Knowledge Graph.
- **Bonus**: discutere come gestire la scadenza o la cancellazione sicura dei nodi temporanei.
- **Bonus 2**: Visualizza un graph selezionando una persona, azienda ecc. e connetti tutti i suoi documenti come nodi. Segnala i vari tipi di documenti con colori diversi.

ESERCIZIO: KNOWLEDGE GRAPH PER L'ANONIMIZZAZIONE

Esercizio in gruppi: Creare un Knowledge Graph per l'Anonimizzazione e il Ripristino

Compito per il gruppo

1.Definire la struttura del grafo:
- Decidere quali proprietà (es. nome, cognome, id, iban, indirizzo, email) mettere nei nodi.
- Definire relazioni utili (es. APPARTIENE_A, HA_IBAN, ECC).

2.Pipeline di anonimizzazione:
- Dopo il NER, per ogni entità, crea nodo con proprietà e una chiave/ID univoca (es. ID_PERSONA_001).
- Nel testo originale, sostituisci l'entità con l'ID.

3.Ripristino dei dati:
- Scrivi una funzione Python che, leggendo il testo anonimizzato, usa Neo4j per trovare e rimettere i dati reali usando l'ID.

4.Testate la pipeline:
- Esempio pratico con almeno due tipi di entità (es. persona e iban) su un piccolo testo aziendale.

Spunti:
- Le proprietà dei nodi possono essere tutte le info utili (anche hash o timestamp).
- Relazioni possono collegare entità tra loro (es. IBAN → Persona).

VECTOR DATABASE

Cosa sono:

I Vector Database gestiscono **embeddings** (cioè rappresentazioni numeriche di testi, immagini, dati), permettendo ricerche e confronti efficienti nello spazio vettoriale.

Esempi: **FAISS, Pinecone, Qdrant**

Quando si usano:

- Ideali per Retrieval-Augmented Generation (RAG)
- Semantic search avanzata (trovare contenuti simili “nel significato”)
- AI generativa (usare conoscenza esterna in tempo reale)

Caratteristiche:

- Scalano facilmente su grandi volumi di dati non strutturati
- Supportano ricerche “fuzzy” (similitudine, non solo corrispondenza esatta)
- Integrabili in pipeline AI e sistemi di produzione

METRICHE E VALUTAZIONE DELLA QUALITÀ DEL GRAFO

Copertura nodi/relazioni

- Percentuale di entità rilevanti effettivamente rappresentate come nodi.
- Rapporto tra nodi creati e numero di entità reali nei dati.
- Esempio: "Il 98% dei clienti, fornitori e contratti presenti nei dati aziendali sono presenti nel grafo."

Risposta a query complesse

- Capacità del grafo di rispondere a domande articolate che richiedono più passaggi o join.
- Valutare se tutte le relazioni utili sono modellate per ottenere la risposta corretta.
- Esempio: "Il grafo consente di trovare tutti i clienti che hanno almeno tre contratti in corso, con i relativi importi e date."

Tempo di esecuzione query

- Tempo necessario per eseguire query tipiche o avanzate (es. ricerca di pattern, aggregazioni).
- Importante per grafi di grandi dimensioni: si misura la scalabilità e l'ottimizzazione delle strutture dati.
- Esempio: "La query per trovare tutti i fornitori collegati ad almeno due contratti si esegue in meno di 0,2 secondi."

Un buon grafo aziendale deve essere completo, rispondere a query business-critical e mantenere tempi di risposta bassi anche su grandi volumi di dati.

FAISS VS PINECONE: CONFRONTO TRA VECTOR DATABASE

FAISS

Libreria open source sviluppata da Facebook AI Research.

Caratteristiche:

- Installabile localmente (on-premise)
- Altissime prestazioni su grandi quantità di dati (milioni di vettori)
- Supporta molteplici algoritmi di indicizzazione e ricerca approssimata
- Richiede gestione e scalabilità a carico dello sviluppatore

Quando si usa:

- Progetti di ricerca, prototipi o sistemi che restano in azienda
- Quando si vuole pieno controllo sull'infrastruttura

Limiti:

- Nessuna interfaccia web nativa
- Non è un servizio cloud: serve gestire storage, backup, scalabilità, ecc.

FAISS VS PINECONE: CONFRONTO TRA VECTOR DATABASE

Pinecone

Servizio cloud gestito specializzato in Vector Database.

Caratteristiche:

- SaaS: nessuna installazione richiesta, si accede via API
- Scalabilità automatica (storage e performance)
- Alta disponibilità, backup automatici
- Integrazione semplice con pipeline AI e framework ML

Quando si usa:

- Applicazioni di produzione, prodotti SaaS, servizi web
- Team che preferiscono delegare la gestione tecnica

Limiti:

- Servizio a pagamento
- I dati risiedono su cloud esterno (privacy da valutare)

CONFRONTO: PROPERTY GRAPH VS RDF VS VECTOR DB

Aspetto	Property Graph	RDF	Vector DB
Struttura	Flessibile (nodi, archi, proprietà)	Standardizzata (triple sogg-pred-ogg)	Matrici/vettori numerici
Query	Cypher, Gremlin, GQL	SPARQL	Ricerca per similarità
Uso	Relazioni aziendali, workflow	Linked Data, semantica web	RAG, semantic search, AI
Vantaggio	Intuitivo, ottimo per reti complesse	Interoperabile, dati collegati	Scalabile per grandi dati
Esempio	Neo4j	Virtuoso, Apache Jena	FAISS, Pinecone, Qdrant

GRAZIE PER L'ATTENZIONE