Profice

AI ACADEMY

Applicare l’Intelligenza Artificiale nello sviluppo software

AI ACADEMY

Tool-calling & Agent Design
24/06/2025

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

- Comprendere cosa si intende per agente AI e come si differenzia da chatbot o LLM classici.
- Analizzare le principali architetture di agenti (ReAct, Plan-&-Execute, CrewAI).
- Imparare a progettare e integrare tool esterni tramite specifiche (JSON, schema) e function-calling.
- Applicare lo standard OpenAI function-calling per tool-calling automatico.
- Gestire memoria delle azioni e contesto conversazionale negli agenti.
- Implementare guardrail e controlli di sicurezza nell'uso dei tool.
- Progettare, testare e valutare agenti AI reali con notebook demo (LangChain, CrewAI).

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.

DOMANDE?

Discussione dei risultati

ESEMPIO DI NER

```python
from transformers import pipeline

ner_pipe = pipeline(
    "ner",
    model="Davlan/bert-base-multilingual-cased-ner-hrl",
    aggregation_strategy="simple"
)

text = "Mario Rossi ha ricevuto un bonifico sull’IBAN IT60X0542811101000000123456."
entities = ner_pipe(text)

# Stampa risultato: "Leonardo → PERSON", ecc.
entities = ner_pipe(text)
for ent in entities:
    print(f"{ent['word']} → {ent['entity_group']}")
```

ESEMPIO DI PIPELINE RAG

```
# Caricamento del testo da file
loader = TextLoader(«contratto.txt", encoding="utf-8")
docs = loader.load()

# Divisione del testo in chunk
splitter = RecursiveCharacterTextSplitter(chunk_size=500, chunk_overlap=50)
chunked_docs = splitter.split_documents(docs)

# Embedding & indicizzazione (FAISS locale)
embeddings = OpenAIEmbeddings()
vectorstore = FAISS.from_documents(chunked_docs, embeddings)
```

ESEMPIO DI PIPELINE RAG

```
# Definizione la pipeline di RAG
retriever = vectorstore.as_retriever()
llm = OpenAI(model="gpt-3.5-turbo")  # O il modello desiderato
rag_chain = RetrievalQA.from_chain_type(
    llm=llm,
    retriever=retriever,
    return_source_documents=True
)
```

```
# Esecuzione di una domanda sul documento (RAG)
query = "Quali sono la durata e la scadenza del contratto?"
result = rag_chain({"query": query})

print("Risposta:", result["result"])
for doc in result["source_documents"]:
    print("Fonte:", doc.page_content[:80], "...")
```

DEFINIZIONE "AGENTE"

Agente AI:

Un’entità (software) capace di prendere decisioni autonome, pianificare sequenze di azioni e usare strumenti esterni (tool), spesso ragionando passo-passo e adattandosi al contesto, con memoria delle azioni svolte.

Riferimenti

- Nei LLM: “chatbot” o “ChatGPT” di base **non è** un agente vero e proprio;
  un agente LLM richiede come minimo:

  - **memoria**
  - **pianificazione**
  - **tool-calling**

- Un root prompt molto lungo soltanto simula alcune capacità, ma non è sufficiente.

QUANDO UN AGENTE NON È SOLO UN CHATBOT

Un agente AI non si limita a generare testo:

- Decide autonomamente **quale tool usare**
- **Può pianificare** più azioni in sequenza
- **Mantiene memoria** delle interazioni e dello stato

La differenza chiave: **l'agente ragiona e agisce oltre il singolo prompt**

Esempio:

- Un semplice chatbot risponde e basta
- Un agente che deve organizzare una call
  - consulta un calendario per procurare i dati
  - prenota una sala
  - conferma l'invito e lo inoltra agli invitati

ARCHITETTURE AGENTICHE PRINCIPALI: PANORAMICA

1. **ReAct (Reason + Act):**
   - L'agente alterna momenti di ragionamento ("pensa") e azione ("agisci").
   - Può decidere se consultare un tool, chiedere chiarimenti, o rispondere direttamente.
   - Favorisce trasparenza e tracciabilità del processo decisionale.
2. **Plan-&-Execute:**
   - L'agente prima crea un piano completo (sequenza di task), poi lo esegue step by step.
   - Pianificazione e azione sono separati, permettendo maggiore controllo.
   - Utile per task complessi o multi-step (es. workflow aziendali).
3. **CrewAI:**
   - Struttura un agente come un team virtuale, con ruoli assegnati: planner, executor, validator, ecc.
   - Gli agenti lavorano in parallelo o in sequenza, ognuno focalizzato su uno specifico aspetto del compito.
   - Favorisce modularità, collaborazione e specializzazione.
4. **MoE (Mixture of Experts):**
   - L'agente seleziona dinamicamente il "modulo esperto" migliore per il task richiesto.
   - Gli "esperti" possono essere modelli LLM, funzioni, API o sottosistemi diversi.
   - Permette scalabilità e alta efficienza su task eterogenei.

ESEMPI DI ARCHITETTURE AGENTICHE REALI

1. **ReAct in pratica:**
   Un agente customer-care che, davanti a una richiesta, decide se rispondere direttamente, consultare una FAQ, oppure estrarre dati da un sistema CRM.
2. **Plan-&-Execute:**
   Un agente per prenotazioni aziendali che prima pianifica i passi (es. “Verifica disponibilità sale” → “Prenota sala” → “Invia conferma”) e poi li esegue nell’ordine giusto.
3. **CrewAI:**
   Un agente progettista che affida la scrittura di codice a uno specialista, la revisione a un altro, la validazione finale a un terzo.
4. **MoE:**
   Un agente che, per rispondere a domande legali, attiva un modulo specializzato in normativa; per domande tecniche, un modulo di engineering.

INTEGRAZIONE DI TOOL ESTERNI NEGLI AGENTI AI

Tool-calling:
Permette all'agente di utilizzare risorse esterne (API, funzioni, database, web search) durante l'esecuzione.

Specifica tool (JSON/schema):
Ogni tool viene descritto tramite uno schema strutturato (tipicamente JSON), che definisce:

- Nome e descrizione del tool
- Parametri richiesti
- Tipo di output atteso

Function-calling (OpenAI, LangChain):
Gli agenti possono invocare funzioni in modo automatico, scegliendo quale tool usare sulla base del contesto.

Vantaggi:

- Espande le capacità dell'agente
- Permette azioni reali (es. prenotare, calcolare, cercare)
- Favorisce modularità e manutenzione

ESEMPI PLANNER VS EXECUTOR (CREWAI)

Richiesta utente:
Riporta gli ultimi tre contratti che hanno lavori in corso.

Planner
Suddivide il compito:

- Cercare tutti i contratti con stato “lavori in corso”
- Ordinare per data di inizio lavori (o data aggiornamento)
- Selezionare i tre più recenti
- Preparare la risposta per l’utente

Executor
Esegue ogni step:

- **Ricerca contratti:** Interroga il database/document management per filtrare i contratti “lavori in corso”.
- **Ordinamento:** Ordina i risultati per data.
- **Selezione:** Estrae i tre contratti più recenti.
- **Output:** Compila una breve scheda informativa per ciascun contratto e la restituisce all’utente.

ESERCIZO CODICE PLANNER VS EXECUTOR (CREWAI)

```
from crewai import Agent, Task, Crew

contratti = []

# Definizione dell’agente Planner CrewAI
planner = Agent(
    name="PlannerContratti",
    role="Pianificatore di richieste documentali",
    goal="Identificare i tre contratti più recenti con lavori in corso",
    tools=[seleziona_contratti_in_corso_tool]
)
# Definizione dell’agente Executor CrewAI
executor = Agent(
    name="ExecutorContratti",
    role="Executor",
    goal="Creare una scheda riassuntiva dei contratti trovati dal planner",
    tools=[genera_scheda_contratti_tool]
)
```

ESERCIZO CODICE PLANNER VS EXECUTOR (CREWAI)

```
# Definizione dei task CrewAI
task_planner = Task(
    description="Trova e restituisci gli ultimi tre contratti con lavori in corso.",
    expected_output="Elenco dei contratti (cliente, nome_progetto, data_inizio)"
)

task_executor = Task(
    description="Ricevi la lista dal planner e genera una scheda leggibile per l'utente.",
    expected_output="Testo riassuntivo per i tre contratti"
)

# Composizione Crew
crew = Crew(
    agents=[planner, executor],
    tasks=[task_planner, task_executor]
)

# Esecuzione CrewAI
result = crew.run()
print(result)
```

ESEMPIO CONCRETO DI INTEGRAZIONE TOOL

Scenario:
Un agente riceve la richiesta: “Converti 100 dollari in euro e verifica il tasso di cambio attuale.”

Tool coinvolti:

- API di conversione valuta (con schema JSON: amount, currency_from, currency_to)
- Web search per recuperare il tasso aggiornato

Flusso:

1. L’agente riconosce che serve chiamare la funzione di conversione.
2. Compila i parametri nel formato richiesto dallo schema.
3. Riceve il risultato e lo include nella risposta.
4. Può effettuare più chiamate se servono dati accessori.

GESTIONE MEMORIA, CONTESTO E SICUREZZA DI AGENTI

Memoria conversazionale:

- L’agente registra le azioni svolte, le risposte date e i tool chiamati.
- Può richiamare informazioni dalle interazioni precedenti per mantenere coerenza e continuità.

Gestione del contesto:

- L’agente mantiene lo stato della conversazione e il contesto operativo (task in corso, parametri, risultati intermedi).
- Gestisce la finestra di token per non perdere informazioni rilevanti.

Guardrail e sicurezza:

- Si implementano controlli per limitare l’accesso o l’uso improprio dei tool.
- Verifiche su input/output per prevenire azioni indesiderate (es. injection, accesso a dati riservati).
- Logging delle chiamate e delle azioni per audit e debug.

SCRATCHPAD & CONTEXT TOKENS

Scratchpad

È uno spazio di lavoro temporaneo dove l'agente appunta idee, risultati intermedi, domande e risposte parziali mentre svolge un task.

- Consente di "pensare ad alta voce", annotare ipotesi o calcoli, e organizzare il ragionamento step-by-step.
- Molto utile per task complessi (es. problem solving multi-step, pianificazione) in cui serve "tenere traccia" del percorso logico.

Context tokens

Rappresentano la quantità di testo (input + output + memoria) che l'agente può gestire contemporaneamente,
detta anche context window.

- Una finestra troppo piccola porta a "dimenticare" parti della conversazione o del task.
- Una finestra più grande permette di mantenere più memoria e dettagli, ma ha limiti di performance e costo.
- Gestire i token in modo efficace è fondamentale: si selezionano solo le informazioni più rilevanti per il prompt corrente, usando memoria breve (scratchpad) e memoria lunga (cronologia, knowledge base).

ELEMENTI FONDAMENTALI DI LANGCHAIN

LangChain è un framework open-source Python (e ora anche JavaScript) per costruire applicazioni che usano Large Language Models (LLM) in modo strutturato, componibile e scalabile.

Document Loader

- Modulo per caricare dati da file, siti web, database, API.
- Supporta molti formati: PDF, DOCX, TXT, HTML, ecc.

Text Splitter (Chunking)

- Suddivide i documenti in “chunk” gestibili dal modello (ad esempio, paragrafi o blocchi da 500 token).
- Fondamentale per analisi e retrieval efficienti.

Embedding Model

- Trasforma testo o chunk in vettori numerici (“embedding”) per la ricerca semantica.
- Compatibile con modelli OpenAI, HuggingFace, MiniLM, ecc.

ELEMENTI FONDAMENTALI DI LANGCHAIN

Vector Store

- Database specializzato che memorizza e indicizza gli embedding.
- Esempi: FAISS, Pinecone, Qdrant.

Retriever

- Modulo che ricerca, tramite similarità, i chunk più rilevanti dati una query.
- Può combinare più strategie: similarity search, keyword, hybrid.

LLM Chain & Agent

- Catene di moduli che orchestrano chiamate LLM, gestione memoria, tool-calling, ragionamento.
- Gli “agent” permettono interazione multi-step, decisioni autonome, uso tool esterni.

Memory

- Memorizza lo storico delle interazioni e i dati temporanei (scratchpad).
- Migliora coerenza e continuità nelle conversazioni lunghe.

DOMANDE?

PAUSA

GRAZIE PER L'ATTENZIONE