# L’AI che parla la tua lingua

DevFest Pescara 2024
08/11/2024

DataMasters

# WHO I AM

GIUSEPPE **MASTRANDREA**
*Lead Teacher*

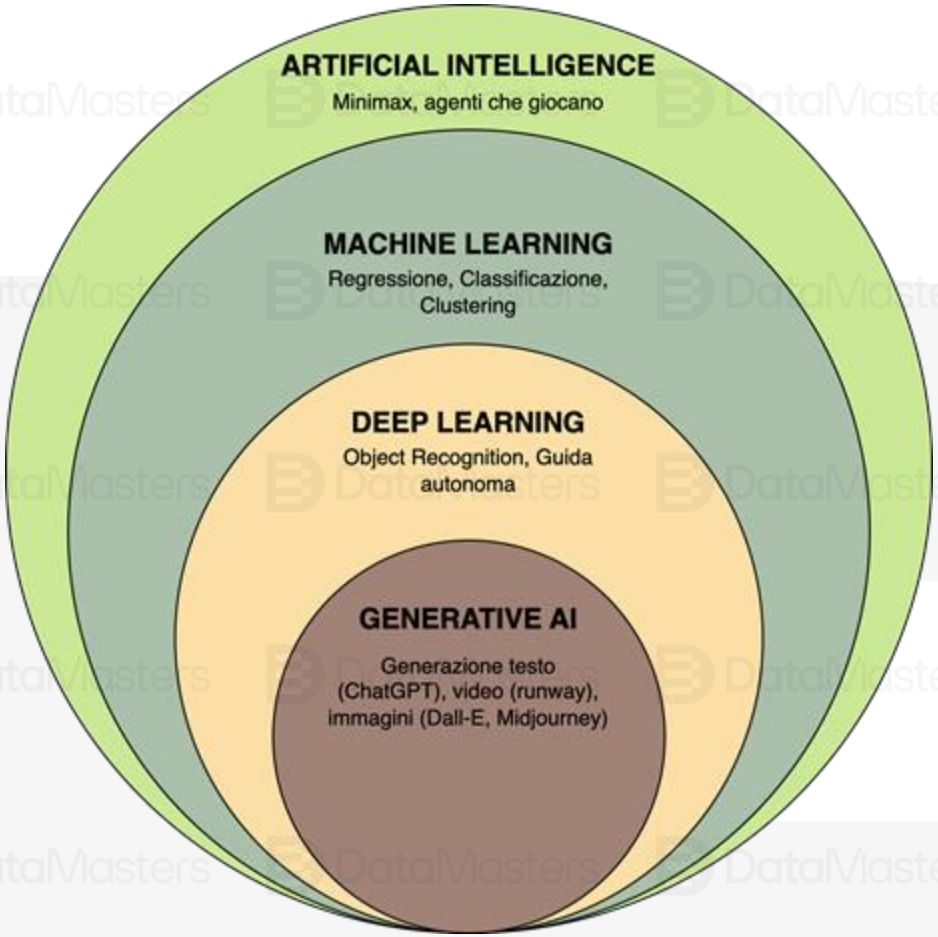
Software engineer

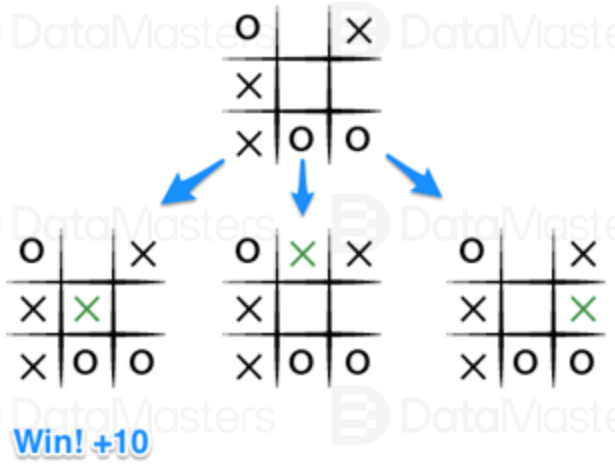
Front End Developer since 2011

Teacher since 2017

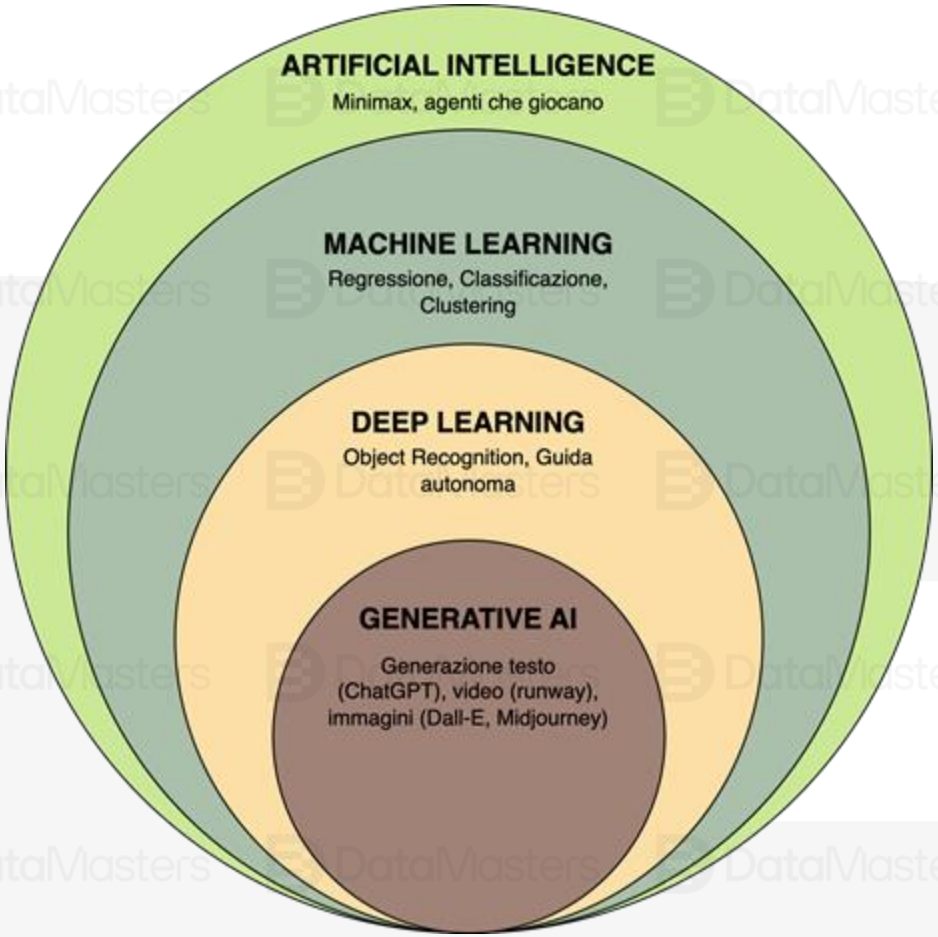
Lead Teacher in Data Masters since 2021

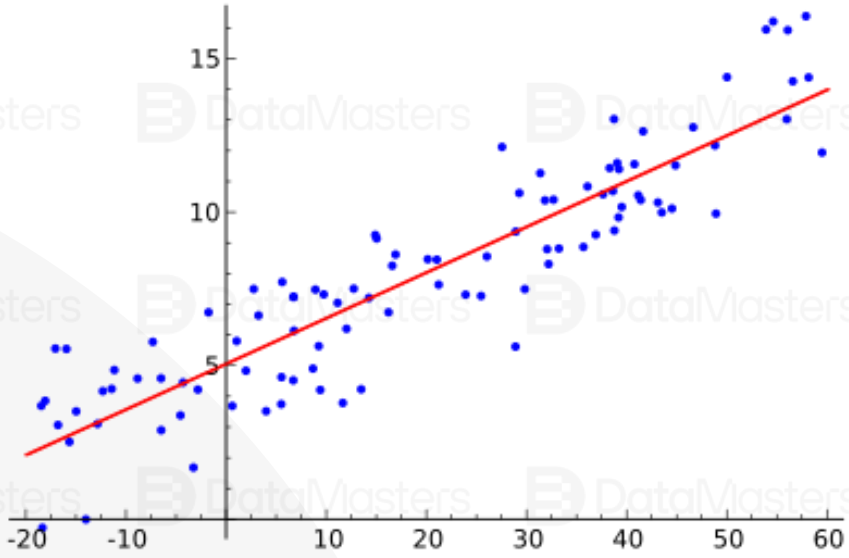
# Artificial Intelligence



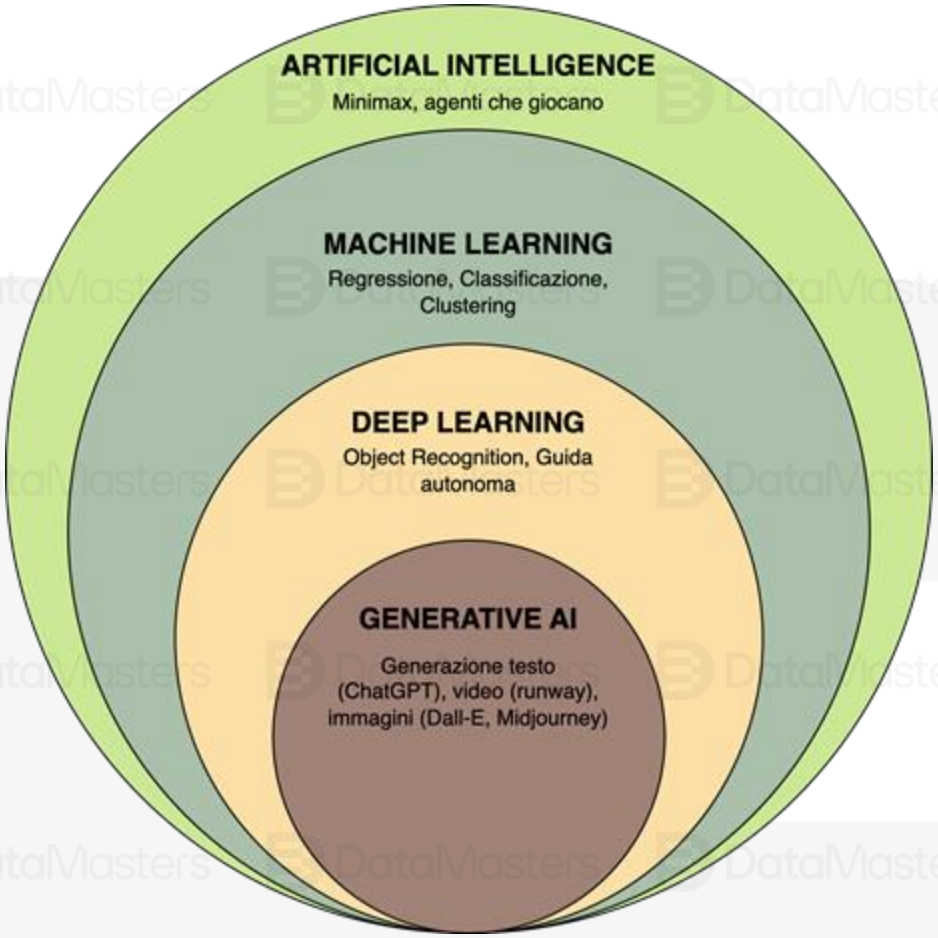



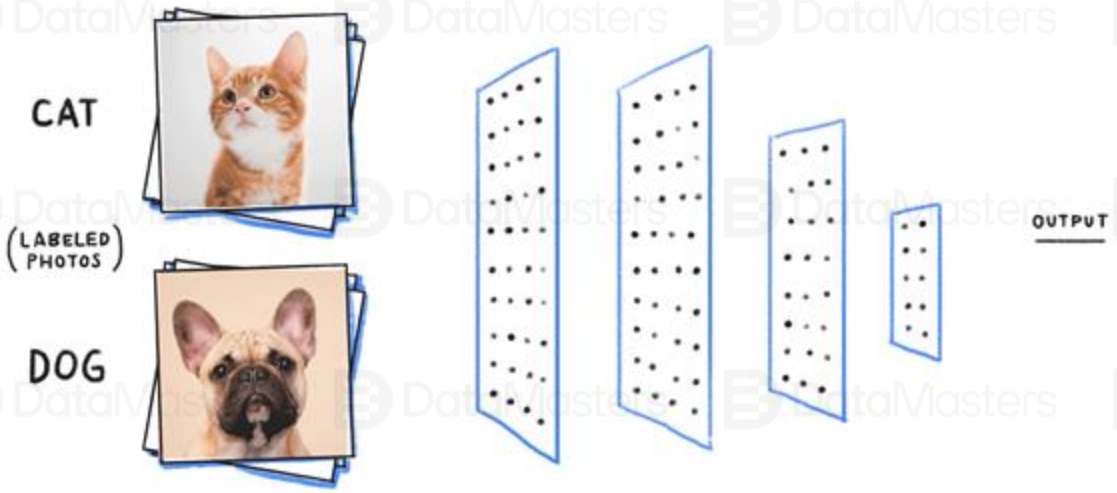
# Machine Learning



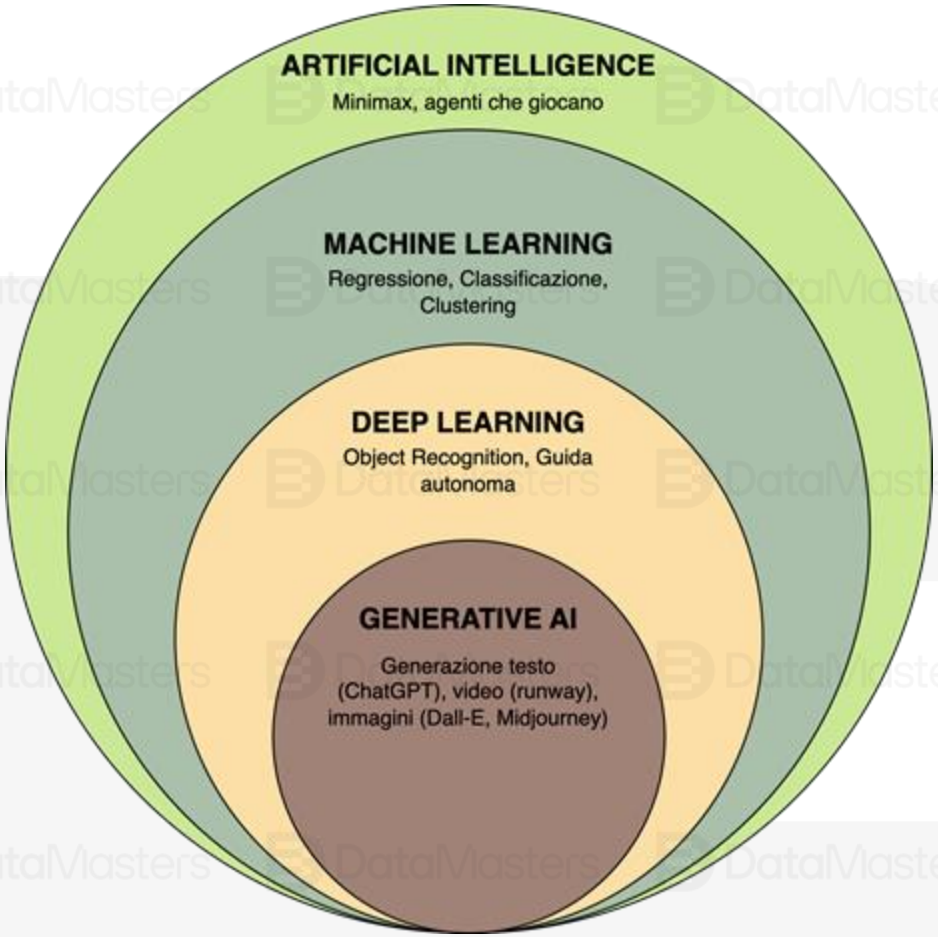



# Deep Learning



# Generative AI




Gemini

LLaMA by Meta

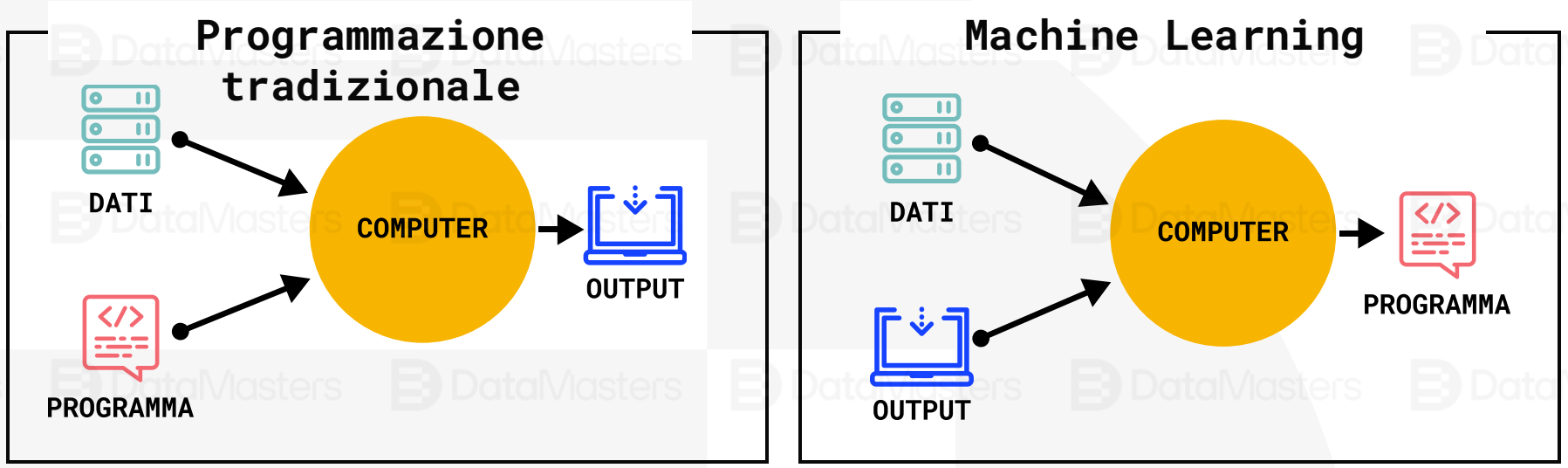
runway

# Modelli predittivi

**Usare dati** per **rispondere a domande**

addestramento

predizione / classificazione



# Apprendimento supervisionato

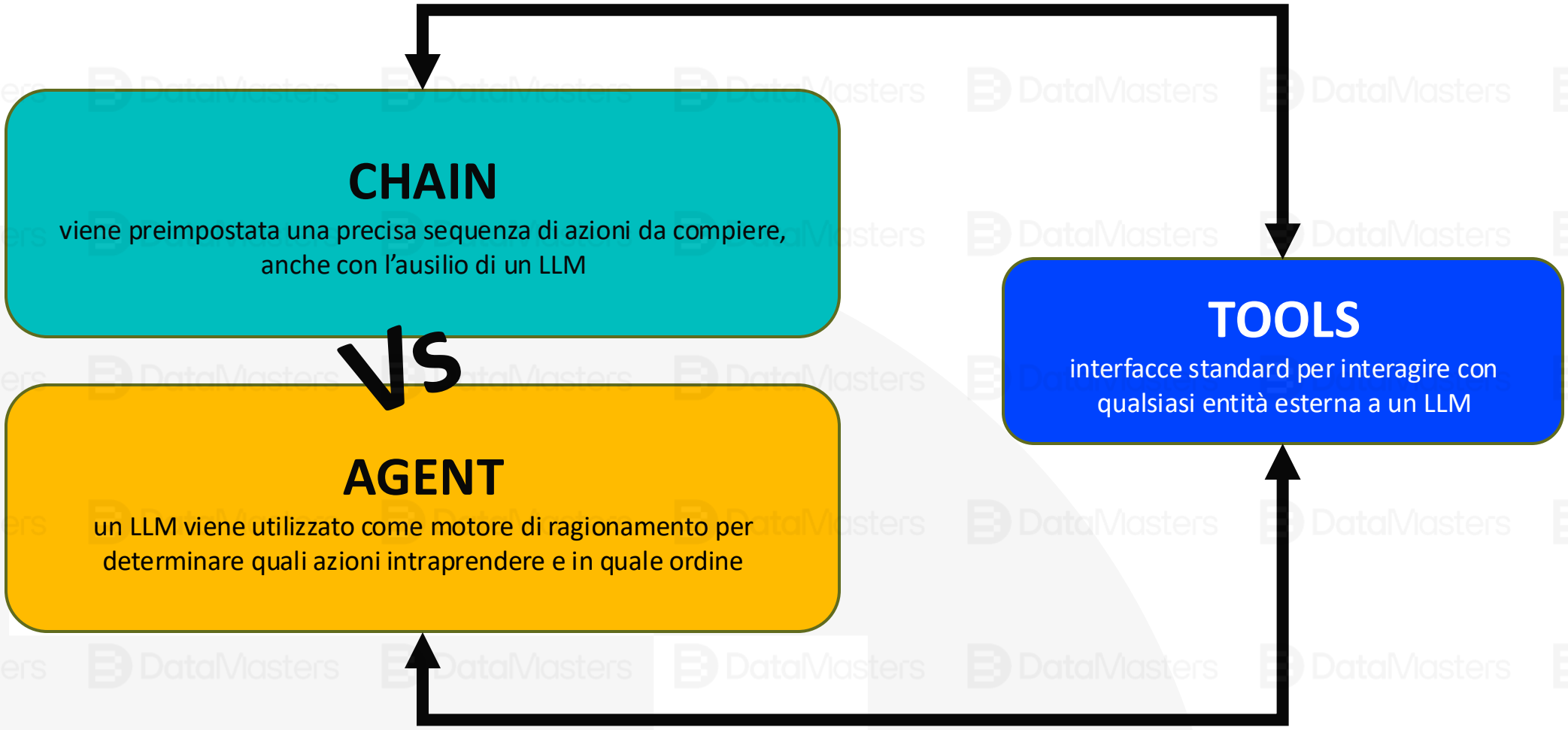
→ Viene fornito un set di dati e si sa come dovrebbe essere il nostro output corretto, supponendo che ci sia una relazione tra input e output.

# …e l’IA generativa?

→ Tutte le moderne applicazioni di IA generativa utilizzano in qualche modo i **Large Language Models** per:

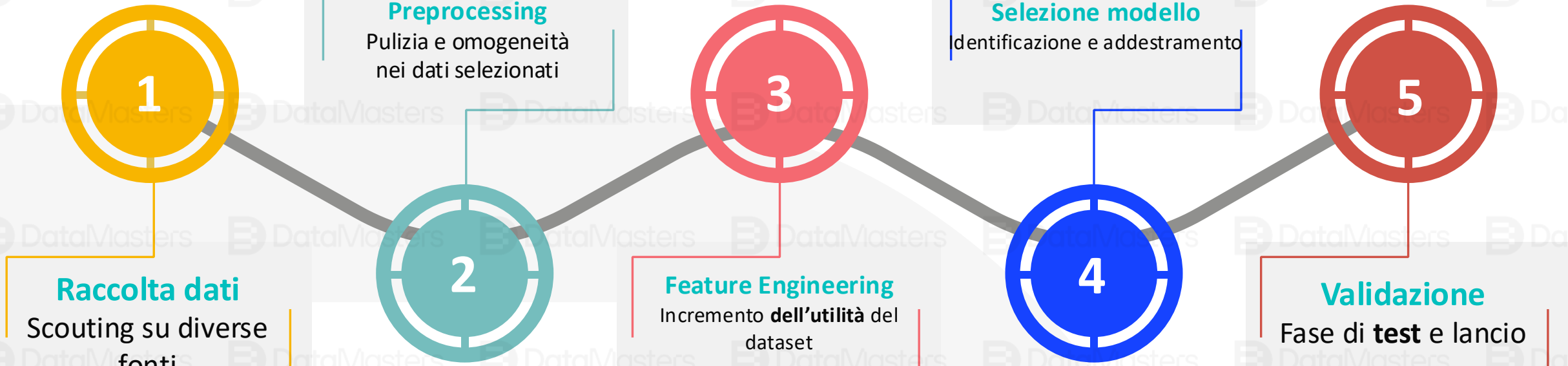
- Decomposizione dei Task
- Esecuzione di sotto-task
- Verifica e Validazione
- Interfacciamento e integrazione
- Interfaccia Utente Avanzata

# …e l’IA generativa?



DataMasters

# Processo di implementazione



# Il mondo oltre le reti neurali

➔ **Come ti convinco a non usare una rete neurale**

- Una rete neurale può richiedere **giorni** (se non **settimane**!) di train e tuning
- Ti piace avere **milioni** di parametri?
- Ti piacciono i **misteri**?
- Quanti strati nascosti diventano **troppi?**

# Il mondo oltre le reti neurali

→ **Decision Trees**

- Non servono neuroni per prendere una buona decisione

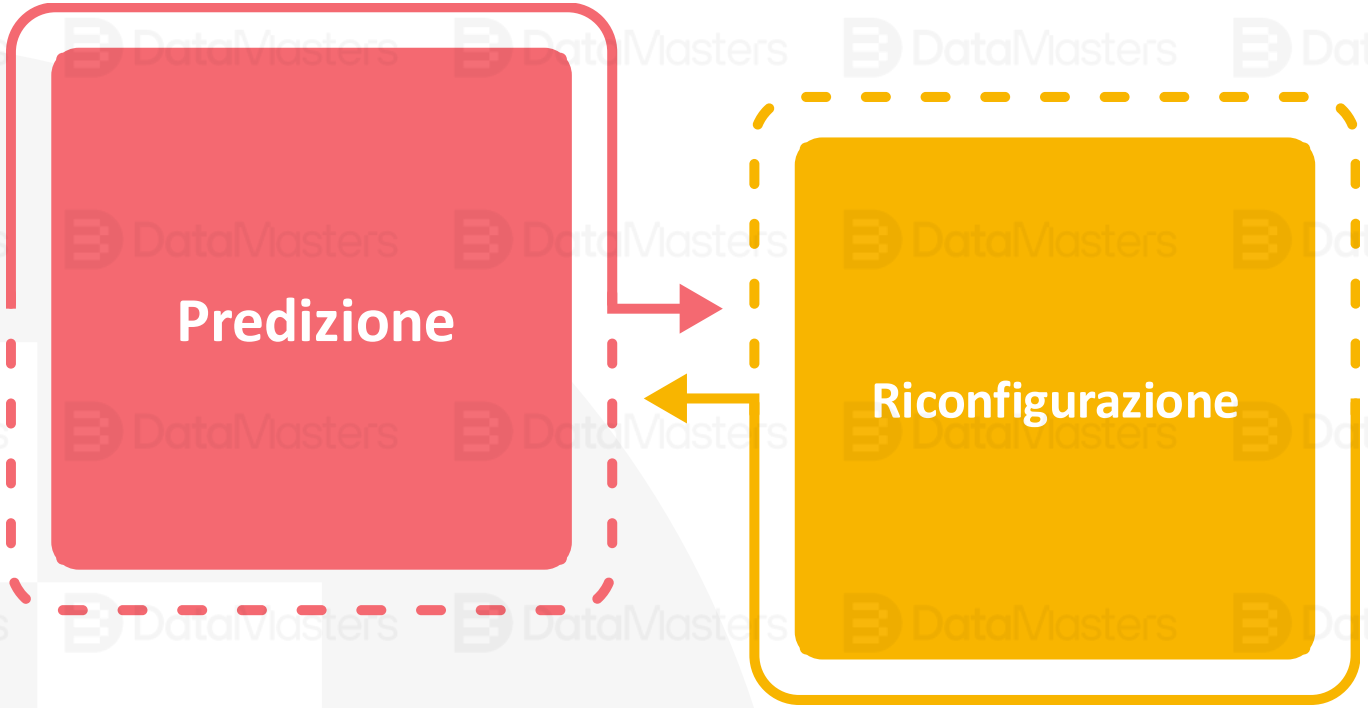
→ **KNN**

- Distanze fra oggetti complessi

→ **XGBoost/CatBoost**

# Train

- **Scopo:** rendere l'algoritmo selezionato capace di dare la risposta giusta sempre più spesso
  - Utilizzo di metriche per poter valutare quantitativamente le performance di diverse configurazioni



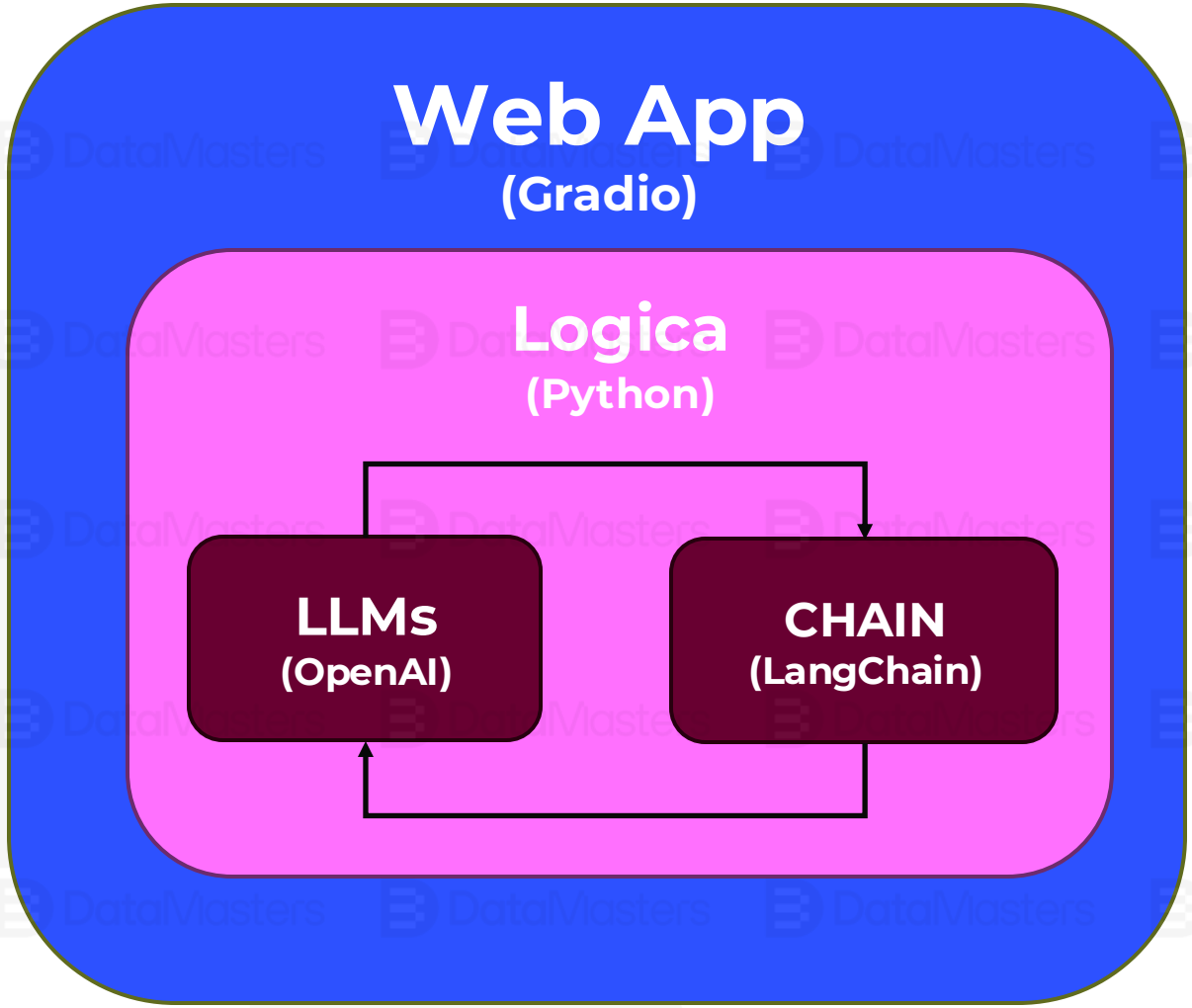
# Validazione e Test

→ In genere si suddivide il set di dati a disposizione in **dati di:**

- **Addestramento**
- **Validazione**
- **Test**

# LLM agentic applications

- In genere si suddivide il set di dati a disposizione in **dati di:**
  - **Addestramento**
  - **Validazione**
  - **Test**



# MLOps

- **MLOps** integra i principi DevOps con il machine learning per automatizzare l'intero ciclo di vita ML, dalla fase di sviluppo fino alla produzione
- I punti chiave includono
  - **Continuous Integration (CI)**
  - **Continuous Delivery (CD)**
  - **Continuous Training (CT)**
- Assicura **automazione** in ogni fase: test, deployment e monitoraggio

# Basta caXXate!

→ È tempo di scrivere un po’di codice

→ Clonate il repo: https://github.com/giuseppemastrandrea/train-prey-deploy

→ Setup di un virtual env

→ `pip install -r requirements.txt`

→ Oppure usa **Google Colab**

- Carica i notebook su Colab
- Metti tutto il materiale in una cartella sul tuo drive

```
from google.colab import drive
drive.mount('/content/drive')
import os
os.chdir("drive/My Drive/<your-folder>")
```

# Creiamo un modello

→ Step 1:
- Carichiamo il dataset

→ Step 2:
- Preprocessing

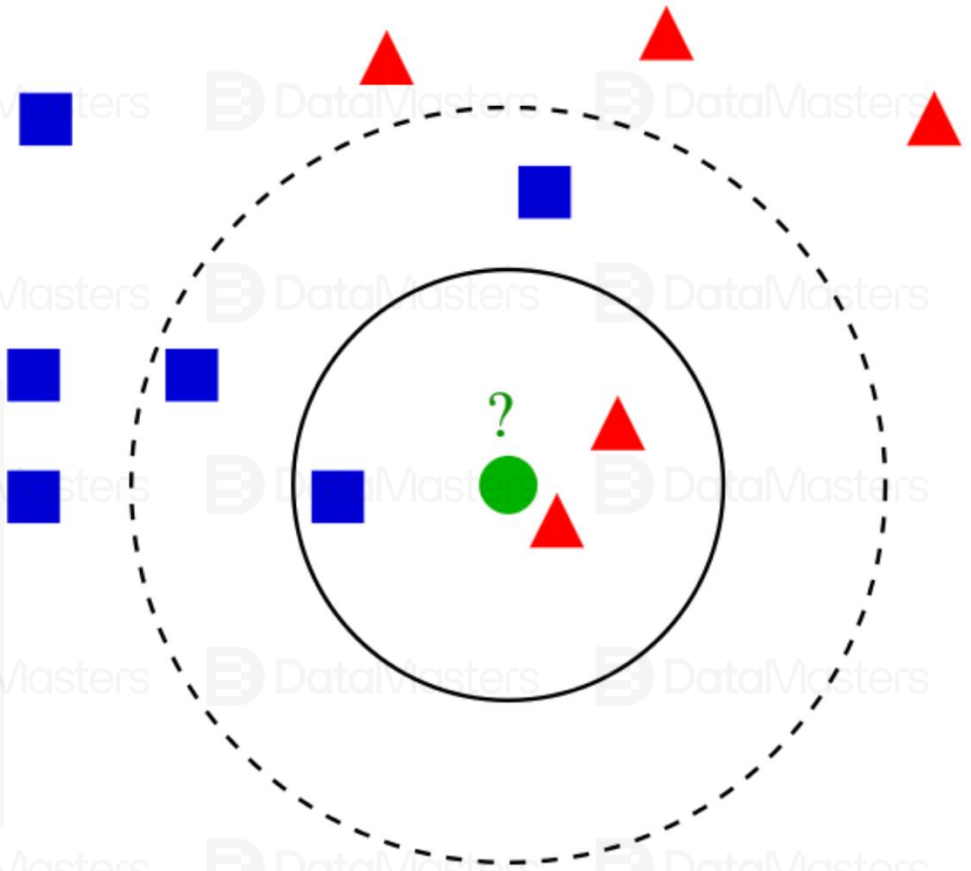
→ Step 3:
- Feature engineering

→ Step 4:
- Train

→ Step 5:
- Test (and pray!)

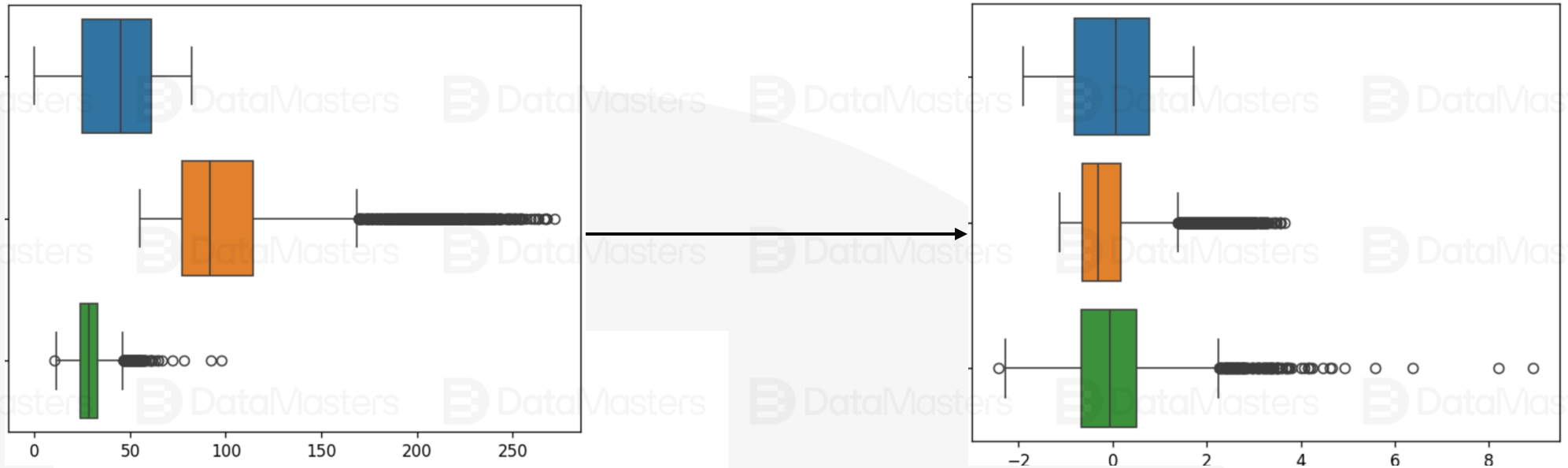
# Preprocessing

- Imputing dei valori mancanti
- KNNImputer
- SimpleImputer



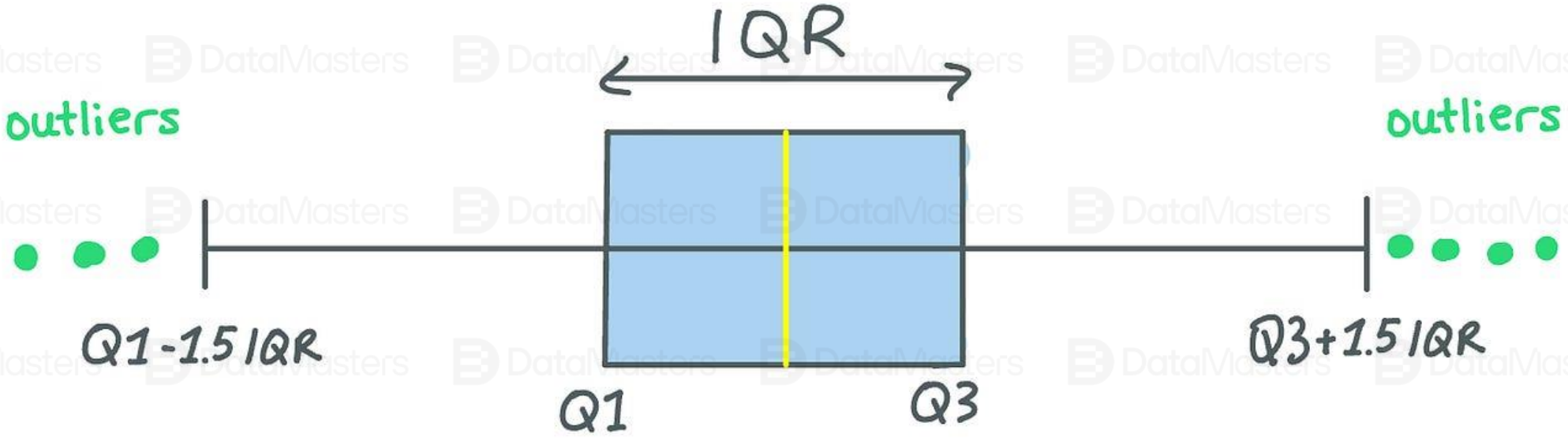
# Preprocessing

→ Standardization vs. Normalization



# Preprocessing

→ Outlier Removal with IQR



# Feature Engineering

→ One Hot Encoding

| id | color |
|---|---|
| 1 | red |
| 2 | blue |
| 3 | green |
| 4 | blue |


| id | color_red | color_blue | color_green |
|---|---|---|---|
| 1 | 1 | 0 | 0 |
| 2 | 0 | 1 | 0 |
| 3 | 0 | 0 | 1 |
| 4 | 0 | 1 | 0 |

# Build and train a model!

→ Neural Network with Keras?

→ Naive Bayes Classifier?

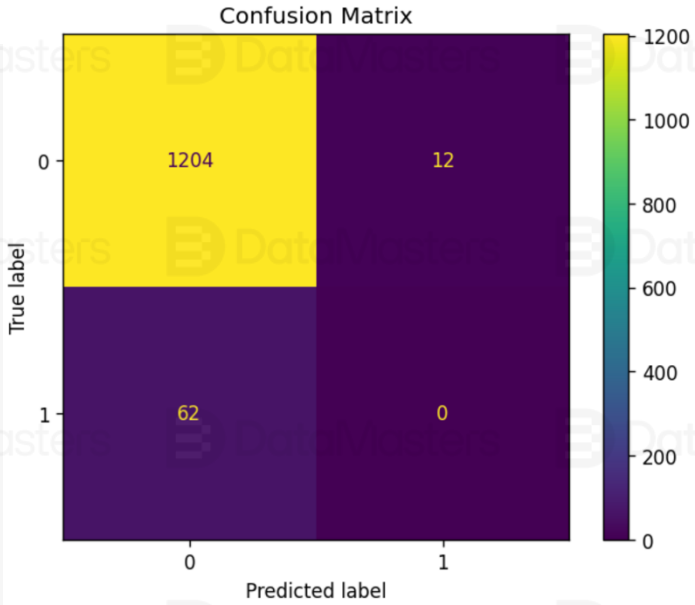
→ KNN Classifier?

## it’s up to you!

# Test and prey

→ Why should you pray?

- Because if test phase it's ok you're ready to **production!**





COFFEE BREAK SIGNORI!
gifs.com

# What’s next?

→ Supponendo che il nostro modello sia perfettamente funzionante…

→ …possiamo inserirlo in un’app powered by LLMs?

**ma certo!**

# What’s next?

→ LangChain: un framework per creare applicazioni powered by LLM

→ LangGraph: un framework per **orchestrare** applicazioni o modelli o chain

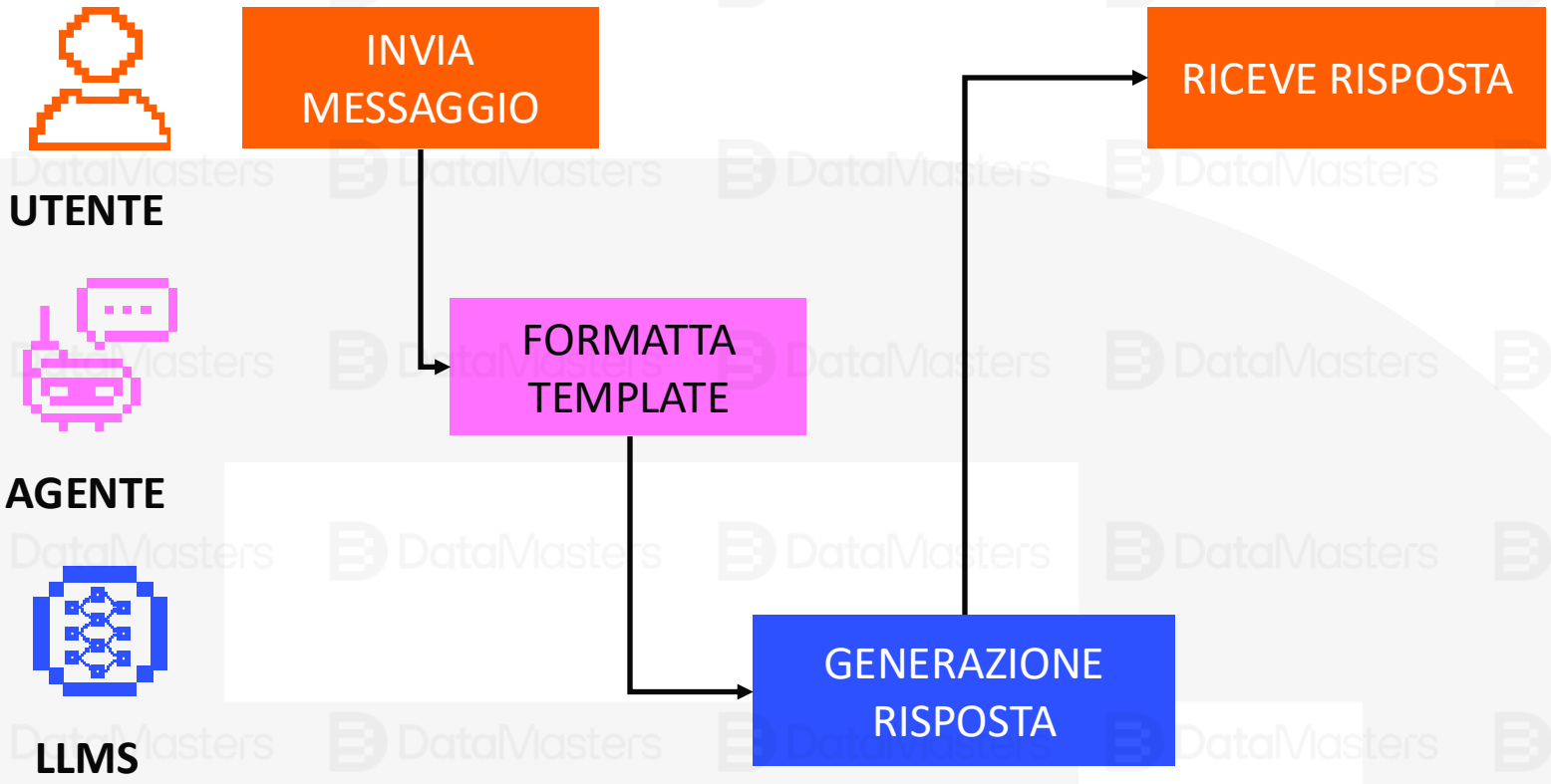
# LangChain

→ Framework per sviluppare applicazioni che sfruttano LLM

→ Componenti principali:

- # Model I/O
- # Chains
- # Memory
- # Retrieval (RAG)
- # Agents
- # Callbacks

# Demo Chatbot: funzionamento



# LangGraph - Pro

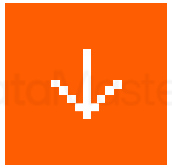
DataMasters LangGraph

**Workflow Ciclici e con Stato**
miglioramento delle capacità di ragionamento

**Flow Engineering**
Facilita un "flusso" iterativo in cui gli LLM possono influenzare le azioni successive

**Multi-Agent**
Facilita lo sviluppo di sistemi multi-agente

# LangGraph - Contro



**Complessità**

Può avere una curva di apprendimento più rapida

**Overhead**

Necessita di complessità inutili nei casi d’uso più semplici

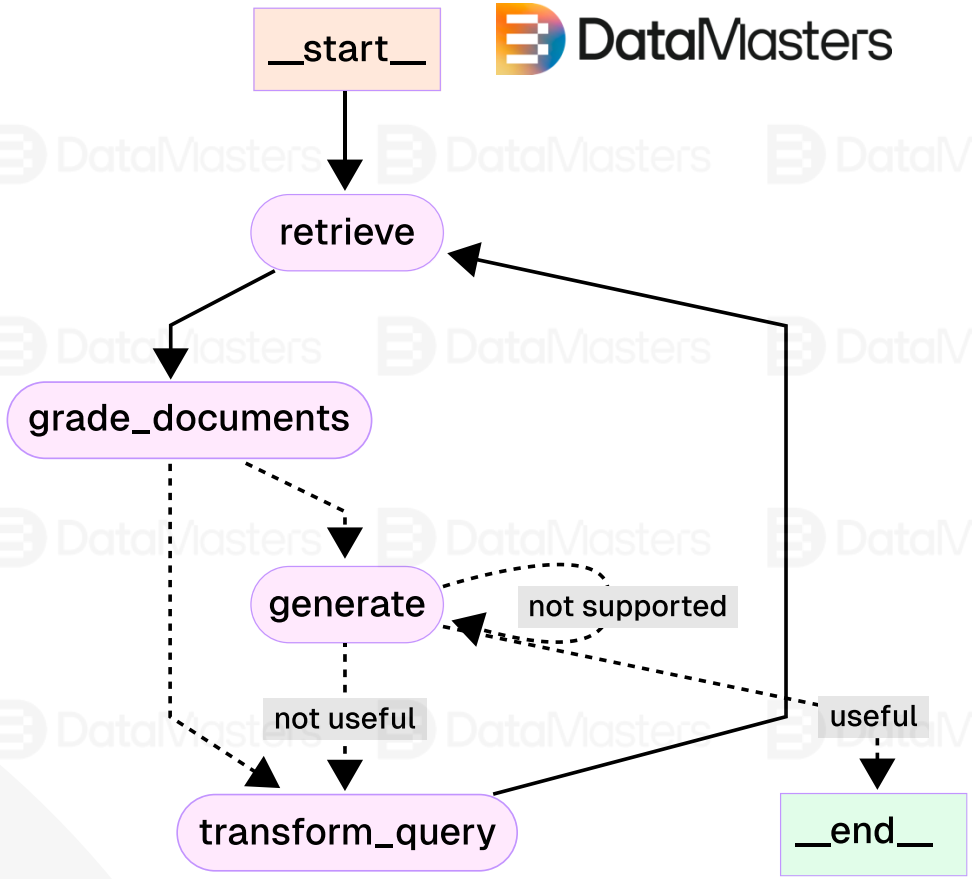
**Documentazione ed esempi limitati**

Supporto ridotto rispetto a LangChain o altri framework

# Il Grafo di LangGraph

Il processo operativo è un **Grafo**, costruito da:

- **STATO**
  struttura dati condivisa tra i nodi
  *(tipicamente un dizionario Python o un oggetto Pydantic)*
- **NODI**
  funzioni Python – eseguono un particolare task
- **ARCHI** (tra nodi)
  funzioni Python – dicono cosa fare dopo



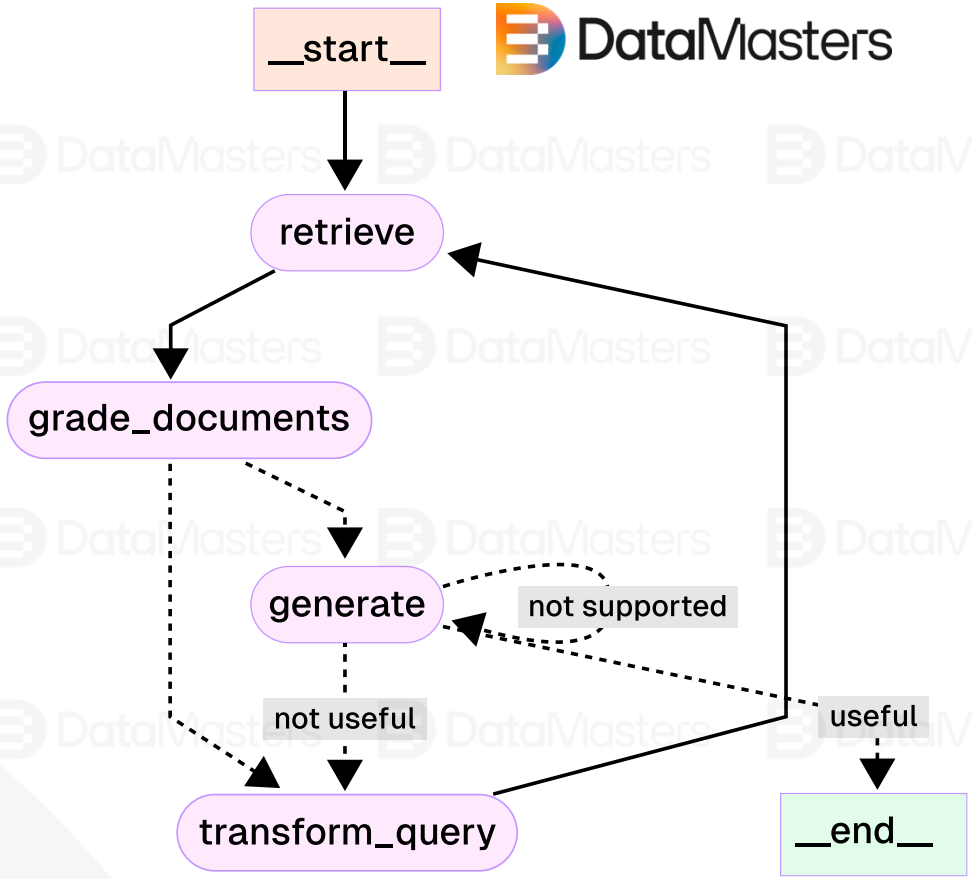

Componendo queste entità è **possibile creare flussi di lavoro complessi** che coinvolgono i cicli e le condizioni utili a gestire uno stato nel tempo.

# Il Grafo di LangGraph

Il processo operativo è un **Grafo**, costruito da:

→ **Ogni NODO**

- Riceve uno Stato in input
- Elabora lo Stato
- Ritorna lo Stato modificato



DataMasters

# …any questions?

DataMasters

# Thanks for your attention!

g.mastrandrea@datamasters.it