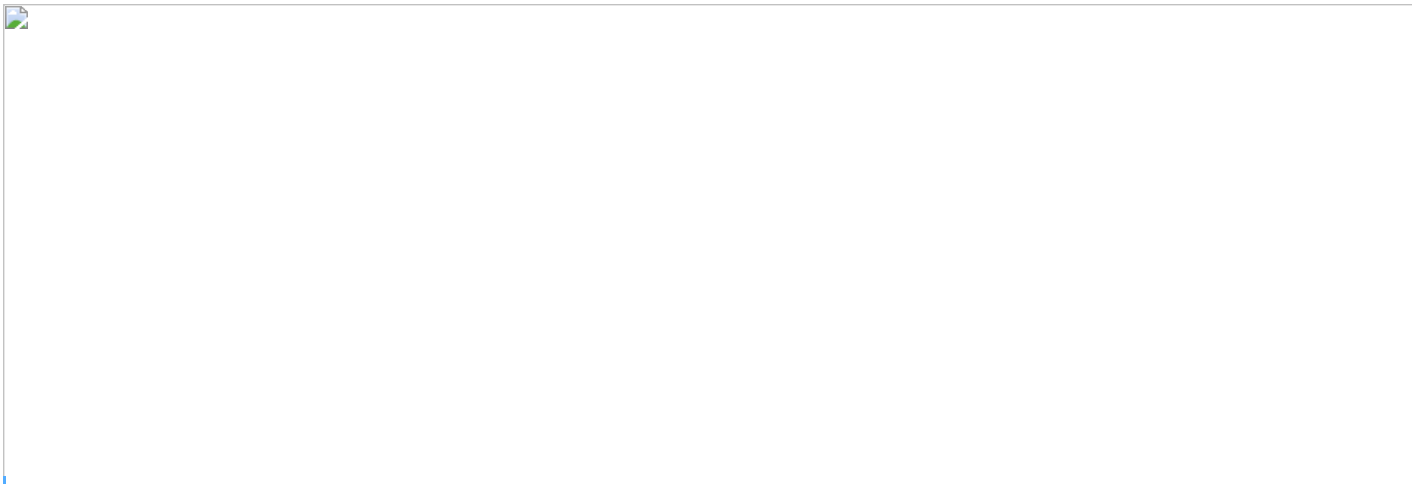
Gli Agenti di Intelligenza Artificiale

DI MASSIMO VAILATI

27/01/2026

INTELLIGENZA ARTIFICIALE AGENTI IA WORKFLOW AUTOMATION OPENAI AI AGENT

LICEO, SCIENTIFICO S.A., TT INFORMATICA



Contenuto: introduzione al concetto di agente di Intelligenza Artificiale (IA), definito come un sistema in grado di percepire, decidere, agire
Attività pratica: sperimentare la logica base di un agente reattivo, creare un agente complesso che analizza la richiesta di un utente e la ridir

Negli ultimi anni, l'Intelligenza Artificiale (IA) è uscita dai laboratori di ricerca per entrare nella nostra vita quotidiana: dagli assistenti vocali co
Ma cosa rende possibile tutto questo? Una delle risposte chiave è il concetto di **agente intelligente**.

Cos'è un agente IA

Un agente di intelligenza artificiale è un sistema in grado di:

- **Percepire** l'ambiente in cui si trova (attraverso sensori, dati o input);
- **Decidere** cosa fare, elaborando le informazioni ricevute;
- **Agire** sull'ambiente per raggiungere un obiettivo tramite attuatori o output digitali.
- **Valutare** i risultati eventualmente adattando le sue decisioni future (se dotato di apprendimento)

In altre parole, un agente è un "attore" che osserva, ragiona e agisce in modo autonomo o semi-autonomo.

Struttura di un agente

Un agente IA è spesso descritto con il modello **PEAS**:

- **P**erformance measure → come si misura il successo dell'agente
- **E**nvironment → l'ambiente in cui opera
- **A**ctuators → i mezzi con cui agisce
- **S**ensors → i mezzi con cui percepisce

Esempio:



Tipi di agenti

Ci sono diversi tipi di agenti, a seconda della loro complessità:

1. **Agenti reattivi semplici** → reagiscono solo agli stimoli (es. "se vedo ostacolo → gira a sinistra").

2. **Agenti basati su modello** → costruiscono una rappresentazione interna del mondo.

3. **Agenti con obiettivi** → pianificano azioni per raggiungere un risultato.

4. **Agenti con apprendimento** → migliorano le proprie decisioni grazie all'esperienza (machine learning).

Esempi nel mondo reale

- **Spotify** → suggerisce musica in base ai gusti dell'utente.
- **Google Maps** → calcola percorsi ottimali, adattandosi al traffico.
- **ChatGPT** → un agente linguistico capace di comprendere e generare testo in modo coerente e informato.

Attività pratica n. 1

Costruiamo un mini-agente reattivo

Far sperimentare agli studenti la logica base di un agente reattivo, anche senza conoscenze avanzate di programmazione.

Attività: "Agente aspirapolvere virtuale"

L'agente si muove in una stanza (griglia) e pulisce le celle sporche. Scriviamo un programma in Python (aspirapolvere.py).

Passaggi:

1. Creiamo l'ambiente

```
import random

# Ambiente: una griglia 5x5
stanza = [[random.choice(['pulito', 'sporco']) for _ in range(5)] for _ in range(5)]
```

2. Definiamo la posizione iniziale dell'agente

```
posizione = [2, 2]  # centro della griglia
```

3. Programmiamo l'agente reattivo

```
def agente(stanza, posizione):
    x, y = posizione
    if stanza[x][y] == 'sporco':
        print("Pulisco", (x,y))
        stanza[x][y] = 'pulito'
    else:
        print("Mi sposto verso", (x,y))
        movimento = random.choice(['su', 'giu', 'sinistra', 'destra'])
        if movimento == 'su' and x > 0: x -= 1
        elif movimento == 'giu' and x < 4: x += 1
        elif movimento == 'sinistra' and y > 0: y -= 1
        elif movimento == 'destra' and y < 4: y += 1
    return [x, y]
```

4. Facciamo agire l'agente più volte

```
for _ in range(10):
    posizione = agente(stanza, posizione)
```

Esempio di esecuzione

```
Stato della stanza:
['pulito', 'sporco', 'pulito', 'sporco', 'sporco']
['sporco', 'pulito', 'pulito', 'sporco', 'sporco']
['pulito', 'sporco', 'pulito', 'sporco', 'pulito']
['pulito', 'sporco', 'pulito', 'pulito', 'pulito']
['pulito', 'pulito', 'sporco', 'pulito', 'sporco']
Mi sposto verso (2, 2)
Pulisco (2, 3)
Mi sposto verso (2, 3)
Pulisco (1, 3)
Mi sposto verso (1, 3)
Pulisco (0, 3)
Mi sposto verso (0, 3)
Mi sposto verso (0, 3)
Mi sposto verso (0, 2)
Mi sposto verso (0, 3)
Stato della stanza:
['pulito', 'sporco', 'pulito', 'pulito', 'sporco']
['sporco', 'pulito', 'pulito', 'pulito', 'sporco']
['pulito', 'sporco', 'pulito', 'pulito', 'pulito']
```

```
['pulito', 'sporco', 'pulito', 'pulito', 'pulito']
['pulito', 'pulito', 'sporco', 'pulito', 'sporco']
Posizione attuale: [1, 3]
```

Discussione finale:

- L'agente prende decisioni autonome?
- Che tipo di agente è (reattivo, basato su modello, con apprendimento)?
- Come potremmo migliorarlo? (es. ricordare dove ha già pulito)

Ecco una versione minima che **migliora il comportamento** ricordando (in un `set`) le celle già pulite, così da evitare di tornarci se possibile (*asp*
Quando la cella attuale è pulita invece di muoversi a caso, l'agente prova a muoversi verso una cella **non ancora visitata** tra i vicini; solo se non

```
import random

# Ambiente: una griglia 5x5
stanza = [[random.choice(['pulito', 'sporco']) for _ in range(5)] for _ in range(5)]

print("Stato iniziale:")
for riga in stanza:
    print(riga)

posizione = (2, 2)      # centro della griglia
visitati = set()        # memorizza le posizioni già pulite o almeno visitate

def vicini(x,y):
    """ritorna celle adiacenti nei limiti"""
    return [(nx,ny) for nx,ny in [(x-1,y),(x+1,y),(x,y-1),(x,y+1)]
            if 0 <= nx < 5 and 0 <= ny < 5]

def agente(stanza, pos):
    x,y = pos

    # marca come visitata
    visitati.add((x,y))

    # se sporca → pulisci
    if stanza[x][y] == 'sporco':
        stanza[x][y] = 'pulito'
        print("Pulisco", (x,y))
        return (x,y)

    # altrimenti cerca tra i vicini un posto non visitato
    candidati = [p for p in vicini(x,y) if p not in visitati]
    if candidati:
        nx,ny = random.choice(candidati)
        print("Mi sposto verso non-visitata", (nx,ny))
        return (nx,ny)

    # fallback: tutto visitato intorno → movimento casuale valido
    nx,ny = random.choice(vicini(x,y))
    print("Mi sposto random", (nx,ny))
    return (nx,ny)

for _ in range(25):
    posizione = agente(stanza, posizione)
```

```
print("\nStato finale:")
for riga in stanza:
    print(riga)
print("Posizione finale:", posizione)
print("Celle visitate:", len(visitati))
```

Esempio di esecuzione

```
Stato iniziale:
['pulito', 'sporco', 'sporco', 'sporco', 'pulito']
['pulito', 'sporco', 'sporco', 'sporco', 'pulito']
['pulito', 'sporco', 'sporco', 'sporco', 'sporco']
['pulito', 'sporco', 'sporco', 'pulito', 'pulito']
['pulito', 'pulito', 'pulito', 'sporco', 'pulito']
Pulisco (2, 2)
Mi sposto verso non-visitata (2, 3)
Pulisco (2, 3)
Mi sposto verso non-visitata (1, 3)
Pulisco (1, 3)
Mi sposto verso non-visitata (0, 3)
Pulisco (0, 3)
Mi sposto verso non-visitata (0, 4)
Mi sposto verso non-visitata (1, 4)
Mi sposto verso non-visitata (2, 4)
Pulisco (2, 4)
Mi sposto verso non-visitata (3, 4)
Mi sposto verso non-visitata (3, 3)
Mi sposto verso non-visitata (3, 2)
Pulisco (3, 2)
Mi sposto verso non-visitata (4, 2)
Mi sposto verso non-visitata (4, 3)
Pulisco (4, 3)
Mi sposto verso non-visitata (4, 4)
Mi sposto random (4, 3)
Mi sposto random (4, 2)
Mi sposto verso non-visitata (4, 1)
Mi sposto verso non-visitata (4, 0)
Mi sposto verso non-visitata (3, 0)
Mi sposto verso non-visitata (2, 0)

Stato finale:
['pulito', 'sporco', 'sporco', 'pulito', 'pulito']
['pulito', 'sporco', 'sporco', 'pulito', 'pulito']
['pulito', 'sporco', 'pulito', 'pulito', 'pulito']
['pulito', 'sporco', 'pulito', 'pulito', 'pulito']
['pulito', 'pulito', 'pulito', 'pulito', 'pulito']
Posizione finale: (2, 0)
Celle visitate: 16
```

Attività pratica n. 2

Costruiamo un semplice agente triage con OpenAI

Creiamo un **agente triage** che riceve la richiesta dell'utente, la analizza e poi la ridireziona a uno dei tre agenti specializzati: **assistenza tecnica**

Di seguito il codice completo in **Python** (*triage_agent.py*), usando l'API di OpenAI. L'idea è:

1. Il **triage agent** analizza il messaggio dell'utente.
2. Determina a quale agente specializzato inviare la richiesta.

3. Chiama il rispettivo agente (che può essere una funzione o un modello con prompt personalizzato).

```
import openai

# Inserisci la tua chiave API
openai.api_key = "YOUR_OPENAI_API_KEY"

# Funzione per chiamare un agente specifico
def call_agent(agent_name, user_message):
    if agent_name == "assistenza_tecnica":
        prompt = f"Sei un esperto di assistenza tecnica. Rispondi alla richiesta dell'utente:\n{user_message}"
    elif agent_name == "assistenza_vendite":
        prompt = f"Sei un esperto di vendite. Rispondi alla richiesta dell'utente:\n{user_message}"
    elif agent_name == "gestione_ordini":
        prompt = f"Sei un esperto nella gestione degli ordini. Rispondi alla richiesta dell'utente:\n{user_message}"
    else:
        return "Agente non trovato."

    response = openai.ChatCompletion.create(
        model="gpt-5-mini",  # Puoi sostituire con il modello che preferisci
        messages=[{"role": "user", "content": prompt}],
        temperature=1
    )
    return response.choices[0].message['content']

# Funzione triage
def triage_agent(user_message):
    # Prompt per determinare il tipo di richiesta
    triage_prompt = f"""
    Sei un agente triage. L'utente ha scritto: "{user_message}".
    Decidi se la richiesta riguarda:
    1. Assistenza tecnica
    2. Assistenza vendite
    3. Gestione ordini

    Rispondi solo con una delle seguenti etichette:
    "assistenza_tecnica", "assistenza_vendite", "gestione_ordini"
    """
    response = openai.ChatCompletion.create(
        model="gpt-5-mini",
        messages=[{"role": "user", "content": triage_prompt}],
        temperature=1
    )
    agent_label = response.choices[0].message['content'].strip()
    return agent_label

# Funzione principale
def main():
    user_message = input("Scrivi la tua richiesta: ")
    agent_label = triage_agent(user_message)
    print(f"Triage: la richiesta verrà inviata a {agent_label.replace('_', ' ')}")
    answer = call_agent(agent_label, user_message)
    print(f"Risposta dell'agente:\n{answer}")

if __name__ == "__main__":
    main()
```

Per poter provare il programma è necessario installare la libreria openai per Python

```
pip install openai
```

Quindi creare una API key OpenAI accedendo con un account (o creandone uno gratuito) su https://platform.openai.com/

Per ottenere una nuova API_KEY cliccare su “Create new secret key”, dare un nome (es. “triage-agent”), quindi copiare subito la chiave che app



È necessario anche avere del credito per l’utilizzo delle API di OpenAI (token). Andare su https://platform.openai.com/settings/organization/bi

Verificare che ci sia almeno una di queste:

- carta di credito attiva
- credito gratuito disponibile
- piano a pagamento attivo

Se non si ha credito, aggiungere: Da Billing → “Add payment method” → aggiungi carta. E’ sufficiente anche un credito di pochi euro.



Primo esempio di esecuzione: Assistenza tecnica (stampante)

Scrivi la tua richiesta: "Non mi funziona più la stampante che ho comprato un mese fa".

Triage: la richiesta verrà inviata a assistenza tecnica

Risposta dell'agente:
Posso aiutarti volentieri. Prima però ho bisogno di qualche informazione per darti indicazioni precise:

Domande rapide:

- Marca e modello della stampante?
- Sistema operativo del computer (Windows 10/11, macOS, Linux)?
- Come è collegata la stampante al computer? (USB, Wi-Fi, Ethernet)
- Cosa succede esattamente quando provi a stampare? (nessuna reazione, errore sul PC, luci lampeggianti, stampa vuota, carta inceppata, r
- Hai già provato a spegnere e riaccendere la stampante e il computer?
- Hai aggiornato driver/firmware o cambiato rete/PC prima che smettesse di funzionare?

Nel frattempo prova questi controlli rapidi (seguendo l'ordine): ...

Commenti e valutazioni

Richiesta utente:

"Non mi funziona più la stampante che ho comprato un mese fa."

Comportamento dell'agente:

- Il **triage** funziona correttamente → la richiesta è classificata come assistenza tecnica.
- La **risposta** dell'agente è molto strutturata: parte con un approccio empatico e chiede informazioni di base, poi guida passo-passo nella dia
- L'agente adotta una **metodologia procedurale**: controlli preliminari, prove progressive, verifica hardware/software, garanzia.
- Usa un tono **professionale e collaborativo**, adatto al supporto clienti.

Punti di forza

- Rilevante e coerente con la categoria scelta.
- Linguaggio chiaro, ordinato, dettagliato.
- Dimostra comprensione del dominio tecnico.

Possibili miglioramenti

- Potrebbe usare il tool`create_ticket()` per aprire una segnalazione automatica dopo aver raccolto i dati.

Secondo esempio di esecuzione: Gestione ordini (ritardo spedizione)

Scrivi la tua richiesta: "Non ho ancora ricevuto il libro che ho ordinato la settimana scorsa e che mi avete detto sarebbe stato spedito entro 3 g

Triage: la richiesta verrà inviata a **gestione ordini**

Risposta dell'agente:
Mi scuso per il disagio - capisco la frustrazione dato che vi era stata promessa la spedizione entro 3 giorni e non avete ancora ricevuto il libro.

- Numero d'ordine (se lo avete)
- Nome e cognome usati per l'ordine
- E-mail o numero di telefono collegati all'ordine
- Indirizzo di spedizione completo
- Titolo del libro e data approssimativa dell'ordine

Nel frattempo, vi chiedo di verificare rapidamente: .....

Commenti e valutazioni

Richiesta utente:

"Non ho ancora ricevuto il libro che ho ordinato la settimana scorsa..."

Comportamento dell'agente:

- Il **triage** smista correttamente a *gestione ordini*.
- L'agente risponde in modo empatico e organizzato: riconosce il disagio, raccoglie dati necessari (ordine, nome, email, indirizzo), suggerisc
- Simula un workflow reale di customer care (controllo, tracking, opzioni di rimborso o rispedizione).

Punti di forza

- Risposta ben contestualizzata.

- Mostra empatia e competenza logistica.
- Spiega con trasparenza i prossimi passi → ispira fiducia.

Possibili miglioramenti

- Se integrato con`update_order()` o`send_email()`, potrebbe generare una risposta automatica operativa (es. simulare una mail di conferr

Proposta di attività da fare in classe

Laboratorio di triage e classificazione

- **Obiettivo**: allenare gli studenti a capire come un agente "decide" a quale reparto assegnare una richiesta.
- **Attività**: fornire 10 messaggi clienti diversi (tecnici, ordini, vendite) e far predire agli studenti a quale agente andranno → poi verificare con l'
- **Discussione**: confrontare i criteri di decisione umani vs AI.

Conclusioni e sviluppi

Negli esempi del triage di questo articolo, l'agente ha risposto molto bene, ma si limita a **generare testo:**

- suggerisce cosa fare,
- fornisce istruzioni,
- non esegue operazioni reali (come creare ticket, inviare mail o cercare ordini).

Per implementare i miglioramenti che abbiamo segnalato o per creare un agente reale utilizzato in un'azienda, serve un **agente "tool-enabled"**

- **leggere e interpretare dati reali** (calendari, documenti, screenshot, note, email);
- **accedere a database** interni o CRM;
- **prendere decisioni sulla base di regole aziendali**;
- **avviare procedure automatiche** (creazione report, invio email, registrazione attività, apertura workflow);
- **interagire con servizi esterni** (Google Calendar, Outlook, ERP, sistemi di ticketing, e-commerce...).

Per ottenere tutto questo, l'agente non basta più da solo: serve una **infrastruttura di automazione** che permetta all'agente di parlare con il mon "coltellino svizzero" che collega: API, servizi cloud, email, database, file, fogli Google. sistemi di ticketing. CRM, webhook (messaggio automati

Con n8n si può costruire un flusso di lavoro visivo basato su nodi, dove **ogni nodo esegue un compito**. Per esempio:

- un nodo riceve una richiesta dall'agente AI;
- un altro aggiorna un database;
- un altro invia una email al cliente;
- un altro crea un evento sul calendario.

In pratica n8n permette all'agente di AI di *agire*nel mondo reale.

