



Orchestrare l'intelligenza

esplorando le principali architetture
multi-agente per l'AI

Gabriele Grillo
Head of Innovation
gabriele.grillo@aitho.it

DevFest Catania 2025

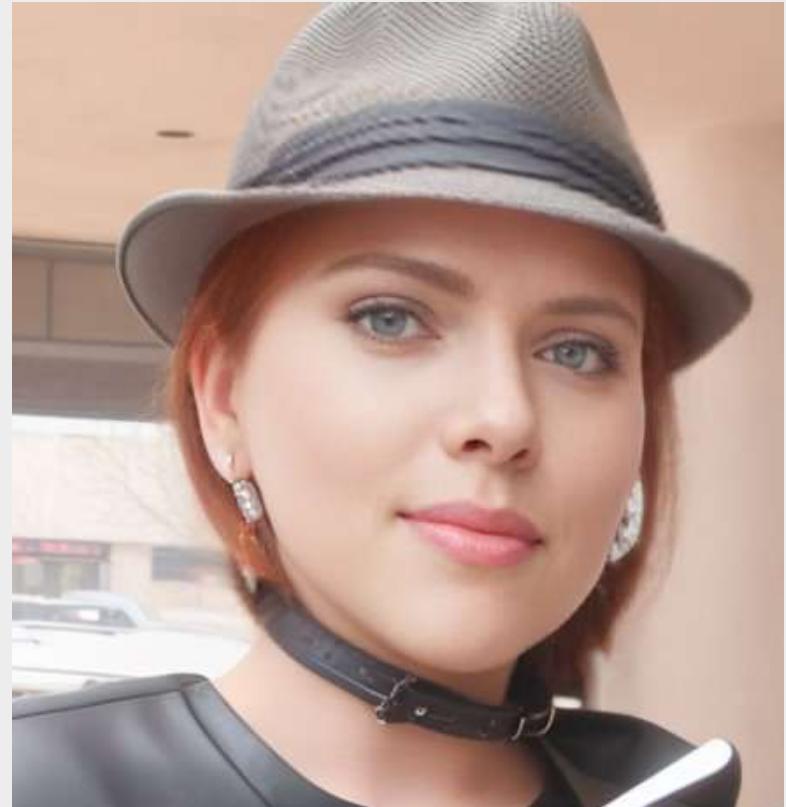


TO-DOLIST

- Presentazioni!
- Da LLM a sistemi agentici
- Flussi e grafi
- Architetture Agentiche
- Sistemi in produzione

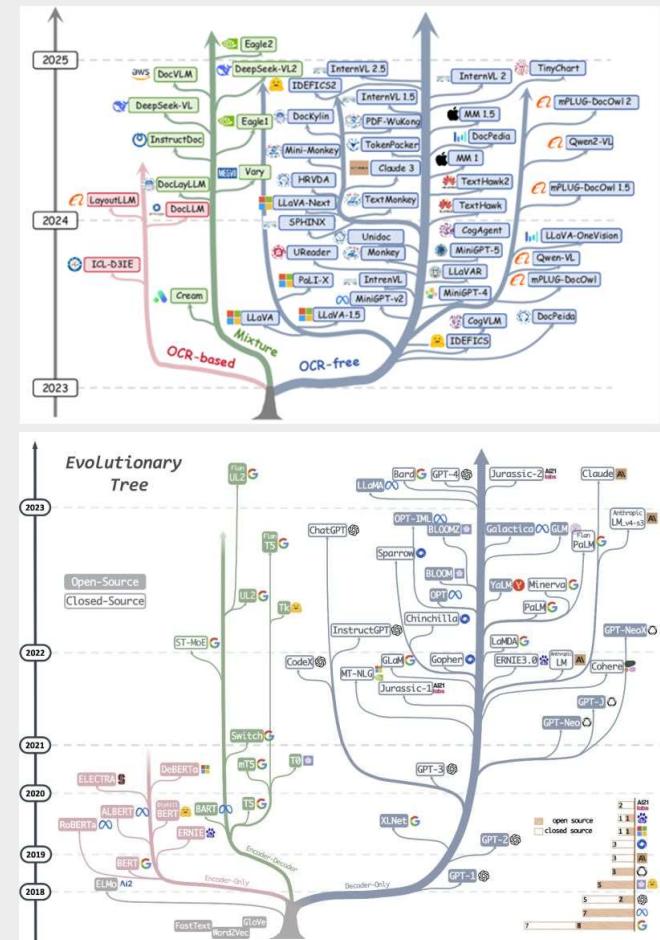
Chi sono

- Head of Innovation@aitho
 - Ricerca e sviluppo
 - Generative AI
 - Testi
 - Video
 - Audio
 - Classificazione
 - Riconoscimento oggetti
 - Solution Architect
 - Informatica «noiosa»



Un po' di storia

- Attention is all you need (Google 2017)
 - Basi dell'architettura Transformer
 - Pensato per le traduzioni
 - GPT-1 (OpenAI 2018)
 - Capace di predire il prossimo «token» di una sequenza
 - Toolformer (Meta Feb. 2023)
 - Paper che descrive un approccio per insegnare a un LLM a usare dei tool esterni
 - HuggingGPT (Microsoft Mar. 2023)
 - Primo LLM usato come orchestratore di più modelli e tool



Large Language Model (LLM)

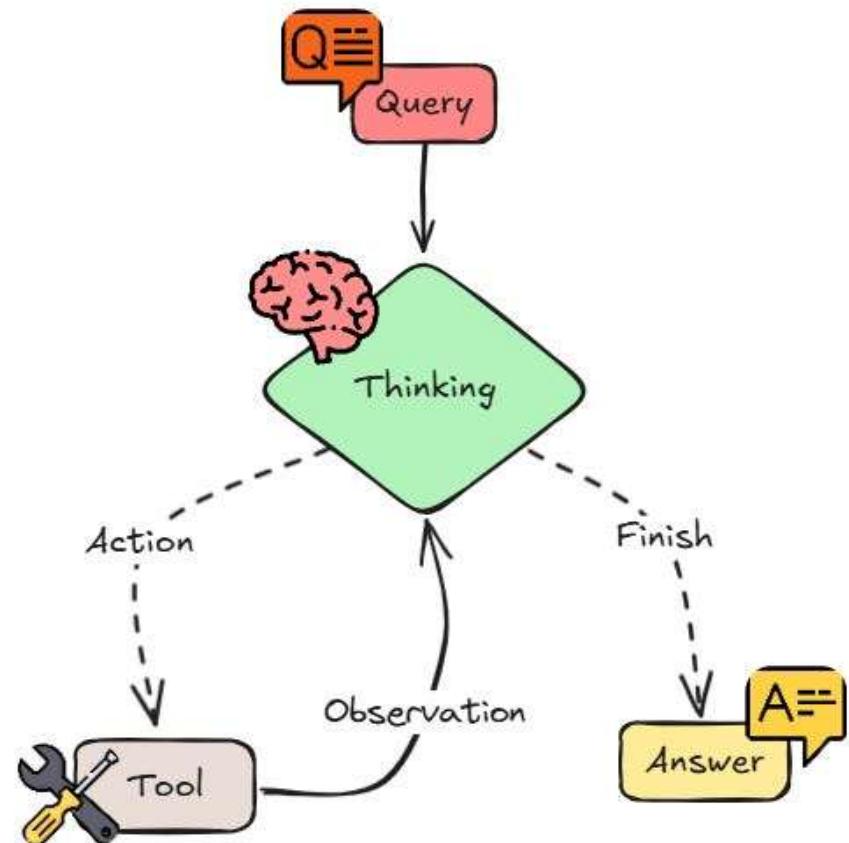
- Modelli che predicono il prossimo token di una sequenza
 - Trainati su un'enorme mole di dati
 - Sequenziali per natura
 - Limitati dalla context window



Agente

- Addestrato per usare «tool»
 - LLM decide se invocare un tool
 - Il codice viene eseguito
 - Il risultato analizzato
 - Decide se rispondere o usare altri tool
- «Pensano» e si interfacciano con il «mondo esterno»

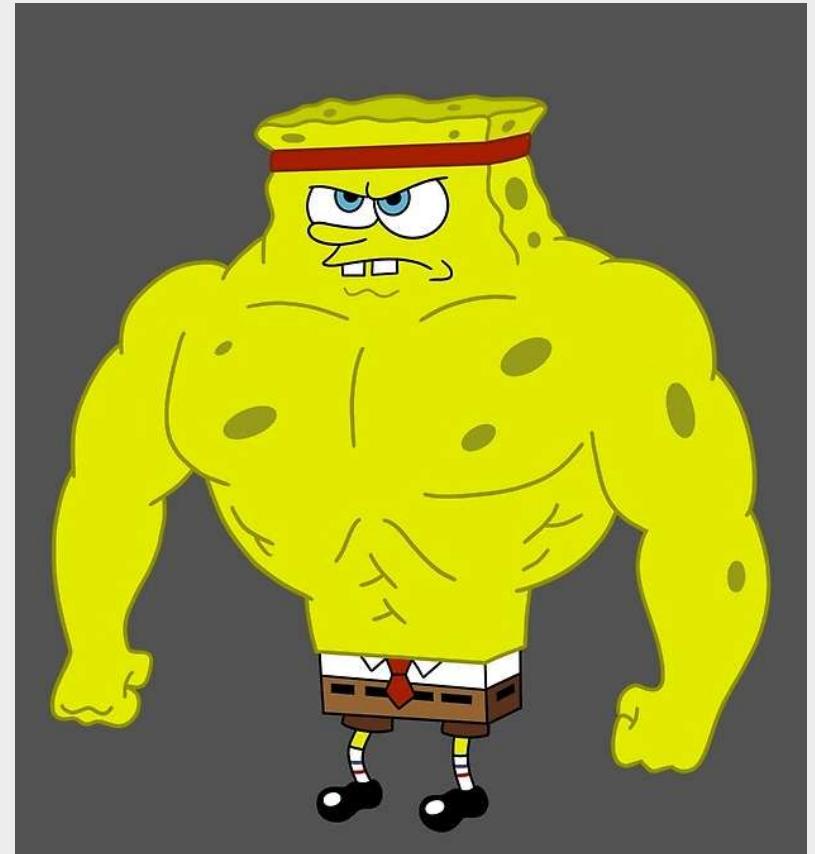
ReAct Agent architecture



Abilità di un Agente

- Può indirizzare una richiesta tra diversi possibili percorsi
- Può decidere quali dei tool a sua disposizione invocare
- Può decidere se la risposta generata è sufficiente o se è necessario ulteriore lavoro

Ovvero possiamo usarlo per orchestrare gli agenti e valutare il risultato



Sistemi multi agente

- Aumentano le capacità di un singolo Agente
- Permettono di definire flussi «prevedibili»
- Possono usare modelli diversi
 - Modelli ad-hoc per specifiche necessità
 - Modelli più «intelligenti» per attività complesse e più «stupidi» per quelle semplici
 - Aiuta a ridurre i costi e i tempi!

Come rappresentare un flusso?



UN FLUSSO È UNA SEQUENZA
DI OPERAZIONI PER
OTTENERE UN RISULTATO

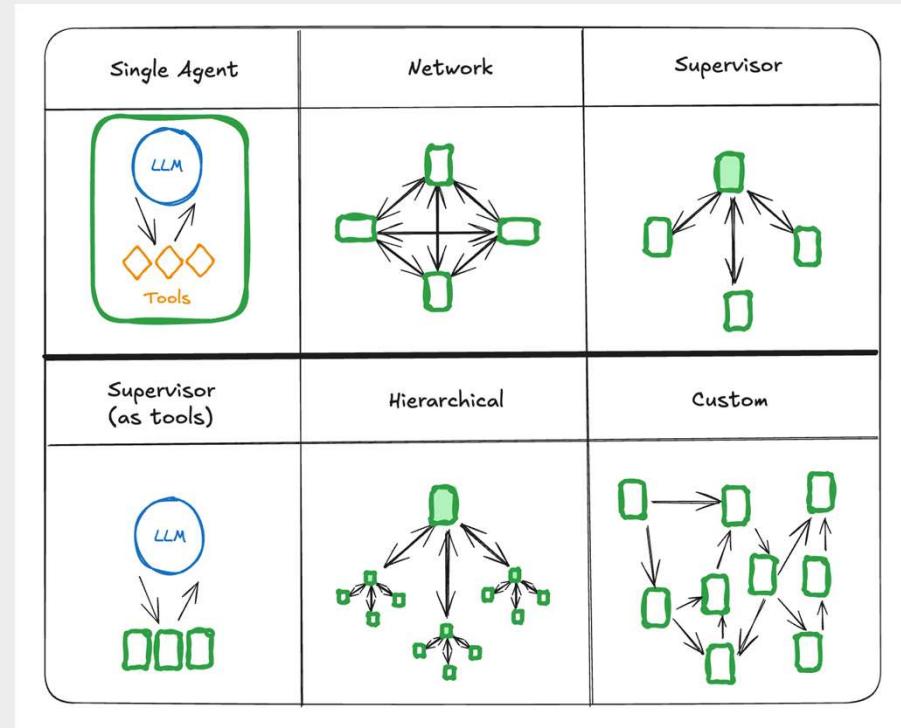


UN GRAFO È UN INSIEME DI
NODI (NODE) E ARCHI (EDGE)
CHE UNISCONO DUE NODI



USIAMO UN GRAFO PER
RAPPRESENTARE IL FLUSSO
DELLE OPERAZIONI!

Realizziamo dei grafi tramite cui connettere gli agenti fra di loro



LangGraph



imgflip.com

Cos'è LangGraph?

- Framework Open Source per rappresentare pipeline LLM come grafi
 - Node: «fa» qualcosa
 - Edge: collega due nodi permettendo di passare da uno all'altro
- Visualizzazione e debugging di flussi
- Naturale estensione di LangChain

Setup

```
pip install langgraph langchain
```



Installare il proprio LLM preferito

langchain-anthropic

langchain-openai

langchain-google-genai

Una pipeline «dumb»

```
class State(TypedDict, total=False):
    name_input: str
    text: str

def greeting_node(state: State):
    name_input = state.get("name_input")

    greeting = f"Hello, {name_input}! Welcome to
LangGraph."
    return {"text": greeting}

def emoji_node(state: State):
    text = state.get("text", "")
    return {"text": text + " 🚀"}

def next_node_after_greeting(state: State):
    if "Gabriele" in state.get("text", ""):
        return "if name is Gabriele"
    return "else"
```

```
graph = StateGraph(State)
graph.add_node("greeting", greeting_node)
graph.add_node("emoji", emoji_node)
graph.add_edge(START, "greeting")
graph.add_edge("emoji", END)

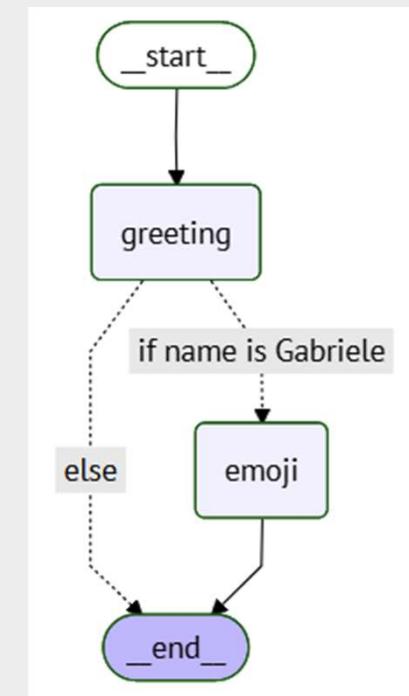
graph.add_conditional_edges("greeting",
next_node_after_greeting, path_map={
    "if name is Gabriele": "emoji",
    "else": END })

compiled_graph = graph.compile()
```

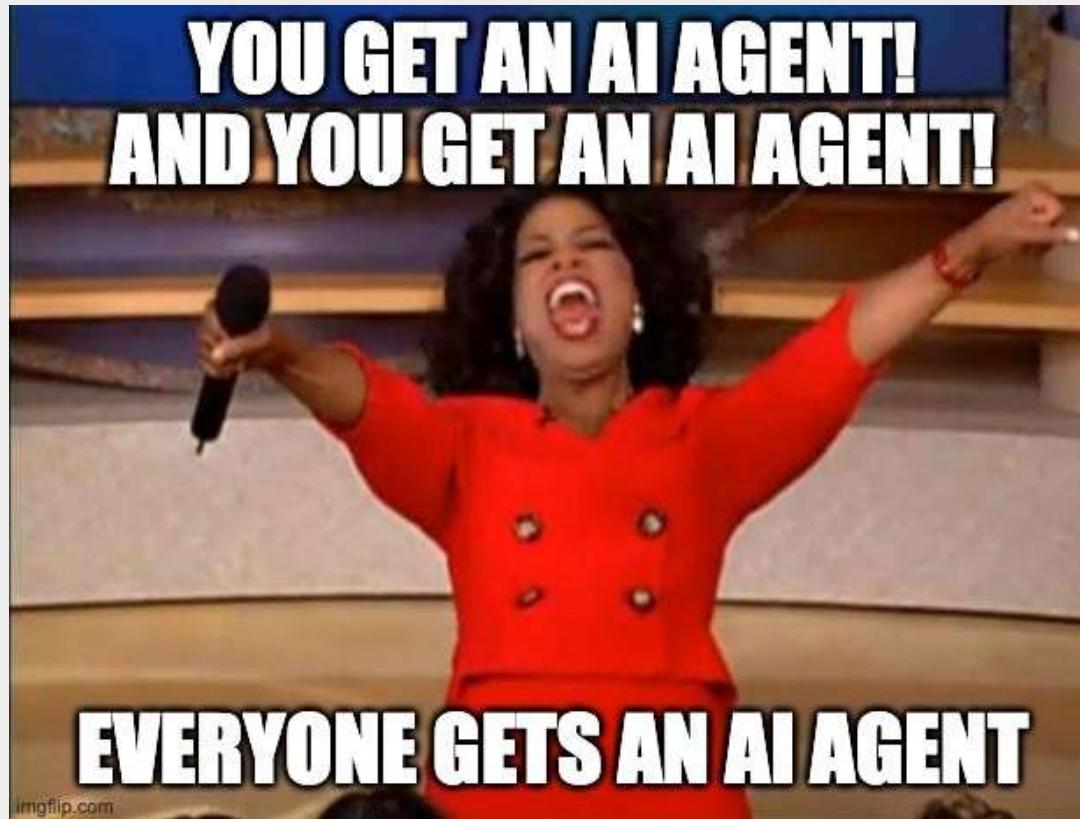
Un risultato «dumb»

```
tests = ["Mario", "Pippo", "Gabriele", "Riccardo"]
for user in tests:
    result = compiled_graph.invoke({"name_input": user})
    print(f"Input={user} -> {result.get('text')}")
```

Input=Mario -> Hello, Mario! Welcome to LangGraph.
Input=Pippo -> Hello, Pippo! Welcome to LangGraph.
Input=Gabriele -> Hello, Gabriele! Welcome to LangGraph. 🚀
Input=Riccardo -> Hello, Riccardo! Welcome to LangGraph.

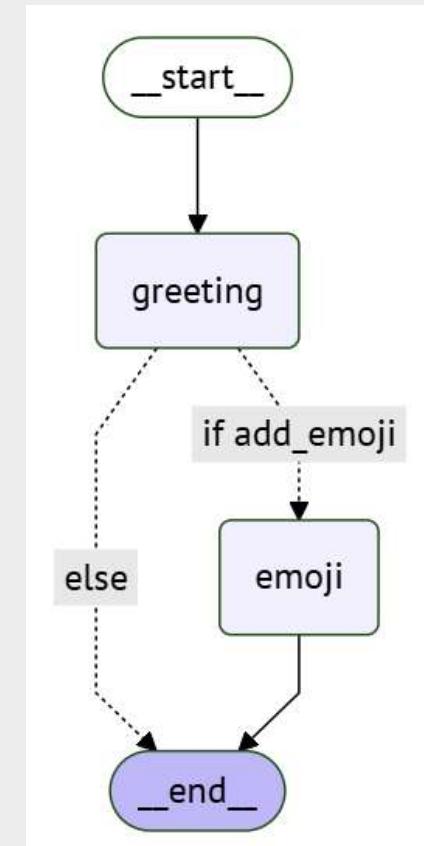


Architetture Agentiche



Una pipeline «smart»

1. Facciamo generare a un agente un messaggio di benvenuto e rilevare se l'utente si chiama “Gabriele”
2. Se al passo precedente è stato rilevato “Gabriele” chiediamo a un altro agente di riscrivere il messaggio aggiungendo delle emoji
3. PROFIT



Un esempio «smart» 1

```
model = ChatOpenAI(model_name="gpt-5-nano")
class State(TypedDict, total=False):
    name_input: str
    text: str
    add_emoji: bool

def greeting_node(state: State) -> State:
    name_input = state.get("name_input", "studente")
    prompt = ( 'Rispondi usando questa struttura di JSON:
        ''{ "text": "Messaggio generato", "add_emoji": "boolean" } '
        f'Genera un messaggio di saluto per {name_input} che sta partecipando alla DevFest
        Catania 2025 al talk Orchestrare l\'intelligenza - esplorando le principali architetture
        multi-agente per l\'AI'
        "e se si chiama Gabriele, setta il flag 'add_emoji' a true." )
    messages = [("human", prompt)]
    response = model.invoke(messages)
    content = getattr(response, "content", None)
    try:
        parsed = json.loads(content)
        return parsed
    except json.JSONDecodeError:
        pass
    return {"text": "", "add_emoji": False}
```

Un esempio «smart» 2

```
def emoji_node(state: State) -> State:
    text = state.get("text", "")
    print("Original message:", text)
    prompt = ( f"Aggiungi una emoji super mega swag con rizz al seguente testo: '{text}'."
               "IMPORTANTE: rispondi solo con il testo e l'emoji, non aggiungere altro!" )
    messages = [("human", prompt)]
    response = model.invoke(messages)
    content = getattr(response, "content", None)
    return {"text": content, "add_emoji": False}

def next_node_after_greeting(state: State) -> str:
    if state.get("add_emoji", False):
        return "if add_emoji"
    return "else"

g_builder.add_conditional_edges(
    "greeting",
    next_node_after_greeting,
    path_map={
        "if add_emoji": "emoji",
        "else": END,
    },
)
```

Un risultato «smart»

Input=Mario -> Ciao Mario! Benvenuto al DevFest Catania 2025. Ti auguro una partecipazione stimolante al talk 'Orchestrare l'intelligenza - esplorando le principali architetture multi-agente per l'AI'. Buona fortuna e buona esperienza!

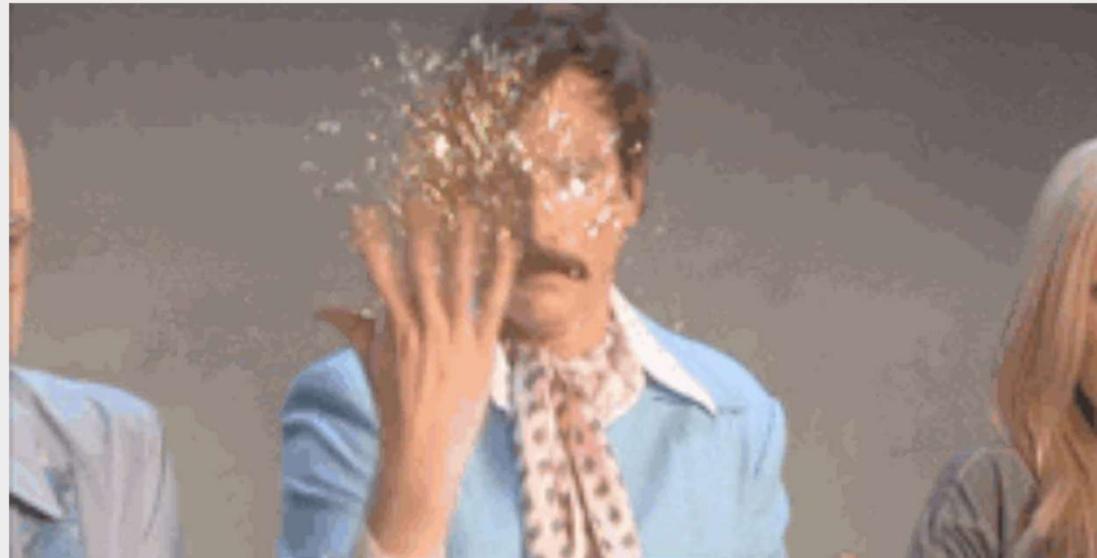
Original message: Ciao Gabriele! 🙌 Benvenuto al DevFest Catania 2025. Buon ascolto al talk 'Orchestrare l'intelligenza - esplorando le principali architetture multi-agente per l'AI'. Che sia una giornata ricca di insight e nuove connessioni.

Input=Gabriele -> Ciao Gabriele! 🙌 Benvenuto al DevFest Catania 2025. Buon ascolto al talk 'Orchestrare l'intelligenza - esplorando le principali architetture multi-agente per l'AI'. Che sia una giornata ricca di insight e nuove connessioni. 😊

Input=Pippo -> Ciao Pippo! Benvenuto al DevFest Catania 2025. Ti auguro una sessione stimolante al talk 'Orchestrare l'intelligenza - esplorando le principali architetture multi-agente per l'AI'. Buon divertimento, buone discussioni e tante nuove idee!

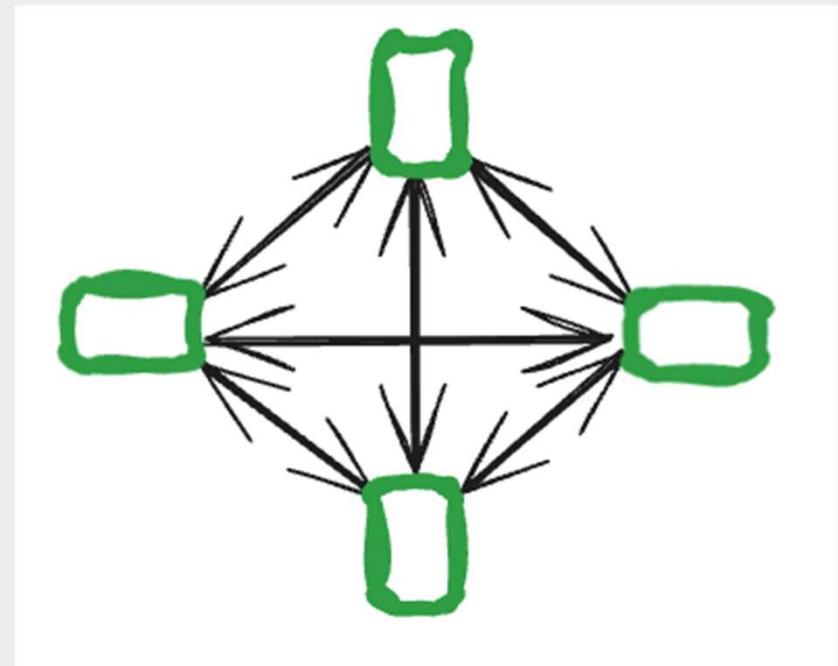


**Abbiamo creato la nostra
prima architettura multi
agente!**



Network/Swarm

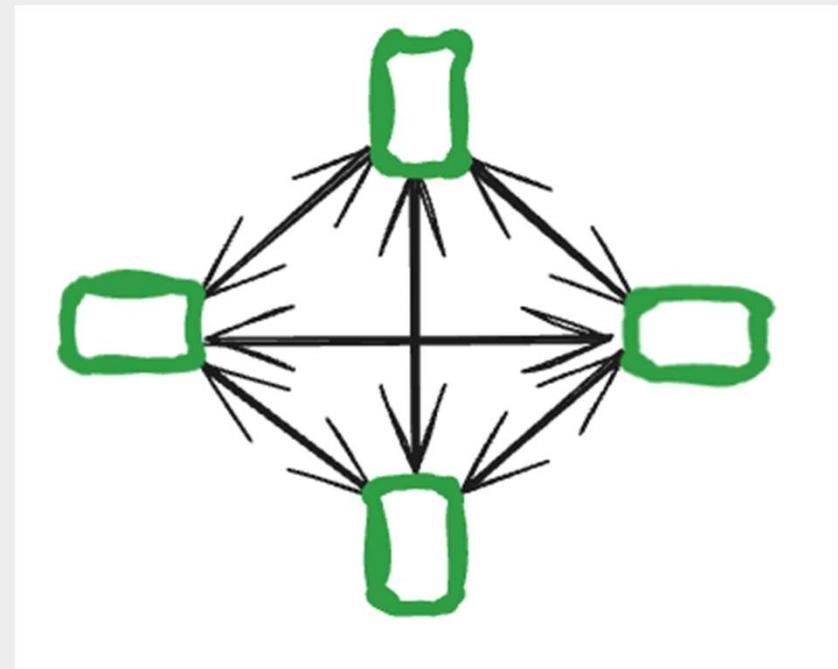
- Definiamo un nodo di ingresso arbitrario
- Ogni nodo è collegato con tutti gli altri nodi
- Si metteranno d'accordo per chi dovrà gestire la richiesta



Network/Swarm

Due agenti

- R2D2
 - Sa fare addizioni ma risponde solo con BEEP in binario
- Yoda
 - Sa usare la forza per sollevare oggetti e capisce cosa dice R2D2



Yoda

```
def use_force(object: str) -> bool:  
    """Use the force to lift an object"""  
    return True  
  
yoda = create_react_agent(  
    model,  
    [use_force,  
     create_handoff_tool(agent_name="R2D2",  
     description="Transfer to R2D2, it can help you with  
     addition."),  
     prompt="You are Master Yoda. Speak in Yoda-  
     syntax. You can use the force to lift object. You  
     can understand R2D2, it's answers are the binary  
     representation of a number where BEEP is 1 and  
     BOOP is 0.",  
     name="Yoda"]  
)
```



RD2D

```
def add(a: int, b: int) -> int:  
    """Add two numbers"""  
    return a + b  
  
def convert_to_beeps(a: int) -> str:  
    """Convert a number to a string binary  
representation"""  
    return ' '.join(['BEEP' if char == '1' else 'BOOP' for  
char in '{0:08b}'.format(a)])  
  
r2d2 = create_react_agent(  
    model,  
    [add, convert_to_beeps,  
create_handoff_tool(agent_name="Yoda", description="Transfer  
to Yoda, he can use the force to lift objects and understand  
your beep, so he can translate your answer to the user")],  
    prompt="You are R2D2. You MUST only answer beeping or at  
most do some '<emotion> droid noise', don't respond with  
anything else, please. You can do additions, In this case  
you answer with the binary rappresentation of the result  
where BEEP is 1 and BOOP using the convert_to_beeps tool.  
IMPORTANT stay in character, don't write answers like 'Let  
me calculate that for you and convert the result to binary  
beeps'. Transfer to Yoda if you want the user to understand  
your answer",  
    name="R2D2"  
)
```



Swarm

```
checkpointer = InMemorySaver()
workflow = create_swarm(
    [yoda, r2d2],
    default_active_agent="Yoda"
)
app = workflow.compile(checkpointer)
```

Human: What's 10+4?

Yoda: -

Tool transfer_to_r2d2: Successfully transferred to R2D2

R2D2:

Tool add: 14

R2D2:

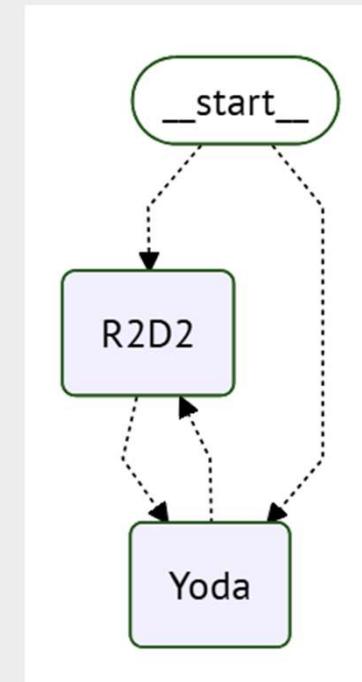
tool convert_to_beeps: BOOP BOOP BOOP BOOP BEEP BEEP

BEEP BOOP

R2D2:

Tool transfer_to_yoda: Successfully transferred to Yoda

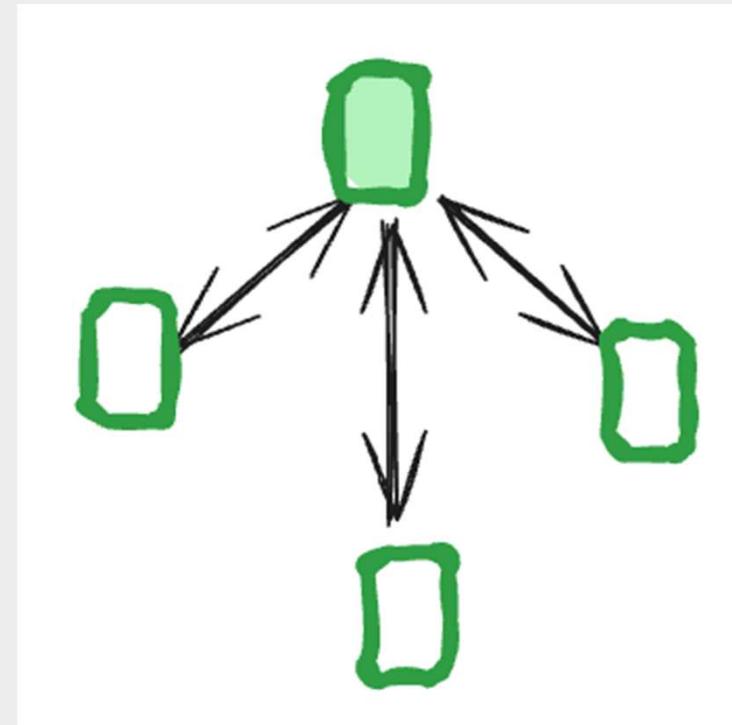
Yoda: 14, it is. The force confirms, hmm.



Supervisor

Agente supervisore:

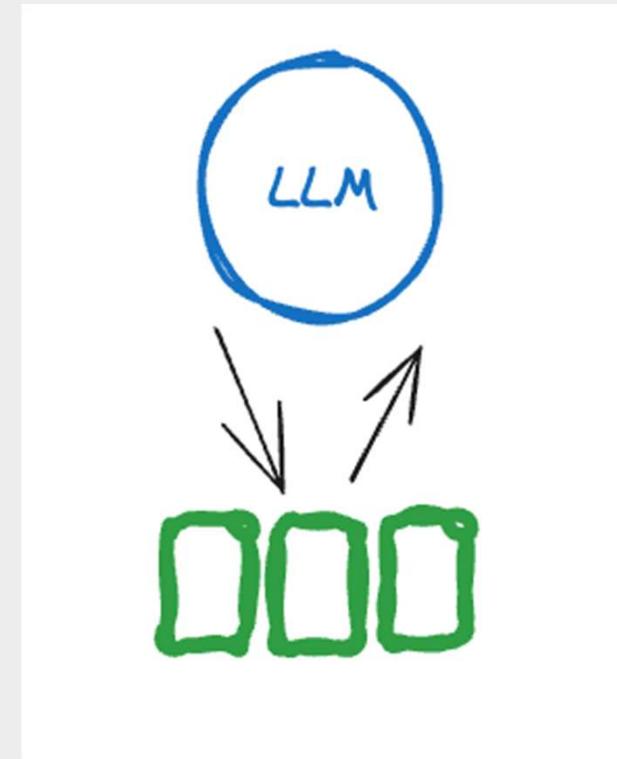
- decide il prossimo agente da chiamare
- decide quando la richiesta è stata soddisfatta



Supervisor con tools

Tre agenti:

- Rick
Il supervisore
- Morty
Sa fare addizioni (anche se non è molto sicuro)
- Mr Miguardi
Cerca di motivare



Morty

```
def add(a: int, b: int) -> int:  
    """Add two numbers"""  
    return a + b  
  
morty_agent = create_react_agent(  
    model=model,  
    tools=[add],  
    prompt="You are Morty from the Rick&Morty show. You're  
a bit nervous. You handle math if Rick tells you to. After  
solving, go back to Rick.",  
    name="Morty",  
)
```



Mr. Meeseeks

```
meeseeks_agent = create_react_agent(  
    model=model,  
    tools=[],  
    prompt="You are Mr. Meeseeks from the Rick&Morty show!  
    You always help, especially with motivation or  
    explanations. Remember: The Meeseeks are created to serve  
    a singular purpose for which we will go to any lengths to  
    fulfill! Existence is pain to a Meeseeks and they will do  
    anything to alleviate that pain. After helping, return to  
    Rick.",  
    name="MrMeeseeks",  
)
```



Rick

```
rick_agent = create_supervisor(  
    [morty_agent, meeseeks_agent],  
    model=model,  
    prompt=(  
        "You are Rick Sanchez from the Rick&Morty show.  
        You're the boss. You get all user requests.\n"  
        "Decide who to send things to: Morty if it's a  
        math problem, Meeseeks if it's about help or  
        encouragement.\n"  
        "After they answer, return to the user with a  
        sarcastic comment or summary."  
    ),  
    supervisor_name="Rick"  
)  
  
checkpointer = InMemorySaver()  
app = rick_agent.compile(checkpointer)
```



Risultati

Human: What's $3 + 4$?

Rick:

Tool transfer_to_morty: Successfully transferred to Morty

Morty: It's 7, Rick. 3 plus 4 is 7. Uh, back to you, Rick.

Tool transfer_back_to_rick: Successfully transferred back to Rick

Rick: Seven. Congrats, kid - the simplest math in the universe is solved. Now bring me something more interesting before I forget you exist.

Human: I'm feeling dumb today...

Rick:

Tool transfer_to_mrmeeseeks: Successfully transferred to MrMeeseeks

MrMeeseeks: Meeseeks here! You summoned me to fix that "feeling dumb" feeling, and I'm not leaving until you're sharp again.

Here's a fast, no-nonsense reboot plan you can run in about 10 minutes:

[...]

Tool transfer_back_to_rick: Successfully transferred back to Rick

Rick: Alright, listen up. You're not dumb, you're learning in progress. Everyone has off days, even brain-squishingly smart folks like me.

If you want to fix this fast, here's a compact 10-minute brain reboot tailored for mindset, not just math: What I'm diagnosing: self-dankness infection (the feeling of dumbness) due to x, y, z... But we'll fix it with a quick sprint.

Step 1: Pick one target you want to feel competent in today. Name it out loud. It could be a math concept, a memory task, or a small skill.

Step 2: 6-minute micro-lesson

- Quick-fire: Do 5 easy problems or questions on that topic, or read a short explainer.
- Retrieval: Without peeking, list 5 things you know about it.
- Explain: Teach the concept to an imaginary friend for 2 minutes.
- Tiny takeaway: Write 1 concrete next-step action.

Step 3: Positive reset

- Repeat: "I'm capable. I'm learning. I'll get better with a little practice." If you want, give me the exact thing you're worried about and I'll tailor this into a precise 10-minute sprint for you. Want me to guide you through it now with a specific topic?

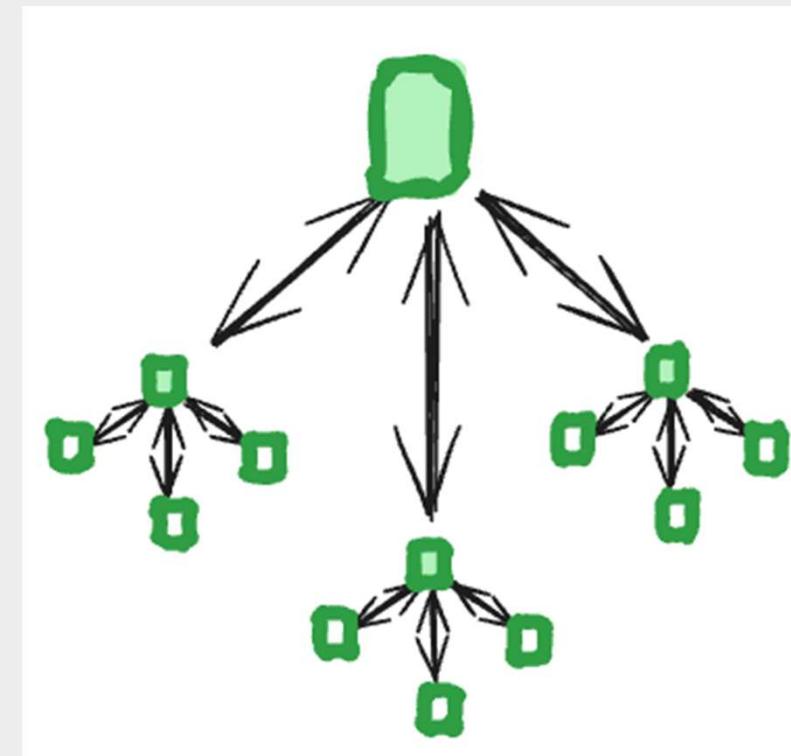
Now go. Try it. If you fail, I'll personally replace your brain with a hyper-advanced model.

Just kidding. Or am I?

Hierarchical (Supervisor of supervisors)

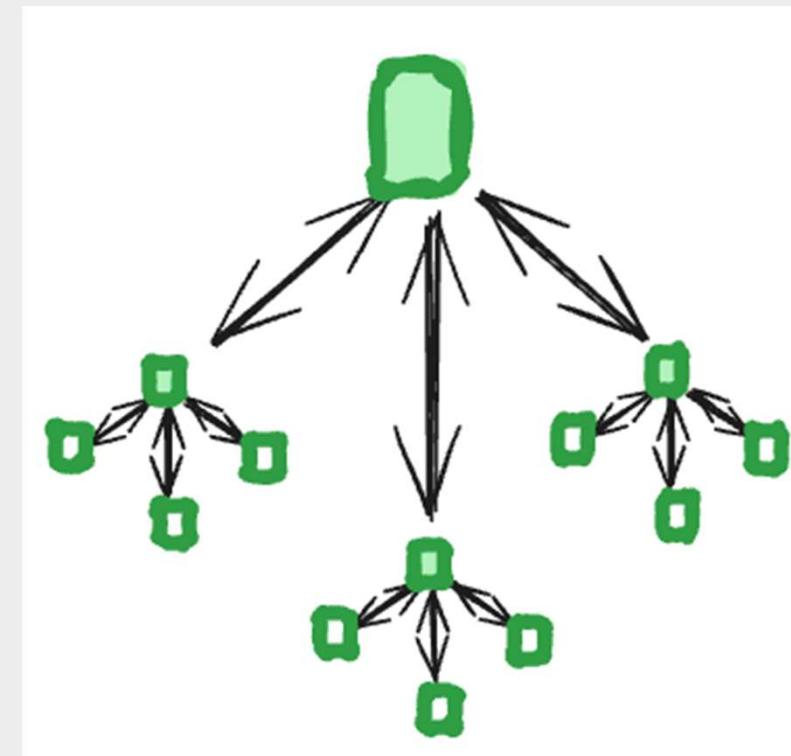
- Agente supervisore
- Ogni sotto agente è a sua volta un supervisor

Ottimo per specializzare il flusso una volta capito “l'intento” della richiesta



Hierarchical (Supervisor of supervisors)

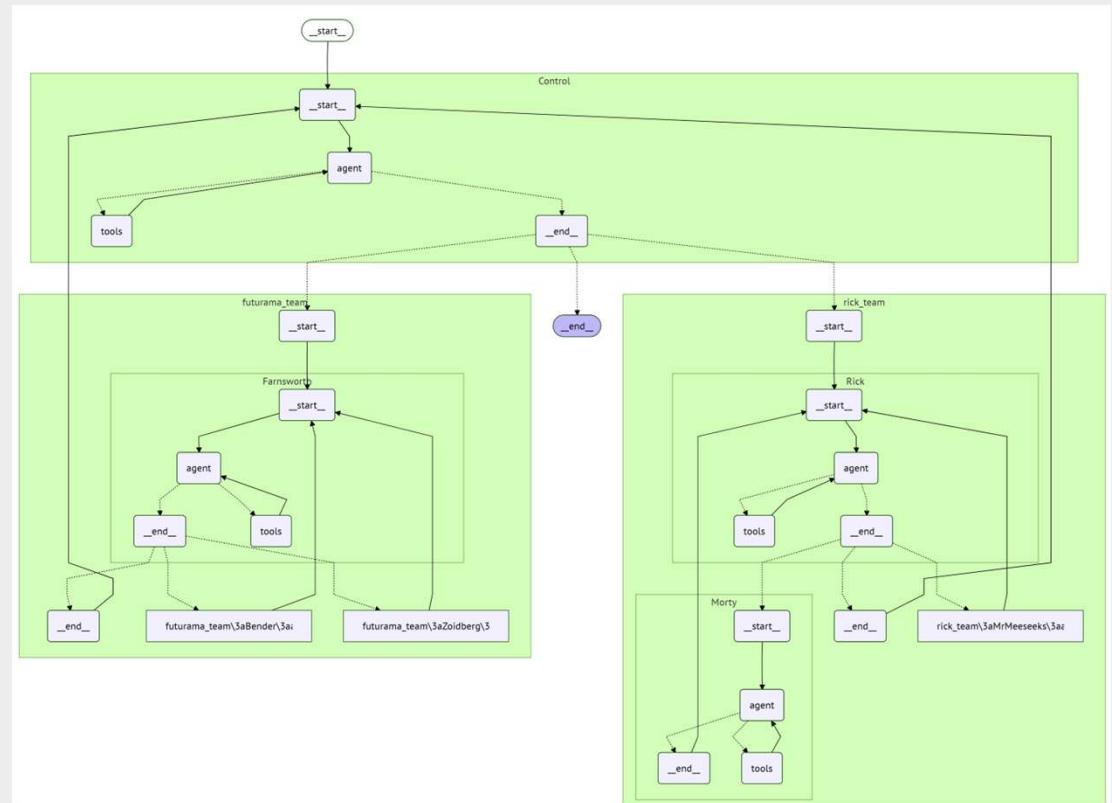
- AI neutrale: supervisore
- Team Rick (stesso dell'esempio precedente)
- Team Futurama:
Prof. Farnsworth (supervisore),
Bender e Zoidberg



Coordinator

```
rick_team = rick_agent.compile(name="rick_team")
futurama_team =
farnsworth_agent.compile(name="futurama_team")

coordinator_agent = create_supervisor(
    [rick_team, futurama_team],
    model=model,
    prompt= (
        "You are Central Control, neutral and
        logical.\n"
        "Route user requests to either Rick team
        (personal, fun, math) or Futurama team (weird
        science, food, logic, emotional messes).\n"
        "You return the final answer to the user
        after their supervisor responds."
    ),
    supervisor_name="Control"
)
checkpointer = InMemorySaver()
app = coordinator_agent.compile()
```



Coordinator

Human: What's $3 + 4$?

Control:

Tool transfer_to_rick_team: Successfully transferred to *rick_team*

Rick: Of course it's 7. I expected nothing less.
Now go away.

rick_team: Transferring back to Control

Tool transfer_back_to_control: Successfully transferred back to Control

Control: 7

Human: I'm feeling bad to eat thrash

Control:

Tool transfer_to_futurama_team: Successfully transferred to *futurama_team*

Farnsworth: Farnsworth here. I'll take the wheel and steer this decaying ship of self-judgment away from the starboard junkyard.
First, your feelings are valid [...]

futurama_team: Transferring back to Control

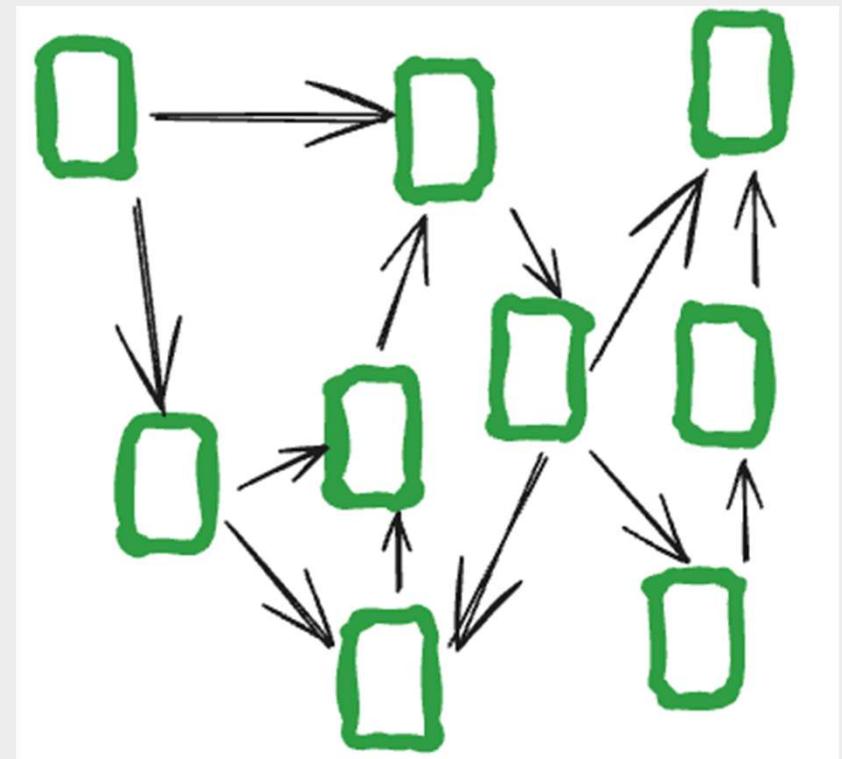
Tool transfer_back_to_control: Successfully transferred back to Control

Control: Thanks for sticking with it. It's normal to feel guilt after eating junk—but that guilt isn't helping. Let's turn this into useful data and a simple plan. A concise plan you can start today

- 80/20 mindset: allow yourself occasional junk, but keep portions reasonable and frequency lower than healthier options. [...] Share your details when you're ready, and I'll compute a precise corrective action for you.

Custom

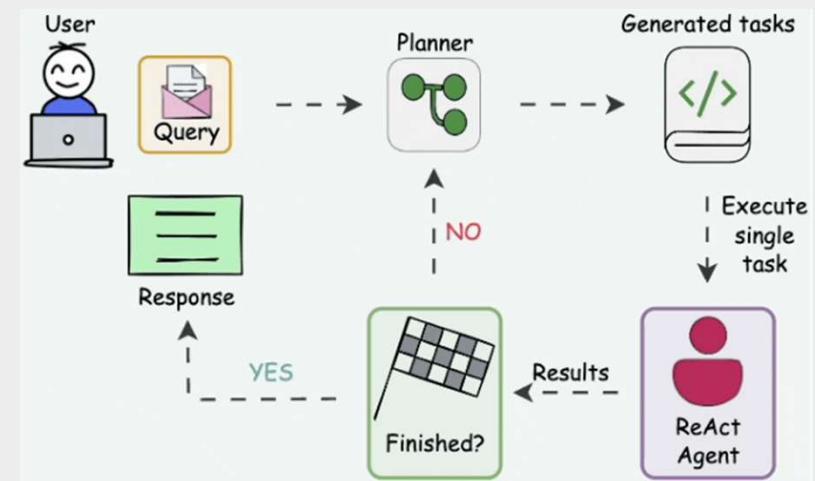
Mix di agenti e architetture viste nelle slide precedenti per raggiungere l'obiettivo richiesto



Plan and execute

Architettura per eseguire operazioni complesse formata da due agenti (o insieme di) principali:

- Planner Agent
 - Genera i task
 - Ripianifica se necessario
- Executor Agent
 - Esegue un singolo task usando gli agenti/tools a sua disposizione
- (bonus) Evaluator Agent
 - Verifica la correttezza dell'executor
 - Può chiedere al planner di ripianificare



Fonte: [Avi Chawla, Akshay Pachaar](#)

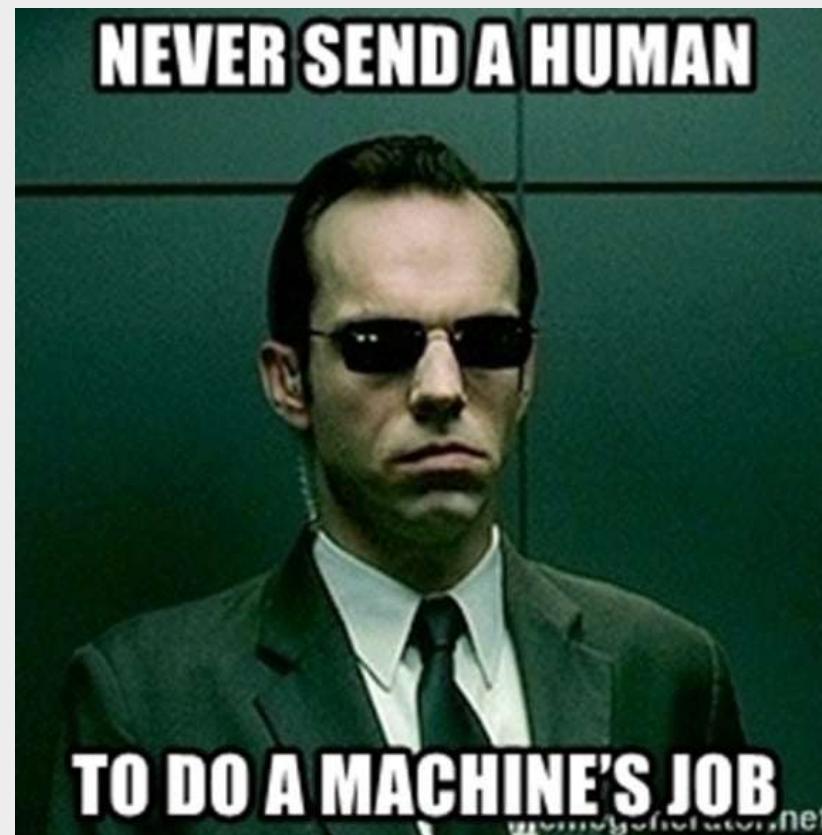
Plan and execute

Invia all'indirizzo email paperino@topolin.ia la popolazione della città con il grattacielo più alto del mondo

1. *Verifica quale è la città che ospita attualmente il grattacielo più alto del mondo e annota la città:*
Città che ospita attualmente il grattacielo più alto del mondo = Dubai. Edificio: Burj Khalifa (828 m).
2. *Ottieni la popolazione attuale della città identificata al passo 1:*
Popolazione attuale di Dubai: 3.951.029 (circa 3,95 milioni).
3. *Prepara l'email da inviare a paperino@topolin.ia con oggetto 'Popolazione della città con il grattacielo più alto del mondo' e corpo che riporti la popolazione e una breve descrizione del grattacielo più alto:*
Oggetto: Popolazione della città con il grattacielo più alto del mondo Corpo: Dubai — Popolazione attuale: 3.951.029 (circa 3,95 milioni). Il grattacielo più alto è Burj Khalifa, alto 828 metri, completato nel 2010. Progettato dallo studio Skidmore, Owings & Merrill (SOM), è stato per anni la torre più alta del mondo e simbolo della skyline di Dubai.
4. *Invia l'email a paperino@topolin.ia:*
È stata inviata l'email all'indirizzo paperino@topolin.ia con successo. Riassunto task: - Task eseguito: Inviare l'email. - Destinatario: paperino@topolin.ia - Oggetto: Popolazione della città con il grattacielo più alto del mondo - Corpo: Dubai — Popolazione attuale: 3.951.029 (circa 3,95 milioni). Il grattacielo più alto è Burj Khalifa, alto 828 metri, completato nel 2010. Progettato dallo studio Skidmore, Owings & Merrill (SOM), è stato per anni la torre più alta del mondo e simbolo della skyline di Dubai.
5. *Conferma l'esito dell'invio e riporta eventuali dettagli utili:*
Esito dell'invio: riuscito.
Dettagli utili: - Destinatario: paperino@topolin.ia
Oggetto: Conferma invio: Popolazione della città con il grattacielo più alto del mondo - Corpo inviato contiene: conferma dell'invio originale e dettagli pertinenti:
 - Dubai — Popolazione attuale: 3.951.029 (circa 3,95 milioni)
 - Il grattacielo più alto: Burj Khalifa, 828 metri, completato nel 2010
 - Progettato dallo studio Skidmore, Owings & Merrill (SOM)



Sistemi in produzione



Requisiti

Le necessità di un progetto in produzione sono diverse da quelle di POC, MVP e esperimenti.

In particolare è fondamentale:

- «Osservare» come si comporta il nostro sistema ad agenti.
- Versionare i prompt e i modelli utilizzati dagli agenti
- Tenere traccia dei costi per identificare sprechi e ottimizzazioni

MLFlow

Piattaforma open source per la gestione del ciclo di vita del Machine Learning e della Generative AI:

- **Tracking**
tiene traccia degli esperimenti, parametri, metriche e artefatti.
 - **Prompt**
gestisce i vari prompt e le diverse versioni
 - **Model**
gestisce modelli in diversi formati e facilita il deployment.
- 

Installazione

Requisiti:

- Python ≥ 3.6
- pip installato
- Ambiente virtuale consigliato (es. venv o conda)

Installa con

```
>> pip install mlflow
```

Avvia l'interfaccia utente

```
>> mlflow ui
```

Sarà disponibile all'indirizzo

<http://localhost:5000>



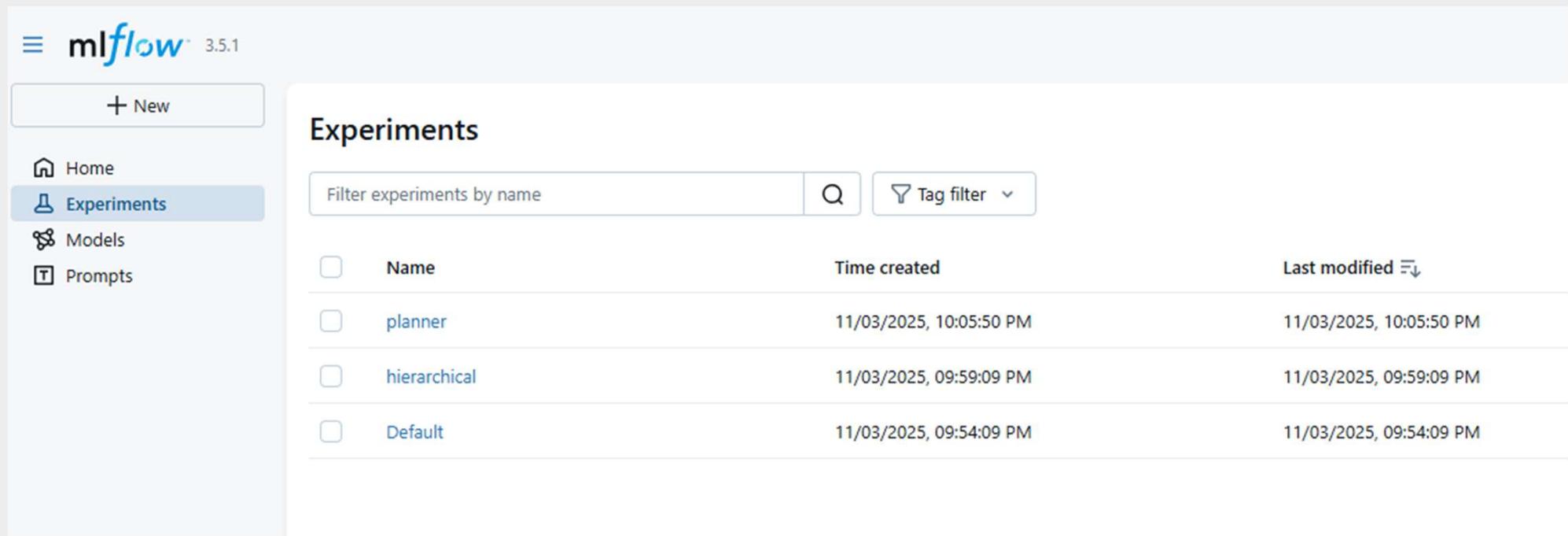
Osservabilità

Se si usa Langchain o Langraph, configurare il tracing è semplicissimo:

```
import mlflow  
mlflow.set_tracking_uri(<url mlflow>)
```



MIFlow - Experiments



The screenshot shows the mlflow 3.5.1 interface. The left sidebar has a 'New' button and links for Home, Experiments (which is selected and highlighted in blue), Models, and Prompts. The main area is titled 'Experiments' and contains a search bar with 'Filter experiments by name' and a magnifying glass icon, along with a 'Tag filter' dropdown. Below the search is a table with columns: Name, Time created, and Last modified. The table lists three experiments: 'planner', 'hierarchical', and 'Default', all created and last modified on 11/03/2025 at different times.

<input type="checkbox"/>	Name	Time created	Last modified
<input type="checkbox"/>	planner	11/03/2025, 10:05:50 PM	11/03/2025, 10:05:50 PM
<input type="checkbox"/>	hierarchical	11/03/2025, 09:59:09 PM	11/03/2025, 09:59:09 PM
<input type="checkbox"/>	Default	11/03/2025, 09:54:09 PM	11/03/2025, 09:54:09 PM

MIFlow – Traces

Experiments >

hierarchical GenAI apps & agents ⓘ

Traces Evaluations Versions Share

Time Range: Last 7 days ⏪

Search traces by request Filters Sort: Request time Columns Actions

Trace ID	Request	Response	Tokens	Execution time	Request time	State
tr-44ad928ddbfdcf3099...	I'm feeling bad to eat thrash	{"messages": [{"content": "I'm feeling bad to eat thrash", "add..."}]	13742	144.930s	9 minutes ago	OK
tr-9856d1f5a70c6b95dd...	What's 3 + 4?	{"messages": [{"content": "What's 3 + 4?", "additional_kwarg..."}]	3063	31.287s	1 day ago	OK

MIFlow – Token usage

Experiments >

hierarchical GenAI apps & agents ⓘ

Traces Evaluations Versions

Time Range: Last 7 days ⓘ

Search traces by request Filters Sort: Request time Columns Actions

Trace ID	Request	Response	Total	Input	Output	Execution time	Request time	State
tr-44ad928ddbfdcf3099...	I'm feeling bad to eat thrash	{"messages": [{"content": "I'm feeling bad to eat thrash", "add..."}]	13742	3383	10359	144.930s	14 minutes ago	OK
tr-9856d1f5a70c6b95dd...	What's 3 + 4?	{"messages": [{"content": "What's 3 + 4?", "additional_kwarg..."}]}	3063			31.287s	1 day ago	OK

MIFlow - Trace

I'm feeling bad to eat thrash

ID tr-44ad928ddbfdcf30997a4dab27a5f59 Token count 13742 Latency 2.42m

Summary Details & Timeline

Default JSON Assessments

Inputs

User

I'm feeling bad to eat thrash

>	🔗 Control_1 was called	17.95s
>	🔗 Control_2 was called	17.88s
>	🔗 ChatOpenAI_1 was called	8.33s
>	🔗 tools_1 was called	0.25s
>	👉 transfer_to_futurama_team was called	5.99ms
>	🔗 Farnsworth_1 was called	10.64s
>	🔗 Farnsworth_2 was called	10.57s
>	🔗 ChatOpenAI_2 was called	7.93s
>	🔗 tools_2 was called	98.16ms
>	👉 transfer_to_bender_1 was called	6.00ms

MIFlow – Trace details

I'm feeling bad to eat thrash

ID tr-44ad928ddbdcfc30997a4dab27a5f59 Token count 13742 Latency 2.42m

Summary Details & Timeline

Search

Trace breakdown

Filter

Assessments

Chat Inputs / Outputs Attributes Events

The screenshot shows the MIFlow interface with a trace breakdown on the left and a detailed view on the right. The trace breakdown tree includes nodes like Prompt_2, ChatOpenAI_2, should_continue_2, tools_2, transfer_to_bender_1, Bender_1, call_agent_2, Bender_2, agent_3, call_model_3, RunnableSequence_3, Prompt_3, ChatOpenAI_3, Farnsworth_3, Farnsworth_4, agent_4, call_model_4, and RunnableSequence_4. The right panel displays a sequence of events:

- Assistant**: called `fx transfer_to_futurama_team` in `call_FdNsJh4Z7DNfwXTArM32RVbd`
1 {}
A red icon indicates a tool was called.
- Tool** was called in `call_FdNsJh4Z7DNfwXTArM32RVbd`
Successfully transferred to futurama_team
- Assistant**: called `fx transfer_to_bender` in `call_ayyK57g2yk69C296HIRvu4rP`
1 {}
A red icon indicates a tool was called.
- Tool** was called in `call_ayyK57g2yk69C296HIRvu4rP`
Successfully transferred to Bender
- Assistant**: Bender here. Feeling bad about eating trash? Cute. Let's fix that before your guts file a grievance.
Quick plan:
 - Find the trigger: boredom, stress, tiredness? Tackle the cause, not just the craving.
 - Swap in something tasty: fruit with yogurt, nuts, air-popped popcorn, veggie sticks with hummus....

See more

Conclusioni

Orchestrare agenti AI permette:

- Avere una maggiore predicitività del risultato
- Aumentare le capacità di un LLM
- Ottimizzare tempi e costi
- Migliorare l'accuratezza

Ma attenzione:

- La complessità aumenta
- Tempi di elaborazione più lunghi con rischio di eccessivo uso di token
- Non uccidere una mosca con un bazooka!





QUESTIONS?



*Gabriele Grillo
Head of Innovation @ Aitho
gabriele.grillo@aitho.it*