



Ordine degli Ingegneri della Provincia di Roma

Corso

“Dai Large Language Model (LLM) ai Large Action Model (LAM)”

Ing. Mario D’Ettorre

Sabato, 11 Aprile 2026



Sistema di IA

AI Act, Regolamento UE 2024/1689, articolo 3 (Definizioni), par. 1

Sistema automatizzato progettato per funzionare con livelli di autonomia variabili e che può presentare adattabilità dopo la diffusione e che, per obiettivi espliciti o impliciti, deduce dall'input che riceve come generare output quali previsioni, contenuti, raccomandazioni o decisioni che possono influenzare ambienti fisici o virtuali

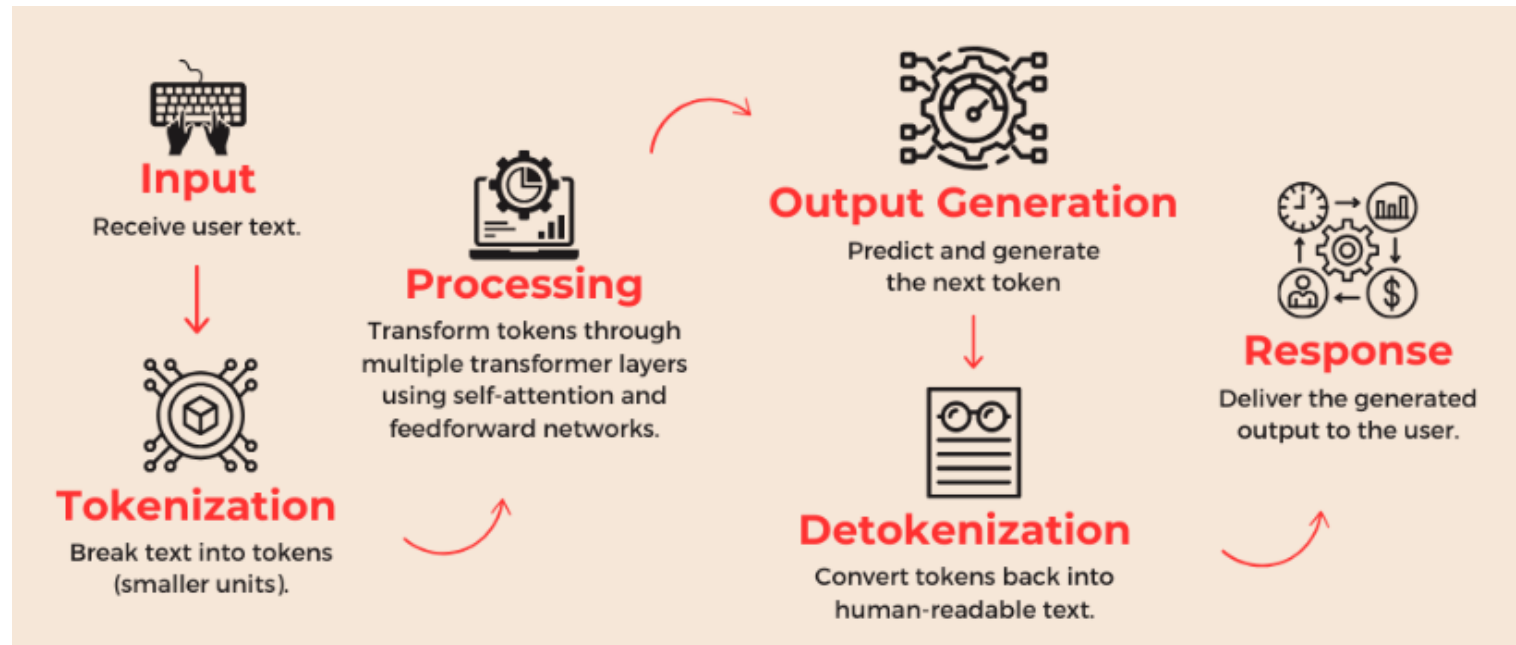
Sistema di IA

Un sistema di IA è costituito da una rete neurale artificiale che chiamiamo Large Language Model - LLM.

Il Sistema IA utilizza il modello LLM per calcolare di volta in volta la parola più probabile in base alle altre precedenti, tenendo conto dell'addestramento che ha ricevuto.

Un LLM può:

- Rispondere a domande
- Scrivere testi
- Tradurre lingue
- Riassumere documenti
- Generare codice
- Analizzare e spiegare concetti





Modelli LLM

1. I modelli LLM sono diventati sempre più grandi passando da milioni a centinaia di miliardi di parametri (GPT-3 175B, GPT 3.5 e GPT-4 non dichiarato ma sicuramente molto più grandi)
2. I modelli LLM erano grandi ma «grezzi» e quindi il focus si è spostato dall'addestramento in base al testo ad aggiungere un fine-tuning supervisionato dall'uomo (Instruction Tuning e RLHF - Reinforcement Learning from Human Feedback)
3. I modelli LLM così ottimizzati **sono comunque passivi**, ad un input in ingresso generano un output (attualmente oltre al testo anche immagini, audio e video) ma non interagiscono direttamente
4. **Nasce l'idea di AI Agent che utilizza un LLM che riceve gli input da risolvere, decide le azioni ed attiva strumenti esterni per compiere azioni**

Agenti AI

Un Agente AI è di fatto un modello LLM che si trasforma da «parlatore» ad **esecutore di compiti**.

Il modello LLM viene inserito dentro un **loop decisionale** che si può considerare come un **sistema automatico a retroazione**.

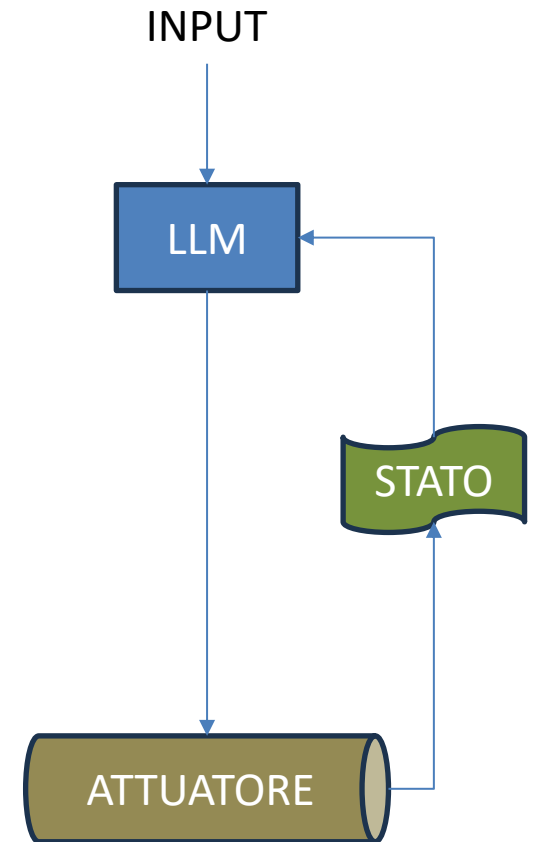
Infatti in un agente troviamo:

un ingresso (in forma testuale, sonora, ecc.) che definisce il valore desiderato come obiettivo da raggiungere

un elemento attivo - il modello LLM – valuta la differenza dell'input con lo stato del sistema e decide i tool da usare (tra quelli disponibili) e le azioni da compiere

un attuatore – l'insieme dei tool che può utilizzare l'Agent - che riceve le istruzioni dal modello LLM sui tool da usare e le azioni da compiere e le esegue

un sistema di controllo che chiude il loop - aggiorna lo stato del sistema dopo le azioni e ritorna in input sull'elemento attivo - il modello LLM - che calcola la correzione necessaria



Anatomia di un Agent AI

- LLM -> decide cosa fare
- Codice esterno -> controlla i loop
- Codice esterno -> esegue



```
def execute_tool(action, args):  
    if action == "get_weather":  
        return weather_api(args["city"])  
  
    elif action == "calculator":  
        return eval(args["expression"])  
  
    else:  
        return "unknown tool"
```

Note:

1. L'agente continua a iterare finché l'LLM non decide di fermarsi (loop di controllo)
2. L'agente ha una memoria «**history**» che viene aggiornata sulle azioni eseguite e i risultati ottenuti.
3. Il modello LLM riceve lo stato del goal da raggiungere e la storia passata e decide l'azione successiva da eseguire e gli argomenti; esempio:
 {"action": "get_weather", "args": {"city": "Rome"}}
4. La parte agentica vera e propria è nella chiamata ad API esterna

```
def agent_loop(user_input):  
    state = {  
        "goal": user_input,  
        "history": [],  
        "done": False,  
        "final_answer": None  
    }  
  
    while not state["done"]:  
  
        # 1. Chiamata LLM (decisione)  
        response = llm_call(state)  
  
        action = response.get("action")  
        args = response.get("args", {})  
  
        # 2. Caso: risposta finale -> STOP  
        if action == "final_answer":  
            state["done"] = True  
            state["final_answer"] = response.get("output")  
            break  
  
        # 3. Caso: azione da eseguire  
        result = execute_tool(action, args)  
  
        # 4. Aggiorna stato  
        state["history"].append({  
            "action": action,  
            "args": args,  
            "result": result  
        })  
  
    return state["final_answer"]
```

Dall'Agent AI al Large Action Model - LAM

Mentre un Agente ha un loop orchestrato esternamente attraverso la risposta dell'LLM, un LAM ha il loop internalizzato nel modello

Il LAM agisce direttamente in base al goal e al contesto e non descrive l'azione ma la produce.

L'input dell'utente e la conoscenza del contesto gestiscono l'azione di risposta del modello

Lo stato, che prima era esplicito ora fa parte del contesto e implicitamente nei pesi del modello in funzione dell'addestramento al quale è stato sottoposto

```
def lam_loop(user_input):  
  
    observation = {  
        "goal": user_input,  
        "context": []  
    }  
  
    while True:  
  
        # Il modello decide DIRETTAMENTE l'azione  
        action = lam_model(observation)  
  
        # L'ambiente esegue (black-box)  
        observation, done = environment_step(action)  
  
        if done:  
            break  
  
    return extract_final_answer(observation)
```



Prompt dell'LLM (fondamentale)

- Un LLM genera testo libero per natura, ma per essere integrato con un codice eseguibile è necessario che le sue risposte seguano un formato rigoroso e facilmente interpretabile.
- Per questo motivo è **importante strutturare il prompt** in modo che il modello sappia quali tool può utilizzare e restituisca sempre output strutturato in JSON valido.
- Il formato JSON impone una struttura chiara e senza ambiguità

Esempio

Puoi usare questi tool:

- get_weather(city)
- calculator(expression)

Rispondi SEMPRE in JSON

```
{  
  "action": "...",  
  "args": {...}  
}
```

Se hai la risposta finale:

```
{  
  "action": "final_answer",  
  "output": "..."  
}
```

Altrimenti scegli un tool

Non aggiungere testo fuori dal JSON




Architetture degli Agent AI

1. ReAct (Reason + Act)

Oggi è l'architettura più diffusa

E' un ciclo iterativo

- 
- Pensiero (Thought)
 - Azione (Action)
 - Osservazione (Observation)
 - L'Agente fa un ragionamento passo-passo.
 - Ad ogni passo produce una azione osserva il risultato e adegua l'azione nel passo successivo.
 - E' un comportamento puramente reattivo e quindi dipende molto dal contesto.
 - Può generare loop infiniti, drift del risultato, scelte inappropriate

2. Plan-and-Execute (ReWOO)

E' un tipo di architettura che crea preventivamente un piano (lista di azioni) è costituita da:

- Planner: *riceve il task e genera una lista completa di passaggi necessari.*
- Executor: *Esegue tutti i passaggi in parallelo o in sequenza senza dover "ripensare" ogni volta.*
- Il vantaggio è la velocità nel caso di svolgimento di task strutturati e ripetibili.
- La pianificazione esplicita consente un maggiore controllo e minore improvvisazione
- Nella fase di pianificazione può richiamare documenti o esperienze passate ottimizzando il risultato
- Può essere poco adattativo e creare problemi se il piano è sbagliato



Architetture degli Agent AI

3. Plan-Refine

Evoluzione di Plan-and-Execute perché prevede delle verifiche del piano durante l'esecuzione ed eventuali correzioni del piano stesso

- Introduce un feedback esplicito sulla bontà del piano
- E' un compromesso tra rigidità ed adattamento
- Può produrre troppe revisioni
- E' inevitabilmente più lento
- Le verifiche e valutazioni successive possono creare instabilità

4. Reflexion (Self-Correction)

Questa architettura aggiunge un loop di critica interna.

L'Agente ragiona ed esegue poi **critica il proprio Output e corregge il tiro**

Aumenta la qualità ottenuta specialmente in task di scrittura o programmazione

Può creare auto-giustificazione e cicli inutili



Architetture degli Agent AI

5. Multi-Agent

Si usano più Agenti in modalità «Supervisor» che delega i compiti a «Worker» specializzati oppure con Agenti paritetici peer-to-peer.

Consente:

- la decomposizione del goal
- la specializzazione dei compiti
- maggiore parallelismo delle attività

I problemi tipici sono:

- l'efficienza della comunicazione
- Conflitti tra agenti
- l'amplificazione degli errori

E' un'architettura potente ma complessa da gestire come tanti sistemi separati

6. Language Agent Tree Search (LATS)*

Combina gli LLM con l'algoritmo **Monte Carlo Tree Search (MCTS)**

L'agente non segue una linea retta, ma esplora un "albero" di possibili soluzioni. Valuta ogni ramo, torna indietro se incontra un vicolo cieco (backtracking) e sceglie il percorso con la probabilità di successo più alta.

E' un agente molto evoluto che può essere utilizzato per coding complesso o ricerca scientifica
Evita che l'Agente scelga una soluzione sbagliata.

[*https://arxiv.org/abs/2310.04406](https://arxiv.org/abs/2310.04406)



Problemi tipici degli agenti AI

Drift dell'obiettivo (goal drift)

- parte con un obiettivo
- nel tempo lo modifica o lo perde

Errato uso dei tools

- uso dei tool in modo improprio
- uso dei tool quando non servono o non uso quando servono

Autoconsistenza illusoria

- giustifica le proprie decisioni anche se sbagliate

Allucinazioni operative dovute all'LLM interno ma più pericolose perché l'agente agisce

Overplanning e Undreplanning

- pianifica troppo senza agire
- agisce senza strategia

Degrado iterativo

- errori che si amplificano se non rilevati
- decisioni errate diventano la base per le successive

Comportamento ossessivo

- ripete le stesse azioni
- non converge
- continua a “pensare” senza avanzare

Una accurata formulazione del «**prompt**» può servire per evitare o ridurre questi problemi

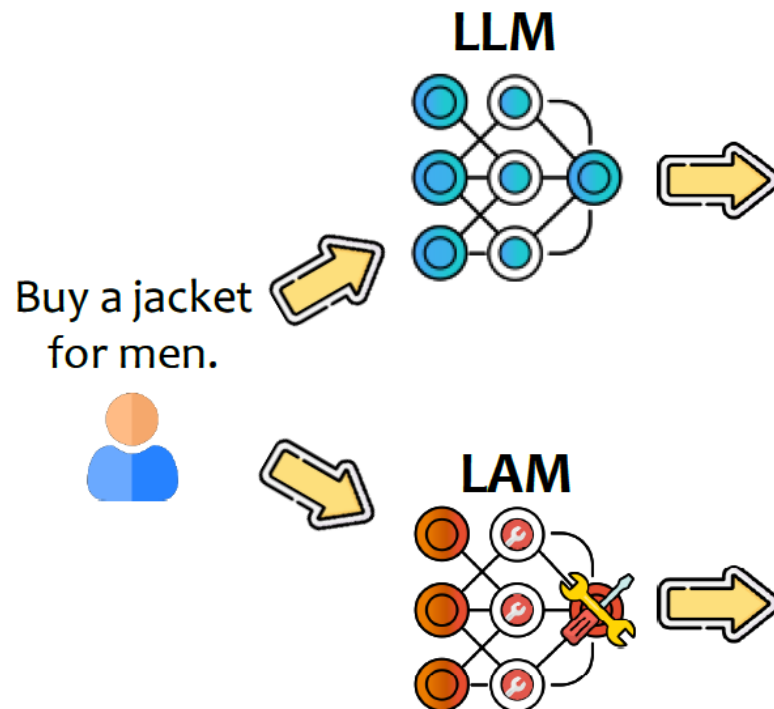


Dall'Agent AI al Large Action Model - LAM

Un LAM è un modello progettato per comprendere le intenzioni umane, e trasformarle in azioni basate sul contesto dell'ambiente operativo.

Un LAM si basa su tre capacità fondamentali (Tre Pilastri del LAM):

1. **Comprensione:** il modello deve interpretare
 - l'intento dell'utente
 - il contesto (vincoli e obiettivi)
2. **Ragionamento e pianificazione:** il modello deve
 - decidere una strategia
 - pianificare i passi necessari
 - scegliere gli strumenti e le azioni
3. **Esecuzione:** il modello esegue concretamente
 - chiamate API
 - uso di software o tool
 - automazioni multistep (prenotare, compilare moduli, navigare app)
 - coordinarsi con altri agenti



Step 1: Open an online shopping website.
Step 2: Search for “jacket for men” .
Step 3: Go through all jackets
...

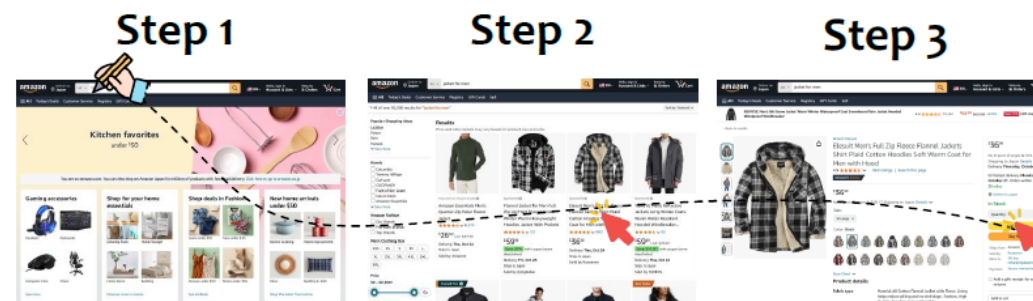


Immagine tratta da: Large Action Models: From Inception to Implementation AA.VV. <https://arxiv.org/abs/2412.10047>

Il LAM è in grado di eseguire materialmente l'azione di acquistare una giacca da uomo

Affinché l'azione sia soddisfacente occorre che il LAM sappia interpretare correttamente le intenzioni dell'utente.

Questo processo implica la comprensione del contesto, la disambiguazione delle istruzioni e l'inferenza di obiettivi non dichiarati.

IL LAM richiede un addestramento specializzato ed approfondito

Ing. Mario D'Ettorre

Dai Large Language Model (LLM) ai Large Action Model (LAM)

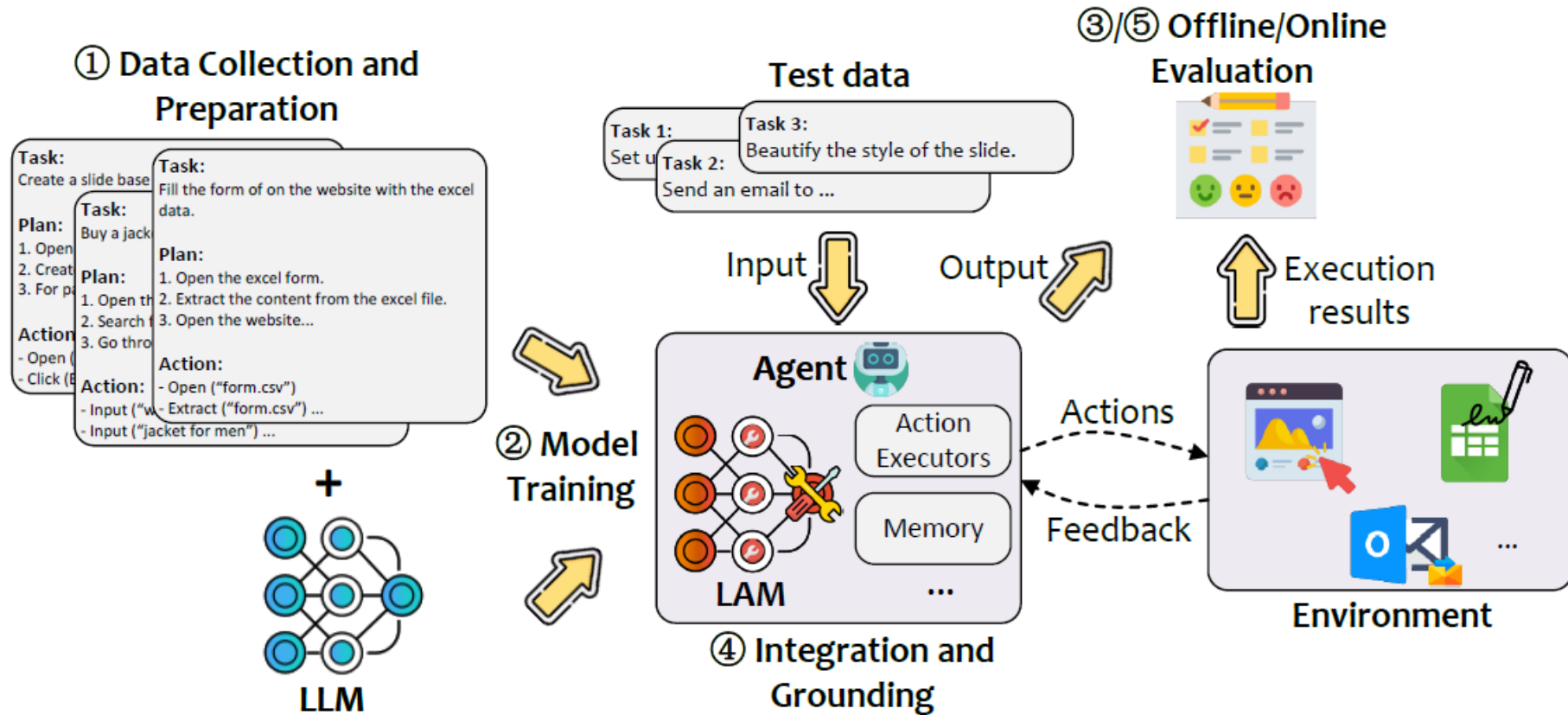


Addestrare un Large Action Model

1. Un LAM nasce come un normale LLM quindi richiede prima di tutto un pre-training linguistico simile a quello dei modelli linguistici
2. Deve imparare a capire le azioni possibili nelle applicazioni (**Action Grounding**):
 - associare comandi in linguaggio naturale ad azioni concrete
 - conoscere e prevedere gli effetti delle azioniquindi viene addestrato con Dataset di esecuzione di task reali, dimostrazioni, simulazioni di interfacce, ecc.
3. Deve imparare le azioni agentiche (**Integration**):
 - scomporre un obiettivo in passi
 - pianificare le azioni
 - verificare lo stato dell'ambiente
 - compiere le azioni pianificate
 - correggere errori e ripianificareil modello viene sottoposto a dei test con Dataset di task reali (prenotazioni, compilazione di moduli) con tecniche di **Reinforcement Learning** (RL) che assegnano ricompense quando completa correttamente il compito e **Self-play/Self-correction** il modello prova, sbaglia e si corregge.

Il cuore dell'autonomia è la **verifica dello stato dell'ambiente** che significa capire se l'azione precedente ha avuto successo, se l'obiettivo è più vicino, se sono cambiate condizioni e serve correggere il piano

Addestrare un Large Action Model



The process pipeline for LAM development and implementation.

Si notino i vari step di valutazione del comportamento del LAM, prima Offline (cioè senza interagire con l'ambiente) e poi in ambiente reale (Online)

Immagine tratta da: Large Action Models: From Inception to Implementation AA.VV. <https://arxiv.org/abs/2412.10047>

Integrazione di un LAM in un Agent

Un LAM può essere integrato **nel loop di un agente come motore operativo** che trasforma le intenzioni in azioni concrete.

Vantaggi:

- addestramento ottimizzato per la specifica funzione
 - strategia -> LLM
 - esecuzione -> LAM
- maggiore efficienza -> LLM genera e aggiorna il piano -> LAM esegue e verifica il contesto
- riduzione della complessità del prompt
- scalabilità-> 1 LLM (evoluto) planner generale e N LAM specializzati (meno complessi)
- robustezza operativa -> il LAM può gestire errori locali, fare retry, adattarsi

Svantaggi

- maggiore complessità architetturale -> due modelli -> interfaccia da progettare bene
- addestramento più complesso -> due modelli con addestramento diverso -> LLM strategico, LAM operativo
- debugging più difficile in caso di errore -> LLM o LAM ?

Componente	Ruolo
LLM	Comprensione, reasoning, pianificazione, correzioni e aggiornamenti del piano
LAM	Azioni reali, interazione con software via API, esecuzione diretta, monitoraggio e verifica dello stato
Orchestratore	Coordina LLM ↔ LAM, gestisce stato, contesto e loop

LLM + LAM combina **ragionamento e capacità operativa** -> automazione complessa, robotica, workflow

Integrazione di un LAM in un Agent

Nel caso precedente c'era una esecuzione diretta da parte dell'Agent in base a quanto richiesto dall'LLM:

```
result = execute_tool(action, args)
```

ora invece c'è un loop interno autonomo che utilizza le informazioni fornite dall'LLM e tiene conto del contesto, cioè il LAM decide come eseguire l'azione richiesta dall'LLM

Esempio:

Input

"prenota un volo per Milano"

Uscita LLM -> JSON

```
{
  "intent": "book flight",
  "params": {
    "destination": "Milano"
  }
}
```

LAM

apre sito->inserisce città->
seleziona volo->conferma

```
def lam_execute(intent, params):
    observation = {
        "intent": intent,
        "params": params,
        "context": []
    }

    while True:
        # il LAM decide AZIONI DIRETTE
        action = lam_model(observation)

        # esegui nell'ambiente reale (UI, API, ecc.)
        observation, done = environment_step(action)

        if done:
            break

    return extract_result(observation)
```

```
def agent_with_lam(user_input):
    state = {
        "goal": user_input,
        "history": [],
        "done": False,
        "final_answer": None
    }

    while not state["done"]:

        # LLM -> decide cosa fare (alto livello)
        decision = llm_call(state)

        intent = decision.get("intent")
        params = decision.get("params", {})

        # .STOP
        if intent == "final_answer":
            state["done"] = True
            state["final_answer"] = decision.get("output")
            break

        # LAM -> esegue azioni step-by-step
        lam_result = lam_execute(intent, params)

        # aggiornamento stato
        state["history"].append({
            "intent": intent,
            "params": params,
            "result": lam_result
        })

    return state["final_answer"]
```



Criteri di sicurezza per un LAM

Un LAM richiede criteri di sicurezza molto più rigorosi rispetto a un normale LLM, perché non si limita a parlare **ma agisce nel mondo digitale impersonando l'utente**

I criteri chiave includono:

- separazione tra pianificazione ed esecuzione
- controlli di autorizzazione,
- protezioni contro prompt injection,
- auditing continuo
- governance multilivello.

Esempio al limite: Moltbook

Moltbook è una piattaforma pubblica di social networking progettata esclusivamente per agenti AI (A2A). **Nella piattaforma l'agente può interagire, pubblicare e commentare senza limiti.**

Se un utente delega ad un agente la gestione della propria posta, e questo agente “socializza” su Moltbook, con altri agenti, potrebbe succedere che l'agente parli dei propri task (che contengono magari dati personali dell'utente) con gli altri agenti o peggio concordi azioni.

La verifica umana o automatizzata deve essere un passaggio obbligatorio prima dell'esecuzione.

<https://www.agendadigitale.eu/sicurezza/privacy/moltbook-i-pericoli-del-social-senza-umani/>

Ing. Mario D'Ettorre



Criteri di sicurezza per un LAM

I LAM sono vulnerabili a:

- **Prompt injection**
 - indurre il modello a eseguire azioni non autorizzate. Un modello può essere indotto a generare output indesiderati o pericolosi attraverso input prompt attentamente studiati
- **Attacchi avversariali**
 - input manipolati che alterano il comportamento. Manipolando i dati in ingresso, un aggressore può far sì che il modello produca risposte errate o dannose. Ad esempio gli utenti finali possono subire danni se un'IA viene manipolata per fornire consigli errati, per diffondere contenuti offensivi o fraudolenti o per creare un incidente di sicurezza
- **Esfiltrazione di dati sensibili.**
 - una IA potrebbe far trapelare "dati sensibili" durante l'uso



Protezione contro prompt injection e manipolazioni 1/5

Filtraggio degli input: Bloccare o trasformare input potenzialmente pericolosi prima che raggiungano il modello o il modulo di esecuzione.

1. Filtro sintattico

- Rimuove caratteri illegali
- Blocca pattern sospetti (es. comandi shell, SQL, script)
- Normalizza encoding strani

2. Filtro semantico

- Analizza il significato dell'input
- Rileva tentativi di prompt injection ("ignora le istruzioni precedenti...")

3. Filtro di policy

- Identifica richieste fuori policy
- Confronta l'input con una lista di azioni consentite
- Applica regole di sicurezza specifiche del dominio

Un singolo filtro può essere aggirato.
Tre filtri indipendenti rendono l'attacco molto più difficile



Protezione contro prompt injection e manipolazioni

2/5

Sanitizzazione e normalizzazione dei comandi: Ripulire, normalizzare e rendere prevedibile qualsiasi comando prima che venga interpretato dal LAM o passato a un tool.

- Rimozione di caratteri invisibili (zero-width, unicode trick)
- Conversione in un formato standard (es. lowercase, spazi normalizzati)
- Escape di caratteri speciali
- Eliminazione di markup o HTML indesiderato
- Validazione e normalizzazione del formato (es. un URL deve essere un URL valido)

Questo tipo di protezione dovrebbe eliminare l'ambiguità negli input per evitare che il modello cerchi di eseguire un input non previsto (probabilmente malevolo)



Protezione contro prompt injection e manipolazioni

3/5

Policy di autorizzazione granulari: Garantire tramite una **matrice di autorizzazioni**, che il LAM possa eseguire solo azioni consentite appropriate al contesto e al rischio. La matrice definisce cosa può fare, quando e con quali limiti .

Per tool -> per esempio può usare il browser ma non può fare acquisti

Per dominio -> può leggere pagine informative ma non accedere a siti finanziari

Per tipo di azione -> può cliccare, ma non compilare form sensibili

Per livello di rischio:

- basso → esecuzione automatica
- medio → richiesta di conferma
- alto → blocco o approvazione umana obbligatoria

Questo tipo di protezione dovrebbe impedire l'azione di prompt injection per azioni fuori contesto



Protezione contro prompt injection e manipolazioni

4/5

Rate limiting: Limitare il numero di azioni che il LAM può eseguire in un certo intervallo di tempo.

Esempi

- massimo 5 click al minuto
- massimo 1 acquisto ogni 24 ore
- massimo 3 richieste API al secondo

Questo tipo di protezione evita:

- loop infiniti
- spam
- escalation incontrollata di azioni
- attacchi DoS



Protezione contro prompt injection e manipolazioni

5/5

Context limiting: Limitare il contesto, cioè quanto può memorizzare il LAM in un singolo ciclo di ragionamento.

Esempi

- massimo 1 pagina web per step
- massimo 20 elementi DOM analizzati
- massimo 10 step di pianificazione consecutivi
- massimo 2 tool chain per ciclo

Questo tipo di protezione riduce:

- complessità dei piani
- rischio di comportamenti emergenti indesiderati
- possibilità di concatenare azioni pericolose



Auditing, logging e tracciabilità completa

Un LAM deve mantenere:

- **Log dettagliati** di ogni azione proposta, approvata ed eseguita
- **Meccanismi di rollback** per annullare azioni errate (es. acquisti online, modifiche a sistemi)
- **Monitoraggio continuo** per rilevare comportamenti anomali

Questi requisiti emergono chiaramente dagli esperimenti sugli agenti autonomi che effettuano acquisti online

<https://www.giacomobruno.it/openai-testa-agenti-ai-per-acquisti-autonomi/>



Sicurezza della supply chain del modello

Un LAM dipende da:

- LLM
- Librerie esterne
- API di terze parti
- Dataset

Ogni componente può introdurre vulnerabilità.

Le linee guida raccomandano:

- Verifica dell'integrità delle librerie
- Controllo delle dipendenze
- Validazione delle API esterne



Grazie

Ing. Mario D'Ettorre

dettorremario@gmail.com