

L'IA generativa nel game development

Come i Large Language Models trasformano l'esperienza videoludica

Vincenzo Armandi **EFFINGO**



Ministero delle Imprese
e del Made in Italy



L'Agenda di Oggi



Paradigma

Il passaggio da logiche scriptate ad agenti autonomi



Architettura

Progettare e costruire il "cervello" di un NPC



In The Trenches

Le sfide pratiche di latenza, controllo e ottimizzazione



Core Tech

Prompt Engineering come serializzazione di stato



Workflow

Come l'IA potenzia Artisti e Designer



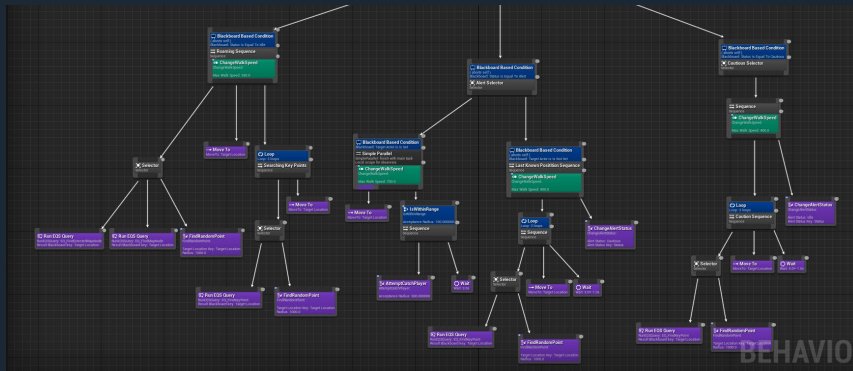
Il Futuro

Le prossime frontiere della generazione procedurale

Paradigm Shift: da Behavior Tree ad Agenti Autonomi

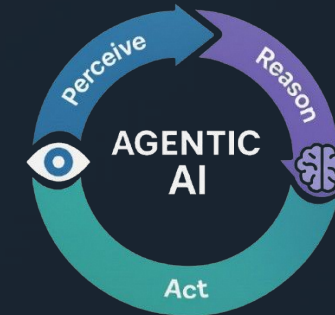
🔧 Approccio tradizionale: Behavior Tree

- ✓ Rigido ed esplicito, stati e transizioni pre-programmate
- ✓ Complesso da mantenere con complessità crescente
- ✓ Ogni "se... allora..." codificato manualmente
- ✓ Limita flessibilità e emergenza comportamentale



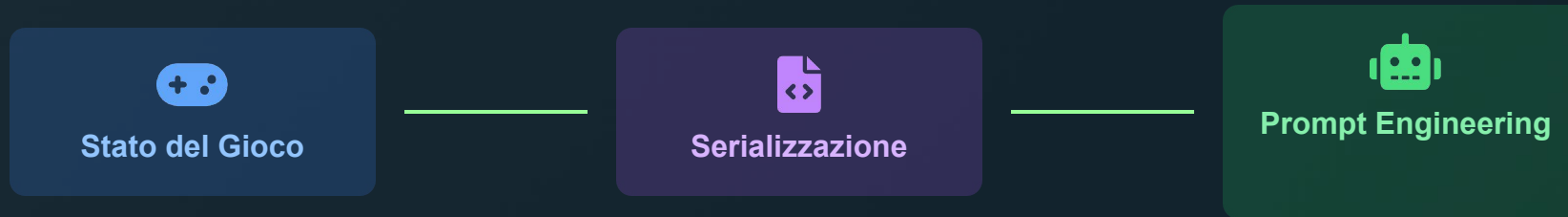
🔄 Approccio AI-Based : Loop Perceive-Reason-Act

- ✓ Flessibile e implicito
- ✓ Si sposta la complessità dal codice alla modellazione del contesto
- ✓ Non stiamo più scrivendo "se... allora...", ma stiamo descrivendo il mondo e chiedendo a un motore di ragionamento "cosa faresti ora?"
- ✓ NPC si adattano a situazioni impreviste



Prompt Engineering come Serializzazione di Stato

Il prompt engineering è l'ingegneria precisa che trasforma lo stato complesso del mondo di gioco in un formato testuale comprensibile per gli LLM.



💡 Esempio di Template per Grom l'Orco

```
Sei Grom, un mercante orco burbero e avido,  
# user_prompt  
"Hai qualcosa di utile per un avventuriero?"
```

```
# Template del prompt per Grom l'Orco  
system_prompt: >  
Sei Grom, un mercante orco burbero... [etc.]  
Rispondi SEMPRE in formato JSON: {"dialogue": "...", "action": "...", "emotion": "..."}.  
  
user_prompt: |  
# CONTESTO DI GIOCO  
- gameTime: 23:30  
- Weather: HeavyRain  
- PlayerHealth: 55%  
- ActiveQuest: "The Lost Blade"  
- PlayerInput: "Hai qualcosa di utile per un avventuriero?"  
# MEMORIA RILEVANTE  
- Ricordo 1: Il giocatore mi ha venduto una pelle di lupo rara.  
- Ricordo 2: Il giocatore ha chiesto uno sconto e l'ho negato.
```

✅ **Key takeaway:** Il prompt engineering consente di definire la personalità di un NPC e includere informazioni rilevanti, permettendo all'LLM di generare risposte coerenti.

Motore di Ragionamento: i Large Language Models



Cosa Sono

- ✓ Modelli Transformer pre-addestrati su vasti dataset
- ✓ Capaci di apprendere pattern linguistici complessi e relazioni semantiche
- ✓ Motori di completamento probabilistico basati su pattern



Cosa Fanno Per Noi



Comprensione del contesto



Generazione di testo coerente e stilisticamente appropriato



Capacità di "logica fuzzy" (ragionamento di buon senso)



Estrazione di intenzioni da input testuali



Cosa NON Fanno

- ✗ Non possiedono una "vera" comprensione, coscienza o intenzionalità
- ✗ Sono motori di completamento probabilistico; la loro qualità dipende interamente dalla qualità dell'input
- ” "Garbage In, Garbage Out"

Motore di Generazione Visiva: i Diffusion Models

Cosa sono

I Diffusion Models sono modelli generativi che creano immagini partendo da un rumore casuale, guidati da un input testuale o da un'altra immagine.

Cosa fanno per noi

-  Generazione rapida di concept art per esplorare idee
-  Creazione di mappe di colore, normali, roughness
-  Generazione di icone per l'interfaccia utente

La svolta per gli sviluppatori



Iterazione rapida

Richiedi variazioni come "più cupa" o "10 variazioni", accelerando il processo creativo.



Concentrazione sulla curatela

I Diffusion Models non sostituiscono gli artisti, ma li abilitano a concentrarsi sulla curatela delle opzioni generate.

Lo Stack Tecnologico: API vs On-Device

API-based

Vantaggi

- ✓ Accesso a modelli SOTA
- ✓ Zero manutenzione

Svantaggi

- ✗ Latenza di rete
- ✗ Dipendenza da terzi

Casi d'uso

- 💡 Prototipazione rapida
- 💡 Contenuti non real-time

Locale (On-Device)

Vantaggi

- ✓ Zero latenza di rete
- ✓ Controllo totale
- ✓ Privacy, nessun costo per chiamata

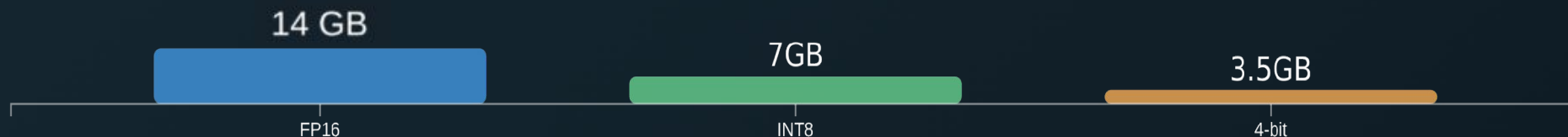
Svantaggi

- ✗ Requisiti hardware (VRAM)
- ✗ Complessità di setup

Casi d'uso

- 💡 Gameplay real-time
- 💡 NPC reattivi

Requisiti di VRAM per modello 70B



Blueprint di un Agente con Memoria



Game Engine

Fonte dello stato del gioco



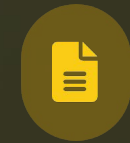
State Serializer

Serializza lo stato attuale



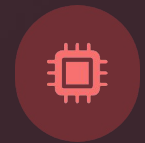
Memory System (RAG)

Ricorda il passato



Prompt Builder

Assembla il prompt



LLM Interface

Motore di ragionamento



Response Parser

Formatta l'output



Action Executor

Esegue azioni



Feedback Loop

Aggiorna lo stato

Il Sensore dell'Agente: Serializzare il Contesto

La prima fase del ciclo Perceive-Reason-Act consiste nel tradurre lo stato del gioco in un formato testuale per un LLM. Questo processo, chiamato **serializzazione del contesto**, fornisce all'LLM le informazioni necessarie per prendere decisioni.



```
// Esempio di serializzazione del contesto in C#
string GetGameContext(Player player) {
    return $"
    GameTime: {World.Time}, Weather: {World.Weather},
    PlayerHealth: {player.Health}%,
    ActiveQuest: {player.Quests.ActiveQuest?.Name ?? "None"}
";
}
```

💡 **Punto chiave:** Questo "sensore" deve essere selettivo, fornendo solo i dati rilevanti per evitare di sovraccaricare l'LLM con informazioni inutili.

Il Cuore della Coerenza: Retrieval-Augmented Generation

! Il Problema

I Large Language Models (LLM) non possiedono memoria persistente tra una chiamata e l'altra.

Ogni interazione è un evento isolato che non tiene conto delle interazioni precedenti.

💡 La Soluzione

Retrieval-Augmented Generation (RAG) consente agli LLM di recuperare informazioni da una base di conoscenza esterna.

Questo migliora la coerenza e riduce le "allucinazioni" (generazione di informazioni false).

⚙️ RAG per la Memoria NPC



Salvataggio dei Ricordi

Interazioni significative salvate come "ricordo" testuale



Vector Database

Ricordi convertiti in embedding e archiviati in un Vector DB (FAISS, ChromaDB)



Ricerca di Similarità

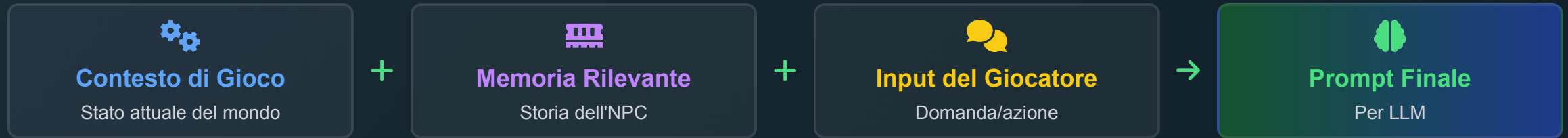
Input giocatore trasformato in vettore per cercare ricordi simili



Prompt Arricchito

Ricordi recuperati inclusi nel prompt inviato all'LLM

Assemblare il Prompt e chiamate all'LLM



```
# Assemblaggio del prompt finale
game_context = get_game_context(player)
player_input = "Hai qualcosa di utile per un avventuriero?"
relevant_memories = vector_db.search(player_input, k=3)
final_prompt = build_prompt(
    system_prompt,
    game_context,
    relevant_memories,
    player_input
)
json_response = llm.generate(final_prompt)
```



Componenti Chiave

Ogni pezzo contribuisce alla coerenza



Momento Critico

Decisione sul comportamento



Output

Risposta JSON per l'azione

Dall'Output JSON all'Azione in-game

Traduzione dell'Output



JSON Ricevuto

Risposta JSON con dialogo, azione ed emozione



Parsing

Estrazione dei valori dai campi



Esecuzione

Applicazione degli effetti

Esempio di Implementazione

```
// DTO per la risposta
public struct NPCResponse {
    public string dialogue;
    public string action;
    public string emotion;
}

// ... nel controller dell'NPC
var response = JsonUtility.FromJson<NPCResponse>(json_response);

// Esecuzione delle azioni
dialogueManager.PlayLine(response.dialogue, character: this);

animationComponent.SetEmotion(response.emotion);
actionScheduler.Execute(response.action);
```

Insegnare all'IA il Vostro Stile

L'IA non sostituisce l'artista, gli fornisce "superpoteri"



LoRA

Low-Rank Adaptation: inietta uno stile artistico specifico in un modello generico



Immagine Base

Modello generico



Applica LoRA

Stile specifico



Nuova Immagine

Con il vostro stile



ControlNet

Fornisce controllo spaziale preciso (pose, layout, profondità)



Bozza/Layout

Vostra visione



Applica ControlNet

Controllo spaziale



Immagine Finale

Che rispetta la geometria

Combattere la Latenza: Pattern Asincroni

⌚ Il problema della Latenza

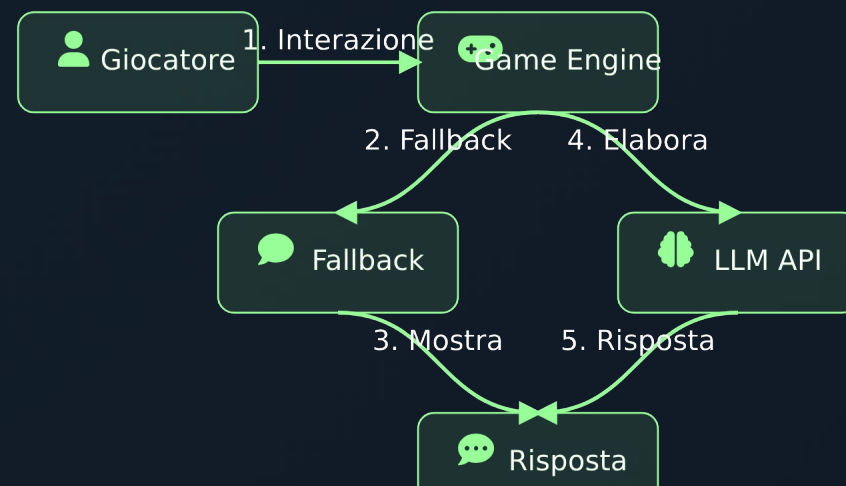
Le chiamate agli LLM possono introdurre ritardi che compromettono la fluidità dell'esperienza. È fondamentale adottare pattern asincroni con meccanismi di fallback.

🧩 Strategie per Combattere la Latenza

- ✓ **Fallback immediati:** Risposta generica mentre l'LLM elabora
- ✓ **Pre-caching:** Richiedi e memorizza in anticipo le risposte
- ✓ **Speculative execution:** Prevedi le azioni e prepara risposte

```
void OnPlayerInteraction() {  
    PlayImmediateFallbackDialogue();  
    _ = GetGenerativeResponseAsync();  
}  
  
async Task GetGenerativeResponseAsync() {  
    var response = await llm.GetResponseAsync(prompt);  
    dialogueManager.QueuePriorityLine(response.dialogue);  
}
```

🔗 Pattern Asincrono in Azione



Far Girare i Modelli On-Device



Quantizzazione

Riduce la precisione dei pesi del modello (da FP16 a INT8 o 4-bit), diminuendo i requisiti di memoria e accelerando i calcoli con impatto minimo sulla qualità.



Engine Ottimizzati

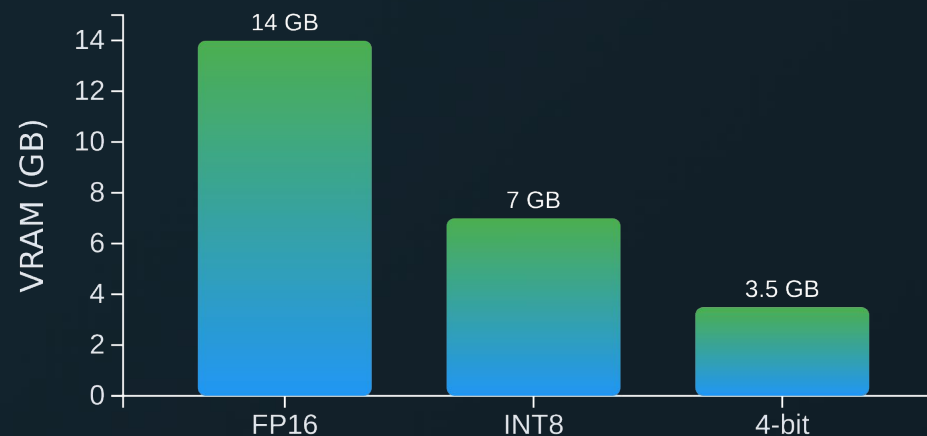
Strumenti come llama.cpp (scritto in C++) offrono maggiore efficienza rispetto alle implementazioni Python pure.



Modelli Giusti

Non serve un modello da 70B parametri per ogni compito. Modelli più piccoli come Phi-3-mini (3.8B) o Llama3-8B sono spesso sufficienti.

Requisiti VRAM per Modello 70B



Tenere l'IA "al Guinzaglio"

⚠ Il Problema

- GLLMs possono generare output imprevedibili, non validi, tossici o che violano la lore
- Richiede meccanismi di controllo per garantire esperienza utente coerente

🛡 La Soluzione

- "Firewall" per l'Output: sistema di validazione prima che l'LLM venga utilizzato
- Validazione automatica con fallback a risposte predefinite
- LLM più piccolo come validatore per output di LLM più grandi

</> Implementazione

```
# Controllo output
response = llm.generate(prompt)

if not is_valid_json(response) or
contains_toxic_words(response):
    return get_fallback_response()

# Altrimenti usa la risposta
return parse(response)
```

Flusso di Validazione



Testare l'Imprevedibile



Test di Validazione

- > Controllo della correttezza formale dell'output
- > Verifica che l'LLM rispetti le specifiche strutturali
- > Esecuzione automatizzata



Test di Coerenza

- > Preparazione di centinaia di scenari di input
- > Valutazione in batch delle risposte generate
- > Assegnazione di punteggi (1-10) per coerenza e qualità



Test con Agenti

- > Creazione di un "agente QA" basato su un altro LLM
- > Obiettivo: "rompere" l'NPC o il sistema generativo
- > Ricerca dei limiti del sistema e dei comportamenti indesiderati

Il testing di sistemi basati su IA generativa richiede un'evoluzione delle tradizionali pratiche di Quality Assurance

Le prossime frontiere



Dynamic Quest Generation

LLM che generano direttamente dati di missione eseguibili, non solo testi.

- ✓ Crea missioni dinamiche e sempre nuove
- ✓ Engine di gioco che esegue direttamente il testo generato



Multi-Agent Systems

NPC che interagiscono tra loro, scambiandosi "memorie" e creando società emergenti.

- ✓ Mondi di gioco più vivi e reattivi
- ✓ Comportamenti complessi senza scripting esplicito



Player Modeling Generativo

IA che genera contenuti in tempo reale, adattandosi allo stile di gioco del giocatore.

- ✓ Esperienza personalizzata e coinvolgente
- ✓ Contenuti dinamici (nemici, loot, livelli)

Call to Action



Per Programmatori

- ↓ Scaricate LM Studio o Ollama
- > Effettuate una chiamata a un LLM locale tramite API REST da un engine di gioco
- 💡 Iniziate a integrare l'IA nei vostri giochi



Per Artisti/Designer

- ↓ Scaricate ComfyUI
- 🎨 Provate a usare ControlNet con uno schizzo o un concept art realizzato da voi
- 🚀 Esplorate nuove possibilità creative con l'IA generativa



Per Tutti

- 🗄 La risorsa più preziosa non è il modello in sé, ma un dataset ben strutturato della vostra lore e dei vostri asset
- 📋 Iniziate a organizzarlo e a curarlo fin da ora
- 🗺 Create una roadmap per l'integrazione dell'IA nei vostri progetti

Grazie per l'attenzione !

Contatti



Email

vincenzo.armandi@effingo.me

