

UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria
Corso di Laurea in Ingegneria dell'Informazione per Videogame
e Realtà Virtuale

Agentic Choreographer: verso l'orchestrazione LLM per generazione di coreografie basate sul dataset AIST++

Agentic Choreographer: Toward LLM Orchestration for Generating Choreographies Based on the AIST++ Dataset

Candidato:
Leonardo Gottani

Relatore:
Dott. Lorenzo Stacchio

Correlatore:
Prof. Adriano Mancini

Anno Accademico 2025-2026



UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria
Corso di Laurea in Ingegneria dell'Informazione per Videogame
e Realtà Virtuale

Agentic Choreographer: verso l'orchestrazione LLM per generazione di coreografie basate sul dataset AIST++

Agentic Choreographer: Toward LLM Orchestration for Generating Choreographies Based on the AIST++ Dataset

Candidato:
Leonardo Gottani

Relatore:
Dott. Lorenzo Stacchio

Correlatore:
Prof. Adriano Mancini

Anno Accademico 2025-2026

Università Politecnica delle Marche
Facoltà di Ingegneria
Corso di Laurea in Ingegneria dell'Informazione per Videogame e Realtà Virtuale
Via Brecce Bianche – 60131 Ancona (AN), Italia

Alla mia famiglia, a Eleonora e a me stesso.

Sommario

La presente tesi descrive l'architettura e lo sviluppo di un sistema agentico in grado di trasformare una richiesta espressa dall'utente in linguaggio naturale, tramite prompt testuale, in una coreografia composta da mosse di danza appartenenti al dataset AIST++, successivamente predisposte per la visualizzazione in ambiente 3D mediante Unity. L'obiettivo principale è realizzare un processo automatizzato che permetta all'utente di descrivere la coreografia desiderata senza conoscere la nomenclatura dei file presenti nel dataset. L'architettura adotta un approccio modulare che integra un'interfaccia web, un backend Flask, un database vettoriale ChromaDB, un workflow agentico basato su LangGraph e sul paradigma ReAct e un Large Language Model. La generazione della coreografia è organizzata in una pipeline composta da acquisizione e interpretazione della richiesta, dal recupero delle mosse di danza rilevanti, dalla selezione e dall'ordinamento delle stesse in una coreografia e dalla validazione dell'output. In particolare, viene adottato un approccio Retrieval-Augmented Generation (RAG) per limitare il contesto fornito al modello linguistico e consentirgli di operare esclusivamente su un insieme ridotto di mosse recuperate dal database vettoriale. La preparazione dei dati prevede l'analisi della naming rule del dataset AIST++, attraverso la quale vengono estratti i principali metadati delle mosse, successivamente trasformati in rappresentazioni vettoriali mediante embedding semantici generati con il modello all-MiniLM-L6-v2. Questo processo permette di trasformare la struttura originaria del dataset in una base di metadati interrogabile dal sistema di retrieval. Il modello NVIDIA Nemotron 3 Ultra 550B viene impiegato per interpretare la richiesta e comporre la coreografia attraverso tre strumenti Python: Parsing, Search e Build. Per aumentare la robustezza del sistema, la componente agentica è affiancata da meccanismi di supervisione, validazione e fallback. Tali controlli permettono di verificare il rispetto del numero di mosse richiesto e la validità dei riferimenti alle mosse presenti nel database, consentendo la produzione degli output finali. Il sistema integra, pertanto, le capacità interpretative e selettive di un Large Language Model con procedure di retrieval semantico e controllo, realizzando una pipeline orientata alla generazione e alla successiva visualizzazione delle coreografie in ambiente 3D mediante plug-in dedicato per Unity. L'architettura presenta tuttavia alcuni limiti, tra cui la riduzione del margine decisionale del modello quando l'insieme delle mosse recuperate semanticamente è ristretto, il possibile intervento massiccio del fallback e l'assenza di parametri dedicati alla continuità cinematica tra movimenti consecutivi.

Indice

1 Introduzione **1**

2 Stato dell'arte **4**

2.1 AI per la generazione e pianificazione di coreografie 4
2.2 LLM e sistemi agentici per danza e performance 6
2.3 Sistemi agentici per ambienti tridimensionali 7
2.4 Confronto e posizionamento del sistema proposto 7

3 Metodologia **10**

3.1 Architettura del sistema 10
3.2 Panoramica dell'architettura 12
3.3 Sistema Retrieval-Augmented Generation 13
3.4 Dataset AIST++ 14
3.5 Indicizzazione del dataset AIST++ 15
3.5.1 Mappatura dei generi e dei livelli di difficoltà 15
3.5.2 Determinazione del BPM 16
3.5.3 Analisi della nomenclatura dei file 16
3.5.4 Costruzione dei metadati 16
3.5.5 Serializzazione dell'indice 17
3.5.6 Integrazione dell'indice nell'architettura complessiva 17
3.6 Database Vettoriale 17
3.6.1 Modello di Embedding 17
3.6.2 Arricchimento semantico 18
3.6.3 Costruzione dei Document LangChain 18
3.6.4 Generazione del Database Vettoriale Chroma 20
3.7 Livello di presentazione: interfaccia web 20
3.8 Backend Flask 21
3.9 Il livello agentico 22
3.9.1 Architettura agente ReAct 23
3.9.2 System Prompt 24
3.10 LangGraph e LangChain 26
3.10.1 Large Language Model 26
3.10.2 Organizzazione del Workflow 27
3.10.3 Il paradigma ReAct 28
3.10.4 Supervisione dello stato 28
3.10.5 ReAct supervisionato 29

3.10.6 Controlli di sicurezza 30
3.10.7 Flusso di esecuzione 31
3.10.8 Ripristino della Pipeline (fallback) 31
3.11 Tool di Parsing 32
3.11.1 Estrazione semantica e gestione errori 32
3.11.2 Formato della risposta dell'LLM 33
3.11.3 Normalizzazione e Controllo 34
3.11.4 Composizione del payload 35
3.12 Tool di Retrieval 36
3.12.1 Chroma come database vettoriale 36
3.12.2 Normalizzazione e costruzione della Query 37
3.12.3 Retrieval semantico 37
3.12.4 Filtraggio dei candidati 38
3.12.5 Fallback e rilassamento filtri 38
3.13 Tool di Build 39
3.13.1 Selezione delle sequenze tramite LLM 39
3.13.2 Validazione e completamento 39
3.13.3 Composizione coreografia e output finale 40
3.13.4 Fallback 41
3.13.5 Costruzione della playlist e tracciabilità dell'output 41
3.13.6 Generazione di `animation.txt` 43
3.14 Gestione dello stato del workflow 43
3.15 Flusso completo di esecuzione 44
3.16 Vantaggi dell'architettura 45
3.17 Limiti architetturali 46

4 Risultati **48**
4.1 Obiettivo della fase di test 48
4.2 Risultati dell'interpretazione delle richieste 48
4.3 Risultati complessivi del workflow 49
4.4 Risultati della ricerca nel dataset AIST++ 49
4.5 Rispetto del numero di mosse richiesto 49
4.6 Analisi del comportamento dell'agente e del meccanismo di fallback . 50
4.7 Analisi della composizione della coreografia 50
4.8 Esportazione dell'output e integrazione con il sistema 3D 51
4.9 Limiti della valutazione sperimentale 51

5 Discussioni **53**
5.1 Interpretazione dei risultati 53
5.2 Ruolo della supervisione nel workflow agentico 54
5.3 Efficacia del meccanismo di fallback 54
5.4 Retrieval semantico e gestione del contesto 55

5.5 Architettura ibrida tra componenti deterministiche e generative . . . 56
5.6 Qualità della coreografia e limiti interpretativi dei risultati 56
5.7 Implicazioni dei risultati per il sistema sviluppato 57

6 Conclusioni **58**
6.1 Conclusioni e contributi del lavoro 58
6.2 Limiti del lavoro 59
6.3 Sviluppi futuri 60

Elenco delle figure

3.1 Pipeline di esecuzione del sistema. 12
3.2 Flusso del processo di Retrieval-Augmented Generation. 13
3.3 Flusso semplificato della comunicazione tra client, backend Flask e workflow agentico. 21
3.4 Workflow dei tools 27
3.5 Ciclo iterativo ReAct (Thought–Action–Observation) dell'agente coreografo. 29
3.6 Fasi fondamentali della pipeline 30
3.7 Pipeline di estrazione, parsing e normalizzazione della risposta LLM. 34
3.8 Pipeline di retrieval vettoriale e filtraggio progressivo delle mosse candidate. 37
3.9 Pipeline sequenziale di validazione, controllo adiacenza e serializzazione degli output. 40

Elenco delle tabelle

3.1 Principali componenti software dell'architettura del sistema. 10
3.2 Mappatura dei codici dei generi di danza nel dataset AIST++. . . . 15
3.3 Struttura dei metadati salvati nell'indice JSON ed esempio di istanza. 17
3.4 Confronto tra il ruolo del Large Language Model (LLM) e dell'Agente AI nell'architettura. 23
3.5 Ripartizione dei ruoli e delle responsabilità tra i componenti del sistema. 27
3.6 Campi del payload generati dal tool di estrazione semantica. 35
3.7 Specifica dettagliata dei campi del payload JSON di output. 42
3.8 Rappresentazione matriciale della struttura del file `animation.txt`. 43

4.1 Sintesi dei risultati della fase di test 52

Capitolo 1

Introduzione

Negli ultimi anni si è affermata l'Agentic Artificial Intelligence (Agentic AI), che rappresenta un cambio di paradigma rispetto ai sistemi di Generative AI (GenAI). Tali sistemi possono infatti affrontare compiti complessi attraverso processi che comprendono elementi di ragionamento, pianificazione e riflessione, oltre alla capacità di interagire con l'ambiente circostante e utilizzare strumenti esterni[1]. A differenza di un sistema generativo tradizionale, generalmente orientato alla produzione di un output a partire da un input, un sistema agentico può organizzare il raggiungimento di un obiettivo attraverso una sequenza di operazioni, utilizzando le informazioni ottenute durante l'esecuzione per guidare le fasi successive[1]. L'impiego di questi sistemi risulta particolarmente interessante nei contesti in cui una richiesta iniziale, formulata liberamente dall'utente, deve essere trasformata in una sequenza strutturata di operazioni. Tra le caratteristiche che contraddistinguono l'Agentic AI assume un ruolo centrale anche il ragionamento multi-step, in quanto, insieme agli elementi introdotti precedentemente, consente di estendere le capacità dei modelli generativi oltre la sola produzione di contenuti, permettendo di utilizzare informazioni provenienti da fonti esterne e di delegare specifiche operazioni a componenti software specializzati[1]. All'interno di questo scenario si colloca il problema affrontato dalla presente tesi, che riguarda la generazione automatica di coreografie di danza a partire da una descrizione testuale fornita dall'utente. L'obiettivo è realizzare un sistema in grado di interpretare una richiesta espressa in linguaggio naturale, individuandone gli elementi rilevanti, e di trasformarla in una coreografia composta da sequenze di mosse di danza appartenenti al dataset AIST++. Il risultato prodotto dal sistema non consiste quindi esclusivamente in una risposta testuale, ma in una rappresentazione strutturata della coreografia, successivamente utilizzabile per la visualizzazione delle animazioni all'interno di un ambiente tridimensionale in Unity. Questa fase si inserisce in continuità con il precedente lavoro di Luca Felcher [2], che ha sviluppato il plug-in dedicato all'importazione e alla visualizzazione delle animazioni in Unity. La presente tesi ne estende l'utilizzo integrandolo all'interno di un processo automatico di generazione della coreografia a partire da una richiesta espressa in linguaggio naturale. La difficoltà principale risiede nella necessità di collegare componenti appartenenti a domini differenti, infatti, se da un lato il linguaggio naturale utilizzato dall'utente è caratterizzato da un'elevata libertà espressiva e non definisce

necessariamente in maniera esplicita tutti i parametri necessari alla costruzione della coreografia, dall'altro, il dataset AIST++ contiene un insieme strutturato di mosse di danza che devono essere individuate e selezionate secondo criteri coerenti con la richiesta. Diventa dunque necessario introdurre un livello intermedio capace di interpretare l'input, estrarre le informazioni utili, individuare le mosse pertinenti all'interno del dataset e organizzare i risultati in una struttura compatibile con il plug-in. Per affrontare tale problema è stata progettata un'architettura agentica basata su un *Large Language Model (LLM)*, integrato all'interno di un workflow supervisionato. L'agente non opera, inoltre, direttamente sul dataset completo, ma dispone di strumenti (*tools*) software specifici attraverso i quali vengono eseguite le operazioni necessarie alla generazione della coreografia. Il processo comprende l'interpretazione della richiesta, la normalizzazione e validazione delle informazioni estratte, la ricerca delle mosse rilevanti e la costruzione della coreografia finale. L'utilizzo di strumenti esterni consente di delegare le operazioni a componenti software dedicate, lasciando al modello linguistico il compito di interpretare la richiesta, valutare le informazioni disponibili e comporre la coreografia nel rispetto dei vincoli definiti dal sistema. L'impiego dei tools rappresenta infatti uno degli elementi caratterizzanti dei sistemi agentici, poiché permette di estendere le funzionalità del modello oltre la sola generazione testuale. Un ulteriore elemento dell'architettura è costituito dall'impiego di tecniche di *Retrieval-Augmented Generation* (RAG). Anziché fornire al modello l'intero contenuto del dataset, le informazioni relative alle mosse di danza vengono indicizzate e rese disponibili attraverso un database vettoriale. In questo modo è possibile recuperare, in funzione della richiesta dell'utente, un insieme limitato di mosse candidate da sottoporre successivamente al processo di selezione. Il paradigma RAG consente infatti di selezionare da una base di conoscenza esterna una porzione di informazioni rilevanti da includere nel contesto del modello, riducendo la quantità di dati da elaborare. Nel sistema sviluppato, il coordinamento delle operazioni è realizzato attraverso *LangGraph*, che avendo il ruolo di orchestratore, permette di rappresentare il processo come un workflow composto da stati e transizioni, mentre il paradigma Reasoning and Acting (*ReAct*)[3] viene utilizzato per combinare il ragionamento del modello con l'esecuzione degli strumenti disponibili. L'agente può quindi analizzare la richiesta, determinare quale operazione eseguire e utilizzare il risultato ottenuto come informazione per le fasi successive della pipeline. Questo approccio consente di realizzare un processo in cui la generazione della coreografia non è ottenuta mediante una singola chiamata al modello, ma attraverso una successione controllata di operazioni. A supporto di tale meccanismo è stato inoltre introdotto un sistema di supervisione dello stato dell'esecuzione, finalizzato a mantenere il controllo sul workflow e a garantire il rispetto dei vincoli definiti nella richiesta iniziale, con lo scopo di mitigare i fenomeni definiti di *agent drift*, ossia deviazioni rispetto al comportamento previsto dell'agente. La pipeline distingue le diverse fasi dell'elaborazione e mantiene le informazioni prodotte durante l'esecuzione, impedendo che il modello possa modificare arbitrariamente parametri già determinati o alterare i

vincoli specifici del sistema, come ad esempio il numero di mosse facenti parte della coreografia richiesto dall'utente. Questa soluzione permette di combinare la flessibilità interpretativa dell'LLM con vincoli implementati a livello software. L'intero processo è infine integrato all'interno di un'architettura software costituita da un'interfaccia web e da un backend sviluppato in Flask. L'interfaccia consente all'utente di inserire la descrizione della coreografia e di visualizzare il risultato dell'elaborazione, mentre il backend coordina l'esecuzione dell'agente e dei relativi tools. La coreografia generata viene quindi rappresentata mediante un file testuale strutturato che associa le mosse di danza ai relativi ballerini e ne permette la visualizzazione delle animazioni in Unity mediante un plug-in dedicato. La presente tesi descrive la progettazione e lo sviluppo di un sistema agentico basato su LLM per la costruzione automatica di coreografie a partire da richieste espresse in linguaggio naturale. Il sistema integra l'Agentic AI, il paradigma ReAct e tecniche di *Retrieval-Augmented Generation* (RAG) per interpretare la richiesta dell'utente, recuperare semanticamente movimenti preesistenti dal dataset AIST++ e comporli in una coreografia coerente con i vincoli definiti. A supporto del processo sono impiegati un workflow supervisionato per controllare le diverse fasi dell'esecuzione e strumenti software specializzati per le operazioni di interpretazione, ricerca, validazione e costruzione dell'output. L'architettura è inoltre integrata con un backend Flask, un'interfaccia web e un plug-in Unity sviluppato nel precedente lavoro di Felcher[2], permettendo di completare il flusso dalla descrizione testuale alla visualizzazione tridimensionale della coreografia. I principali contributi della tesi risiedono quindi nella combinazione di un agente LLM, di un meccanismo di retrieval semantico e di una supervisione esplicita del workflow, con l'obiettivo di automatizzare il processo di composizione della coreografia mantenendo separate le capacità decisionali del modello dalle operazioni software deterministiche. La tesi è così strutturata: il Capitolo 2 presenta lo stato dell'arte sull'impiego dell'intelligenza artificiale, degli LLM e dei sistemi agentici nella generazione e nella gestione di coreografie e performance tridimensionali; il Capitolo 3 descrive la metodologia adottata e le principali scelte progettuali del sistema illustrando l'architettura e l'implementazione delle diverse componenti; il Capitolo 4 presenta i risultati della fase sperimentale; il Capitolo 5 discute i risultati della precedente fase. Infine, il Capitolo 6 riassume i principali contributi e risultati della tesi, evidenzia i limiti del sistema e propone possibili sviluppi futuri.

Capitolo 2

Stato dell'arte

2.1 AI per la generazione e pianificazione di coreografie

L'impiego dell'intelligenza artificiale nella generazione di coreografie ha conosciuto una progressiva evoluzione, passando da sistemi basati prevalentemente sulla generazione automatica del movimento a soluzioni nelle quali componenti intelligenti possono assumere un ruolo di pianificazione, coordinamento e adattamento. Nell'ambito dei sistemi agentici, il problema della coreografia può essere affrontato non soltanto come generazione di una sequenza di movimenti, ma come un processo articolato nel quale più operazioni vengono coordinate in funzione di un obiettivo. Nonostante la rapida diffusione degli approcci agentici in numerosi domini applicativi, la loro applicazione specifica alla generazione di coreografie risulta ancora relativamente recente. La letteratura individua, infatti, differenti forme di agenticità, che spaziano da sistemi multi-agente basati su regole e meccanismi di auto-organizzazione fino a sistemi più recenti basati su Large Language Model, capaci di utilizzare il linguaggio naturale per analizzare, pianificare o modificare il movimento. Il confronto tra tali approcci permette di individuare le principali caratteristiche dei sistemi agentici per la generazione di coreografie e di mettere in evidenza aspetti talvolta solo marginalmente esplorati.

Li et al. [4] introducono il dataset AIST++ e un framework per la generazione di movimenti di danza in 3D, realistici e sincronizzati, a partire da un brano musicale sfruttando l'architettura *(Full-Attention-Cross-modal Transformer)* (FACT) che include un *deep cross-modal transformer block* allenato per prevedere N mosse future.

Siyao et al. [5] propongono un nuovo framework di *music-to-dance*, Bailando, che presenta due componenti chiave: una *choreographic memory*, che impara a sintetizzare unità di danza significative passando da una sequenza di pose 3D a un codebook quantizzato, e un modello *Generative Pre-trained Transformer* (GPT) per comporle in danze coerenti con la musica. A differenza di Li et al. [4], considerano il problema non solo come sintesi diretta del movimento, ma anche come composizione di unità di danza precedentemente apprese.

Tseng et al. [6] sviluppano *Editable Dance GEneration* (EDGE), un metodo per la generazione di coreografie modificabili basato su modelli di tipo diffusion, in grado di creare danze realistiche, fisicamente plausibili e coerenti con la musica di input,

offrendo funzionalità di modifica tra cui il condizionamento basato sulle singole articolazioni e l'interpolazione dei movimenti *in-betweening*. La generazione è quindi affiancata dalla possibilità di controllare e modificare il movimento prodotto.

Questi lavori evidenziano un'evoluzione dalla generazione diretta del movimento verso approcci nei quali assumono maggiore importanza la rappresentazione delle unità coreografiche e la loro composizione. Il problema può quindi essere interpretato non soltanto come generazione di nuove traiettorie di movimento, ma anche come organizzazione di elementi già disponibili.

Un'impostazione differente è presentata da De Filippo e Milano [7], che propongono un metodo basato su tecniche di Intelligenza Artificiale simbolica per la pianificazione di coreografie eseguite da robot. Il problema viene descritto mediante stati, sequenze di movimento, vincoli e requisiti relativi alla musica, che vengono utilizzati dal sistema per costruire una coreografia coerente con le condizioni specificate. L'approccio è orientato a ridurre il costo computazionale dei metodi basati sull'apprendimento e, al tempo stesso, a rendere più semplice l'inserimento di vincoli estetici e musicali. Questo lavoro assume particolare rilevanza perché introduce una concezione della coreografia come problema di pianificazione, anziché come semplice generazione di movimento. Le sequenze vengono infatti selezionate e organizzate sulla base di regole e vincoli espliciti, permettendo al sistema di adattarsi a differenti requisiti definiti dall'utente. L'approccio non utilizza, tuttavia, Large Language Model e non presenta il meccanismo di interazione dinamica tra modello linguistico e strumenti tipici dei moderni sistemi agentici. Una direzione complementare è sviluppata da Braccini et al. [8], che studiano l'impiego della *Swarm Intelligence* per la generazione automatica di coreografie eseguite da agenti artificiali. L'idea alla base del lavoro consiste nell'utilizzare interazioni locali e meccanismi decentralizzati per ottenere comportamenti collettivi emergenti. Il sistema non dipende quindi da un singolo agente centrale che determina l'intera coreografia, ma sfrutta la coordinazione tra più agenti e le proprietà di auto-organizzazione tipiche dei sistemi swarm. Gli agenti rappresentano entità autonome coinvolte direttamente nella performance e la coreografia emerge dalle loro interazioni, introducendo il concetto di generazione distribuita e autonoma della coreografia, caratterizzata da coordinamento, autonomia e gestione di vincoli globali. Braccini et al. [9] estendono il precedente lavoro proponendo una prima implementazione del modulo di *Feature Mapping*, nel quale una rete neurale apprende la trasformazione tra caratteristiche musicali e parametri utilizzati nella simulazione degli agenti. Gli autori introducono inoltre una componente di valutazione umana attraverso questionari, utilizzando i risultati ottenuti per analizzare e calibrare le funzioni di fitness e gli iperparametri impiegati nella generazione. Il passaggio verso un paradigma human-in-the-loop è particolarmente significativo perché evidenzia come la generazione automatica della danza possa essere concepita come un processo iterativo, nel quale il risultato prodotto dal sistema viene valutato e successivamente utilizzato per modificare il comportamento dell'algoritmo. Anche in questo caso, tuttavia, l'architettura non è basata su modelli linguistici. Nel complesso, gli approcci

descritti mostrano il passaggio dalla generazione diretta del movimento verso forme di composizione e pianificazione della coreografia. I modelli generativi permettono di sintetizzare e modificare sequenze di danza, mentre gli approcci basati sulla pianificazione e sulla Swarm Intelligence introducono meccanismi espliciti di organizzazione, coordinamento e autonomia.

2.2 LLM e sistemi agentici per danza e performance

L'evoluzione dei Large Language Model ha introdotto una nuova possibilità: utilizzare il linguaggio naturale non soltanto come input per un modello generativo, ma anche come interfaccia attraverso la quale un agente può analizzare un risultato, formulare istruzioni e coordinare più fasi dell'elaborazione. Nell'ambito della danza e della performance, questo approccio permette di utilizzare uno o più agenti non soltanto per generare nuovi movimenti, ma anche per comporre, analizzare e modificare le coreografie. Particolarmente significativo è il lavoro di Shang et al. [10], che affronta il problema del raffinamento del movimento di danza e nasce dall'osservazione che i sistemi text-to-motion tradizionali sono prevalentemente basati su interazioni single-turn e non affrontano in modo adeguato la valutazione qualitativa e la correzione del movimento. Gli autori propongono quindi il framework *DanceAgent*, basato su un *LLM agent* e su una sequenza iterativa di tre fasi: valutazione del movimento, generazione di istruzioni testuali e successivo raffinamento del movimento. Il sistema introduce quindi un ciclo nel quale il risultato di una fase viene utilizzato come informazione per quella successiva. L'agente non si limita a generare una risposta iniziale, ma analizza il risultato disponibile, individua aspetti da correggere e produce istruzioni che guidano una nuova elaborazione. La componente agentica risiede pertanto nella capacità di mantenere una storia delle interazioni e di utilizzare tale informazione per coordinare il processo iterativo. Un ulteriore sviluppo è rappresentato dal lavoro di Yu e Zhang [11] che considera la generazione della performance come un processo multi-agente nel quale differenti componenti cooperano per analizzare il feedback del pubblico e adattare dinamicamente musica, coreografia e controllo della performance. L'architettura considera quindi la performance come un sistema dinamico nel quale le decisioni relative alla musica e alla danza vengono coordinate con le informazioni provenienti dall'ambiente. L'obiettivo non è soltanto generare una coreografia iniziale, ma consentire una forma di adattamento in tempo reale in funzione dei dati osservati, introducendo una caratteristica tipica dei sistemi agentici: la capacità di modificare il comportamento sulla base di nuove informazioni disponibili durante l'esecuzione. I lavori di Shang et al. [10] e Yu e Zhang [11] si differenziano dal sistema sviluppato nella presente tesi. DanceAgent [10] rappresenta, infatti, un'evoluzione significativa rispetto ai sistemi di generazione diretta del movimento e il suo obiettivo principale consiste nel valutare e raffinare una performance esistente. Non viene affrontato, tuttavia, il problema della trasformazione di una richiesta testuale da parte dell'utente in una coreografia costruita mediante la selezione di

mosse di danza appartenenti a una base di conoscenza esterna. Il lavoro di Yu e Zhang [11], invece, adotta un'architettura multi-agente e l'input principale è costituito dal feedback del pubblico e dalle informazioni relative alla performance mentre il sistema qui sviluppato è orientato a una richiesta testuale formulata direttamente dall'utente e utilizza un singolo agente LLM dotato di tools specializzati.

2.3 Sistemi agentici per ambienti tridimensionali

Utile per comprendere l'evoluzione degli agenti verso applicazioni in ambienti tridimensionali, è il lavoro di He et al. [12]. Gli autori sviluppano un sistema, *Cutscene Agent*, che utilizza un'architettura basata su LLM agent per trasformare descrizioni narrative in sequenze cinematografiche tridimensionali. Un agente direttore coordina differenti agenti specializzati, dedicati ad animazione, cinematografia e sound design, mentre un toolkit basato su Model Context Protocol (MCP) permette agli agenti di interagire bidirezionalmente con il game engine e di osservare lo stato della scena durante l'esecuzione. Sebbene Cutscene Agent non sia specificamente dedicato alla generazione di coreografie di danza, il lavoro è particolarmente rilevante per il problema affrontato nella presente tesi, poiché dimostra come un LLM agent possa coordinare strumenti esterni e componenti di animazione all'interno di un ambiente 3D. Il sistema introduce inoltre un ciclo chiuso nel quale l'agente osserva lo stato dell'ambiente e utilizza tale informazione per guidare le operazioni successive. Cutscene Agent opera principalmente sul processo di creazione di una scena cinematografica completa, utilizzando più agenti specializzati. L'architettura sviluppata, invece, si concentra sulla costruzione della sequenza coreografica mediante un singolo agente e strumenti software dedicati.

2.4 Confronto e posizionamento del sistema proposto

I lavori analizzati mostrano come l'impiego dell'intelligenza artificiale nella generazione e nella gestione delle coreografie si sia progressivamente esteso dalla sintesi diretta del movimento verso forme di pianificazione, coordinamento e adattamento della performance. Pur presentando obiettivi e architetture differenti, gli approcci considerati possono essere ricondotti a quattro principali filoni: la generazione diretta del movimento, la pianificazione e la coordinazione mediante sistemi simbolici o swarm, il raffinamento e l'adattamento attraverso LLM e sistemi multi-agente e, infine, l'orchestrazione di strumenti software all'interno di ambienti tridimensionali. Il primo filone comprende i sistemi orientati alla generazione diretta del movimento, come quelli proposti da Li et al. [4], Siyao et al. [5], e Tseng et al. [6]. In questi approcci il problema principale consiste nella sintesi di nuove sequenze di movimento, eventualmente condizionate dalla musica o da specifici vincoli. L'introduzione di rappresentazioni intermedie e di meccanismi di modifica, come nel caso di Bailando ed EDGE, mostra tuttavia come il problema si sia progressivamente spostato dalla

sola generazione di traiettorie verso la possibilità di organizzare e controllare unità di movimento. Il secondo filone considera invece la coreografia come un problema di pianificazione e coordinazione. De Filippo e Milano [7] utilizzano tecniche di Intelligenza Artificiale simbolica per organizzare sequenze di movimento sulla base di stati, vincoli e requisiti musicali, mentre Braccini et al. [8] adottano un paradigma di Swarm Intelligence nel quale il comportamento collettivo emerge dall'interazione decentralizzata tra più agenti. In questi sistemi l'agenticità non è necessariamente legata alla generazione di nuove traiettorie, ma alla capacità di organizzare e coordinare entità e azioni secondo determinate condizioni. Il terzo filone è rappresentato dai sistemi basati su LLM agent e architetture multi-agente, nei quali il linguaggio naturale assume un ruolo nel processo decisionale. DanceAgent utilizza un agente linguistico per valutare e raffinare iterativamente una performance, mentre il lavoro di Yu e Zhang [11] estende il paradigma a un'architettura multi-agente in grado di adattare dinamicamente diversi aspetti della performance sulla base del feedback disponibile. In questo caso l'agente viene utilizzato principalmente come componente decisionale e di coordinamento, capace di interpretare informazioni e determinare le operazioni successive, piuttosto che come semplice generatore di movimento. Infine, il quarto filone, con il lavoro di He et al. [12] mostra, attraverso *Cutscene Agent*, come gli agenti LLM possano essere impiegati per orchestrare strumenti e componenti specializzati all'interno di ambienti tridimensionali, trasformando una descrizione espressa in linguaggio naturale in una sequenza cinematografica tridimensionale. Sebbene il problema affrontato non sia specificamente quello della composizione coreografica, questo lavoro evidenzia la possibilità di utilizzare un LLM come livello di coordinamento tra l'intenzione dell'utente, gli strumenti disponibili e lo stato di un ambiente 3D. Il confronto tra questi quattro filoni evidenzia quindi differenti livelli ai quali può essere introdotta l'agenticità. Nei sistemi di generazione diretta essa interviene principalmente sul movimento prodotto, negli approcci di pianificazione e Swarm Intelligence riguarda la coordinazione di azioni ed entità autonome, nei sistemi basati su LLM viene utilizzata per interpretare informazioni, valutare risultati e determinare le operazioni successive, negli ambienti tridimensionali, infine, può assumere il ruolo di coordinatore tra un obiettivo espresso in linguaggio naturale e un insieme di strumenti software. Il sistema sviluppato nella presente tesi utilizza l'agenticità principalmente a livello di processo. L'obiettivo non è generare direttamente nuove traiettorie articolari né controllare individualmente i movimenti dei ballerini, ma trasformare una richiesta espressa dall'utente in linguaggio naturale in una coreografia composta da movimenti già presenti nel dataset AIST++. L'agente LLM assume quindi il ruolo di coordinatore delle diverse operazioni necessarie alla costruzione della coreografia, mentre la ricerca dei movimenti e le successive operazioni di elaborazione vengono affidate a strumenti specializzati. L'architettura combina in particolare tre aspetti: interpretazione della richiesta in linguaggio naturale, retrieval semantico dei movimenti e composizione della coreografia. La richiesta dell'utente viene prima interpretata e strutturata, successivamente vengono

recuperate dal database vettoriale le sequenze maggiormente pertinenti e, infine, le mosse selezionate vengono organizzate per costruire la coreografia richiesta. A differenza dei sistemi che utilizzano l'LLM direttamente per produrre il contenuto del movimento, il modello opera quindi su un insieme di elementi preesistenti recuperati da una base di conoscenza. Un ulteriore elemento distintivo riguarda la separazione tra decisione e operazione: L'LLM è responsabile dell'interpretazione della richiesta e del coordinamento del processo, mentre le operazioni di retrieval, validazione e produzione dell'output sono realizzate attraverso componenti software specializzati. Il workflow è inoltre sottoposto a una supervisione dello stato che vincola l'esecuzione alle diverse fasi previste. Rispetto ai lavori precedentemente analizzati, il sistema della presente tesi non introduce quindi un nuovo metodo di generazione del movimento, non considera gli agenti come danzatori autonomi e non utilizza un'architettura multi-agente per adattare la performance. Il contributo proposto consiste nell'impiego di un agente LLM supervisionato come livello di coordinamento tra linguaggio naturale, base di conoscenza e strumenti software, con l'obiettivo di automatizzare il passaggio da una descrizione testuale a una coreografia composta da movimenti del dataset AIST++ e predisposta per la successiva visualizzazione in ambiente tridimensionale.

Capitolo 3

Metodologia

3.1 Architettura del sistema

Il sistema sviluppato ha l'obiettivo di trasformare una richiesta espressa dall'utente in linguaggio naturale tramite prompt testuale, in una coreografia composta da mosse di danza, appartenenti a diversi generi, recuperate dal dataset AIST++ e successivamente visualizzate all'interno del game engine Unity tramite il plug-in dedicato. L'architettura integra diversi componenti software, ciascuno con una responsabilità specifica, riepilogati nella Tabella 3.1. Il sistema è stato suddiviso in più componenti per la separazione delle responsabilità e per facilitarne lo sviluppo: il retrieval delle mosse di danza dal database vettoriale, l'interpretazione della richiesta dell'utente, la selezione delle mosse e la costruzione dell'output hanno richiesto tools e controlli dedicati. L'architettura non si basa esclusivamente sulla capacità generativa dell'LLM; questo, infatti, è soltanto uno dei componenti del sistema e viene utilizzato all'interno di un processo più ampio nel quale sono presenti dataset esterni, strumenti di ricerca, vincoli applicativi, procedure di validazione e in particolare un workflow agentico basato sul framework open-source LangGraph.

Tabella 3.1: Principali componenti software dell'architettura del sistema.

Componente	**Implementazione**	**Responsabilità**
Interfaccia web	*HyperText Markup Language* (*HTML*) + JavaScript (*JS*)	Acquisizione della richiesta dell'utente in linguaggio naturale e presentazione dei risultati dell'elaborazione.
Backend	Flask	Gestione delle richieste provenienti dall'interfaccia web e coordinamento dell'esecuzione della pipeline.
Conversione del dataset	Script Python	Conversione dei dati e dei metadati del dataset AIST++ in un formato strutturato *JavaScript Object Notation* (`JSON`) utilizzabile dal sistema.

Continua nella pagina successiva

Componente	**Implementazione**	**Responsabilità**
Indicizzazione del dataset	ChromaDB	Indicizzazione delle mosse AIST++ mediante rappresentazioni vettoriali e memorizzazione nel database vettoriale per consentire il recupero semantico.
Workflow agentico	LangGraph e ReAct	Orchestrazione delle diverse fasi di elaborazione mediante un agente basato sul paradigma ReAct e con supervisione dello stato del workflow.
Retrieval-Augmented Generation	ChromaDB e LLM	Recupero delle mosse semanticamente più pertinenti rispetto alla richiesta dell'utente e fornitura dei candidati al modello linguistico.
Large Language Model	Nemotron 3 Ultra 550B	Interpretazione della richiesta, valutazione dei candidati recuperati e selezione delle mosse da utilizzare nella composizione della coreografia.
Validazione e fallback	Script Python	Verifica dei dati prodotti dal modello, applicazione dei vincoli deterministici e gestione delle condizioni di errore mediante meccanismi di fallback.
Generazione dell'output	Script Python	Generazione dei file contenenti la coreografia generata, successivamente utilizzati per la visualizzazione delle animazioni in Unity mediante il plug-in dedicato.

3.2 Panoramica dell'architettura

Il funzionamento complessivo del sistema può essere descritto prendendo come riferimento la pipeline rappresentata nella Figura 3.1.

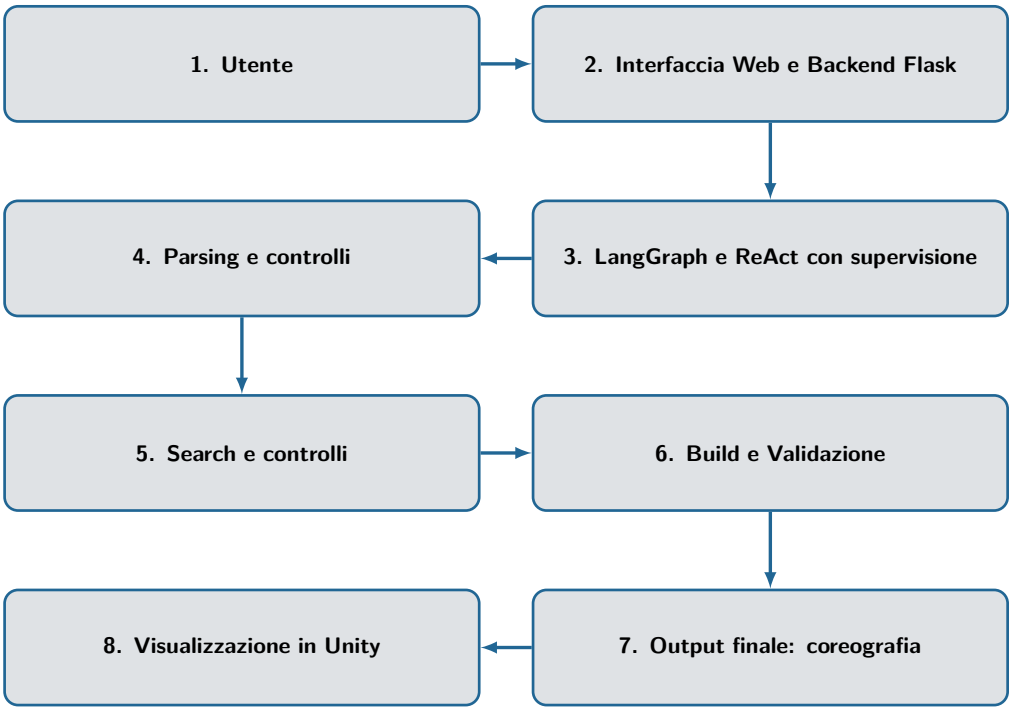


Figura 3.1: Pipeline di esecuzione del sistema.

Si possono dunque individuare cinque fasi principali del flusso di esecuzione:

1. **Acquisizione della richiesta dell'utente:** l'utente descrive la coreografia desiderata tramite prompt testuale attraverso l'apposita interfaccia.
2. **Interpretazione e strutturazione del prompt:** la richiesta viene interpretata e strutturata generando una query in formato `.json`
3. **Recupero delle mosse candidate dal database vettoriale:** la query viene utilizzata per la ricerca nel database vettoriale delle mosse con la maggiore similarità semantica
4. **Selezione e validazione delle mosse di danza facenti parte della coreografia:** il modello linguistico compone la coreografia selezionando tra le mosse ricevute dal database vettoriale.
5. **Esportazione e visualizzazione della coreografia in ambiente 3D:** viene generato il file `animation.txt` utilizzato dal plug-in per la visualizzazione delle animazioni in Unity

Ogni fase produce un output che viene utilizzato dallo step successivo della pipeline.

3.3 Sistema Retrieval-Augmented Generation

Il sistema adotta un approccio di Retrieval-Augmented Generation (RAG), l'LLM riceve informazioni recuperate da una base di dati esterna prima di generare l'output. La motivazione principale dell'utilizzo del RAG è evitare di fornire al modello linguistico l'intero database delle mosse di danza disponibili. Il dataset AIST++ effettivamente indicizzato dal progetto non rappresenta l'intero dataset, le mosse indicizzate sono comunque un numero elevato, circa quattrocento, e fornire tutte queste informazioni per ogni richiesta comporta un aumento significativo della dimensione del contesto, dei tempi di elaborazione e delle risorse necessarie. Il sistema utilizza quindi un livello di retrieval per individuare le mosse più pertinenti alla richiesta dell'utente. Il modello linguistico non riceve l'intero database, ma solamente un insieme ridotto di candidati sui quali può operare per costruire la coreografia. Il processo è costituito da due componenti principali: un modulo di retrieval, che ricerca nel database le mosse semanticamente pertinenti alla richiesta iniziale, e un modello generativo, che analizza i candidati recuperati e li utilizza per costruire la coreografia. Questa architettura permette quindi di mantenere il contesto fornito all'LLM entro dimensioni gestibili, evitando di trasferire al modello l'intero database ad ogni richiesta e concentrando la sua elaborazione solamente sulle mosse maggiormente pertinenti.

Il processo può essere rappresentato come da Figura 3.2.

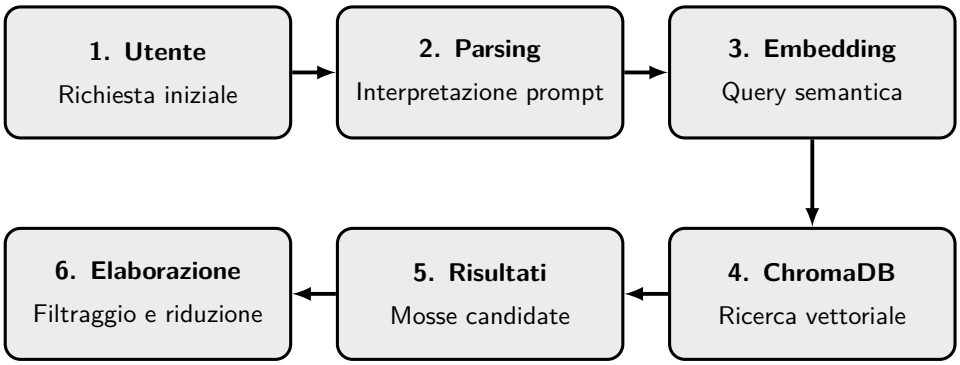


Figura 3.2: Flusso del processo di Retrieval-Augmented Generation.

3.4 Dataset AIST++

L'AIST Dance Video Database (AIST Dance DB) rappresenta il primo benchmark condiviso su larga scala dedicato alla street dance, concepito per promuovere la ricerca accademica e lo sviluppo di modelli avanzati nell'ambito del Dance Information Processing. Caratterizzato dall'inclusione di sequenze video ad alta risoluzione accompagnate da tracce musicali originali e prive di vincoli di copyright, il dataset è stato progettato per stimolare la formulazione di nuovi task computazionali in ambito scientifico, tra cui: classificazione del genere di movimento, identificazione del ballerino e stima della tecnica e della complessità [13].

Il dataset AIST++ raccoglie un'ampia collezione di sequenze di movimento memorizzate in vari formati digitali (ad esempio .pkl). La caratteristica strutturale fondamentale del dataset risiede nella sua naming rule (convenzione di denominazione dei file), la quale codifica in modo rigoroso i metadati essenziali della mossa direttamente all'interno del nome del file. Ciascuna sequenza segue un formato di denominazione standard espresso tramite la seguente struttura:

Tale convenzione viene decodificata per estrarre i seguenti metadati:

- **Codice di genere (`genre_code`):** identifica lo stile coreografico della mossa tra i 10 generi previsti dal dataset.
- **Codice di situazione (`situation_code`):** utilizzato dal sistema per determinare il livello di complessità o difficoltà della sequenza (rispettivamente 'facile' e 'difficile').
- **Identificativo del ballerino (`dancer_id`):** distingue l'esecutore della mossa.
- **Codice musicale (`music_code`):** associa il brano audio d'accompagnamento e permette al sistema di ricavare la frequenza di tempo espressa in *Beats Per Minute* (BPM).

L'adozione di questa naming rule permette al sistema di eseguire il parsing dei file in fase di preelaborazione ed estrarre tutti i metadati necessari.

3.5 Indicizzazione del dataset AIST++

Per consentire all'agente di effettuare la ricerca delle mosse di danza, il progetto prevede una fase preliminare di indicizzazione del dataset AIST++. Questa fase è implementata nel modulo `aist_index.py`, il cui compito è analizzare i file presenti nella directory del dataset, estrarre le informazioni codificate nei relativi nomi e costruire un indice in formato JSON contenente i metadati associati a ciascuna mossa.

3.5.1 Mappatura dei generi e dei livelli di difficoltà

Lo script definisce una struttura `genre_map` che associa i codici dei generi di danza presenti nel dataset alle relative descrizioni testuali. Per ciascun genere viene inoltre definito un intervallo di BPM associato. Sono presenti i seguenti codici, riepilogati in Tabella 3.2:

Tabella 3.2: Mappatura dei codici dei generi di danza nel dataset AIST++.

Codice	**Genere di Danza**
gBR	Breaking
gPO	Popping
gLO	Locking
gMH	Middle Hip-hop
gLH	LA style Hip-hop
gHO	House
gWA	Waacking
gKR	Krump
gJB	Ballet Jazz
gJS	Street Jazz

`KEYWORD_TO_GENRES` stabilisce una corrispondenza tra termini testuali e codici di genere, comprende sia termini che identificano direttamente uno stile, sia descrizioni relative alle caratteristiche della coreografia. Alcuni termini talvolta possono essere associati a più generi. La gestione della complessità viene invece realizzata attraverso `COMPLEXITY_MAP`, che associa i codici sBM (Basic Movement) e sFM (Freestyle Movement) rispettivamente alle etichette facile e difficile. Questa è stata una scelta adottata dal sistema per semplificare la distinzione tra mosse che potrebbero essere considerate di semplice esecuzione (sBM) e mosse che potrebbero essere considerate più complesse (sFM). Queste strutture traducono la naming rule dei file utilizzata dal dataset AIST++ in una rappresentazione testuale che può essere utilizzata dal sistema.

3.5.2 Determinazione del BPM

Un ulteriore elemento estratto durante l'indicizzazione è il BPM della sequenza. La funzione `get_exact_bpm()` riceve il codice del genere e il codice musicale e ricava dal secondo il relativo indice numerico attraverso un'espressione regolare. Nel caso in cui il codice non possa essere interpretato correttamente, viene utilizzato il valore predefinito di 100 BPM. La funzione utilizza due sequenze di valori BPM differenti. Per il genere House (gHO) vengono considerati i valori [110, 115, 120, 125, 130, 135], mentre per gli altri generi vengono utilizzati [80, 90, 100, 110, 120, 130]. L'indice ottenuto dal codice musicale viene utilizzato per selezionare il valore corrispondente; qualora l'indice superi la lunghezza della lista, viene utilizzato l'ultimo valore disponibile. Il BPM viene quindi memorizzato direttamente nell'indice, rendendo disponibile tale informazione senza doverla ricostruire durante le successive operazioni di ricerca.

3.5.3 Analisi della nomenclatura dei file

La funzione `build_index()` riceve il percorso della directory principale del dataset, lo converte in un percorso assoluto e inizializza una struttura index vuota destinata a contenere i metadati estratti. Per individuare i file che rispettano la convenzione di denominazione prevista dal dataset viene definita un'espressione regolare (*regex*). L'espressione permette di identificare all'interno del nome del file diversi componenti strutturali: il codice del genere, il codice di situazione, l'identificativo del ballerino, il codice musicale e l'estensione del file. La funzione attraversa ricorsivamente la directory del dataset tramite `rglob(*)` e considera esclusivamente i file con estensione .pkl, .bvh o .npy. I file che non rispettano la convenzione prevista vengono ignorati. Per ogni file valido vengono quindi estratti i diversi componenti attraverso i gruppi della regular expression. In particolare, vengono identificati `genre_code`, `situation_code`, `dancer_id` e `music_code`.

3.5.4 Costruzione dei metadati

Una volta estratte le informazioni dal nome del file, `build_index()` costruisce la relativa voce dell'indice. Per il genere viene recuperata la descrizione testuale attraverso `genre_map`; contemporaneamente viene calcolato il BPM mediante `get_exact_bpm()`. Per ogni sequenza viene quindi generato un record, riepilogati in Tabella 3.3.

Tabella 3.3: Struttura dei metadati salvati nell'indice JSON ed esempio di istanza.

Campo (Chiave)	**Descrizione**	**Valore d'esempio**
`id`	Identificativo univoco della sequenza	`gBR_sBM_cAll_d04_mBR0_ch01`
`path`	Percorso del file nel dataset	`motions\gBR_sBM_cAll_d04_mBR0_ch01.pkl`
`extension`	Estensione del file	`.pkl`
`genre_code`	Codice del genere di danza	`gBR`
`genre_name`	Nome descrittivo del genere	`Breaking`
`situation_code`	Codice della situazione	`sBM`
`music_code`	Codice della traccia musicale	`mBR0`
`bpm`	Frequenza di tempo in BPM	`80`
`difficulty`	Livello di difficoltà	`facile`
`dancer_id`	Identificativo del ballerino	`d04`

La funzione `_relative_or_absolute()` tenta di calcolare il percorso relativo e, nel caso in cui questo non sia possibile, restituisce il percorso disponibile.

3.5.5 Serializzazione dell'indice

L'indice viene serializzato in un file JSON mediante `json.dump()`, con codifica UTF-8, preservando i caratteri non ASCII e applicando una formattazione indentata per migliorarne la leggibilità. Al termine viene stampato il numero di file indicizzati e il percorso del file generato. Il file JSON costituisce quindi una rappresentazione strutturata e persistente dei metadati del dataset AIST++.

3.5.6 Integrazione dell'indice nell'architettura complessiva

Dal punto di vista architetturale, `aist_index.py` rappresenta uno strato di preparazione dei dati (data preparation) posto a monte del sistema, lo script trasforma la struttura originaria del dataset AIST++ basata sulla naming rule dei file in una base di dati contenente metadati. Il dataset `aist_index.json` verrà poi utilizzato come base per la generazione del database vettoriale Chroma.

3.6 Database Vettoriale

Successivamente all'indicizzazione dei metadati, il sistema necessita di tradurre le informazioni testuali delle mosse di danza in rappresentazioni vettoriali ad alta densità (dense embeddings), consentendo la ricerca per similarità semantica tramite database vettoriale. Tale processo è interamente gestito dallo script `ingest_chroma.py`, il cui scopo è trasformare il dizionario di metadati `aist_index.json` in un database vettoriale denominato *ChromaDB* persistente nella directory `./chroma_db`.

3.6.1 Modello di Embedding

Il modulo si avvale dell'integrazione tra *LangChain* e *ChromaDB* (`langchain_chroma`). La generazione dei vettori è affidata al modello pre-addestrato *sentence-*

transformers/all-MiniLM-L6-v2 [14] mediante la classe *HuggingFaceEmbeddings*. Questo è progettato per proiettare frasi e sequenze testuali in uno spazio vettoriale ad alta densità a 384 dimensioni [14]. Contiene circa 22,7 milioni di parametri e presenta una dimensione su disco inferiore agli 80 MB. Le caratteristiche chiave di all-MiniLM-L6-v2 sono:

- **Codifica semantica delle frasi (*sentence encoding*):** il modello mappa qualsiasi testo di input in uno spazio vettoriale ad alta densità, sintetizzando il significato semantico complessivo della stringa all'interno di un vettore denso a 384 dimensioni.
- **Efficienza computazionale e finestra di contesto:** progettato per elaborare sequenze fino a un massimo di 256 word pieces (sub-word token), il modello risulta adatto all'analisi dei metadati espansi del dataset, permettendo un'elaborazione estremamente rapida, leggera e a basso impatto sulle risorse hardware. Il codice forza, inoltre, l'esecuzione degli embedding sulla CPU locale.
- **Addestramento mediante *contrastive learning*:** il modello è stato addestrato secondo un paradigma di apprendimento contrastive learning, ottimizzando la sua capacità di misurare le relazioni di affinità e calcolare la similarità semantica tra frasi differenti.

3.6.2 Arricchimento semantico

Per fare in modo che la ricerca vettoriale basata sulla similarità del coseno (cosine similarity) risponda in modo efficace ai prompt espressi dagli utenti in linguaggio naturale, il modulo applica una tecnica di espansione semantica (semantic expansion) prima della vettorizzazione. I codici sintetici del dataset AIST++ vengono arricchiti tramite tre dizionari:

- `GENRE_EXPANSION`: associa a ciascun codice di stile di danza una varietà di sinonimi, descrittori stilistici e termini d'uso comune, consentendo al database vettoriale di associare la mossa anche se l'utente ricerca parole chiave generiche.
- `DIFFICULTY_EXPANSION`: traduce il livello di complessità (facile, medio, difficile) in termini legati al contesto di esecuzione della mossa di danza.
- `MOOD_BY_GENRE`: correla i generi musicali a chiavi emotive e attitudinali (mood).

3.6.3 Costruzione dei Document LangChain

La funzione `build_document()` trasforma ciascun record del dataset `aist_index.json` in un oggetto document di LangChain. La struttura del documento si compone di due sezioni fondamentali:

- **Contenuto testuale (`page_content`):** è una stringa descrittiva costruita concatenando l'identificativo (ID) della mossa, la denominazione dello stile, i termini espansi da `GENRE_EXPANSION` e `MOOD_BY_GENRE`, la frequenza in BPM, la difficoltà, il codice di situazione, il tipo di movimento ed i codici di musica e ballerino, rappresenta l'input inviato al modello di embedding per la generazione del vettore.

- **Dizionario dei metadati (metadata):** contiene le informazioni scalari non vettorizzate (id, path, extension, `genre_code`, genre_name, `situation_code`, `music_code`, bpm, difficulty, `dancer_id`). Dal momento che ChromaDB rifiuta campi con valore `None`, lo script applica un filtraggio preventivo delle chiavi vuote, i metadati vengono infine salvati nel database insieme al vettore.

3.6.4 Generazione del Database Vettoriale Chroma

La funzione principale `build_chroma_db()` coordina l'intero workflow di generazione del database vettoriale:

- **Gestione del flag `-reset`:** se il modulo viene eseguito passando da riga di comando l'argomento `-reset`, la directory preesistente ./chroma_db viene completamente eliminata tramite `shutil.rmtree()`, permettendo la possibilità di ricostruire il database da zero in caso di aggiornamenti al file `aist_index.json`.

- **Caricamento e scansione:** la funzione `load_dataset()` verifica la presenza di `aist_index.json`, ne legge la struttura e la converte in un elenco di oggetti document ordinati univocamente per identificativo.

- **Indicizzazione e salvataggio:** con l'invocazione di `Chroma.from_documents()`, i documenti vengono vettorizzati in batch sulla *Central Processing Unit* (CPU) e serializzati in modo permanente sul disco all'interno della directory di destinazione.

Il database vettoriale così generato costituisce la memoria semantica a supporto del tool `search_aist_dataset`, che permette all'agente ReAct di eseguire ricerche ibride (semantiche e filtri su metadati) in tempi di risposta ridotti.

3.7 Livello di presentazione: interfaccia web

Il primo livello dell'architettura è rappresentato dall'interfaccia web che è la fase in cui l'utente entra in contatto con il sistema e in cui può inserire una richiesta tramite prompt testuale contenente le caratteristiche della coreografia desiderata. La richiesta può includere, ad esempio:

- lo stile di danza
- il livello di difficoltà (facile o difficile)
- il numero di mosse desiderate
- i bpm minimi e massimi
- il mood desiderato

L'interfaccia ha il compito di inviare la richiesta dell'utente al backend Flask. Non è necessario che l'utente conosca i nomi dei file presenti nel dataset AIST++ o i codici utilizzati per identificare le animazioni, il prompt testuale verrà infatti successivamente interpretato dall'LLM che estrarrà le informazioni corrette da utilizzare nelle fasi successive della pipeline di esecuzione. Questa scelta permette di realizzare

un'interazione ad alto livello e semplificata per l'utente che può descrivere ciò che desidera ottenere, mentre la gestione dei dati, la ricerca delle mosse di danza e la costruzione della coreografia vengono eseguite automaticamente dal sistema. Dal punto di vista dell'architettura, l'interfaccia è separata dalla logica dell'agente. Questa non si occupa infatti, ad esempio, di interrogare direttamente ChromaDB o di chiamare l'LLM, queste operazioni sono delegate al backend Flask, che rappresenta il livello intermedio tra la parte di interfaccia grafica e il sistema agentico. La separazione tra interfaccia e logica applicativa offre diversi vantaggi, tra cui: consentire di modificare l'interfaccia senza intervenire sul workflow e centralizzare la gestione delle richieste nel backend Flask.

3.8 Backend Flask

Il backend è stato realizzato utilizzando *Flask*, un micro-framework in linguaggio Python utilizzato per lo sviluppo di applicazioni e siti web basato sullo strumento Werkzeug *Web Server Gateway Interface* (WSGI) e presenta il motore di template Jinja2. Definito micro-framework in quanto possiede un nucleo semplice ma estendibile. Inoltre non c'è uno strato di astrazione per i componenti che forniscono funzionalità per le quali esistono già librerie di terze parti [15]. Flask svolge il ruolo di collegamento tra l'interfaccia utente e il sistema agentico. Quando l'utente invia una richiesta, il backend la riceve e avvia il processo di generazione della coreografia. Il backend ha principalmente le seguenti responsabilità: ricevere il prompt dell'utente, avviare il workflow dell'agente, coordinare la comunicazione tra interfaccia e logica applicativa, restituire lo stato dell'elaborazione, restituire il risultato finale e rendere disponibili i file prodotti dal sistema.

Il compito di Flask consiste nel gestire la comunicazione tra i diversi livelli dell'applicazione, il cui flusso semplificato è illustrato nella Figura 3.3.

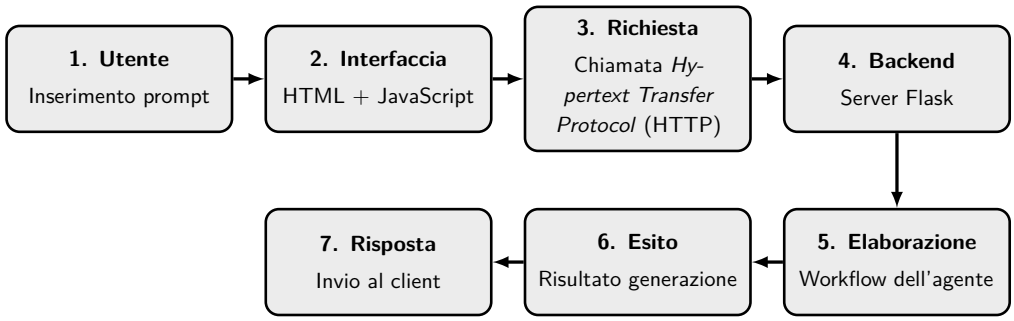


Figura 3.3: Flusso semplificato della comunicazione tra client, backend Flask e workflow agentico.

Questa organizzazione separa il livello di presentazione e il livello di elaborazione. L'interfaccia si occupa di mostrare le informazioni, mentre Flask gestisce le richieste e il collegamento con il sistema agentico.

3.9 Il livello agentico

Il nucleo del sistema è costituito dall'agente AI. A differenza di una semplice chiamata a un Large Language Model, che si limiterebbe a generare testo isolato rispondendo direttamente in un unico passaggio, l'agente AI è in grado di svolgere compiti complessi in autonomia, sfruttando il ragionamento (pianificazione, scomposizione, riflessione), utilizzare strumenti esterni, gestire lo stato della conversazione ed eseguire un workflow strutturato su più fasi[1]. L'agente coreografo è stato sviluppato come un sistema software capace di coordinare una serie di operazioni finalizzate alla generazione della coreografia richiesta dall'utente.

È importante distinguere tra LLM e agente. Come si evince da Tabella 3.4, l'LLM è il modello linguistico utilizzato per interpretare le informazioni e selezionare le mosse di danza, mentre l'agente rappresenta il livello di coordinamento del sistema. Questo risultato è stato implementato attraverso un workflow basato su LangGraph e sul paradigma *ReAct*, l'agente può utilizzare tre strumenti specifici, chiamati *tools*, sviluppati in linguaggio Python, che permettono al sistema di interrogare il database vettoriale a partire dall'interpretazione del prompt dell'utente, elaborare e filtrare i risultati della ricerca e costruire la sequenza di movimenti che compone la coreografia richiesta. L'agente non genera direttamente la coreografia attraverso un'unica operazione come avverrebbe nel caso della *GenAI*, ma coordina una sequenza di attività specializzate, nelle quali ogni fase contribuisce alla trasformazione progressiva del prompt iniziale in un output contenente le mosse di danza facenti parte della coreografia e visualizzabili all'interno di Unity. Il workflow mantiene inoltre uno stato interno che consente di monitorare l'avanzamento dell'elaborazione e verificare che le principali fasi del processo siano state effettivamente completate prima di produrre l'output finale. Il confronto viene riepilogato in tabella 3.4.

Tabella 3.4: Confronto tra il ruolo del Large Language Model (LLM) e dell'Agente AI nell'architettura.

Dimensione	**Large Language Model (LLM)**	**Agente AI (LangGraph / ReAct)**
Ruolo principale	Ragionamento semantico e interpretazione	Coordinamento globale e orchestrazione del workflow
Ambito operativo	Generazione di testo e scelte in un singolo passaggio	Esecuzione sequenziale multi-step con memoria di stato
Uso degli Strumenti	Identifica l'intenzione e sceglie quale tool attivare	Invoca ed esegue i tools Python
Accesso ai Dati	Opera unicamente sulle informazioni nel contesto	Interroga direttamente il database vettoriale ChromaDB
Gestione dello Stato	Elaborazione priva di stato (*stateless*)	Mantiene la memoria condivisa dell'avanzamento
Validazione e Fallback	Non verifica la conformità dei vincoli rigidi	Esegue controlli deterministici e fallback in Python
Output e File System	Restituisce strutture dati/JSON testuali	Genera e serializza i file finali

3.9.1 Architettura agente ReAct

L'architettura dell'agente ReAct è implementata all'interno del file `agent.py` attraverso il framework LangGraph, viene utilizzato il modello linguistico *NVIDIA Nemotron 3 Ultra 550B* [16] tramite ChatOpenAI reindirizzata su *Application Programming Interface* (API) di OpenRouter. La scelta del modello in questione è stata effettuata tenendo in considerazione la possibilità di fruirne gratuitamente e da remoto, seppure con limitazioni relative al numero di chiamate al modello giornaliere, e il numero di parametri totali (550 miliardi) e attivi (55 miliardi), superiore ad altri modelli gratuiti. Per rendere il comportamento dell'LLM il più possibile deterministico durante l'elaborazione delle richieste e la chiamata dei tools, il modello è configurato con `temperature` nulla. Sono inoltre impostati parametri di controllo quali un limite massimo di 1024 token per risposta, l'assenza di tentativi automatici aggiuntivi e un timeout di 120 secondi. Per limitare l'accumulo dei messaggi nella finestra di contesto (*context window*), nel file `agent.py` è stata inserita la gestione della memoria a breve termine basata sulla funzione `trim_messages`. Il trimmer è configurato con un limite massimo di 6000 token, utilizzando `count_tokens_ap-`

`proximately` per la stima della lunghezza del contesto. La strategia last consente di mantenere i messaggi più recenti, mentre `include_system=True` consente la conservazione del messaggio di sistema che corrisponde al prompt guida dell'agente, `pre_model_hook` applica tale operazione prima di ogni elaborazione del modello, fornendo quindi all'LLM una versione ridotta della cronologia disponibile. L'agente viene istanziato attraverso `create_react_agent`, a cui vengono forniti l'LLM, tutti i tools che possono essere utilizzati (`all_tools`), il prompt guida `UNIVERSAL_PROMPT`, il componente `MemorySaver` per la gestione dello stato in memoria e l'hook di preelaborazione. `MemorySaver` consente di mantenere lo stato dell'agente tra le successive iterazioni del workflow durante l'esecuzione. L'agente ReAct è inserito in una struttura di supervisione esterna. In particolare, `agent.py` non garantisce che vengano sempre eseguiti tutti gli strumenti nell'ordine previsto; il controllo del workflow è affidato al modulo `supervised_react.py`, che verifica l'effettiva esecuzione dei tre tools della pipeline: `parse_dance_prompt`, `search_aist_dataset` e `build_sequence_plan`. Qualora il comportamento ReAct non porti a conclusione uno degli step previsti, il supervisore interviene completando il workflow mediante un fallback in Python. La componente ReAct mantiene la capacità di coordinare gli strumenti, mentre il supervisore ne vincola l'esecuzione al workflow sviluppato.

3.9.2 System Prompt

La definizione delle direttive di sistema e dei vincoli comportamentali che guidano l'agente ReAct nell'elaborazione delle coreografie è centralizzata all'interno dello script prompts.py attraverso la costante `SYSTEM_PROMPT`. Il prompt assegna all'agente un ruolo specializzato nella generazione di coreografie e ne formalizza il ciclo di ragionamento imponendo una sequenza obbligatoria nell'invocazione dei tools:

1. **`parse_dance_prompt`:** per l'estrazione semantica e strutturata dei parametri dalla richiesta dell'utente.
2. **`search_aist_dataset`:** per l'interrogazione del database vettoriale e il recupero delle mosse compatibili.
3. **`build_sequence_plan`:** per la composizione finale della coreografia.

Per consentire il rigore operativo e mitigare il rischio di allucinazioni da parte del modello linguistico remoto (NVIDIA Nemotron tramite OpenRouter), `SYSTEM_PROMPT` stabilisce una serie di regole. Tra queste, si richiede l'immutabilità della richiesta originale dell'utente, il divieto di inventare identificativi (ID) o sequenze di movimento non presenti nel dataset, e di effettuare la ricerca delle mosse solamente all'interno del database vettoriale ChromaDB. Dal punto di vista della logica di query, il prompt codifica una chiara separazione dei ruoli: i mood e gli aspetti emotivi guidano la ricerca per similarità semantica in Chroma, mentre il genere, il tempo

in BPM e la difficoltà agiscono come filtri. In questo modo, l'agente ReAct viene confinato all'interno di un perimetro operativo controllato, demandando all'LLM il compito di valutare e ordinare i candidati estratti dal tool `search_aist_dataset`.

3.10 LangGraph e LangChain

LangGraph [17] è un framework e runtime di orchestrazione open source sviluppato da LangChain Inc. [18] per la creazione e la gestione di agenti e flussi di lavoro di Intelligenza Artificiale caratterizzati da stato persistente, esecuzione multi-step e possibilità di interazione tra diversi componenti. A differenza delle tradizionali pipeline lineari e acicliche, LangGraph permette di rappresentare il processo attraverso un grafo orientato, nel quale i nodi rappresentano le diverse operazioni e gli archi definiscono il percorso dell'esecuzione. Il grafo può inoltre contenere percorsi condizionali e cicli, consentendo all'agente di ripetere determinate operazioni o modificare il proprio comportamento sulla base dei risultati ottenuti. Il funzionamento di LangGraph si basa principalmente su tre elementi:

- **Stato (*State*):** una struttura dati condivisa che rappresenta la memoria di lavoro del workflow e contiene le informazioni necessarie durante l'esecuzione.
- **Nodi (*Nodes*):** funzioni Python che ricevono lo stato corrente, eseguono una specifica operazione e restituiscono eventuali aggiornamenti allo stato.
- **Archi (*Edges*):** collegamenti tra i nodi che determinano il passaggio da una fase all'altra. Gli archi condizionali permettono inoltre di scegliere dinamicamente il nodo successivo sulla base dello stato corrente.

LangGraph non sostituisce LangChain, ma svolge un ruolo complementare. LangChain fornisce le astrazioni e le integrazioni necessarie per interagire con i modelli linguistici, gestire i prompt e collegare l'agente ai relativi strumenti, mentre LangGraph si occupa dell'orchestrazione del processo e della gestione dello stato. Nel progetto, LangGraph non genera direttamente la coreografia, ma coordina le diverse operazioni necessarie per trasformare il prompt dell'utente nell'output finale e permette all'agente di interagire con i tools durante l'esecuzione. Le varie fasi del processo condividono lo stesso stato, che viene aggiornato progressivamente e utilizzato per trasferire le informazioni tra i diversi nodi. Lo stato contiene, tra le altre, le seguenti informazioni: prompt originale dell'utente, intento estratto dal prompt, numero di sequenze richieste, risultati della ricerca, mosse candidate, coreografia finale.

3.10.1 Large Language Model

Il large language model utilizzato dall'agente è *NVIDIA Nemotron 3 Ultra 550B*, accessibile attraverso la piattaforma OpenRouter che viene utilizzata come livello di accesso al modello tramite un'API compatibile con OpenAI, consentendone l'integrazione nell'applicazione attraverso LangChain. Nel sistema, Nemotron costituisce il componente linguistico e decisionale dell'agente. Il modello riceve il prompt dell'utente, le istruzioni definite nel sistema e le informazioni disponibili nello stato e, attraverso il paradigma ReAct, determina quali operazioni eseguire e quali tools utilizzare. Le operazioni specifiche vengono quindi delegate ai tools

sviluppati in Python. Il modello riceve i risultati delle operazioni eseguite e li utilizza per determinare come procedere nel workflow. In questo modo, i diversi componenti svolgono ruoli distinti come illustrato nella Tabella 3.5.

Tabella 3.5: Ripartizione dei ruoli e delle responsabilità tra i componenti del sistema.

Componente	**Ruolo e Responsabilità**
`OpenRouter`	Fornisce il livello di accesso API al modello `Nemotron`
`LangChain`	Integra il modello linguistico e i relativi tools all'interno dell'agente
`Nemotron`	Interpreta la richiesta e determina le azioni da intraprendere
`LangGraph`	Orchestra la struttura del workflow e gestisce lo stato condiviso
`Tools Python`	Eseguono concretamente le operazioni logiche richieste dal workflow
`ChromaDB`	Gestisce la ricerca per similarità semantica nel database vettoriale

3.10.2 Organizzazione del Workflow

Nel sistema sono integrati principalmente tre tools che l'agente può invocare:

- `parse_dance_prompt`
- `search_aist_dataset`
- `build_sequence_plan`

`parse_dance_prompt` l'LLM interpreta la richiesta dell'utente ed estrae le informazioni necessarie per definire i parametri della ricerca, `search_aist_dataset` utilizza questi parametri per interrogare il database vettoriale e recuperare le mosse di danza più pertinenti, `build_sequence_plan` utilizza i candidati recuperati e il modello linguistico per costruire la coreografia richiesta e generare l'output finale. I tre tools rappresentano quindi le principali operazioni attraverso cui la richiesta dell'utente viene trasformata progressivamente nell'output finale. La loro esecuzione viene coordinata dall'agente all'interno del workflow. La struttura del processo può essere rappresentata come da Figura 3.4.

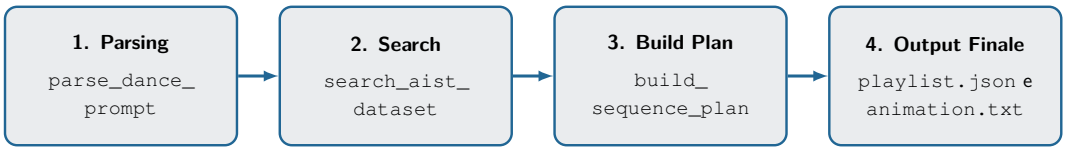


Figura 3.4: Workflow dei tools

Questa suddivisione permette di attribuire una responsabilità specifica a ciascuno strumento e di separare le diverse fasi dell'elaborazione, rendendo il workflow più strutturato e controllabile.

3.10.3 Il paradigma ReAct

Il sistema adotta il paradigma ReAct (*Reasoning and Acting*), un approccio in cui il modello linguistico alterna fasi di ragionamento interno (*Thought*) all'esecuzione di azioni concrete (*Action*) tramite l'invocazione di strumenti esterni (*Tools*), osservandone infine i risultati (*Observation*). Il modello può decidere quale strumento utilizzare per portare avanti il processo. L'utilizzo di strumenti separati presenta un vantaggio rispetto a una singola richiesta al modello. Ogni operazione può essere controllata e può produrre un risultato utilizzabile dalle fasi successive.

3.10.4 Supervisione dello stato

Lasciare un Large Language Model completamente libero nell'esecuzione del paradigma ReAct può comportare il rischio di fenomeni di *agent drift*, ovvero una deviazione dell'agente dal comportamento previsto durante l'esecuzione. In particolare, l'elevata flessibilità nella selezione degli strumenti può portare il modello a modificare il flusso operativo previsto, interrompere prematuramente la pipeline o effettuare chiamate non coerenti con lo stato corrente del workflow.

Per garantire una maggiore robustezza dell'esecuzione, al paradigma ReAct viene affiancato un supervisore dello stato gestito tramite LangGraph. Lo stato condiviso del workflow mantiene tre flag di avanzamento:

- ***parsed*:** attesta l'avvenuta estrazione e normalizzazione della richiesta dell'utente mediante il tool `parse_dance_prompt`.
- ***searched*:** conferma l'esecuzione della ricerca per similarità vettoriale e il recupero dei candidati validi mediante `search_aist_dataset`.
- ***built*:** comunica l'avvenuta selezione, validazione e serializzazione della playlist finale mediante `build_sequence_plan`.

Gli archi condizionali del grafo verificano tali flag prima di consentire il passaggio alla fase successiva. In questo modo, l'agente non può procedere verso una fase per la quale non siano state soddisfatte le condizioni necessarie. Ad esempio, se l'agente ReAct tenta di invocare `build_sequence_plan` quando *searched* è ancora impostato a `False`, la logica di supervisione impedisce la transizione e reindirizza l'esecuzione verso la fase di retrieval.

Il paradigma ReAct interviene principalmente nell'orchestrazione dinamica delle operazioni. Quando l'utente invia un prompt in linguaggio naturale, il modello linguistico Nemotron analizza la richiesta, le istruzioni disponibili e lo stato corrente del workflow, determinando quale strumento debba essere invocato per far progredire l'esecuzione. I risultati prodotti dal tool vengono quindi restituiti al modello come osservazione, consentendogli di valutare l'azione successiva.

Un ulteriore impiego del modello linguistico si verifica durante la costruzione della sequenza mediante `build_sequence_plan`. In questa fase, il modello analizza i

metadati associati alle mosse candidate e seleziona quelle più coerenti con la richiesta dell'utente, considerando informazioni quali `dancer_id`, BPM, stile e progressione della sequenza. La decisione del modello è successivamente sottoposta ai controlli implementati dal sistema prima della produzione dell'output finale.

L'esecuzione del ciclo ReAct può essere quindi schematizzata nelle seguenti tre fasi, riepilogate nella Figura 3.5:

1. **Ragionamento (*Thought*)**: Nemotron riceve il prompt dell'utente, le istruzioni di sistema e lo stato condiviso del workflow. Sulla base delle informazioni disponibili, determina l'azione successiva.

2. **Azione (*Action*)**: il modello produce una chiamata a uno dei tre tool Python disponibili.

3. **Osservazione (*Observation*)**: il tool invocato esegue l'operazione richiesta e restituisce al modello i risultati dell'esecuzione. Questi risultati costituiscono la nuova osservazione sulla quale il modello basa l'iterazione successiva.

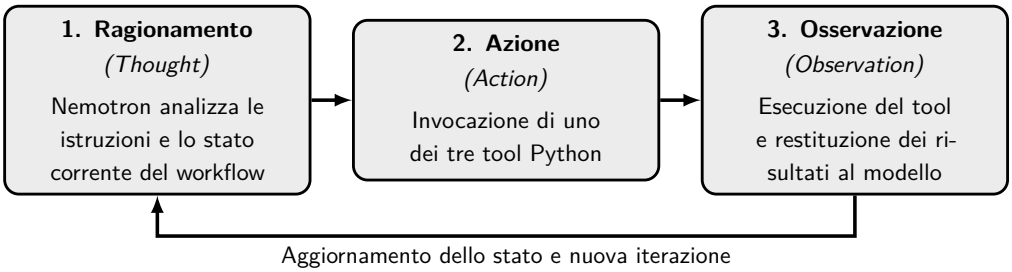


Figura 3.5: Ciclo iterativo ReAct (Thought–Action–Observation) dell'agente coreografo.

Il ciclo ReAct non opera quindi in maniera completamente autonoma, ma è incapsulato all'interno di una macchina a stati che ne controlla l'avanzamento tra le diverse fasi della pipeline. In questo modo, la capacità interpretativa e la flessibilità del modello vengono mantenute, mentre i vincoli strutturali necessari alla corretta esecuzione del workflow vengono imposti dal livello di `supervisione`.

3.10.5 ReAct supervisionato

La `supervisione` descritta nella precedente sezione è implementata concretamente nel componente `supervised_react.py`, che introduce un livello di controllo esterno rispetto al ciclo ReAct. Il componente opera come una macchina a stati che monitora l'avanzamento dell'agente e verifica che le operazioni vengano eseguite secondo l'ordine previsto.

La necessità di questo livello di controllo deriva dal fatto che il modello linguistico, pur essendo in grado di selezionare dinamicamente i tool, non costituisce da solo

una garanzia sul rispetto dei vincoli strutturali della pipeline. Durante l'esecuzione possono infatti verificarsi comportamenti quali l'alterazione del prompt originale, l'interruzione del flusso dopo una sola ricerca, la sostituzione di risultati validi con nuove interrogazioni non riuscite oppure la produzione di una coreografia con un numero di mosse differente da quello richiesto dall'utente.

Il supervisore interviene pertanto come livello di controllo tra la decisione dell'agente e l'avanzamento del workflow, quando la transizione proposta dal modello è compatibile con lo stato corrente, l'esecuzione può procedere normalmente. In caso contrario, il controller impedisce la transizione non valida e indirizza il workflow verso la fase necessaria per soddisfare i prerequisiti mancanti.

Questa architettura combina due proprietà differenti: da un lato, la flessibilità del paradigma ReAct, che permette al modello di interpretare la richiesta e selezionare dinamicamente gli strumenti, dall'altro, la determinazione del workflow garantita dalla macchina a stati, che impedisce all'agente di discostarsi dai vincoli fondamentali della pipeline.

Il supervisore non sostituisce quindi il modello linguistico nella fase decisionale, ma ne limita il comportamento entro uno spazio di transizioni ritenute valide dal sistema. La logica di controllo costituisce di conseguenza un livello di verifica strutturale che contribuisce a rendere l'esecuzione più robusta e prevedibile.

3.10.6 Controlli di sicurezza

Lo script `supervised_react.py` permette il rispetto della sequenza operativa, illustrata in Figura 3.6, dei tre tools:

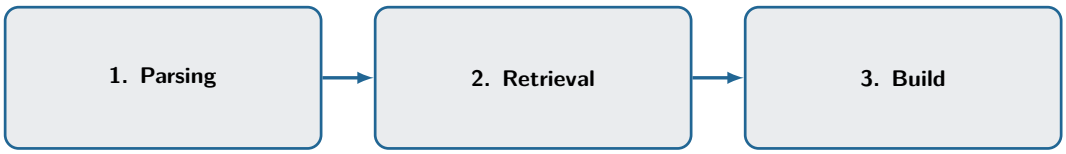


Figura 3.6: Fasi fondamentali della pipeline

Il controller implementa tre principali controlli di sicurezza:

- **Immutabilità della richiesta utente:** impedisce all'agente di sostituire o riscrivere la richiesta originale durante le iterazioni del ciclo ReAct.
- **Persistenza della ricerca valida (`_best_search_json`):** evita che una ricerca non riuscita sovrascriva un precedente risultato di ricerca valido ottenuto da ChromaDB. La funzione analizza lo stato della sessione e seleziona prioritariamente il valore memorizzato in `last_successful_search_json` rispetto a `last_search_json`.
- **Validazione del conteggio delle sequenze:** verifica che il numero di mosse inserite nella coreografia corrisponda esattamente al parametro richiesto dall'utente

3.10.7 Flusso di esecuzione

La funzione principale `run_supervised_react_workflow()` coordina la risposta del sistema combinando l'esecuzione dell'agente LangGraph con i meccanismi di controllo. Il flusso operativo si articola nelle seguenti fasi sequenziali:

1. **Inizializzazione dello Stato:** il controller resetta lo stato del workflow tramite `reset_status()` e prepara il dizionario `supervision` per il tracciamento di tutte le metriche di fallback.

2. **Esecuzione dell'Agente ReAct:** viene invocato l'agente sincrono mediante la funzione `_invoke_react_agent()`, impostando il limite di ricorsione a 16 passaggi. Se l'agente completa l'esecuzione senza sollevare eccezioni, la variabile react_completed viene impostata a `True`.

3.10.8 Ripristino della Pipeline (fallback)

Nel caso in cui ReAct interrompa la pipeline di esecuzione, il controller `supervised_react.py` interviene eseguendo direttamente le funzioni Python dei tool mancanti:

1. **Garantire l'analisi semantica (*parse fallback*):** se nello stato del workflow (status) non è presente il risultato last_parse_json, il controller invoca direttamente il tool `parse_dance_prompt` passando il prompt utente integro.

2. **Garantire la ricerca vettoriale (*search fallback*):** il controller tenta di recuperare l'ultima ricerca ChromaDB valida mediante `_best_search_json()`. Nel caso in cui non vi siano ricerche valide memorizzate, viene eseguito il tool `search_aist_dataset` utilizzando i parametri estratti dalla fase di parsing.

3. **Garantire la composizione della coreografia (*build fallback*):** se l'agente non ha eseguito `build_sequence_plan`, il controller invoca direttamente il tool inoltrando i candidati estratti dal database vettoriale e il numero di mosse richieste.

Validazione numero mosse: il controller misura il numero di mosse presenti nella coreografia generata (`_playlist_generated_count`) confrontandolo con quello richiesto dall'utente. Nel caso in cui non coincidano, la coreografia viene immediatamente scartata e rigenerata dal controller.

3.11 Tool di Parsing

Il primo tool, denominato `parse_dance_prompt`, costituisce la componente di *Natural Language Understanding* (NLU) del sistema e ha il compito di analizzare il testo inserito dall'utente in linguaggio naturale, trasformando la richiesta in un insieme strutturato di parametri formalizzati e serializzati in formato JSON. Tale rappresentazione costituisce l'input per le successive fasi del workflow e consente una codifica coerente dei generi di danza secondo le convenzioni e i codici definiti dal dataset AIST++.

Il tool è organizzato secondo una pipeline articolata in tre fasi principali:

1. **Estrazione semantica e gestione delle eccezioni:** il modello LLM interpreta la richiesta dell'utente ed estrae i principali elementi. Un meccanismo di gestione delle eccezioni consente la continuità dell'esecuzione in caso di errori di comunicazione, timeout o output non valido.

2. **Normalizzazione deterministica e controllo dell'output:** i dati prodotti dall'LLM vengono verificati e normalizzati mediante regole deterministiche. Questa fase corregge eventuali discrepanze, vincola i generi ai codici ammessi da AIST++ e impedisce l'introduzione di parametri non esplicitamente richiesti dall'utente.

3. **Composizione del payload e persistenza dello stato:** i parametri validati vengono organizzati nel payload finale e serializzati in formato JSON. Il risultato viene inoltre memorizzato nello stato globale della pipeline.

Nemotron interpreta la richiesta, ma non ha l'ultima parola sui vincoli strutturali in quanto BPM e numero di sequenze, ad esempio, vengono sottoposti a controlli.

3.11.1 Estrazione semantica e gestione errori

Nella prima fase, la richiesta utente viene preventivamente controllata tramite la funzione `_get_locked_prompt` e successivamente inoltrata al modello LLM Nemotron 3 Ultra, distribuito via OpenRouter API. L'integrazione con l'LLM è configurata con una temperatura pari a 0, riducendo il grado di variabilità nella generazione dei token.

Il modello viene istruito mediante un *System Prompt* a compiere le seguenti operazioni:

- Interpretare le entità della richiesta mappa-generi secondo i codici ufficiali AIST++.

- Estrarre attributi secondari quali il livello di difficoltà, il tipo di movimento, il mood espressivo, il numero di sequenze richieste, BPM massimi e minimi.

- Restituire una stringa serializzata in formato JSON conforme allo schema della classe `DanceIntent`.

Per garantire la continuità dell'esecuzione, l'intero blocco relativo all'invocazione remota del modello è racchiuso all'interno di una struttura di gestione delle eccezioni *try-except*. In caso di errori di rete, timeout della chiamata API o anomalie nella sintassi del JSON restituito dall'LLM, il sistema intercetta l'eccezione, registra l'evento nei log mediante un messaggio di warning e inizializza un oggetto `DanceIntent` neutro. In questo modo, il fallimento dell'LLM non determina l'interruzione della pipeline: l'esecuzione prosegue attraverso un fallback deterministico, lasciando alle successive fasi di normalizzazione e validazione il compito di stabilire i valori effettivamente utilizzabili.

3.11.2 Formato della risposta dell'LLM

Affinché la risposta generata dal modello linguistico possa essere utilizzata dal programma, l'output testuale restituito dall'LLM deve essere convertito in una struttura dati interpretabile dal codice Python. Il modello viene istruito tramite *System Prompt* a restituire un oggetto JSON contenente le mosse selezionate e il relativo ordine. Tuttavia, le risposte dei modelli linguistici possono presentare variazioni rispetto al formato atteso, come l'inclusione di blocchi markdown. Il sistema integra una pipeline dedicata all'estrazione, al parsing, alla normalizzazione e alla validazione della risposta, illustrata nella Figura 3.7:

- **Ricezione dell'output grezzo:** acquisizione della stringa testuale prodotta dal modello.
- **Estrazione della struttura JSON:** la funzione `_estrai_json_da_testo` individua la porzione di codice valida, separando l'oggetto JSON dal testo contiguo o estraendolo dai marcatori markdown.
- **Parsing e conversione:** deserializzazione della stringa tramite `json.loads()` in oggetti nativi Python (dizionari e liste).
- **Normalizzazione dello schema:** la funzione `_normalizza_output_llm` uniforma le chiavi e le strutture restituite, riconducendo i dati al tipo interno `SequenzaScelta` (basato sulle coppie posizione-`indice_candidato`).
- **Recupero di emergenza:** qualora la sintassi JSON risulti non idonea, il sistema applica espressioni regolari per isolare ed estrarre direttamente gli indici numerici dal testo, evitando l'interruzione del workflow.

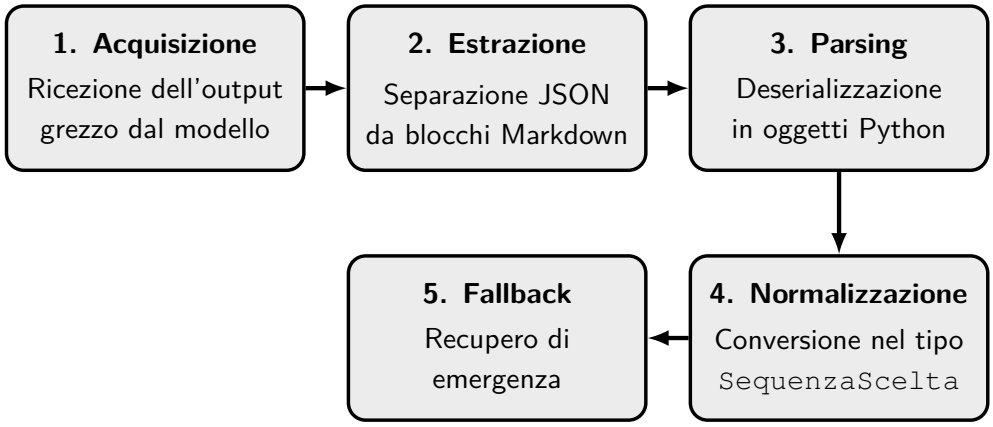


Figura 3.7: Pipeline di estrazione, parsing e normalizzazione della risposta LLM.

Una volta normalizzata la risposta, la struttura dati viene passata alla pipeline di validazione per i controlli.

3.11.3 Normalizzazione e Controllo

I modelli linguistici tendono ad applicare inferenze non richieste (*over-inference*), associando vincoli quantitativi a termini qualitativi. Ad esempio, il termine energica potrebbe essere interpretato autonomamente dal modello come un intervallo specifico di BPM, nonostante l'utente non abbia fornito alcun vincolo temporale. Per evitare che tali inferenze introducano parametri non richiesti nella pipeline, il sistema applica una fase di post-processing in cui i dati estratti dall'LLM vengono verificati e, se necessario, corretti.

1. **Mapping e normalizzazione dei generi:** la funzione `normalizza_generi` confronta le stringhe estratte dall'LLM con il testo originale e con il dizionario dei codici ammessi, correggendo eventuali errori di allineamento e controllando l'appartenenza del risultato al dominio del dataset AIST++.

2. **Controlli sui parametri temporali (BPM):** viene eseguito un controllo booleano tramite la funzione `utente_ha_chiesto_bpm_o_tempo(prompt)`. Se nella richiesta non sono presenti valori numerici di BPM né indicazioni esplicite relative alla velocità (come lento, veloce) gli eventuali valori di `min_bpm`, `max_bpm` e `tempo_description` precedentemente prodotti dall'LLM vengono sovrascritti e impostati a `None`. In questo modo, il sistema impedisce che un'inferenza del modello venga trasformata automaticamente in un vincolo di ricerca.

3. **Parsing quantitativo e strutturale:** il numero di mosse richieste viene validato attraverso la funzione `estrai_n_sequenze_da_testo`. Quando la richiesta non contiene un'indicazione numerica esplicita, il parametro viene impostato al valore predefinito di sistema `DEFAULT_NUM_SEQUENCES`. Questo

permette di preservare il vincolo quantitativo espresso direttamente dall'utente, evitando che il modello possa modificarlo durante l'interpretazione semantica (`DEFAULT_NUM_SEQUENCES`).

3.11.4 Composizione del payload

Alla fine delle fasi di estrazione e validazione, i parametri normalizzati vengono organizzati in un dizionario Python che costituisce il payload finale. Il dizionario include i campi come da Tabella 3.6.

Tabella 3.6: Campi del payload generati dal tool di estrazione semantica.

Chiave Payload	**Tipo Dato**	**Descrizione**
`genre_codes`	`List[str]`	Lista dei codici di genere normalizzati (AIST++)
`difficulty`	`str`	Livello di difficoltà della mossa
`motion_type`	`str`	Tipologia di movimento richiesto
`min_bpm` / `max_bpm`	`int`	Intervallo di tempo/ritmo target
`tempo_description`	`str`	Descrizione testuale del tempo/ritmo
`mood_keywords`	`List[str]`	Parole chiave sull'atmosfera/stile
`n_sequenze`	`int`	Numero totale di sequenze da generare
`prompt_originale`	`str`	Testo integrale inviato dall'utente

Il dizionario viene quindi serializzato in formato JSON prima di essere restituito alla pipeline. Contestualmente, il modulo aggiorna lo stato globale dell'applicazione memorizzando il risultato nella variabile `LAST_PARSED_INTENT` e invocando la funzione `_set_workflow_status`. La persistenza di questo stato consente di mantenere una rappresentazione coerente dell'intento dell'utente durante l'esecuzione della pipeline e rende le informazioni estratte disponibili in modo sincrono alle fasi successive dell'orchestrazione coreografica. In questo modo, il risultato della fase di parsing costituisce il riferimento strutturato utilizzato dai successivi strumenti per la ricerca e la composizione delle sequenze di movimento.

3.12 Tool di Retrieval

Il secondo tool, denominato `search_aist_dataset`, ha il compito di individuare, all'interno del dataset AIST++ precedentemente vettorializzato, le mosse di danza compatibili con la richiesta dell'utente elaborata dal tool `parse_dance_prompt`. La ricerca viene effettuata mediante il database vettoriale Chroma, utilizzando un approccio ibrido che combina la similarità semantica del contenuto testuale con filtri applicati ai metadati associati alle sequenze.

3.12.1 Chroma come database vettoriale

Chroma viene utilizzato all'interno dell'architettura come database vettoriale per memorizzare e recuperare in modo efficiente le informazioni associate alle sequenze di movimento. Durante la fase di preparazione dei dati, le descrizioni e i metadati associati alle mosse di danza vengono trasformati in *embedding*. Gli embedding rappresentano numericamente il contenuto semantico delle informazioni e consentono di effettuare confronti di similarità tra la richiesta dell'utente e le mosse presenti nel database. Quando l'utente invia una richiesta, il processo si articola nelle seguenti fasi:

1. La query viene convertita in un embedding vettoriale.
2. ChromaDB confronta la rappresentazione della richiesta con quelle memorizzate nel database tramite similarità vettoriale.
3. Il sistema restituisce le sequenze che presentano la maggiore similarità semantica con la query.

Nel progetto, la dimensione del retrieval iniziale è determinata dal parametro `SEARCH_K` che viene utilizzato per recuperare un insieme di massimo quaranta candidati, questi vengono successivamente filtrati dal codice Python per restituire al modello le dieci mosse candidate con la similarità maggiore. Questa scelta presenta due vantaggi principali:

- **Riduzione del rischio di esclusione prematura:** evita di scartare sequenze che, pur avendo un punteggio vettoriale leggermente inferiore, possiedono metadati altamente compatibili con i vincoli del prompt.
- **Applicazione di filtri specifici:** permette di eseguire controlli rigidi in Python sul pool di candidati prima di passare i dati all'LLM.

Il retrieval non produce direttamente la coreografia finale. Il suo risultato è un pool di mosse candidate, ovvero insieme di possibili mosse che verranno analizzate negli step successivi.

Il processo può quindi essere rappresentato come da Figura 3.8.

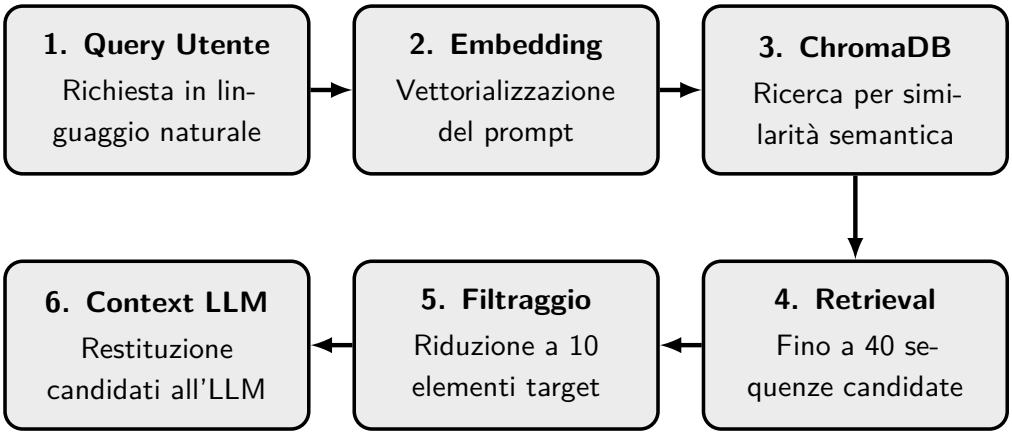


Figura 3.8: Pipeline di retrieval vettoriale e filtraggio progressivo delle mosse candidate.

3.12.2 Normalizzazione e costruzione della Query

Il tool riceve in input i parametri in formato JSON prodotti dalla fase di parsing, tra cui codici di genere, mood, difficoltà, BPM e numero di sequenze richieste. Prima della ricerca, tali parametri vengono sottoposti a una normalizzazione. In particolare, i generi vengono verificati mediante `normalizza_generi`, il livello di difficoltà attraverso `normalizza_difficulty` e le parole chiave relative al mood mediante `normalizza_mood`. I parametri temporali vengono invece sottoposti a un ulteriore controllo: BPM e descrizioni della velocità vengono utilizzati come filtri solamente quando la richiesta originale dell'utente contiene un'indicazione esplicita relativa al tempo o alla velocità. Eventuali valori introdotti autonomamente dal modello vengono ignorati. Successivamente viene costruita una query semantica composta dal prompt originale, dai codici dei generi, dai relativi nomi descrittivi e dalle parole chiave del mood. Questa query viene utilizzata per interrogare lo spazio vettoriale di Chroma attraverso una ricerca per similarità.

3.12.3 Retrieval semantico

La query viene utilizzata per effettuare una similarity_search su Chroma, recuperando inizialmente un insieme di `SEARCH_K` candidati semanticamente rilevanti. I risultati ottenuti vengono poi sottoposti a un filtraggio basato sui metadati delle mosse. Questo permette di distinguere due criteri differenti: la similarità semantica determina quali sequenze sono potenzialmente rilevanti rispetto al contenuto della richiesta, mentre i filtri tecnici verificano la compatibilità dei candidati con la richiesta esplicita dell'utente. I risultati vengono infine normalizzati e tramite l'id delle mosse vengono rimosse le mosse duplicate. Il numero di mosse richiesto dall'utente viene recuperato attraverso `_get_forced_n_sequenze`, preservando la richiesta originale.

3.12.4 Filtraggio dei candidati

Successivamente al recupero iniziale da ChromaDB, le mosse candidate vengono sottoposte a una fase di filtraggio basata sui metadati. Questo passaggio, eseguito in codice Python all'interno del tool `search_aist_dataset`, serve a ridurre l'insieme dei candidati prima che venga dato in input all'analisi dell'LLM. I filtri applicati controllano la coerenza delle mosse rispetto a diversi parametri:

- **Stile di danza e genere:** verifica della corrispondenza diretta o della compatibilità tra lo stile richiesto dall'utente e le etichette dei dati AIST++.
- **Informazioni temporali e metriche (BPM):** scarto delle sequenze la cui frequenza temporale si discosti dai vincoli target richiesti dall'utente (applicato solo in presenza di un vincolo esplicito nel prompt).
- **Livello di difficoltà ed esecuzione:** valutazione della complessità tecnica del movimento basata sulla situazione AIST++ (sBM per facile, sFM per difficile).

Oltre al filtraggio la funzione di selezione dei candidati (`diversifica_candidati`), consente l'unicità delle opzioni basata sull'identificatore univoco (*id*) e seleziona le sequenze con la migliore rilevanza semantica restituita dal database vettoriale. Durante la fase di generazione della coreografia viene applicata la regola di adiacenza: una stessa mossa non può essere riutilizzata all'interno della coreografia due volte consecutivamente (distanza minima pari a un passo). Questo vincolo viene esplicitato sia come istruzione nel prompt di sistema dell'LLM (`build_sequence_plan`), sia controllato dal codice Python, che scarta eventuali duplicati adiacenti durante la validazione della risposta dell'LLM e nel ciclo di fallback automatico. Infine, in fase di esportazione, il `dancer_id` viene utilizzato per raggruppare le mosse relative a ciascun ballerino all'interno del file di output (`animation.txt`) per il plug-in di visualizzazione in Unity.

3.12.5 Fallback e rilassamento filtri

Se il numero di candidati ottenuti dalla prima ricerca non è sufficiente, il tool applica una serie di fallback. Inizialmente viene eseguita una seconda ricerca semantica più ampia, mantenendo i filtri tecnici relativi a genere e, se richiesti, BPM. Se questa procedura non produce un numero sufficiente di candidati, il filtro relativo alla difficoltà può essere temporaneamente rilassato, mantenendo il vincolo di genere. Tale rilassamento viene registrato esplicitamente nel campo `filtri_rilassati`, evitando che la modifica dei criteri di ricerca rimanga nascosta all'utente o alle fasi successive della pipeline. I risultati vengono memorizzati nella cache tramite `salva_cache_chroma` e restituiti attraverso un payload JSON contenente la query semantica utilizzata, i filtri applicati, il numero di corrispondenze ottenute con i criteri originari, gli eventuali filtri rilassati e l'elenco dei candidati recuperati.

3.13 Tool di Build

Il terzo tool, denominato `build_sequence_plan`, si occupa della composizione della coreografia selezionando e ordinando le mosse precedentemente recuperate dal database Chroma dal tool `search_aist_dataset`. Il tool riceve come input il prompt originale, il numero di sequenze richieste e l'insieme dei candidati recuperati da Chroma. Dopo una fase iniziale di normalizzazione e validazione, viene costruito un insieme diversificato di candidati che viene sottoposto al modello Nemotron. Il modello riceve per ogni candidato i relativi metadati, tra cui identificativo della sequenza, genere, BPM, difficoltà, codice musicale e `dancer_id`. La composizione viene effettuata attraverso tre passaggi principali:

3.13.1 Selezione delle sequenze tramite LLM

Nemotron riceve i candidati forniti dal sistema e viene istruito a selezionare esattamente il numero di sequenze richiesto e a ordinarle secondo una progressione coreografica composta da introduzione, sviluppo, picco e finale. La selezione deve essere effettuata utilizzando i metadati disponibili e la richiesta originale dell'utente. Il modello non dispone tuttavia di informazioni sulle pose, sulle articolazioni o sulla continuità fisica dei movimenti. Il risultato prodotto da Nemotron è un JSON contenente gli indici dei candidati selezionati e una breve motivazione relativa al criterio di scelta e ordinamento. Gli indici costituiscono un riferimento ai candidati reali forniti in input.

3.13.2 Validazione e completamento

La risposta del modello viene sottoposta a una fase di validazione Python. Il sistema verifica che ogni indice selezionato appartenga effettivamente all'insieme dei candidati disponibili e che non venga utilizzato più di una volta. Per ogni indice valido, i metadati vengono recuperati e completati direttamente dai dati provenienti da Chroma. Nel caso in cui Nemotron restituisca un numero insufficiente di selezioni valide, Python completa automaticamente la coreografia utilizzando candidati di Chroma non ancora utilizzati. Questo meccanismo impedisce che un errore o una selezione incompleta del modello provochi la perdita del numero di mosse richiesto. Anche in caso di JSON non valido, il sistema tenta di recuperare gli indici dalla risposta testuale del modello; qualora ciò non sia possibile, viene utilizzato il fallback Python. La fase di validazione verifica che le selezioni dell'LLM corrispondano a mosse realmente esistenti e rispettino i vincoli della coreografia. La pipeline di validazione esegue i seguenti controlli sequenziali, illustrati in Figura 3.9:

- **Controllo dell'intervallo degli indici:** verifica che gli indici proposti dal modello rientrino nel range dei candidati forniti nel contesto, indici non validi vengono scartati.

- **Risoluzione dei metadati:** associazione di ciascun indice valido alla relativa mossa presente nei risultati di ChromaDB.
- **Applicazione della regola di adiacenza:** verifica che la mossa correntemente valutata non sia identica a quella immediatamente precedente. Questo permette il riutilizzo della stessa mossa all'interno della coreografia, ma ne impedisce la ripetizione consecutiva (distanza minima di un passo).
- **Completamento deterministico (*fallback*):** se il numero di mosse valide individuate dall'LLM risulta inferiore al numero richiesto dall'utente, il codice Python completa automaticamente la coreografia selezionando i candidati rimanenti più rilevanti.
- **Generazione degli output:** a validazione completata, le mosse vengono serializzate nel file `playlist.json`. Viene infine generato il file `animation.txt`, nel quale i riferimenti alle animazioni vengono raggruppati per `dancer_id` per essere visualizzati mediante il plug-in dedicato per Unity.

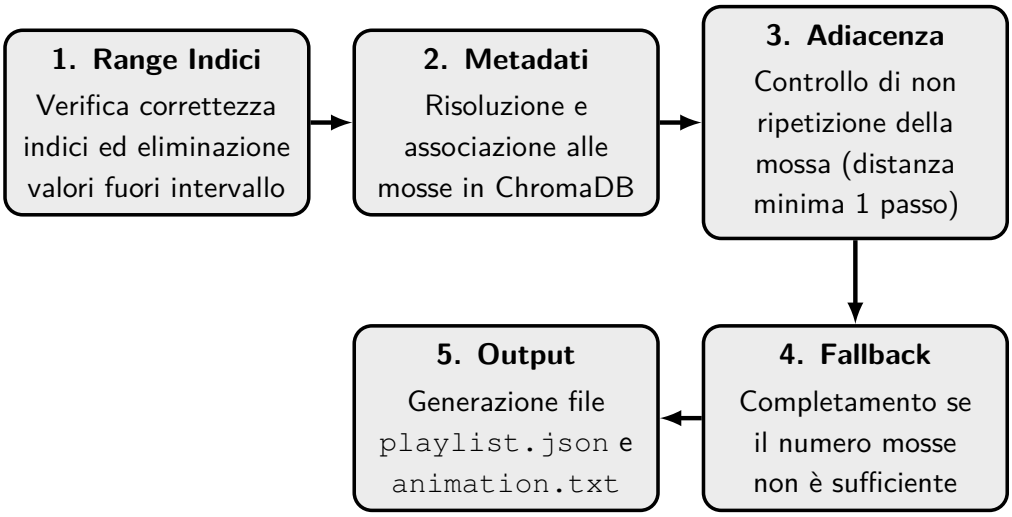


Figura 3.9: Pipeline sequenziale di validazione, controllo adiacenza e serializzazione degli output.

Questo permette una distinzione tra la componente generativa (scelta e ordinamento forniti dall'LLM) e la componente di controllo (verifica formale, vincoli strutturali e generazione dei file).

3.13.3 Composizione coreografia e output finale

Una volta ottenute le mosse finali, queste vengono ordinate e numerate secondo la posizione all'interno della coreografia. Il sistema crea la struttura `playlist`, contenente le mosse ordinate, la descrizione della coreografia, il criterio di selezione, il prompt originale, la durata stimata e le informazioni relative all'utilizzo del modello

e agli eventuali completamenti effettuati da Python. La durata stimata è calcolata prendendo come riferimento un valore di default di otto secondi per ogni mossa AIST++. La playlist viene infine serializzata e salvata nel file `playlist.json`. Parallelamente, il sistema genera `animation.txt`, raggruppando gli identificativi delle animazioni in funzione del relativo `dancer_id`; ogni riga contiene quindi le animazioni associate a uno specifico ballerino. Il risultato finale è una composizione di sequenze già presenti nel dataset, nella quale Nemotron svolge il ruolo di selettore e ordinatore semantico, mentre Python mantiene il controllo sui vincoli strutturali, sulla validità degli identificativi e sul completamento della playlist in caso di errori.

3.13.4 Fallback

Per aumentare la robustezza dell'architettura e prevenire blocchi dell'esecuzione, il sistema integra un meccanismo di fallback implementato in Python. L'attivazione del livello di fallback si verifica al manifestarsi di una delle seguenti condizioni:

- **Mancata risposta o errore dell'LLM:** problemi di congestione della rete, timeout della chiamata API verso OpenRouter o errori interni del servizio;
- **Fallimento del parsing sintattico:** output testuali non interpretabili come JSON e non recuperabili tramite espressioni regolari;
- **Invalidità delle selezioni:** presenza di indici fuori intervallo o non corrispondenti ai candidati forniti nel contesto;
- **Incompletezza della sequenza:** numero di mosse restituite insufficiente o scarto di candidati a causa della violazione del vincolo di adiacenza.

In ciascuno di questi scenari, il sistema può proseguire l'esecuzione mediante il completamento tramite script Python, applicando in modo rigoroso gli stessi vincoli definiti per l'LLM, in particolare la regola di adiacenza, assicurando che nemmeno i candidati inseriti dal fallback possano generare ripetizioni immediate consecutive. L'esito dell'esecuzione viene infine tracciato all'interno del file di output `playlist.json`. La struttura dati include infatti campi di metadati dedicati che registrano il numero di selezioni valide prodotte dal modello, il numero di sequenze colmate automaticamente dal codice Python e lo stato di attivazione del fallback.

3.13.5 Costruzione della playlist e tracciabilità dell'output

Una volta completata la fase di selezione e validazione, il sistema costruisce la playlist finale organizzando le mosse nell'ordine previsto dalla coreografia. La playlist viene serializzata nel file `playlist.json` attraverso una struttura JSON contenente sia le mosse selezionate sia una serie di metadati utili alla descrizione e alla tracciabilità del processo. La struttura JSON viene riepilogata in Tabella 3.7.

Tabella 3.7: Specifica dettagliata dei campi del payload JSON di output.

Campo (Chiave JSON)	**Descrizione e Funzione**
`success`	Valore booleano che indica la corretta conclusione del processo di pianificazione.
`sequenze_ordinate`	Lista delle mosse di danza selezionate, disposte nell'ordine sequenziale di esecuzione.
`descrizione_coreografia`	Descrizione testuale sintetica della coreografia generata.
`criterio_scelta`	Descrizione del criterio utilizzato per la selezione e l'ordinamento delle sequenze.
`durata_stimata_sec`	Stima della durata complessiva della coreografia, espressa in secondi.
`prompt_originale`	Testo della richiesta originale dell'utente, conservato per mantenere la tracciabilità.
`numero_sequenze_richieste`	Numero di sequenze espressamente richiesto dall'utente nel prompt.
`numero_sequenze_generate`	Numero di sequenze effettivamente presenti nella playlist finale generata.
`source`	Indicatore della sorgente dei dati utilizzata per il retrieval (es. `ChromaDB`).
`generata_da`	Modalità di costruzione (es. selezione da modello `Nemotron` vs completamento Python).
`scelte_llm_valide`	Numero di sequenze proposte dall'LLM che hanno superato i controlli di validità.
`scelte_completate_da_python`	Numero di mosse scelte automaticamente dal codice Python tramite fallback.
`fallback_interno_python`	Flag booleano che segnala l'avvenuta attivazione del meccanismo di completamento interno.

La presenza di questi metadati consente di mantenere la tracciabilità dell'output, distinguendo il contributo del modello linguistico dalle operazioni effettuate dal codice Python. In particolare, `scelte_llm_valide` e `scelte_completate_da_python` permettono di determinare quante sequenze sono state effettivamente selezionate dal modello e quante sono state aggiunte successivamente dal sistema per completare la coreografia richiesta. Prima della serializzazione, le sequenze vengono inoltre ordinate in base al campo posizione, che identifica l'ordine progressivo delle componenti della coreografia. Il codice assegna quindi nuovamente la numerazione delle posizioni a partire da 1, in modo che la struttura finale presenti un ordinamento temporale coerente prima dell'esportazione. Il file `playlist.json` costituisce infine l'output del processo di generazione e viene successivamente utilizzato durante la generazione del file `animation.txt`, nel quale le animazioni vengono raggruppate in funzione del ballerino di appartenenza tramite il `dancer_id`.

3.13.6 Generazione di `animation.txt`

Parallelamente alla serializzazione della playlist in formato JSON, la pipeline genera un file di testo denominato `animation.txt`. Questo file testuale viene utilizzato come input al plug-in dedicato per la visualizzazione in Unity. La struttura di `animation.txt` si basa sulle seguenti regole di raggruppamento e ordinamento:

- **Raggruppamento per ballerino:** ogni riga del file testo corrisponde a un singolo `dancer_id` identificato durante la fase di selezione.
- **Ordinamento sequenziale:** all'interno della medesima riga, i file vengono elencati in ordine cronologico da sinistra a destra, separati da uno spazio.

La Tabella 3.8 illustra un esempio con due ballerini.

Tabella 3.8: Rappresentazione matriciale della struttura del file `animation.txt`.

Riga	Mossa 1	Mossa 2	Mossa 3	Mossa 4
1	gBR_sBM_cAll_d05_mBR0_ch10.json	gBR_sBM_cAll_d05_mBR5_ch01.json	—	—
2	gBR_sBM_cAll_d06_mBR2_ch01.json	gBR_sBM_cAll_d06_mBR3_ch01.json	gBR_sBM_cAll_d06_mBR4_ch01.json	gBR_sBM_cAll_d06_mBR5_ch01.json

In questo esempio:

- La prima riga definisce la sequenza ordinata di due mosse assegnata al primo ballerino (d05).
- La seconda riga specifica la sequenza di quattro mosse per il secondo ballerino (d06).
- Gli identificativi delle stringhe rispecchiano la naming rule del dataset AIST.

3.14 Gestione dello stato del workflow

La gestione dello stato è un elemento centrale dell'architettura del progetto. Le diverse fasi del sistema devono poter accedere alle informazioni prodotte dalle fasi precedenti. Lo stato può contenere informazioni quali: prompt originale, risultato del parsing, numero di sequenze richiesto, risultati della ricerca, candidati disponibili, selezioni prodotte dall'LLM, eventuali errori, playlist finale e stato di completamento del processo. Conservare il prompt originale inserito dall'utente è particolarmente importante in quanto permette di verificare che le operazioni successive svolte dal sistema restino coerenti con la richiesta iniziale. Durante l'esecuzione il modello, specialmente nei casi di modelli non particolarmente performanti come Gemma4:e4b, potrebbe produrre una riformulazione della richiesta o interpretare erroneamente alcuni vincoli. Un altro elemento importante è la memorizzazione del numero di sequenze richiesto. Questo parametro viene mantenuto separatamente per evitare che una risposta dell'LLM possa modificare arbitrariamente la quantità di sequenze

da produrre. Lo stato consente inoltre di mantenere i risultati dell'ultima ricerca ChromaDB. Questi risultati possono essere riutilizzati dalla fase di *build* senza dover eseguire nuovamente la ricerca. L'architettura utilizza *cache* per conservare sia la lista dei risultati sia una mappa indicizzata dagli *id* delle sequenze. Questa organizzazione facilita il recupero delle informazioni relative a un candidato.

3.15 Flusso completo di esecuzione

L'intera pipeline di esecuzione si articola nelle seguenti fasi sequenziali:

1. **Interfaccia utente e gestione della richiesta:** l'utente inserisce una richiesta in linguaggio naturale attraverso l'interfaccia web. L'input viene inoltrato al backend sviluppato in Flask, che inizializza e gestisce lo stato del workflow dell'agente e avvia l'esecuzione della pipeline.

2. **Parsing del prompt ed estrazione dei vincoli:** il tool `parse_dance_prompt` analizza la richiesta dell'utente ed estrae le informazioni necessarie alla ricerca, tra cui stile o genere, eventuali caratteristiche espressive, livello di difficoltà, vincoli temporali e numero di sequenze richieste.

3. **Recupero vettoriale e filtraggio dei candidati:** i parametri estratti vengono utilizzati per costruire la query semantica destinata a ChromaDB. Il tool `search_aist_dataset` esegue la ricerca vettoriale e recupera un insieme iniziale di candidati. Le sequenze vengono successivamente sottoposte a filtri basati sui metadati, come stile, difficoltà e BPM, quando richiesti esplicitamente. I risultati vengono inoltre deduplicati sulla base dell'identificatore univoco (id).

4. **Riduzione e preparazione del candidate pool:** attraverso *`build_sequence_plan`*, i candidati recuperati vengono ulteriormente elaborati e diversificati tramite la funzione `diversifica_candidati`. Questa fase elimina le ridondanze basandosi sugli identificatori univoci e seleziona le opzioni a maggior rilevanza semantica da sottoporre all'LLM.

5. **Selezione e ordinamento tramite Nemotron:** il modello linguistico riceve il prompt originale, il numero di mosse di danza richiesto e il pool di candidati. Sulla base delle istruzioni definite nel prompt di sistema, il modello seleziona gli indici dei candidati e li ordina per costruire la progressione della coreografia.

6. **Parsing e normalizzazione della risposta dell'LLM:** la risposta generata dal modello viene acquisita come testo e passata alla funzione *`_estrai_json_da_testo`*, che individua e interpreta la struttura JSON eventualmente presente. Successivamente, `normalizza_output_llm` converte i diversi possibili formati restituiti dal modello nello schema interno utilizzato dal programma. In caso di JSON non valido, il sistema può tentare di recuperare gli indici direttamente dalla risposta testuale.

7. **Validazione deterministica dell'output:** il codice Python verifica che gli indici restituiti dal modello appartengano effettivamente all'insieme dei candidati disponibili e che rientrino nell'intervallo valido; inoltre, viene applicata la regola di adiacenza. Questi controlli vengono eseguiti in modo che l'output finale rispetti i vincoli del sistema anche in presenza di eventuali imprecisioni del modello, nei casi gestiti dalla pipeline.

8. **Completamento tramite Python:** qualora il modello non fornisca un numero sufficiente di selezioni valide, Python completa automaticamente le posizioni mancanti utilizzando candidati non ancora selezionati. Nel risultato finale vengono inoltre distinti il numero di selezioni valide effettuate dall'LLM e il numero di sequenze aggiunte successivamente dal codice Python.

9. **Serializzazione dei file di output:** la sequenza finale viene ordinata e serializzata nei formati utilizzati dalle fasi successive del sistema:

   - **`playlist.json`:** contiene le sequenze ordinate e i relativi metadati, inclusi `numero_sequenze_richieste`, `numero_sequenze_generate`, `scelte_llm_valide`, `scelte_completate_da_python` e le informazioni relative alla modalità di generazione.
   - **`animation.txt`:** contiene gli identificativi dei file di animazione, raggruppati per `dancer_id`.

3.16 Vantaggi dell'architettura

L'architettura sviluppata presenta i seguenti vantaggi:

- **Modularità:** i componenti del sistema sono separati in moduli distinti con responsabilità ben definite (parsing del prompt, recupero vettoriale su ChromaDB, orchestrazione semantica tramite LLM, validazione e serializzazione). Questa separazione facilita la manutenzione e consente l'aggiornamento o la sostituzione di singole componenti (come l'adozione di un diverso vector store o di un nuovo modello linguistico) senza modificare la logica globale.

- **Controllo sui dati:** vincolando le selezioni di Nemotron al solo pool di candidati restituito dal dataset AIST++, il sistema consente di mitigare il rischio che l'LLM generi riferimenti a file di animazione non presenti nella base di conoscenza.

- **Flessibilità d'uso:** l'interfaccia basata sul linguaggio naturale permette all'utente di esprimere richieste stilistiche, temporali ed espressive ad alto livello.

- **Robustezza e tolleranza agli errori:** la pipeline di validazione e i cicli di fallback gestiscono risposte dell'LLM sintatticamente errate, incomplete o prive

di coerenza con i vincoli della coreografia (come la violazione della regola di adiacenza), consentendo che il sistema produca una playlist valida ed eseguibile nei casi gestiti da fallback.

- **Tracciabilità dell'output:** il salvataggio di `playlist.json` conserva un quadro informativo completo della generazione, registrando il prompt originale, il contributo del modello e l'eventuale intervento correttivo del codice Python.
- **Integrazione diretta con l'ambiente 3D Unity:** la generazione del manifesto `animation.txt`, formattato per riga in base al `dancer_id`, fornisce al motore grafico una struttura di caricamento dei metadati.
- **Estendibilità:** l'architettura è predisposta per accogliere nuovi dataset di motion capture, modelli linguistici con contesti più ampi, ulteriori filtri sui metadati.

3.17 Limiti architetturali

I principali limiti dell'architettura si articolano nei seguenti punti:

- **Contrazione della capacità decisionale dell'LLM:** All'interno della funzione `build_sequence_plan`, il set di candidati estratti da ChromaDB viene filtrato e ridotto mediante `diversifica_candidati` per rientrare nella finestra di contesto del modello linguistico. Quando l'utente richiede un numero elevato di sequenze e il candidate pool inviato a Nemotron è contenuto, il margine di selezione del modello si riduce drasticamente. In questi contesti, l'LLM perde gran parte della sua funzione selettiva e semantica, venendo di fatto forzato a includere quasi tutti i candidati disponibili per coprire la lunghezza richiesta.
- **Attenuazione dell'intenzione semantica:** qualora l'output dell'LLM risulti incompleto, sintatticamente non valido o privo di elementi, entra in funzione il ciclo di completamento Python. Tale meccanismo seleziona i candidati rimanenti secondo una logica first-fit basata unicamente sulla disponibilità nell'elenco di ChromaDB. Sebbene ciò consenta la generazione di una playlist valida, un intervento massiccio del fallback rischia di diminuire la coerenza stilistica definita dal prompt dell'utente.
- **Assenza di continuità cinematica:** il sistema non calcola metriche di continuità fisica, di conseguenza, una successione di movimenti ritenuta coerente dall'LLM dal punto di vista stilistico può produrre transizioni brusche durante la riproduzione continua dell'animazione.
- **Coreografia con più ballerini:** La serializzazione del file `animation.txt` raggruppa le tracce per riga in base al `dancer_id` estratto dai metadati

delle sequenze AIST++. Tuttavia, la pipeline non verifica se la richiesta originale fosse intesa per un singolo esecutore: qualora il processo di retrieval o l'LLM selezionino movimenti con `dancer_id` differenti, `animation.txt` genererà automaticamente righe distinte per ciascun `dancer_id`, distribuendo la sequenza su più personaggi nell'ambiente Unity anziché mantenerla su un unico ballerino.

- **Riduzione rigida dell'input nel Parsing del Prompt:** il tool di parsing `parse_dance_prompt` converte il testo in linguaggio naturale in un dizionario strutturato di parametri chiave (stile, BPM, difficoltà) necessari a interrogare ChromaDB. Le caratteristiche espressive, le metafore narrative o le descrizioni del prompt che non trovano una corrispondenza diretta nei campi metadati del dataset vengono filtrate nella fase iniziale, non contribuendo alla composizione della query vettoriale.

Capitolo 4

Risultati

4.1 Obiettivo della fase di test

La fase di testing ha avuto l'obiettivo di verificare il comportamento del sistema agentico durante l'intero processo di generazione della coreografia. In particolare, la valutazione si è concentrata su quattro aspetti principali: la capacità di interpretare correttamente la richiesta dell'utente, l'efficacia della fase di retrieval delle mosse dal database vettoriale, il rispetto del numero di mosse richiesto e la robustezza del sistema agli errori. Il risultato delle esecuzioni dei test viene registrato nel file di log `agent_debug.log`. La maggior parte delle richieste è stata formulata utilizzando lo stile Street Jazz, diversificando il numero di mosse, gli attributi qualitativi e alcune caratteristiche semantiche della richiesta. Questa impostazione ha permesso di verificare il comportamento del sistema in presenza di richieste simili ma non identiche e, soprattutto, di osservare se variazioni nel testo producessero una corretta modifica dei parametri estratti.

4.2 Risultati dell'interpretazione delle richieste

Uno dei primi elementi valutabili è l'interpretazione del prompt inserito dall'utente. Nel log sono presenti complessivamente 20 sessioni di generazione analizzate. Non tutte le sessioni raggiungono correttamente la fase di interpretazione, a causa di problemi esterni, come il congestionamento del traffico verso i server NVIDIA. Considerando quindi esclusivamente le sessioni nelle quali il parser restituisce un output utilizzabile dalla fase successiva, il numero di mosse richiesto dall'utente viene mantenuto correttamente in tutti i casi osservati. Questo risultato è particolarmente significativo perché il numero di mosse costituisce un vincolo esplicito della richiesta e deve essere mantenuto fino alla generazione finale. Anche l'interpretazione delle caratteristiche qualitative mostra una certa coerenza. Ad esempio, per le richieste contenenti il termine energico, il parser ricorre sistematicamente a un insieme di parole chiave relative a dinamismo ed energia, mentre nel caso di richieste contenenti il termine divertente vengono utilizzate parole chiave quali allegro, festoso e giocoso.

4.3 Risultati complessivi del workflow

L'analisi delle sessioni mostra che il sistema è in grado di completare la generazione della coreografia anche in presenza di alcuni errori durante l'esecuzione del percorso principale dell'agente. In particolare, il log evidenzia la presenza di sessioni nelle quali il percorso ReAct non viene completato correttamente, ma il sistema riesce comunque a proseguire attraverso il meccanismo di fallback implementato nel backend. Questo comportamento permette di distinguere il completamento del percorso ReAct dal completamento dell'intero processo di generazione: il mancato completamento del percorso principale non comporta necessariamente l'interruzione dell'elaborazione, poiché il sistema dispone di una procedura di recupero basata sui dati già ottenuti nelle fasi precedenti.

4.4 Risultati della ricerca nel dataset AIST++

La seconda componente valutata riguarda la fase di retrieval effettuata mediante Chroma. Rispetto alle 20 sessioni analizzate, nelle chiamate che restituiscono elementi validi il numero di risultati varia principalmente tra 30 e 42, con un valore medio di 35,8 risultati validi per chiamata. Questo mostra che il sistema riesce generalmente a produrre un insieme di candidati utilizzabili. Nei casi standard, il numero di candidati osservato è frequentemente pari a 40, corrispondente al valore configurato per `SEARCH_K`. Il numero può tuttavia risultare inferiore quando non sono presenti abbastanza elementi che soddisfano contemporaneamente i filtri applicati alla ricerca. Quando il numero di candidati individuati non è sufficiente, la logica implementata nel backend può intervenire attraverso una procedura di fallback che completa il pool dei candidati rilassando progressivamente alcuni dei filtri utilizzati nella ricerca. Gli elementi così recuperati vengono quindi aggiunti ai candidati già individuati, permettendo di ottenere un insieme sufficientemente ampio per la successiva fase di composizione. Un ulteriore elemento osservato riguarda il numero di candidati effettivamente sottoposti al modello per la composizione della coreografia. Tutti i test che raggiungono questa fase presentano un pool ridotto a 10 elementi, secondo il limite configurato per `MAX_CANDIDATES_FOR_LLM`. Tale scelta consente di contenere la quantità di informazioni trasmesse al modello e, conseguentemente, il rischio di superamento dei limiti di contesto o di timeout.

4.5 Rispetto del numero di mosse richiesto

Il rispetto del numero di mosse rappresenta uno dei principali requisiti funzionali del sistema, poiché il valore specificato dall'utente deve essere conservato durante l'intero workflow. In tutti i test conclusi con il ReAct supervisionato, il numero di mosse della coreografia generata coincide con quello richiesto dall'utente. Questo risultato evidenzia l'utilità della supervisione dello stato nella gestione di tale vincolo.

In presenza di errori nel comportamento dell'agente o nella risposta del modello, il sistema utilizza infatti il numero richiesto già memorizzato nello stato del workflow, evitando che tale valore venga modificato durante l'esecuzione.

4.6 Analisi del comportamento dell'agente e del meccanismo di fallback

Uno degli aspetti più rilevanti emersi dall'analisi dei log riguarda la frequenza con cui il percorso ReAct principale non viene completato e il sistema deve ricorrere alla procedura di recupero controllato. Su 20 sessioni di generazione, 10 riportano esplicitamente il messaggio «ReAct non ha completato la sessione», corrispondenti al 50% delle sessioni analizzate. In 9 di questi 10 casi, pari al 90% delle sessioni con ReAct incompleto, il modello non restituisce una struttura JSON valida e, di conseguenza, non produce una selezione di mosse utilizzabile per la costruzione della coreografia. Le anomalie registrate comprendono l'assenza di un JSON nella risposta, strutture JSON non corrette e contenuti non interpretabili nel formato atteso. Il meccanismo di fallback consente al sistema di recuperare questi casi, utilizzando i candidati già individuati durante la fase di retrieval e procedendo alla costruzione della sequenza attraverso la logica implementata nel backend. Il recupero permette quindi di portare a termine la generazione anche quando il modello non è in grado di fornire direttamente un output valido. Questo comportamento evidenzia il ruolo della supervisione dello stato e delle procedure di recupero nel ridurre la dipendenza dal completamento corretto del ciclo ReAct. È tuttavia necessario distinguere il completamento tecnico della generazione dalla qualità della coreografia: il ricorso al fallback consente di ottenere un output, ma non dimostra di per sé che la sequenza prodotta presenti la stessa qualità o coerenza di una composizione realizzata correttamente dal modello.

4.7 Analisi della composizione della coreografia

Il log fornisce inoltre alcune informazioni qualitative relative al criterio utilizzato dal modello durante la costruzione della sequenza finale. Il modello non si limita, almeno nelle risposte osservate, a restituire un insieme non ordinato di movimenti, ma associa alla scelta delle mosse un criterio di organizzazione temporale della coreografia. Tuttavia, si tratta di un criterio dichiarato dal modello e non di una valutazione oggettiva della qualità estetica o della coerenza del movimento. Il log, infatti, non contiene una valutazione effettuata da esperti o utenti sulla qualità percepita delle coreografie.

4.8 Esportazione dell'output e integrazione con il sistema 3D

Un ulteriore risultato riguarda la produzione degli artefatti utilizzati dalle fasi successive del sistema. Per tutte le sessioni concluse, il log registra l'esportazione di playlist.json e di animation.txt, che costituiscono i risultati strutturati della fase di generazione. Il log permette quindi di verificare funzionalmente il passaggio dalla richiesta espressa in linguaggio naturale alla produzione degli output strutturati necessari alle fasi successive. Non consente tuttavia di valutare il successo dell'integrazione con Unity, poiché la corretta importazione e visualizzazione della coreografia nell'ambiente 3D richiedono una verifica specifica della fase di integrazione.

4.9 Limiti della valutazione sperimentale

I risultati ottenuti consentono principalmente di valutare il comportamento tecnico del sistema e il rispetto di alcuni requisiti funzionali, ma non permettono di determinare in maniera oggettiva la qualità estetica delle coreografie generate. In particolare, il log consente di osservare il comportamento del parser, il numero di candidati recuperati durante la fase di retrieval, il rispetto del numero di mosse richiesto e la produzione degli output strutturati. Non contiene invece una valutazione effettuata da esperti o utenti sulla qualità percepita delle coreografie, sulla naturalezza delle transizioni o sulla coerenza complessiva della composizione. Di conseguenza, le osservazioni relative ai criteri utilizzati dal modello per organizzare le mosse devono essere considerate esclusivamente come informazioni qualitative ricavate dalle risposte del modello e non come una misura oggettiva della qualità della coreografia. Un ulteriore limite riguarda la disponibilità e l'affidabilità del servizio del modello linguistico remoto. Una parte delle sessioni analizzate non completa il percorso ReAct principale a causa di problemi nella generazione dell'output strutturato o di problematiche legate alla disponibilità del servizio. Il meccanismo di fallback consente in questi casi di mantenere operativa la pipeline, ma il ricorso a tale procedura evidenzia comunque una dipendenza da un componente esterno che non è completamente sotto il controllo del sistema. Nel complesso, la campagna sperimentale fornisce evidenze a favore della fattibilità del workflow proposto. I test evidenziano la capacità del sistema di trasformare richieste testuali in intent strutturati, recuperare candidati dal dataset AIST++, mantenere il numero di mosse richiesto nei test conclusi con il ReAct supervisionato e produrre gli output necessari alla fase successiva. Allo stesso tempo, i risultati mostrano che la fase maggiormente sensibile è l'interazione con il modello linguistico, sia per la produzione di output strutturati nel formato atteso sia per la dipendenza dalla disponibilità del servizio remoto. Tali aspetti costituiscono pertanto i principali punti di attenzione per eventuali sviluppi futuri del sistema. I risultati sono riepilogati nella Tabella 4.1.

Tabella 4.1: Sintesi dei risultati della fase di test

Indicatore	**Risultato**	**Interpretazione**
Sessioni di generazione analizzate	**20/20**	La campagna sperimentale comprende complessivamente 20 esecuzioni.
Sessioni con ReAct non completato	**10/20 (50%)**	In metà delle sessioni il percorso ReAct principale non viene completato e viene attivato il meccanismo di recupero.
Sessioni con ReAct incompleto e JSON non valido	**9/10 (90%)**	Nella maggior parte dei casi di ReAct incompleto, il problema riguarda la produzione di un output JSON non valido o non interpretabile.
Numero medio di candidati validi nel retrieval	**35,8**	Il sistema produce mediamente un insieme di candidati utilizzabili per la fase di composizione.
Intervallo osservato dei candidati validi	**30–42**	Il numero di candidati varia in funzione dei filtri applicati e della disponibilità di elementi compatibili.
Numero configurato di candidati per il retrieval	**40**	Nei casi standard il retrieval raggiunge frequentemente il valore configurato.
Numero massimo di candidati sottoposti al modello	**10**	Il pool viene ridotto prima della composizione per contenere la quantità di informazioni trasmesse al modello.
Rispetto del numero di mosse richiesto	**100%** dei test conclusi con ReAct supervisionato	Nei test che completano il percorso ReAct supervisionato, il numero di mosse generato coincide con quello richiesto dall'utente.
Produzione degli output strutturati	**100%** delle sessioni concluse	Le sessioni concluse producono `playlist.json` e `animation.txt`.
Valutazione oggettiva della qualità estetica	**Non effettuata**	Il log non contiene valutazioni da parte di esperti o utenti sulla qualità percepita, sulle transizioni o sulla coerenza della coreografia.
Valutazione dell'integrazione Unity tramite il log	**Non determinabile**	La corretta importazione e visualizzazione nell'ambiente 3D richiedono una verifica specifica della fase di integrazione.

Capitolo 5

Discussioni

5.1 Interpretazione dei risultati

I risultati ottenuti durante la fase di test permettono di formulare alcune considerazioni sul comportamento complessivo del sistema progettato. L'analisi delle 20 sessioni considerate mostra che la pipeline è in grado di gestire le principali fasi del processo che conduce dalla richiesta espressa in linguaggio naturale alla produzione di una coreografia, pur evidenziando alcune criticità legate principalmente all'interazione con il modello linguistico remoto.

Uno degli aspetti più significativi riguarda la capacità del sistema di separare le diverse responsabilità all'interno del workflow, ovvero, l'interpretazione della richiesta, la ricerca delle mosse, la composizione della coreografia e la produzione degli output non sono infatti demandate esclusivamente al modello linguistico, ma vengono inserite all'interno di una pipeline nella quale intervengono anche componenti deterministici. Questa organizzazione consente di mantenere sotto il controllo del software alcune informazioni che devono rimanere stabili durante l'intera esecuzione.

In particolare, il numero di mosse richiesto dall'utente rappresenta un vincolo che deve essere conservato senza variazioni durante il workflow. I test conclusi correttamente con il ReAct supervisionato mostrano che tale vincolo viene mantenuto nella generazione finale, e questo è coerente con la scelta di memorizzare nello stato del workflow le informazioni già determinate nelle fasi precedenti, evitando che il modello possa modificare autonomamente parametri già acquisiti.

I risultati evidenziano, inoltre, che la correttezza del sistema non può essere valutata esclusivamente sulla base del completamento del percorso ReAct. Una parte significativa delle sessioni analizzate non conclude infatti correttamente il percorso principale dell'agente, ma la pipeline può comunque proseguire attraverso il meccanismo di fallback. Ne deriva una distinzione importante tra il comportamento del componente agentico e quello dell'intero sistema: un errore nell'esecuzione del percorso ReAct non coincide necessariamente con un fallimento completo della generazione.

5.2 Ruolo della supervisione nel workflow agentico

I risultati ottenuti permettono di discutere direttamente una delle principali scelte progettuali del sistema, ovvero l'introduzione di una supervisione dello stato sopra il comportamento ReAct.

Il paradigma ReAct permette al modello di alternare l'elaborazione della richiesta all'utilizzo degli strumenti disponibili, ma lascia al modello una parte significativa del controllo sulla sequenza delle operazioni.

La presenza dello stato condiviso e delle condizioni implementate nel workflow consente di ridurre i casi di agent drift, mantenendo memoria dell'avvenuto completamento delle principali fasi dell'elaborazione, le variabili di stato utilizzate dal sistema rappresentano quindi non soltanto informazioni operative, ma anche un meccanismo di controllo sull'esecuzione dell'agente.

I risultati relativi al rispetto del numero di mosse richiesto forniscono un'indicazione a favore di questa scelta. Nei test che completano il percorso ReAct supervisionato, il valore richiesto viene mantenuto fino alla generazione della sequenza finale. Il dato non consente di attribuire il risultato esclusivamente alla supervisione, poiché anche altri componenti del sistema partecipano alla gestione del vincolo, ma mostra che l'introduzione di un controllo esplicito dello stato è compatibile con il mantenimento di parametri funzionali durante l'esecuzione.

La supervisione assume inoltre un ruolo importante nella gestione degli errori poiché, quando il workflow non segue il percorso previsto, il backend può fare riferimento alle informazioni già raccolte nelle fasi precedenti invece di affidarsi nuovamente alla libera interpretazione del modello. In questo modo, il comportamento agentico viene inserito all'interno di una struttura più vincolata, nella quale alcune decisioni possono essere demandate al modello mentre altre rimangono sotto il controllo della logica applicativa.

5.3 Efficacia del meccanismo di fallback

Il comportamento osservato nelle sessioni con ReAct incompleto rappresenta uno degli elementi più rilevanti della valutazione sperimentale.

Nel 50% delle sessioni analizzate il log segnala il mancato completamento del percorso ReAct, nella maggior parte di questi casi, pari al 90%, il problema è associato alla produzione di un output JSON non valido o non interpretabile secondo il formato previsto dal sistema. Questo dato evidenzia una criticità concreta dell'approccio adottato: anche quando il modello dispone delle informazioni necessarie per eseguire il compito, la sua risposta può non rispettare la struttura richiesta dal backend. Questo risultato suggerisce che l'affidabilità del workflow non dipende esclusivamente dalla capacità del modello di individuare mosse coerenti con la richiesta dell'utente, ma anche dalla sua capacità di restituire informazioni strutturate nel formato atteso dal sistema. Diventa importante prevedere controlli in grado di gestire eventuali

risposte non conformi, riducendo la dipendenza dal corretto completamento del ciclo ReAct.

Da questo punto di vista, il meccanismo di fallback assume una funzione essenziale. La logica implementata permette di utilizzare le informazioni già ottenute durante il retrieval e di proseguire con la costruzione della sequenza senza richiedere che il modello completi nuovamente tutte le operazioni previste dal percorso principale.

Il risultato è un'architettura nella quale l'errore del modello non determina necessariamente l'interruzione dell'intera pipeline. Tale caratteristica contribuisce alla continuità operativa del sistema, almeno rispetto agli errori osservati durante la campagna di test.

Il fallback consente quindi la possibilità di produrre un output, ma non costituisce una garanzia della qualità della coreografia ottenuta. Di conseguenza, la robustezza tecnica e la qualità artistica devono essere considerate come proprietà distinte.

Il meccanismo di fallback può quindi essere interpretato come uno strumento di continuità operativa piuttosto che come un sostituto completo del comportamento del modello, il suo scopo principale è ridurre il rischio che un errore causi il fallimento dell'intera pipeline.

5.4 Retrieval semantico e gestione del contesto

I risultati relativi alla fase di retrieval confermano il ruolo centrale del database vettoriale all'interno dell'architettura.

Nelle chiamate che restituiscono elementi validi, il numero di candidati recuperati varia tra 30 e 42, con una media di 35,8 risultati. Il valore risulta coerente con la configurazione del sistema, che utilizza un valore di `SEARCH_K` pari a 40 nei casi standard. Le variazioni osservate dipendono dalla disponibilità di elementi compatibili con i vincoli applicati alla ricerca.

Questo comportamento evidenzia una caratteristica importante dell'approccio adottato: il retrieval non ha l'obiettivo di individuare direttamente la coreografia finale, ma di costruire un insieme di candidati semanticamente compatibili con la richiesta. La composizione della sequenza viene effettuata successivamente, sulla base di un insieme più ristretto di elementi.

La procedura di fallback prevista nella ricerca assume in questo contesto una funzione complementare. Quando l'applicazione simultanea dei filtri produce un numero insufficiente di risultati, il sistema può rilassare progressivamente alcuni vincoli e integrare i candidati recuperati con ulteriori elementi. Tale comportamento evita che criteri di ricerca troppo restrittivi impediscano alla fase successiva di disporre di un insieme sufficientemente ampio per la composizione.

Un ulteriore elemento di interesse è rappresentato dalla riduzione del numero di candidati trasmessi al modello. Sebbene il retrieval possa produrre fino a 40 risultati, il modello riceve un massimo di 10 candidati, secondo il valore configurato per `MAX_CANDIDATES_FOR_LLM`. Questa separazione tra fase di recupero e fase di

composizione consente di limitare la quantità di informazioni presenti nel contesto del modello.

La scelta assume particolare rilevanza all'interno del progetto, nel quale il modello linguistico viene utilizzato attraverso un servizio remoto soggetto a limiti di contesto e a possibili timeout.

Nel complesso, i risultati supportano l'utilità dell'integrazione tra retrieval semantico e generazione linguistica: il database vettoriale svolge il compito di restringere lo spazio delle possibili mosse, mentre il modello opera su un insieme già filtrato di candidati anziché sull'intero dataset.

5.5 Architettura ibrida tra componenti deterministiche e generative

Un elemento che emerge dall'analisi complessiva è la natura ibrida del sistema sviluppato. La pipeline combina infatti componenti basate su comportamento deterministico con una componente generativa costituita dal modello linguistico.

Tra le operazioni maggiormente vincolate rientrano la validazione e normalizzazione dei dati estratti, la gestione dello stato, i filtri applicati al retrieval, la selezione dei candidati disponibili e la produzione dei file di output. Al contrario, l'interpretazione semantica della richiesta e la composizione della sequenza coinvolgono capacità generative del modello.

Questa separazione permette di attribuire al modello le attività per le quali la flessibilità linguistica costituisce un vantaggio, mantenendo invece sotto il controllo del software le operazioni per le quali è richiesta maggiore prevedibilità.

Il comportamento osservato durante i test mostra l'importanza di tale distinzione. Gli errori principali non riguardano infatti esclusivamente la logica applicativa, ma si concentrano soprattutto nella fase di interazione con il modello; in particolare, la produzione di strutture JSON conformi al formato atteso e la presenza di controlli deterministici e di procedure di fallback consentono di limitare l'effetto di tali errori sul resto della pipeline.

L'architettura ibrida può quindi essere considerata una conseguenza diretta della necessità di integrare le capacità generative di un LLM all'interno di un processo software che deve comunque rispettare vincoli funzionali definiti in modo esplicito.

5.6 Qualità della coreografia e limiti interpretativi dei risultati

La valutazione sperimentale mette in evidenza una distinzione tra la correttezza funzionale del sistema e la qualità della coreografia generata.

Dal log è possibile osservare che il modello utilizza, nelle risposte analizzate, un criterio di organizzazione temporale delle mosse, tuttavia, questa osservazione descrive

il comportamento dichiarato dal modello e non costituisce una misura oggettiva della qualità della sequenza prodotta.

Non sono stati infatti raccolti dati provenienti da esperti di danza o da utenti finali relativamente alla qualità percepita delle coreografie, alla naturalezza delle transizioni, alla coerenza complessiva della sequenza o alla corrispondenza tra la richiesta testuale e il risultato dal punto di vista artistico.

Di conseguenza, i risultati ottenuti consentono di trarre conclusioni principalmente sulla dimensione tecnica e funzionale del sistema. È possibile verificare soltanto che una richiesta venga interpretata, che vengano recuperati candidati compatibili, che il numero di mosse venga mantenuto nei test conclusi correttamente e che gli output previsti vengano prodotti.

Una valutazione completa dovrebbe pertanto affiancare alle metriche tecniche anche criteri specifici per la qualità della composizione.

5.7 Implicazioni dei risultati per il sistema sviluppato

Considerati nel loro insieme, i risultati forniscono indicazioni utili sul ruolo delle diverse componenti dell'architettura.

L'interpretazione del prompt consente di trasformare una richiesta formulata in linguaggio naturale in una rappresentazione strutturata utilizzabile dal sistema. Il retrieval restringe quindi lo spazio delle possibili mosse, mentre la fase di composizione utilizza il sottoinsieme recuperato per costruire la coreografia finale. La supervisione dello stato mantiene i vincoli attraverso il workflow e il fallback permette di gestire parte degli errori che si verificano durante l'interazione con il modello.

Ne deriva una pipeline nella quale il risultato finale non dipende da un'unica operazione generativa, ma dall'interazione di più componenti con responsabilità differenti. Tale organizzazione è coerente con l'obiettivo di realizzare un sistema nel quale l'utente possa formulare la richiesta senza dover conoscere direttamente la struttura interna del dataset AIST++.

Allo stesso tempo, la frequenza dei casi di ReAct incompleto mostra che il comportamento del modello costituisce ancora uno dei principali elementi di variabilità dell'intero processo, il sistema dispone di meccanismi per gestire tali situazioni, ma la necessità di ricorrere al fallback in una parte consistente delle sessioni indica che la stabilità dell'interazione con il modello rappresenta un aspetto da migliorare.

Capitolo 6

Conclusioni

6.1 Conclusioni e contributi del lavoro

Il lavoro presentato in questa tesi ha avuto come obiettivo la progettazione e lo sviluppo di un sistema agentico in grado di trasformare una richiesta espressa dall'utente in linguaggio naturale in una coreografia composta da mosse di danza appartenenti al dataset AIST++, predisponendo successivamente gli output necessari alla visualizzazione in ambiente tridimensionale mediante plug-in dedicato in Unity. Per raggiungere tale obiettivo è stata sviluppata un'architettura che integra un Large Language Model all'interno di un workflow agentico basato su LangGraph e sul paradigma ReAct, affiancato da un meccanismo di supervisione dello stato. Il sistema utilizza inoltre un database vettoriale realizzato mediante Chroma e una serie di strumenti software specializzati per l'interpretazione della richiesta, la ricerca dei movimenti e la costruzione della coreografia. L'integrazione con il backend Flask, l'interfaccia web e il plug-in Unity sviluppato nel precedente lavoro di Felcher[2] consente di completare il flusso dalla descrizione testuale alla visualizzazione tridimensionale della coreografia. Il contributo del lavoro svolto consiste nella possibilità di interagire con il sistema attraverso una richiesta formulata liberamente, senza richiedere all'utente una conoscenza diretta della struttura interna del dataset AIST++ e l'organizzazione del workflow agentico attraverso una separazione tra le capacità decisionali del modello linguistico e le operazioni software deterministiche. L'LLM è utilizzato per interpretare la richiesta, comporre la coreografia e coordinare le diverse operazioni, mentre retrieval, validazione e produzione degli output sono affidati a strumenti specializzati. La supervisione dello stato consente inoltre di mantenere sotto controllo l'avanzamento del workflow e di preservare le informazioni già determinate durante l'esecuzione. Nei test conclusi, tale meccanismo ha permesso di mantenere il rispetto del numero di mosse richiesto dall'utente e di generare i file di output correttamente anche in presenza di errori in diverse fasi dell'esecuzione. La valutazione sperimentale ha inoltre evidenziato come il corretto funzionamento di un sistema agentico non dipenda esclusivamente dalla capacità del modello linguistico. In alcune sessioni, il percorso ReAct non è stato completato correttamente, principalmente a causa della produzione di output strutturati non conformi al formato previsto. La presenza di un meccanismo di fallback ha tuttavia consentito di

utilizzare le informazioni già disponibili e di proseguire l'esecuzione. Questo evidenzia come affiancare alla componente generativa meccanismi espliciti di validazione, supervisione e gestione degli errori possa aiutare nel portare a compimento l'intera pipeline. L'architettura sviluppata costituisce pertanto un'applicazione dell'Agentic AI al problema della composizione automatica di coreografie, mantenendo separate le capacità interpretative e decisionali del modello dalle operazioni eseguite in modo deterministico dai componenti software.

6.2 Limiti del lavoro

Nonostante i risultati ottenuti siano coerenti con gli obiettivi funzionali del progetto, il sistema presenta alcuni limiti che devono essere considerati.

Un primo limite riguarda la dipendenza dal modello linguistico remoto; i test hanno evidenziato che il comportamento del modello può variare in funzione della corretta produzione dell'output strutturato e della disponibilità del servizio utilizzato. Questo elemento introduce nella pipeline una componente esterna che non è completamente sotto il controllo dell'applicazione e che può influenzarne il comportamento complessivo.

Un secondo limite è legato alla frequenza con cui il percorso ReAct non viene completato nelle sessioni analizzate. Sebbene il fallback permetta di mantenere operativa la pipeline, il ricorso a tale procedura mostra che il percorso principale non è ancora sufficientemente stabile da poter essere considerato affidabile in tutte le condizioni osservate.

Un ulteriore limite riguarda il numero dei test effettuati, 20 sessioni, utilizzando prevalentemente richieste relative allo stile Street Jazz. I risultati ottenuti descrivono quindi il comportamento del sistema nelle condizioni sperimentate, ma non permettono di estendere automaticamente le conclusioni a tutte le possibili combinazioni di generi, vincoli e richieste formulate dagli utenti.

È inoltre necessario distinguere la correttezza funzionale dalla qualità della coreografia; la valutazione effettuata si è concentrata principalmente sulla corretta interpretazione del prompt utente, sul retrieval delle mosse, sul rispetto dei vincoli e sulla produzione degli output. Non è stata invece introdotta una metodologia specifica per misurare la qualità estetica delle mosse, la naturalezza delle transizioni o la coerenza della composizione dal punto di vista di utenti o esperti di danza.

Tali limiti non hanno impedito, nel contesto dei test effettuati, il raggiungimento degli obiettivi funzionali del progetto, ma definiscono il perimetro entro il quale devono essere interpretati i risultati ottenuti.

6.3 Sviluppi futuri

A partire dai risultati e dai limiti emersi, è possibile individuare diverse direzioni di sviluppo.

Un primo sviluppo potrebbe riguardare la robustezza nella gestione degli output prodotti dal modello linguistico, in relazione ai casi in cui il percorso ReAct non viene completato correttamente a causa della produzione di strutture JSON non conformi al formato previsto. Potrebbero essere introdotti meccanismi più avanzati di validazione, correzione automatica e rigenerazione degli output non validi, con l'obiettivo di ridurre il numero di errori che interrompono il percorso principale e limitare il ricorso alle procedure di fallback.

Un secondo possibile sviluppo deriva dalla dipendenza dal modello linguistico remoto. Il confronto tra differenti modelli potrebbe consentire di valutare l'influenza delle loro caratteristiche sulla capacità di interpretare correttamente le richieste, produrre output strutturati validi e completare in modo stabile il percorso ReAct. Tale confronto permetterebbe inoltre di analizzare il compromesso tra qualità delle risposte, stabilità del comportamento e tempi di esecuzione.

Un ulteriore sviluppo considera l'ampliamento della valutazione sperimentale, necessario per superare il limite rappresentato dalle 20 sessioni analizzate e dalla prevalenza di richieste relative allo stile Street Jazz. Sarebbe quindi opportuno estendere i test a un insieme più ampio e diversificato di richieste, includendo differenti generi di danza, combinazioni di vincoli e descrizioni caratterizzate da diversi livelli di specificità. Una valutazione più estesa permetterebbe di verificare la generalizzabilità dei risultati e di individuare eventuali comportamenti dipendenti da particolari caratteristiche delle richieste.

Per quanto riguarda la valutazione della qualità della composizione coreografica, che nella presente tesi è stata distinta dalla verifica della correttezza funzionale della pipeline, potrebbero essere introdotte metriche automatiche per analizzare la compatibilità tra movimenti consecutivi, la continuità delle transizioni, la varietà delle sequenze e il rispetto dei vincoli semantici presenti nella richiesta. In particolare, l'analisi della continuità tra movimenti consecutivi potrebbe fornire una misura più significativa della qualità della coreografia rispetto alla sola verifica della correttezza dei singoli movimenti recuperati e selezionati.

Infine, la valutazione automatica potrebbe essere affiancata da una valutazione con utenti o persone dotate di competenze specifiche nel settore della danza, mediante questionari o protocolli di valutazione. Tale analisi permetterebbe di considerare aspetti difficilmente misurabili attraverso i soli criteri funzionali, quali la qualità percepita della coreografia, la coerenza complessiva della sequenza, la naturalezza delle transizioni e la corrispondenza tra il risultato generato e la richiesta iniziale dell'utente.

Bibliografia

[1] J. Schneider, "Generative to agentic ai: Survey, conceptualization, and challenges," *arXiv preprint arXiv:2504.18875*, 2025.

[2] L. Felcher, "SMPLHViewer: Gui-based unity solution to visualize smpl-h animations," https://github.com/jaggernout04/SMPLHViewer, 2026.

[3] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "React: Synergizing reasoning and acting in language models," *arXiv preprint arXiv:2210.03629*, 2022.

[4] R. Li, S. Yang, D. A. Ross, and A. Kanazawa, "Ai choreographer: Music conditioned 3d dance generation with aist++," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE, 2021, pp. 13 381–13 392.

[5] L. Siyao, W. Yu, T. Gu, C. Lin, Q. Wang, C. Qian, C. C. Loy, and Z. Liu, "Bailando: 3d dance generation by actor-critic gpt with choreographic memory," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 11 050–11 059.

[6] J. Tseng, R. Castellon, and K. Liu, "Edge: Editable dance generation from music," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2023, pp. 448–458.

[7] A. De Filippo and M. Milano, "Robotic choreography creation through symbolic ai techniques," in *International Conference on Entertainment Computing*. Springer, 2023, pp. 346–351.

[8] M. Braccini, A. De Filippo, M. Lombardi, M. Milano *et al.*, "Swarm intelligence: A novel and unconventional approach to dance choreography creation," in *CEUR WORKSHOP PROCEEDINGS*, vol. 3810. CEUR-WS, 2024, pp. 162–172.

[9] M. Braccini, A. De Filippo, M. Lombardi, and M. Milano, "Towards a human-in-the-loop framework for swarm intelligence-driven dance generation," in *International Conference on Image Analysis and Processing*. Springer, 2025, pp. 625–636.

[10] C. Shang, X. Chen, L. An, J. Zhang, Y. Zhang, Y. Liu, and X. Yang, "Danceagent: Dance movement refinement with llm agent," *IEEE Transactions on Visualization and Computer Graphics*, 2025.

[11] Y. Yu and X. Zhang, "Developing a dynamic music and dance coordination multi-agent system based on real-time audience feedback analysis," *Advanced Electromagnetics*, vol. 15, no. 3, pp. 7841–7850, 2026.

[12] L. He, H. Pang, Q. Gan, X. Shen, Z. Zhang, Y. Liu, G. Fang, B. Liu, K. Sheng, S. Zeng *et al.*, "Cutscene agent: An llm agent framework for automated 3d cutscene generation," *arXiv preprint arXiv:2604.25318*, 2026.

[13] S. Tsuchida, S. Fukayama, M. Hamasaki, and M. Goto, "Aist dance video database: Multi-genre, multi-dancer, and multi-camera database for dance information processing," in *Proceedings of the 20th International Society for Music Information Retrieval Conference, ISMIR 2019*, Delft, Netherlands, Nov. 2019.

[14] SentenceTransformers, "all-MiniLM-L6-v2: Pretrained sentence embedding model," https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2, 2021.

[15] Pallets Projects, "Flask documentation (v3.1.x)," 2026. [Online]. Available: https://flask.palletsprojects.com/

[16] NVIDIA, "Nemotron 3 ultra: Open, efficient mixture-of-experts hybrid mamba-transformer model for agentic reasoning," 2026, white Paper. [Online]. Available: https://research.nvidia.com/labs/nemotron/files/NVIDIA-Nemotron-3-Ultra-Technical-Report.pdf

[17] LangChain, "LangGraph: Low-level orchestration framework for stateful agents," https://github.com/langchain-ai/langgraph, 2024.

[18] H. Chase, "LangChain," https://github.com/langchain-ai/langchain, Oct. 2022.