

UNIMORE

UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Università degli Studi di Modena e Reggio Emilia

Dipartimento di Scienze e Metodi dell'Ingegneria

Corso di Laurea Magistrale in

Ingegneria Gestionale

A. A. 2025/2026

LLM e Agenti AI per il controllo di sistemi IoT: analisi e sperimentazione

Relatore: Prof. Franco Zambonelli

Correlatore: Prof. Stefano Mariani

Laureando: Simone Suraci

Sommario

La seguente tesi sperimentale considera la tematica dell'utilizzo dei Large Language Models (LLM) come interfacce in linguaggio naturale per l'interazione con sistemi IoT e cyber-fisici complessi. In particolare, il lavoro si concentra sul controllo e sull'interrogazione di un ambiente HVAC simulato, nel quale le richieste formulate dall'utente devono essere tradotte in operazioni tecniche compatibili con dispositivi, sensori e attuatori del simulatore.

Nello specifico, la problematica affrontata riguarda la valutazione della capacità degli LLM di veicolare le richieste di un utente all'interno della struttura tecnica di un sistema: il modello deve individuare il dispositivo corretto, selezionare il sensore o l'attuatore appropriato e generare una chiamata eseguibile che rispetti quanto richiesto. Per analizzare questo problema è stata sviluppata un'integrazione tra Smart Building Simulator (SBSim) e il modello Qwen2.5:7B, eseguito localmente tramite Ollama.

A tale scopo è stata progettata un'architettura in grado di confrontare diverse configurazioni di prompt e dataset di richieste a difficoltà crescente, utilizzando strategie di costruzione incrementale del contesto, meccanismi di function calling e flussi di interazione ispirati al paradigma ReAct, al fine di valutare le potenzialità del modello e individuarne i principali limiti.

I risultati mostrano che il modello può interagire efficacemente con il simulatore quando dispone di una rappresentazione minima della struttura e di un supporto applicativo per il mapping tra linguaggio naturale e identificativi tecnici. Al contrario, l'assenza di questi due elementi incide negativamente sulle prestazioni del modello utilizzato, che risulta essere poco efficace nella costruzione autonoma del contesto e nella correzione iterativa delle proprie azioni.

Il contributo specifico del lavoro consiste nello sviluppo dell'integrazione software, nella definizione dei tool, nella progettazione dei prompt e nella valutazione sperimentale dei fattori che influenzano la corretta traduzione delle richieste.

Indice

Introduzione 1
Scenario tecnologico e lavori correlati 5
2.1. Evoluzione dei LLM 5
2.1.1. Qwen2.5 9
2.2. Sistemi agentici basati su LLM 9
2.3. Uso di strumenti esterni (tool use) 11
2.3.1. Function calling 13
2.4. ReAct e interazione iterativa 15
2.5. Ollama 16
2.6. Lavori correlati 17
2.6.1. LLM come interfacce per smart home e IoT 17
2.6.2. Agenti LLM e tool use per l'interazione con sistemi esterni 19
2.6.3. Controllo e monitoraggio di ambienti energetici e HVAC 20
2.6.4. Posizionamento della presente tesi in relazione ai lavori correlati 20
Architettura del sistema 22
3.1. Struttura del sistema 22
3.1.1. SBSim 23
3.1.2. Qwen-Ollama 25
3.1.3. Layer Python 26
3.2. Flusso esecutivo della simulazione 27
3.2.1 Interpretazione della richiesta 29
3.2.2 Elaborazione della richiesta 29
3.2.3 Esecuzione dell'operazione 30
3.2.4 Aggiornamento degli attuatori e avanzamento della simulazione 31
3.3. Tools implementati 32
3.4. Gestione degli attuatori 33
3.4.1. Test action vector vs override 34
Test Sperimentali 37
4.1. Configurazione generale degli esperimenti 38
4.2. Prima fase: confronto esplorativo tra prompt 38
4.2.1. System Prompt utilizzati 39

4.2.2. Dataset ristretto 42
4.2.3. Risultati prima fase 43
4.2.4. Analisi del confronto esplorativo 44
4.3. Seconda fase: confronto con dataset esteso 45
4.3.1. System Prompt utilizzati 45
4.3.2. Dataset esteso 46
4.3.3. Risultati seconda fase 49
4.3.4. Analisi del ruolo della struttura e del resolver 49
4.4. Terza fase: costruzione incrementale del contesto 51
4.4.1. Analisi della costruzione incrementale del contesto 52
4.5. Quarta fase: esperimento ReAct-Like 55
4.5.1. Configurazione dell'esperimento 55
4.5.2. Risultati ottenuti 58
4.5.3. Analisi dell'implementazione iterativa ReAct-like 59
4.6. Quinta fase: esperimento ReAct testuale 64
4.6.1. Configurazione dell'esperimento 64
4.6.2. Risultati ottenuti 66
4.6.3. Analisi dell'implementazione ReAct testuale few-shot 67
Analisi complessiva dei risultati 71
Conclusioni e sviluppi futuri 74
6.1. Conclusioni 74
6.2. Sviluppi futuri 75
Bibliografia 77
Appendici 80
Appendice A – Stralci di codice del layer applicativo 80
A.1 Definizione dei tool disponibili 80
A.2 Gestione delle tool call generate dal modello 81
A.3 Normalizzazione degli argomenti 82
A.4 Risoluzione dei riferimenti ai dispositivi 83
A.5 Esecuzione delle query e dei comandi verso SBSim 84
Ringraziamenti 85

Capitolo Primo

Introduzione

Negli ultimi anni i Large Language Models (LLM) hanno dimostrato notevoli capacità nell'interpretazione del linguaggio naturale e nella risoluzione di compiti complessi, consentendo loro non solo di generare testo, ma anche di interagire attivamente con ambienti software e dispositivi reali, attraverso il processing delle istruzioni dell'utente[1].

Tuttavia, molte applicazioni richiedono che l'utente (o il sistema) fornisca informazioni strutturate, parametri ben formattati o comandi riconducibili a dispositivi e azioni note. Questo limita la naturalezza dell'interazione, soprattutto quando l'utente formula richieste generiche, e motiva lo studio di soluzioni capaci di collegare linguaggio naturale e struttura tecnica del sistema[2]. Rimane quindi aperta la questione di quanto un LLM sia realmente in grado di comprendere autonomamente un sistema sconosciuto, identificandone i componenti, interpretandone la struttura e utilizzando correttamente le funzionalità messe a disposizione.

La rilevanza di questo tema deriva dal fatto che molti sistemi cyber-fisici, tra cui smart building, ambienti IoT e sistemi HVAC, combinano componenti hardware e software, sensori, controllori, funzioni di monitoraggio, automazione e analisi dei dati. Questa articolazione rende l'interazione con tali sistemi non sempre immediata per l'utente finale e si inserisce in un quadro più ampio di barriere all'adozione (facilità d'uso percepita, controllo percepito, privacy, cybersecurity, costi e "ansia" tecnologica)[3]. In questi contesti, pertanto, il linguaggio naturale rappresenta un'interfaccia potenzialmente più intuitiva rispetto a comandi rigidi o configurazioni manuali, ma introduce anche il problema di tradurre richieste spesso generiche, incomplete o formulate ad alto livello in operazioni tecniche precise e compatibili con il sistema controllato[2].

Il presente lavoro si colloca in questo scenario e ha come obiettivo principale l'esplorazione della capacità di un Large Language Model di controllare e interrogare un ambiente HVAC simulato tramite richieste formulate in linguaggio naturale. In particolare, la tesi analizza il ruolo della conoscenza strutturale del sistema, dei meccanismi di mapping semantico e delle possibili strategie di costruzione incrementale

del contesto operativo, valutando sperimentalmente quali elementi siano più rilevanti per ottenere una corretta interpretazione delle richieste.

A tale scopo, in questo lavoro è stata realizzata in ambiente Linux un'integrazione tra il simulatore Smart Building Simulator (**SBSim**)[4] e il modello "**qwen2.5:7b**", appartenente alla famiglia dei modelli Qwen rilasciati da Alibaba[5], dotandolo di strumenti dedicati per l'esplorazione della struttura del sistema, la lettura dei sensori e la modifica dei parametri di controllo. Il programma è stato sviluppato in Python.

L'ambiente scelto per la sperimentazione (SBSim) è un sistema HVAC simulato, composto da zone, dispositivi, sensori e attuatori. Tale ambiente consente di simulare il comportamento di un edificio dotato di sistema HVAC, nel quale le azioni di controllo sugli attuatori influenzano l'evoluzione dello stato dell'ambiente attraverso le leggi fisiche implementate del simulatore. L'ambiente non si limita quindi a rappresentare una semplice struttura statica di dispositivi e parametri, ma permette di osservare l'effetto delle modifiche nel tempo su grandezze come temperatura, consumi energetici, costi, emissioni e comfort[6].

Il sistema presenta una struttura tecnica articolata, con nomi di dispositivi e parametri non sempre immediatamente riconducibili al linguaggio naturale dell'utente. Ad esempio, una richiesta formulata come "dimmi la temperatura della stanza 10" deve essere tradotta nell'identificazione del dispositivo corretto e del relativo sensore disponibile nel simulatore. Analogamente, una richiesta come "aumenta la temperatura dell'acqua della caldaia" deve essere associata all'attuatore corretto del boiler. Questa doppia caratteristica rende SBSim un ambiente adatto sia per verificare l'integrazione operativa tra LLM e simulatore, sia per analizzare la capacità del modello di effettuare il mapping tra linguaggio naturale e struttura tecnica del sistema.

Tale scenario permette quindi di esplorare il problema del **grounding**, ovvero il collegamento tra la richiesta espressa in linguaggio naturale e gli elementi concreti del sistema simulato o del mondo fisico con cui l'LLM interagisce. Il modello deve infatti selezionare la funzionalità coerente con la richiesta, individuare il dispositivo interessato, scegliere il sensore o l'attuatore appropriato e generare comandi compatibili con l'interfaccia programmatica del simulatore[7]. Lo studio di questo passaggio è rilevante

perché rappresenta uno dei nodi principali per l'utilizzo affidabile degli LLM come interfacce verso sistemi esterni complessi.[8]

Per rendere più robusta questa interazione, è stato introdotto un livello intermedio sviluppato in Python, incaricato di validare gli argomenti prodotti dal modello, risolvere riferimenti parziali o semantici ai dispositivi e impedire che richieste incoerenti vengano inoltrate direttamente al simulatore.

È stata posta particolare attenzione non solo all'accuratezza nell'esecuzione dei comandi, ma anche alla capacità del modello di costruire una rappresentazione interna del sistema a seconda delle informazioni disponibili in partenza, valutando quali elementi risultino determinanti per una corretta interpretazione delle richieste dell'utente e quali limiti emergano nel processo di comprensione autonoma del contesto.

La tesi riporta nel secondo capitolo i contenuti bibliografici principali riferiti allo scenario tecnologico e ai lavori correlati. In questa sezione vengono introdotti i concetti principali relativi all'intelligenza artificiale generativa, ai Large Language Models, ai sistemi agentici, al function calling e all'integrazione tra modelli linguistici e sistemi esterni. Vengono inoltre presentate le tecnologie utilizzate nel lavoro (Qwen e Ollama) e i concetti di base relativi a smart building, sistemi HVAC e ambienti simulati per attività di controllo e analisi.

Nel terzo capitolo viene illustrata l'architettura del sistema integrato all'interno dell'ambiente SBSim. Viene inizialmente presentata la struttura dell'ambiente utilizzato, con riferimento a zone, dispositivi, sensori, attuatori e azioni disponibili. Successivamente viene illustrata l'architettura implementata per collegare il modello LLM al simulatore, descrivendo il flusso di elaborazione di una richiesta. Una parte specifica del capitolo è dedicata alle scelte architetturali adottate, tra cui la gestione degli attuatori tramite action vector e override manuale.

Il quarto capitolo presenta i test sperimentali sviluppati. In questa sezione vengono descritti la configurazione degli esperimenti, i dataset di input, i prompt confrontati e i criteri adottati per valutare la correttezza delle risposte. Gli esperimenti sono organizzati in più fasi, con l'obiettivo di analizzare il comportamento del modello al variare del livello

di dettaglio del prompt, della disponibilità della struttura del sistema e dei meccanismi di supporto presenti nel backend.

Il quinto capitolo è dedicato all'analisi dei risultati sperimentali, con l'obiettivo di discutere il ruolo del prompt, della struttura del dominio, dei meccanismi di risoluzione dei riferimenti, della costruzione incrementale del contesto e delle strategie iterative ispirate a ReAct.

Infine, il sesto capitolo presenta le conclusioni del lavoro, sintetizzando i principali risultati ottenuti e individuando alcune possibili direzioni di sviluppo futuro.

Capitolo Secondo

Scenario tecnologico e lavori correlati

2.1. Evoluzione dei LLM

Negli ultimi anni l'intelligenza artificiale generativa (e in particolare i Large Language Models) ha assunto un ruolo centrale nell'elaborazione del linguaggio naturale. Questi modelli sono infatti in grado di interpretare, generare e rielaborare testo in modo flessibile, rendendo possibile il loro utilizzo non solo per attività di generazione linguistica, ma anche come interfacce per sistemi più ampi, quali software e applicazioni esterne.

Uno dei passaggi fondamentali nello sviluppo dei moderni modelli linguistici è rappresentato dall'introduzione dell'architettura **Transformer** da parte di Vaswani et al.[9]. I modelli linguistici precedenti (come le reti ricorrenti RNN o LSTM) elaboravano il testo in modo strettamente sequenziale, calcolando uno stato nascosto dopo l'altro. Questa impostazione impediva di sfruttare pienamente il calcolo parallelo durante l'addestramento, ponendo di conseguenza dei limiti alla scalabilità dei modelli su grandi quantità di dati.

Il Transformer ha rivoluzionato questo approccio: a differenza delle tradizionali reti ricorrenti o convoluzionali, nelle quali l'elaborazione della sequenza avviene in modo progressivo, il Transformer si basa su meccanismi di "attenzione" che permettono al modello di considerare le relazioni tra le diverse parti dell'input in modo più efficace e parallelizzabile. Il modello, in particolare, costruisce una rappresentazione del contesto assegnando pesi diversi ai token presenti nell'input. La **Scaled Dot-Product Attention** costituisce il meccanismo di base tramite il quale il Transformer calcola l'attenzione: il modello confronta Query (Q) e Key (K) per ottenere dei punteggi di rilevanza, normalizza tali punteggi tramite softmax e li utilizza per pesare i Value (V). La **Multi-Head Attention** estende questo principio eseguendo più meccanismi di attention in parallelo, permettendo al modello di catturare relazioni diverse tra i token della stessa sequenza (figura 2.1).

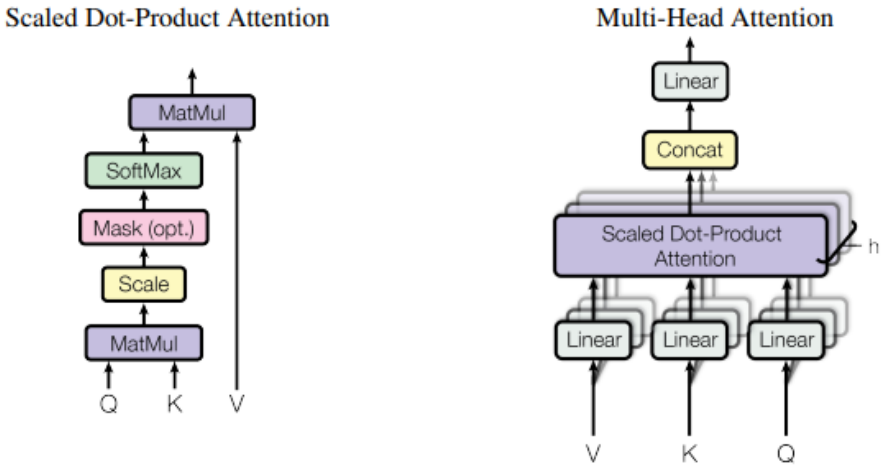


Figura 2.1 Rappresentazione architetturale del meccanismo di attenzione e di attenzione in parallelo.

Fonte: A. Vaswani et al., "Attention Is All You Need", 2017[9]

La struttura originale del Transformer (figura 2.2) è organizzata secondo un'architettura **encoder-decoder**. L'encoder riceve la sequenza di input e la trasforma in una rappresentazione interna contestuale, mentre il decoder utilizza tale rappresentazione per generare progressivamente la sequenza di output.

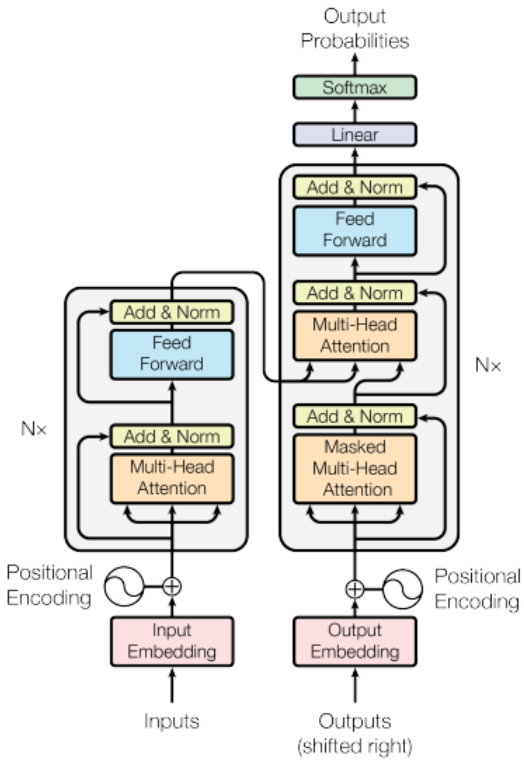


Figura 2.2 Modello architetturale encoder-decoder del Transformer.

Fonte: A. Vaswani et al., "Attention Is All You Need", 2017[9]

Nonostante la maggior parte dei moderni LLM (Qwen, GPT, Llama) adottino solitamente architetture decoder-only derivate dal Transformer, lo schema encoder-decoder originario del Transformer rappresenta il riferimento storico e concettuale da cui si è sviluppata gran parte dei moderni modelli linguistici, ponendo il meccanismo di attenzione al centro dell'elaborazione del testo. Questa caratteristica ha favorito l'addestramento di modelli di dimensioni sempre maggiori, ponendo le basi per lo sviluppo degli LLM contemporanei.

L'efficienza computazionale del Transformer ha quindi permesso ai ricercatori di scalare le dimensioni dei modelli (in termini di parametri) e dei dataset di pre-training di diversi ordini di grandezza. Con l'aumento della scala dei modelli e dei dati utilizzati per l'addestramento, gli LLM hanno progressivamente mostrato capacità più generali di adattamento a compiti diversi. Un contributo importante in questa direzione è rappresentato dal lavoro di Brown et al.[10], nel quale viene evidenziata la capacità dei modelli linguistici di svolgere compiti differenti a partire da istruzioni fornite direttamente nel prompt. In questo paradigma, noto come "**in-context learning**", il

modello non viene necessariamente riaddestrato per ogni specifico compito, ma adatta il proprio comportamento sulla base del contesto ricevuto in input.

L'adattamento al task può avvenire con diversi livelli di informazione fornita nel prompt. Si parla di **zero-shot prompting** quando il compito viene descritto soltanto tramite istruzioni, senza esempi aggiuntivi; di **one-shot prompting** quando viene fornito un singolo esempio dimostrativo; infine di **few-shot prompting** quando il prompt contiene più esempi dimostrativi utili a guidare il comportamento del modello. Queste modalità si distinguono dal tradizionale **fine-tuning** (figura 2.3), nel quale il modello viene invece addestrato ulteriormente su esempi specifici attraverso aggiornamenti dei pesi della rete.

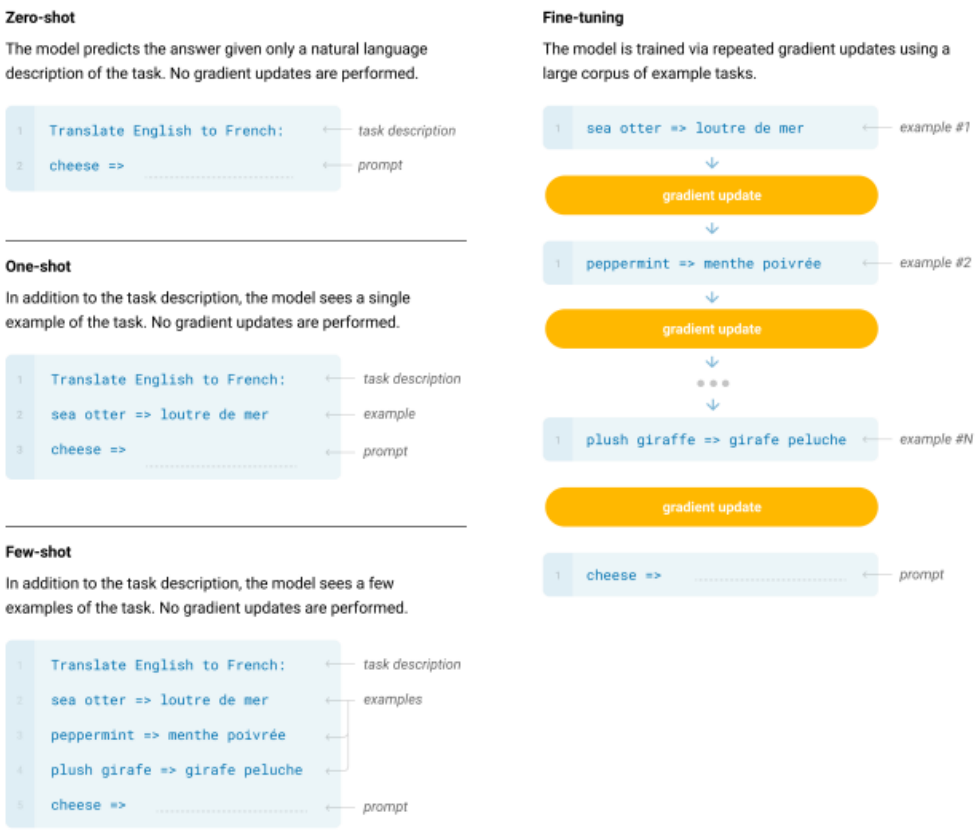


Figura 2.3 Esempi di in-context learning vs fine-tuning tradizionale

Fonte: T. B. Brown et al., "Language Models Are Few-Shot Learners"[10]

2.1.1. Qwen2.5

All'interno di questo scenario si colloca la famiglia di modelli Qwen2.5, che si configura come una serie completa di Large Language Models basati su un'architettura decoder di tipo Transformer. I modelli sono disponibili in diverse dimensioni (da 0.5 a 72 miliardi di parametri), con versioni sia base sia instruction-tuned. Rispetto alle versioni precedenti, le fonti ufficiali riportano miglioramenti nella capacità dei modelli di seguire istruzioni, generare testi lunghi, comprendere dati strutturati e produrre output strutturati[11]. Questi aspetti sono particolarmente rilevanti per il presente lavoro, nel quale il modello non viene utilizzato soltanto per generare testo libero, ma anche per interpretare richieste dell'utente e produrre istruzioni strutturate verso strumenti esterni.

In questa tesi viene utilizzata la variante qwen2.5:7b, corrispondente alla famiglia Qwen2.5-7B. La scelta di utilizzare una variante da 7B (ossia 7 miliardi di parametri) è legata alla necessità di disporre di un modello sufficientemente capace nell'interpretazione del linguaggio naturale e nella generazione di output strutturati, ma al tempo stesso eseguibile localmente con risorse hardware limitate. L'interesse verso modelli di questo tipo non riguarda quindi soltanto la qualità della generazione testuale, ma anche la possibilità di impiegarli come componenti interpretative all'interno di sistemi più ampi. In particolare, un LLM può essere utilizzato per trasformare una richiesta formulata in linguaggio naturale in una decisione operativa, come la scelta di una funzione da richiamare o di un insieme di parametri da fornire a un sistema esterno.

Questa prospettiva costituisce il collegamento con i meccanismi di function calling e con i sistemi agentici, approfonditi nei paragrafi successivi.

2.2. Sistemi agentici basati su LLM

Negli sviluppi più recenti, i Large Language Models vengono spesso utilizzati come componenti centrali di sistemi più ampi, all'interno dei quali interpretano obiettivi e prendono decisioni tramite l'osservazione e l'interazione con l'ambiente circostante. Questa evoluzione porta al concetto di sistema agentico basato su LLM, nel quale il modello linguistico assume un ruolo simile a quello di un controllore centrale, incaricato di comprendere il contesto e selezionare le azioni più appropriate.

Il concetto di agente autonomo precede lo sviluppo degli LLM. Nella definizione classica proposta da Franklin e Graesser, un agente è un sistema situato in un ambiente, capace di percepirlo e di agire su di esso nel tempo, perseguendo determinati obiettivi e influenzando così le percezioni future[12]. Questa definizione mette in evidenza tre elementi fondamentali: la presenza di un ambiente, la capacità di osservare o ricevere informazioni da esso, e la possibilità di compiere azioni che ne modificano lo stato o producono conseguenze rilevanti.

Nel caso degli agenti basati su LLM, tali funzioni vengono reinterpretate alla luce delle capacità dei modelli linguistici. L'ambiente può assumere diverse forme a seconda dell'architettura: esso può essere costituito da una conversazione, da un insieme di documenti, da un software esterno, da una base di conoscenza o da un sistema fisico o simulato. Le osservazioni assumono spesso la forma di messaggi, dati restituiti da strumenti esterni o risultati di precedenti azioni. Le azioni, invece, possono consistere nella generazione di una risposta testuale, nella scelta di una funzione da richiamare, nell'interrogazione di una API (Application Programming Interface) o nella modifica dello stato di un sistema.

Wang et al.[13] descrivono gli agenti autonomi basati su LLM come sistemi composti da più moduli funzionali, tra cui profilo, memoria, pianificazione e azione. Il modulo di profiling consente di definire il ruolo e le caratteristiche dell'agente, orientandone il comportamento rispetto al compito da svolgere. Il modulo di memoria permette di conservare e recuperare informazioni rilevanti, distinguendo tra informazioni presenti nel contesto immediato e conoscenze archiviate esternamente. Il modulo di pianificazione riguarda la capacità di scomporre obiettivi complessi in passaggi intermedi, mentre il modulo di azione rappresenta il collegamento tra le decisioni dell'agente e gli effetti prodotti sull'ambiente (figura 2.4).

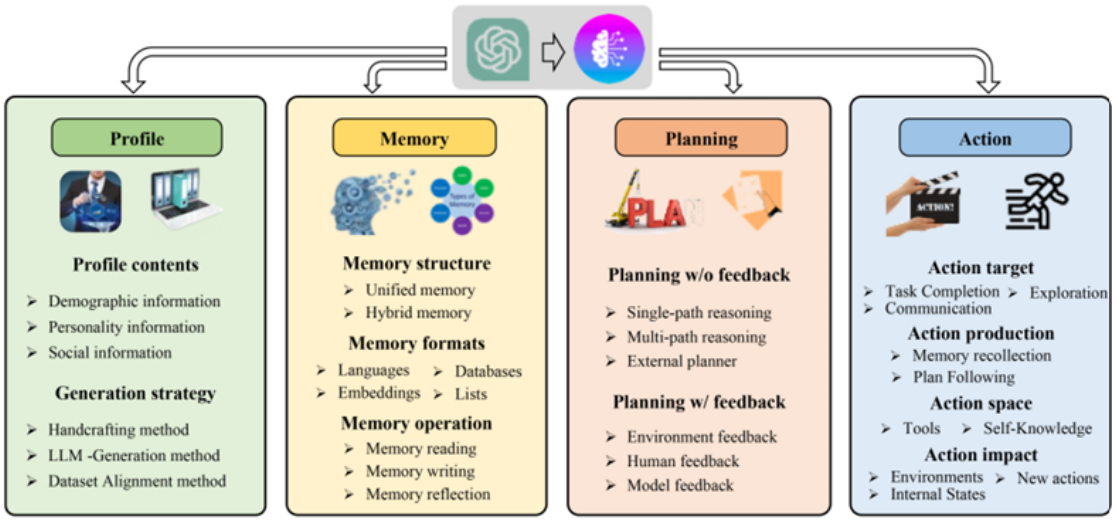


Figura 2.4 Framework concettuale degli agenti autonomi basati su LLM, articolato nei quattro moduli funzionali

Fonte: L. Wang et al., "A Survey on Large Language Model Based Autonomous Agents", 2024[13]

Ai fini del presente lavoro, i moduli più rilevanti sono soprattutto quelli relativi alla pianificazione e all'azione, in quanto occorre valutare se un modello linguistico sia in grado di trasformare una richiesta dell'utente in un'operazione eseguibile su un sistema esterno. In questo senso, l'LLM non agisce come componente autonomo isolato, ma come parte di un'architettura più ampia in cui il modello interpreta le informazioni che recepisce dall'ambiente.

Il paragrafo successivo approfondisce l'utilizzo di strumenti esterni e il meccanismo del function calling come modalità di interazione tra LLM e sistemi software.

2.3. Uso di strumenti esterni (tool use)

Un elemento centrale nei sistemi agentici basati su LLM è la possibilità di utilizzare strumenti esterni, indicata in letteratura con il termine "**tool use**" (figura 2.5). I modelli linguistici, infatti, presentano alcune limitazioni intrinseche: la loro conoscenza deriva dalla fase di addestramento, non hanno accesso diretto allo stato aggiornato di sistemi esterni e possono produrre risposte linguisticamente plausibili ma non necessariamente corrette o verificabili. Inoltre, alcune attività richiedono operazioni deterministiche, accesso a dati aggiornati o interazione con software specifici, aspetti che non possono essere affidati esclusivamente alla generazione testuale.

L’uso di strumenti esterni permette di estendere le capacità operative del modello. Attraverso tool, API o funzioni applicative, l’LLM può accedere a informazioni non presenti nel suo contesto iniziale, eseguire calcoli, interrogare database, recuperare documenti o interagire con componenti software. In questo modo, il modello non viene utilizzato soltanto come sorgente di conoscenza, ma come componente in grado di decidere quando sia necessario ricorrere a uno strumento e come utilizzare il risultato ottenuto.[14]

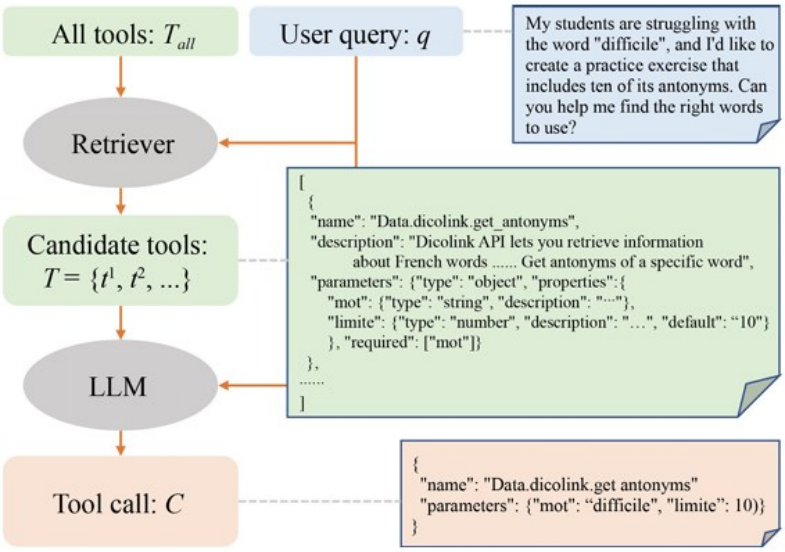


Figura 2.5 Esempio di schema generale del tool use in un sistema basato su LLM

Fonte: Y. Zheng, H. Wei, J. Lu et al., “Investigating effective LLM-based in-context tool use, 2026[14]

Un riferimento importante in questa direzione è il progetto “**Toolformer**” [7], che mostra come un modello linguistico possa essere addestrato a utilizzare API esterne, apprendendo quando chiamare uno strumento, quale strumento selezionare, quali argomenti fornire e come integrare il risultato nella generazione successiva. Sebbene l’approccio di Toolformer riguardi una specifica metodologia di addestramento, il contributo è rilevante dal punto di vista concettuale perché evidenzia il passaggio da modelli puramente testuali a modelli capaci di incorporare strumenti esterni nel proprio processo di generazione.

2.3.1. Function calling

Il function calling rappresenta una modalità pratica per realizzare l'interazione tra LLM e strumenti esterni. In letteratura e nella documentazione tecnica, i termini tool use e function calling sono talvolta utilizzati per indicare lo stesso concetto generale, ossia la capacità di un modello linguistico di interagire con strumenti esterni. La distinzione tra i due termini non va quindi intesa come una separazione teorica rigida, ma soprattutto come una differenza di livello e di implementazione.[15] Mentre il tool use indica in senso generale la capacità di un modello di utilizzare strumenti, il function calling introduce un meccanismo più strutturato: le funzioni disponibili vengono descritte al modello tramite uno schema che specifica il nome della funzione, il suo scopo e i parametri accettati. Il modello può quindi decidere se rispondere direttamente all'utente oppure produrre una chiamata strutturata verso una delle funzioni disponibili.

Questo meccanismo risponde a una differenza fondamentale tra linguaggio naturale e sistemi software: il linguaggio naturale è flessibile, ambiguo e spesso incompleto; al contrario, una funzione software richiede input precisi, tipizzati e coerenti con una specifica interfaccia. Il function calling permette quindi di costruire un ponte tra questi due livelli: da un lato l'utente può formulare una richiesta in forma libera, dall'altro il sistema riceve una rappresentazione più strutturata dell'azione da eseguire.

Il processo può essere descritto in più passaggi (figura 2.6). Inizialmente, l'applicazione fornisce al modello l'elenco dei tool disponibili, includendo per ciascuno una descrizione e lo schema degli argomenti attesi. Quando riceve la richiesta dell'utente, il modello valuta se uno di questi strumenti sia necessario per completare il compito. Se decide di utilizzare un tool, genera una chiamata contenente il nome della funzione e i relativi parametri. L'esecuzione della funzione, tuttavia, non viene svolta direttamente dall'LLM: è il codice applicativo esterno a validare gli argomenti, eseguire l'operazione e restituire eventualmente il risultato al modello.

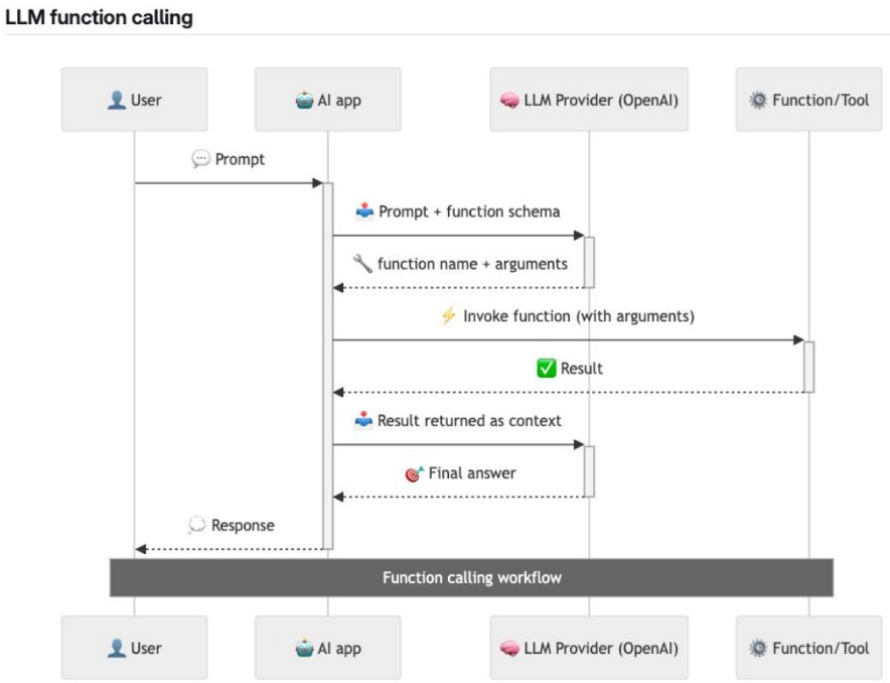


Figura 2.6 Flusso operativo del function calling

Fonte: Awesome Testing, "How does Playwright MCP work?"[16]

Questa separazione tra interpretazione ed esecuzione è particolarmente importante dal punto di vista dell'affidabilità. L'LLM può essere utilizzato per comprendere l'intento dell'utente e proporre una chiamata strutturata, ma l'esecuzione effettiva resta controllata dal sistema esterno, che può verificare la correttezza dei parametri, impedire operazioni non valide e gestire eventuali errori. In questo modo, il function calling consente di sfruttare la flessibilità del linguaggio naturale senza rinunciare al controllo richiesto dalle interfacce software.

Questo passaggio è particolarmente importante nei contesti applicativi in cui la correttezza non dipende solo dalla coerenza linguistica della risposta, ma anche dalla capacità di produrre operazioni valide rispetto a un sistema esterno. In tali scenari, il modello deve essere in grado di selezionare lo strumento appropriato, compilare correttamente i parametri richiesti e utilizzare l'eventuale risultato restituito dallo strumento per completare il task.

Una volta definito il modo in cui un LLM può richiamare strumenti esterni tramite function calling, resta da considerare come tale interazione possa essere organizzata quando una singola chiamata non è sufficiente. Il paragrafo successivo introduce quindi

il paradigma ReAct, che estende l'uso degli strumenti in una logica iterativa basata sull'alternanza tra ragionamento, azione e osservazione.

2.4. ReAct e interazione iterativa

Un ulteriore sviluppo nell'ambito dei sistemi agentici basati su LLM è rappresentato dal paradigma ReAct (Reasoning and Acting)[17]. Tale paradigma nasce dall'idea che il modello possa migliorare il proprio comportamento alternando passaggi di ragionamento e azioni verso l'ambiente esterno. A differenza di un approccio "one-shot" (in cui il modello produce una risposta in un unico passaggio), ReAct prevede una interazione iterativa in cui il modello ragiona sullo stato corrente, seleziona un'azione, riceve un'osservazione e aggiorna il proprio comportamento sulla base delle informazioni ottenute.

Nel paradigma ReAct, il ragionamento viene espresso attraverso paragrafi in linguaggio naturale (figura 2.7), spesso indicati come **Thought**, mentre le azioni rappresentano chiamate verso strumenti, API o ambienti esterni. Dopo ogni azione, l'ambiente restituisce una **Observation**, cioè un'informazione che viene reinserita nel contesto del modello e utilizzata per decidere il passaggio successivo. Questo schema consente di costruire un ciclo ad anello chiuso, nel quale il modello può correggere o completare il proprio comportamento sfruttando progressivamente i risultati delle azioni precedenti.

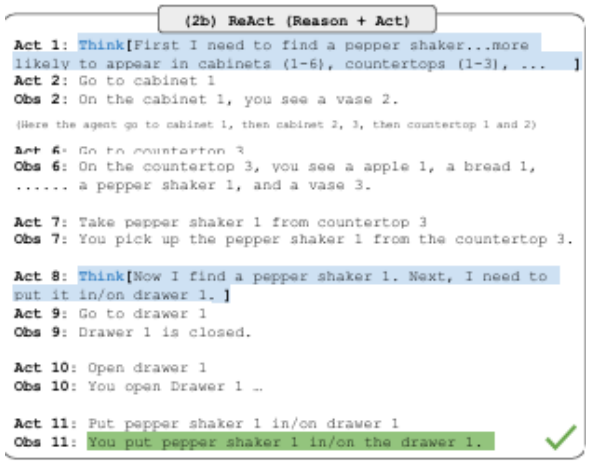


Figura 2.7 Esempio applicativo di ReAct: il modello alterna ragionamento, azioni e osservazioni

Fonte: S. Yao et. al, "ReAct: Synergizing Reasoning and Acting in Language Models", 2023[17]

Il contributo principale di ReAct consiste quindi nell'unire due dimensioni che, prese separatamente, presentano alcuni limiti: il solo ragionamento testuale può aiutare il modello a scomporre un problema o pianificare una risposta, ma non gli consente di verificare le proprie ipotesi rispetto a dati esterni. Al contrario, l'uso di azioni senza una componente esplicita di ragionamento può risultare meno efficace nei compiti che richiedono pianificazione, gestione degli errori o interpretazione del contesto.

L'alternanza tra reasoning e acting, integrata all'utilizzo delle osservazioni precedenti, permette di combinare questi due aspetti, rendendo il modello più adatto a compiti multi-step, nonché a scenari in cui il modello deve interagire con una struttura tecnica esterna e può non disporre inizialmente di tutte le informazioni necessarie per portare a termine correttamente il compito richiesto.[18]

2.5. Ollama

Nel presente lavoro il modello Qwen2.5:7B viene eseguito localmente attraverso **Ollama**, un framework e ambiente di esecuzione open-source per Large Language Models che consente di scaricare ed eseguire modelli linguistici su una macchina locale, esponendoli ad applicazioni esterne tramite API e librerie dedicate[19]. In questo senso, tale framework rappresenta un ambiente di esecuzione e gestione che consente di utilizzare modelli LLM in locale e di interrogarli tramite interfacce, fungendo da ponte tra il codice applicativo e il modello selezionato.

In particolare, per l'inferenza locale Ollama espone un server accessibile tramite endpoint HTTP locale (normalmente all'indirizzo **http://localhost:11434/api**), mentre le librerie ufficiali consentono di integrare le chiamate al modello direttamente in applicazioni Python o JavaScript[20]. Ollama prevede inoltre il supporto al tool calling, attraverso il quale il modello può indicare il nome della funzione da utilizzare e gli argomenti da fornire; l'esecuzione effettiva della funzione rimane tuttavia responsabilità dell'applicazione esterna[21].

Questo aspetto viene applicato nel presente lavoro, nel quale Ollama rende disponibile Qwen come componente interrogabile dal codice Python. Il layer applicativo sviluppato nella tesi riceve quindi l'eventuale tool call prodotta dal modello tramite Ollama, ne valida gli argomenti, la traduce in una richiesta compatibile con SBSim e restituisce al

modello o all'utente il risultato dell'operazione. In questo modo, Ollama permette di eseguire il modello localmente e di renderlo accessibile al codice applicativo.

2.6. Lavori correlati

L'utilizzo dei Large Language Models (LLM) come interfacce in linguaggio naturale per l'interazione con sistemi IoT, edifici intelligenti e ambienti cyber-fisici rappresenta un ambito di ricerca in forte crescita. La progressiva diffusione di dispositivi connessi, sensori e attuatori rende infatti sempre più complessa la gestione di ambienti domotici, energetici e industriali, soprattutto quando l'interazione avviene tramite configurazioni manuali, regole predefinite o interfacce poco flessibili.

In questo contesto, diversi lavori recenti hanno studiato come le capacità linguistiche e di ragionamento degli LLM possano essere utilizzate per rendere più naturale l'accesso a sistemi tecnici complessi. Allo stesso tempo, tali applicazioni mettono in evidenza una criticità ricorrente: nei sistemi fisici o simulati non è sufficiente che il modello produca una risposta plausibile in linguaggio naturale, ma è necessario che l'azione generata sia corretta, sicura e compatibile con la struttura effettiva dell'ambiente controllato.

I lavori correlati possono essere ricondotti a tre filoni principali: l'utilizzo degli LLM come interfacce per smart home e sistemi IoT, l'impiego di agenti LLM e meccanismi di tool use per l'interazione con sistemi esterni, e l'applicazione degli LLM al controllo o al monitoraggio di ambienti intelligenti, energetici e cyber-fisici.

2.6.1. LLM come interfacce per smart home e IoT

Un primo filone di ricerca riguarda l'impiego degli LLM come agenti conversazionali o moduli di interfaccia ad alto livello in ambienti domotici e IoT. L'obiettivo di questi sistemi è consentire all'utente di esprimere intenzioni operative o automazioni desiderate tramite linguaggio naturale, lasciando al modello il compito di interpretare la richiesta e tradurla in azioni compatibili con l'infrastruttura sottostante.

Tra i lavori rilevanti in questo ambito, Sasha (King et al.)[2] propone un processo decisionale sequenziale basato su LLM per la pianificazione di obiettivi domotici. L'idea alla base è utilizzare il modello non solo come generatore di testo, ma come componente

capace di ragionare su uno stato domestico e individuare le azioni necessarie per raggiungere un determinato obiettivo.

Un approccio affine è rappresentato da SAGE (Smart Home Agent with Grounded Execution), sviluppato da Rivkin et al[22]. In questo caso, l'LLM viene inserito all'interno di un'architettura agentica orientata all'esecuzione di comandi in ambiente smart home. Il sistema utilizza strumenti specializzati per gestire preferenze dell'utente, disambiguare dispositivi presenti nell'ambiente e monitorare lo stato dei sensori. L'aspetto più interessante, rispetto al presente lavoro, è il concetto di grounded execution: la richiesta dell'utente deve essere collegata ad azioni effettivamente eseguibili e non rimanere una semplice interpretazione linguistica.

Un ulteriore contributo è rappresentato da IoTGPT (Yu et al.)[23], che affronta alcune criticità tipiche dell'applicazione degli LLM al controllo IoT, come la latenza, il costo delle inferenze e la natura non deterministica delle risposte generate. IoTGPT adotta una strategia basata su decomposizione dei task e memoria: le istruzioni dell'utente vengono scomposte in sotto-compiti, tradotte in comandi API e successivamente memorizzate in una struttura gerarchica. In questo modo, il sistema può riutilizzare sottocompiti già appresi, riducendo il numero di chiamate all'LLM e favorendo una maggiore stabilità del comportamento.

Nel contesto dell'End-User Development e della sostenibilità energetica domestica, Barricelli et al. propongono invece la piattaforma Green Smart Home[24]. Il sistema integra un digital twin dell'abitazione con un agente conversazionale basato su LLM, permettendo all'utente di creare e gestire automazioni domestiche tramite linguaggio naturale. La piattaforma considera anche aspetti legati ai consumi energetici e alla possibile presenza di conflitti tra regole, mostrando come gli LLM possano essere utilizzati non solo per interpretare richieste, ma anche per supportare l'utente nella configurazione di comportamenti più sostenibili e coerenti.

La letteratura evidenzia tuttavia che la traduzione diretta dal linguaggio naturale alle azioni operative presenta ancora diversi rischi. In questa direzione si colloca HomeBench (Li et al.)[25], un benchmark per la valutazione di LLM in scenari smart home con istruzioni valide e invalide, riferite sia a singoli dispositivi sia a più dispositivi contemporaneamente. I risultati mostrano che anche modelli avanzati possono incontrare difficoltà in presenza di richieste ambigue, istruzioni non valide o riferimenti a dispositivi

e funzionalità inesistenti. Questo sembra suggerire che il solo modello linguistico non sia sufficiente a garantire un grounding operativo affidabile.

2.6.2. Agenti LLM e tool use per l'interazione con sistemi esterni

Un secondo filone di lavori riguarda l'impiego degli LLM in contesti nei quali il modello non si limita a generare testo, ma deve produrre azioni eseguibili su un sistema esterno. Rispetto ai concetti generali di tool use, function calling e ReAct già introdotti nei paragrafi §2.3 e §2.4, l'interesse di questi lavori risiede soprattutto nel modo in cui il linguaggio naturale viene collegato a strumenti, API, primitive di controllo o azioni fisiche.

Nel campo della robotica e del controllo embodied, Code as Policies (Liang et al.)[26] propone l'utilizzo di LLM addestrati alla generazione di codice per produrre policy robotiche in Python a partire da istruzioni formulate in linguaggio naturale. In questo approccio, il modello genera programmi che richiamano API di percezione e controllo, trasformando una richiesta dell'utente in una sequenza di operazioni eseguibili dal robot. Il contributo è rilevante perché mostra un possibile modo di collegare linguaggio, rappresentazione dell'ambiente e azione, anche se attraverso la generazione di codice libero e non tramite tool call vincolate come nel presente lavoro.

Un approccio diverso è rappresentato da RT-2 (Brohan et al.)[27], che introduce il concetto di Vision-Language-Action model. In questo caso, il modello integra percezione visiva, istruzioni linguistiche e generazione di azioni robotiche, permettendo di trasferire conoscenza appresa da dati web e vision-language a compiti di controllo fisico. Pur essendo distante dal dominio HVAC, RT-2 è utile per inquadrare il filone più generale dei foundation models applicati all'azione in ambienti fisici.

Questi lavori evidenziano un aspetto comune alla presente tesi: affinché un LLM possa interagire con un sistema esterno, è necessario definire un collegamento tra linguaggio naturale e azioni operative. Nel caso della robotica tale collegamento può assumere la forma di codice, primitive di controllo o azioni motorie; nel caso della presente tesi, invece, esso viene realizzato tramite tool predefiniti, argomenti strutturati e un layer applicativo incaricato di validare la corrispondenza tra richiesta utente, dispositivi, sensori e attuatori del simulatore.

2.6.3. Controllo e monitoraggio di ambienti energetici e HVAC

Infine, un terzo filone riguarda l'applicazione degli LLM al controllo e al monitoraggio di ambienti tecnici specializzati, come sistemi di gestione energetica degli edifici, impianti HVAC e sistemi di automazione industriale.

Nel contesto del Building Energy Management, Zhang et al.[28] propongono un framework di interpretable machine learning control applicato a sistemi energetici degli edifici. In questo caso, l'LLM non viene utilizzato per controllare direttamente l'ambiente in tempo reale, ma per interpretare e spiegare in linguaggio naturale le decisioni prodotte da strategie di controllo basate su machine learning. L'integrazione tra valori di interpretabilità, come gli Shapley values, e conoscenza fisica dell'edificio permette di generare spiegazioni più comprensibili per gli operatori, aumentando la trasparenza del sistema di controllo.

In ambito industriale e cyber-fisico, Xia et al.[29] propongono un'architettura agentica per il controllo di sistemi di automazione industriale. Il framework integra agenti LLM, digital twin, eventi di sistema e chiamate strutturate verso attuatori industriali. Gli agenti ricevono informazioni contestuali sullo stato dell'impianto e le trasformano in piani operativi o comandi di controllo. Questo lavoro è particolarmente rilevante perché mostra come gli LLM possano essere inseriti in architetture software complesse, nelle quali il modello non opera in modo isolato, ma viene supportato da rappresentazioni strutturate, eventi e interfacce di esecuzione.

Nel complesso, i lavori su building energy systems, smart industry e cyber-physical systems confermano che gli LLM possono svolgere un ruolo utile come strato di astrazione tra l'utente e sistemi tecnici complessi. Al tempo stesso, tali lavori mostrano la necessità di affiancare al modello linguistico componenti applicative in grado di fornire contesto, validare le azioni e impedire l'esecuzione di comandi non sicuri o non compatibili con la struttura reale del sistema.

2.6.4. Posizionamento della presente tesi in relazione ai lavori correlati

La presente tesi si colloca all'intersezione dei filoni appena descritti: da un lato, condivide con i lavori su smart home e IoT l'idea di utilizzare un LLM come interfaccia in linguaggio naturale verso un ambiente tecnico composto da dispositivi, sensori e attuatori.

Dall'altro, riprende dai lavori su tool use e agenti LLM il problema della traduzione di una richiesta testuale in una chiamata strutturata verso funzioni software. Infine, si collega ai lavori su building energy systems e sistemi cyber-fisici perché per il caso specifico applica questa logica a un ambiente HVAC simulato, nel quale le azioni del modello devono essere compatibili con la struttura fisica e tecnica del sistema.

Rispetto alla letteratura analizzata, l'obiettivo è maggiormente circoscritto e sperimentale, in quanto il presente lavoro valuta in che misura un LLM locale, eseguito tramite Ollama, sia in grado di interagire con un ambiente HVAC simulato tramite richieste formulate in linguaggio naturale. La scelta di utilizzare Qwen2.5:7B consente inoltre di osservare il comportamento di un modello di dimensioni contenute, eseguito localmente, senza dipendere da API cloud o da modelli proprietari ad alta capacità.

Un altro elemento distintivo del lavoro è l'integrazione con Smart Building Simulator (SBSim), utilizzato come ambiente dinamico composto da zone, dispositivi, sensori e attuatori. A differenza di benchmark basati su scenari domotici prevalentemente discreti o su dataset statici, la sperimentazione considera un sistema HVAC simulato, nel quale l'utente può richiedere la lettura di sensori o la modifica di setpoint associati a componenti come boiler, AHU e regolatori VAV. Questo permette di analizzare non solo la comprensione linguistica della richiesta, ma anche la capacità del modello di collegarla alla struttura tecnica effettiva del simulatore.

Infine, un ulteriore elemento riguarda il ruolo del layer applicativo sviluppato in Python. I risultati sperimentali mostrano che il modello locale non può essere trattato come un agente completamente autonomo capace di operare direttamente sulle API del simulatore. Quando l'utente utilizza espressioni naturali come "caldaia", "unità di trattamento aria" o "stanza 10", il sistema deve ricondurre tali riferimenti agli identificativi tecnici richiesti dal simulatore. Il layer Python svolge quindi una funzione di mediazione in cui normalizza gli argomenti prodotti dal modello, risolve riferimenti parziali o semantici ai dispositivi e impedisce l'esecuzione di chiamate non valide o allucinate.

Capitolo Terzo

Architettura del sistema

3.1. Struttura del sistema

Il sistema sviluppato è composto da tre componenti principali (figura 3.1): il simulatore Smart Building Simulator (SBSim), il modello linguistico Qwen e un layer di integrazione sviluppato in Python, incaricato di collegare il modello al simulatore. L'obiettivo dell'architettura è consentire a un utente di interagire con un ambiente HVAC simulato tramite richieste formulate in linguaggio naturale, traducendo tali richieste in operazioni leggibili o eseguibili all'interno del simulatore.

SBSim rappresenta l'ambiente simulato all'interno del quale avviene l'evoluzione dello stato dell'edificio in relazione allo scorrere del tempo. Esso contiene la struttura dell'edificio, i dispositivi HVAC, i sensori, gli attuatori e le logiche fisiche interne che determinano l'aggiornamento dello stato del sistema nel tempo. In questo lavoro, SBSim viene utilizzato come ambiente operativo all'interno del quale si verifica la capacità del modello di leggere informazioni, modificare parametri di controllo e osservare l'effetto di tali modifiche sulle metriche della simulazione.

Il modello Qwen svolge invece il ruolo di interprete linguistico. Riceve in input una richiesta testuale dell'utente e decide se rispondere direttamente oppure se utilizzare uno dei tool disponibili tramite function calling. Il suo compito è quello di comprendere l'intento della richiesta e tradurlo nella selezione di una funzione e dei relativi argomenti.

Tra il modello e SBSim è stato introdotto un layer Python, che costituisce il nucleo dell'integrazione sviluppata. Questo livello intermedio riceve le tool call prodotte dal modello, normalizza e valida gli argomenti, risolve eventuali riferimenti semantici o parziali ai dispositivi, costruisce una rappresentazione strutturata dell'operazione da eseguire e inoltra la richiesta al simulatore tramite le API previste da SBSim. Tale componente ha quindi una funzione di controllo e mediazione, impedendo che identificativi non validi, parametri inesistenti o richieste incoerenti vengano eseguiti direttamente sull'ambiente simulato.

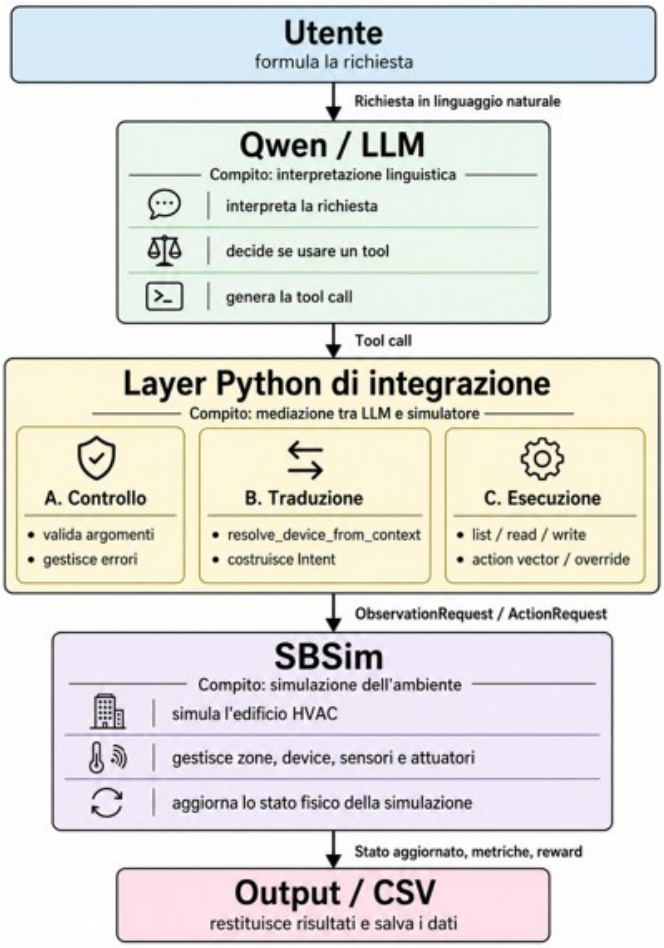


Figura 3.1 Schema semplificato dei componenti del sistema e rispettive relazioni

L'architettura complessiva prevede quindi che la richiesta dell'utente venga inizialmente espressa in linguaggio naturale, interpretata dal modello LLM, trasformata dal layer Python in una richiesta strutturata e infine applicata al simulatore. Il risultato dell'operazione viene poi restituito all'utente e registrato insieme alle metriche prodotte dalla simulazione.

Nei sotto-paragrafi successivi vengono analizzate più in dettaglio le singole componenti.

3.1.1. SBSim

L'edificio simulato SBSim si compone di 128 **zone**, delle quali 127 sono numerate da 1 a 127 e contengono un singolo dispositivo VAV dedicato al monitoraggio e alla regolazione dell'aria all'interno della specifica zona. La zona restante è denominata

"**default_zone**" e contiene un boiler e un'unità di trattamento dell'aria (AHU). Ciascuno di questi dispositivi contiene dei sensori (**measurements)** dedicati alla lettura dei parametri monitorati dal dispositivo e degli attuatori (**setpoints)** per impostare il valore desiderato per i parametri modificabili.

La mappatura dei sensori e attuatori presenti all'interno di ciascuna tipologia di dispositivo è riportata nella tabella sottostante:

Dispositivo	Descrizione Dispositivo	Campo	Tipo	Descrizione Campo
Boiler	Caldaia centrale	supply_water_setpoint	attuatore	Temperatura target dell'acqua calda prodotta
Boiler	Caldaia centrale	heating_request_count	sensore	Numero richieste di riscaldamento dalle zone
Boiler	Caldaia centrale	supply_water_temperature_sensor	sensore	Temperatura reale acqua in uscita
AHU	Unità trattamento aria centrale	supply_air_heating_temperature_setpoint	attuatore	Temperatura target aria calda immessa
AHU	Unità trattamento aria centrale	supply_air_cooling_temperature_setpoint	attuatore	Temperatura target aria fredda immessa
AHU	Unità trattamento aria centrale	cooling_request_count	sensore	Numero richieste di raffrescamento
AHU	Unità trattamento aria centrale	outside_air_temperature_sensor	sensore	Temperatura aria esterna
AHU	Unità trattamento aria centrale	outside_air_flowrate_sensor	sensore	Portata aria esterna
AHU	Unità trattamento aria centrale	supply_air_flowrate_sensor	sensore	Portata aria immessa
AHU	Unità trattamento aria centrale	supply_fan_speed_percentage_command	sensore	Velocità ventilatore (%)
VAV	Regolatore aria di zona	supply_air_damper_percentage_command	attuatore	Percentuale apertura serranda aria
VAV	Regolatore aria di zona	zone_air_temperature_sensor	sensore	Temperatura aria della stanza
VAV	Regolatore aria di zona	supply_air_flowrate_setpoint	sensore	Portata aria target della stanza

3.1.2. Qwen-Ollama

Il modello linguistico utilizzato nell'architettura è qwen2.5:7b, eseguito localmente tramite Ollama. Qwen è una famiglia di Large Language Models sviluppata da Alibaba; nel nome del modello, la sigla 2.5 identifica la versione utilizzata, mentre 7b indica indicativamente la dimensione del modello, pari a circa sette miliardi di parametri. La scelta di utilizzare una versione da 7 miliardi di parametri è intermedia rispetto al trade-off tra capacità di interpretazione e risorse computazionali a disposizione: maggiore è il numero di parametri del modello, maggiore è in genere la capacità rappresentativa del modello, ma maggiore è anche il costo computazionale richiesto per eseguirlo.

Il modello riceve in input il prompt di sistema (system prompt), la richiesta dell'utente (user prompt) e la descrizione dei tool disponibili. La chiamata al modello viene effettuata tramite la funzione ollama.chat(), specificando il modello da utilizzare, la lista dei messaggi e l'elenco delle funzioni richiamabili tramite function calling (figura 3.2). In questo modo il modello può decidere se rispondere direttamente in linguaggio naturale oppure se generare una chiamata a uno dei tool esposti dal sistema.

```
resp = ollama.chat(
    model="qwen2.5:7b",
    messages=[
        {"role": "system", "content": system_prompt},
        {"role": "user", "content": user_text},
    ],
    tools=TOOLS,
    options={
        "temperature": 0, #diminuisce la casualità
        "top_p": 1 #esplora tutte le possibilità
    }
)
```

Figura 3.2 Chiamata al modello Qwen specificato tramite Ollama

La chiamata specifica il modello da utilizzare e passa al modello due messaggi principali: il messaggio di sistema, contenente il system prompt e le istruzioni generali di comportamento, e il messaggio utente, contenente la richiesta formulata in linguaggio naturale.

Alla chiamata vengono inoltre forniti i tool disponibili, definiti nella variabile `TOOLS`, in modo che il modello possa eventualmente richiamare una funzione tramite function

calling anziché limitarsi a generare una risposta testuale. In questo modo Qwen può scegliere autonomamente se rispondere direttamente all'utente oppure produrre una tool call verso una delle funzioni implementate.

È infine possibile impostare alcuni parametri opzionali di generazione. Per il caso specifico, i parametri impostati sono stati utilizzati per rendere più stabile il comportamento del modello durante gli esperimenti. In particolare, il parametro **top_p** è stato impostato pari a 1, cioè al suo valore massimo, evitando così di applicare ulteriori restrizioni all'insieme delle alternative considerate dal modello durante la generazione. Se, ad esempio, questo parametro fosse stato impostato a 0.7, il modello avrebbe limitato la scelta ai token più probabili fino al raggiungimento del 70% di probabilità cumulata. In combinazione con il parametro **temperature** = 0 (par. §4.1), questa impostazione consente di ottenere un comportamento più stabile nella selezione del tool e nella generazione degli argomenti associati alla chiamata. Questa configurazione è stata scelta per rendere più confrontabili le diverse esecuzioni sperimentali.

3.1.3. Layer Python

Il terzo componente dell'architettura è costituito dal layer Python sviluppato per collegare il modello Qwen al simulatore SBSim. Tale layer riceve le richieste generate dal modello tramite function calling e le traduce in operazioni compatibili con le interfacce del simulatore. La presenza di questo livello intermedio consente di evitare che l'output generato dal modello venga applicato direttamente al simulatore senza controlli. Eventuali allucinazioni, riferimenti incompleti, identificativi non validi o parametri non coerenti vengono intercettati dal layer Python prima di essere tradotti in operazioni effettive, impedendo pertanto l'esecuzione di operazioni non compatibili con la struttura dell'ambiente simulato[30].

Le principali funzioni svolte dal layer sono:

- Definizione dei tool disponibili (par. §3.3);
- Gestione delle chiamate ai tool prodotte da Qwen (**esegui_tools)**;
- Normalizzazione degli argomenti (**clean_tool_args**);
- Risoluzione dei riferimenti ai dispositivi (**resolve_device_from_context**);

- Costruzione ed esecuzione delle query/command nel formato richiesto dall'ambiente e inoltro della richiesta a SBSim (**esegui_intent**).

L'implementazione di tali funzioni è riportata, sotto forma di estratti di codice, nell'Appendice A.

Il layer implementato in Python svolge quindi una funzione di mediazione tra interpretazione linguistica e simulazione fisica: mentre Qwen si occupa di comprendere la richiesta dell'utente e selezionare il tool più opportuno, il layer Python verifica, struttura ed esegue l'operazione sul simulatore. Questa separazione consente di sfruttare le capacità interpretative del modello mantenendo al tempo stesso un controllo esplicito sulle operazioni effettivamente applicate all'ambiente SBSim, evitando la generazione di potenziali allucinazioni.

3.2. Flusso esecutivo della simulazione

Nell'architettura implementata viene effettuata una simulazione in ambiente HVAC dalla durata di 24 ore, suddivisa in "episodi" (steps) di 5 minuti ciascuno: a ogni step l'utente può interagire con il simulatore HVAC tramite un input scritto in linguaggio naturale. Il modello LLM interpreta la richiesta e, attraverso meccanismi di function calling, richiama i tool disponibili per esplorare la struttura del sistema, leggere i valori dei sensori o modificare gli attuatori. Le richieste di azione o osservazione generate dal modello LLM attraversano il livello intermedio sviluppato in Python al fine di eliminare potenziali allucinazioni generate.

Completata la fase di elaborazione, il simulatore SBSim esegue l'avanzamento dello stato dell'ambiente aggiornando dinamicamente i dispositivi HVAC, ricalcolando le grandezze fisiche e producendo i nuovi valori di temperatura, consumi e costi energetici. Infine, il risultato dell'operazione richiesta viene restituito all'utente insieme allo stato aggiornato del sistema.

Al termine della simulazione i risultati delle metriche ottenuti in ogni step vengono salvati su un file .csv e resi disponibili all'utente.

Di seguito uno schema completo del flusso (figura 3.3), le cui parti salienti verranno analizzate nei sotto-paragrafi successivi.

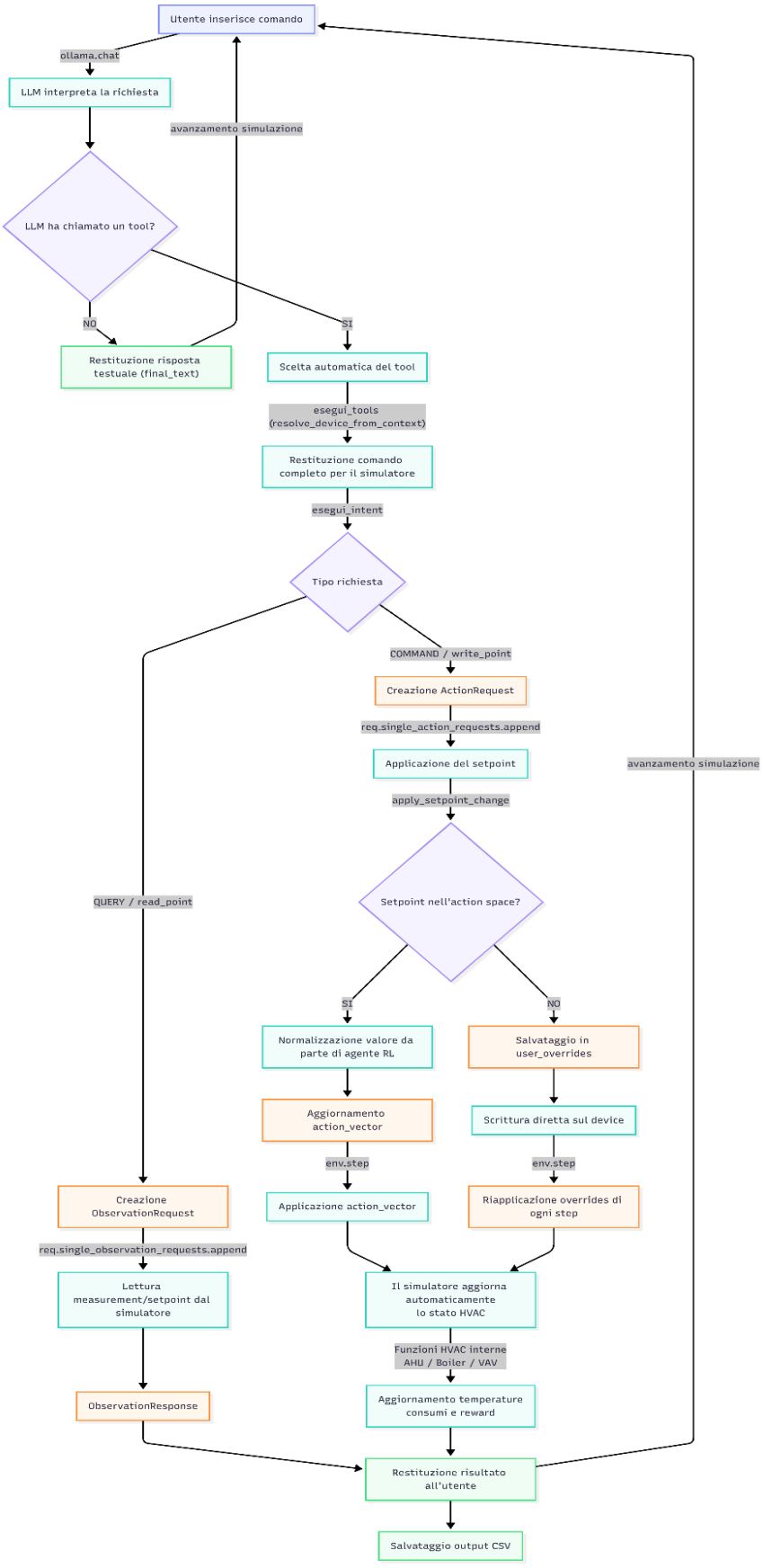


Figura 3.3 Diagramma di flusso della simulazione implementata

3.2.1 Interpretazione della richiesta

Ad ogni step della simulazione l'utente può inserire un comando testuale. Tale comando viene inviato al modello LLM tramite la chiamata **ollama.chat()**, insieme al prompt di sistema e alla lista dei tool disponibili. Il modello analizza la richiesta e decide se rispondere direttamente con testo oppure se richiamare uno dei tool tramite function calling. I tool disponibili sono approfonditi nel paragrafo §3.3.

Nel caso in cui il modello non individui alcun tool da utilizzare, il sistema restituisce semplicemente la risposta testuale generata dal modello. Nel caso invece in cui venga prodotta una tool call, il sistema estrae il nome della funzione richiesta e i relativi argomenti, restituendo al back-end la richiesta generata (figura 3.4).

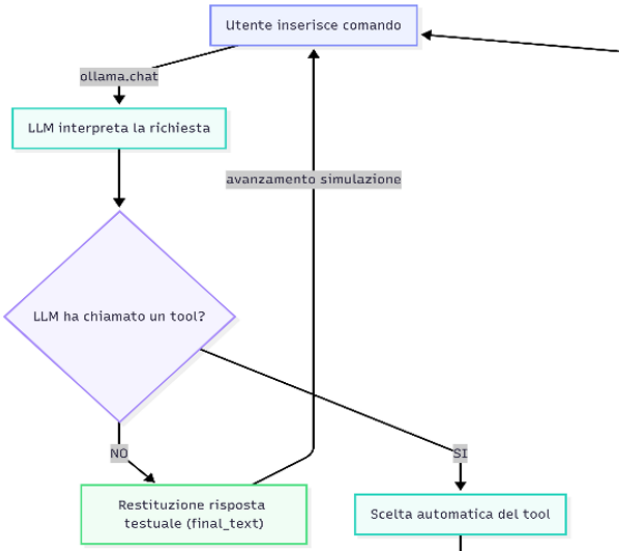


Figura 3.4 Flusso della chiamata a Ollama

3.2.2 Elaborazione della richiesta

La gestione della tool call avviene attraverso la funzione **esegui_tools()**, che rappresenta il punto di collegamento tra l'output del modello e il simulatore. In questa fase gli argomenti generati dall'LLM vengono prima normalizzati tramite la funzione **clean_tool_args()**, che converte i parametri numerici che sono stati restituiti come stringhe dal tool e gestisce eventuali campi vuoti.

Successivamente viene individuato il dispositivo a cui la richiesta si riferisce. Questa operazione è gestita dalla funzione **resolve_device_from_context()**, che permette di risolvere riferimenti espliciti o parziali ai dispositivi del sistema. Tale funzione è

particolarmente utile, poiché gli identificativi dei dispositivi di tipo **boiler** e **air handler** vengono generati dinamicamente dal simulatore (quindi cambiano ad ogni run), impedendo pertanto di dedurli dal solo linguaggio naturale dell'utente.

Una volta definiti dispositivo, parametro e valore richiesto, il sistema costruisce un oggetto **Intent** (figura 3.5), che rappresenta in modo strutturato l'operazione da eseguire. L'Intent può essere di tipo **query**, nel caso di lettura di un valore, oppure di tipo **command**, nel caso di modifica di un attuatore.

```
class Intent(BaseModel):
    type: Literal["command", "query"]
    zone_id: Optional[str] = None
    device_id: Optional[str] = None
    measurement_name: Optional[str] = None
    setpoint_name: Optional[str] = None
    value: Optional[float] = None
```

Figura 3.5 Definizione degli attributi della classe Intent

3.2.3 Esecuzione dell'operazione

L'esecuzione dell'operazione è gestita dalla funzione **esegui_intent()**, illustrata nella figura 3.6, che applica una serie di controlli di coerenza prima di inoltrare la richiesta al simulatore. In particolare, viene verificato che il dispositivo indicato esista, che appartenga eventualmente alla zona specificata e che il sensore o attuatore richiesto sia effettivamente disponibile per quel dispositivo.

Nel caso di una richiesta di lettura, il sistema costruisce una "**ObservationRequest"**, contenente una "**SingleObservationRequest**" relativa al dispositivo e al parametro richiesto (che può essere sia un sensore sia un attuatore). La richiesta viene quindi inviata al simulatore tramite il metodo **building.request_observation()**, che restituisce il valore corrente del parametro selezionato.

Nel caso di una richiesta di modifica, il sistema costruisce invece una "**ActionRequest**", contenente una "**SingleActionRequest**" che include dispositivo, attuatore e valore da impostare. La richiesta viene inoltrata a SBSim tramite il metodo **building.request_action()**, che verifica se l'azione è accettabile e restituisce un codice di

risposta. Se la modifica viene accettata, il risultato viene successivamente gestito dalla logica di aggiornamento degli attuatori.

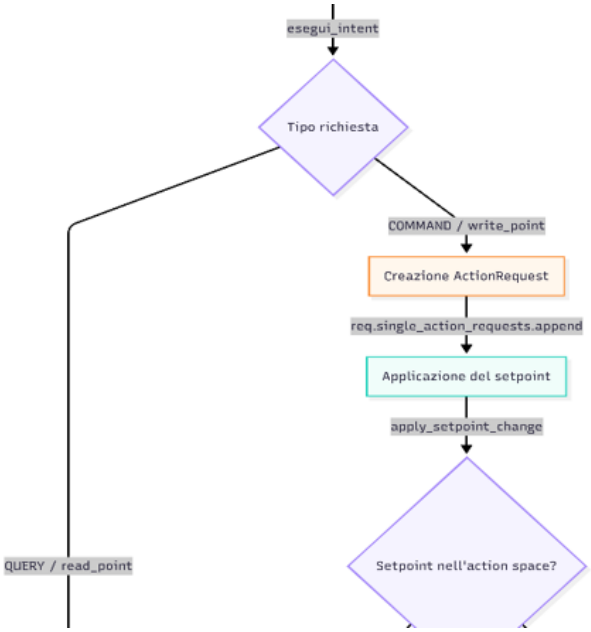


Figura 3.6 Ramificazione del flusso in base al tipo di richiesta

3.2.4 Aggiornamento degli attuatori e avanzamento della simulazione

Per le operazioni di scrittura, dopo l'accettazione della richiesta da parte del simulatore, il sistema stabilisce come mantenere attivo il nuovo valore negli step successivi. Questa logica è implementata nella funzione **apply_setpoint_change()**, che distingue tra due possibili canali di aggiornamento (approfonditi nel paragrafo §3.4).

In entrambi i casi, dopo la gestione dell'attuatore, la simulazione procede tramite **env.step(current_action)**. Questa chiamata fa avanzare il simulatore di uno step temporale, applicando le azioni correnti e aggiornando lo stato dell'ambiente. Il simulatore utilizza quindi i valori correnti per aggiornare le metriche monitorate tramite le funzioni HVAC interne relative ai singoli dispositivi (figura 3.7).

Al termine della simulazione, viene prodotto un file .csv contenente le informazioni relative agli eventuali attuatori modificati, agli override applicati e alle principali metriche della simulazione, misurate in ogni step.

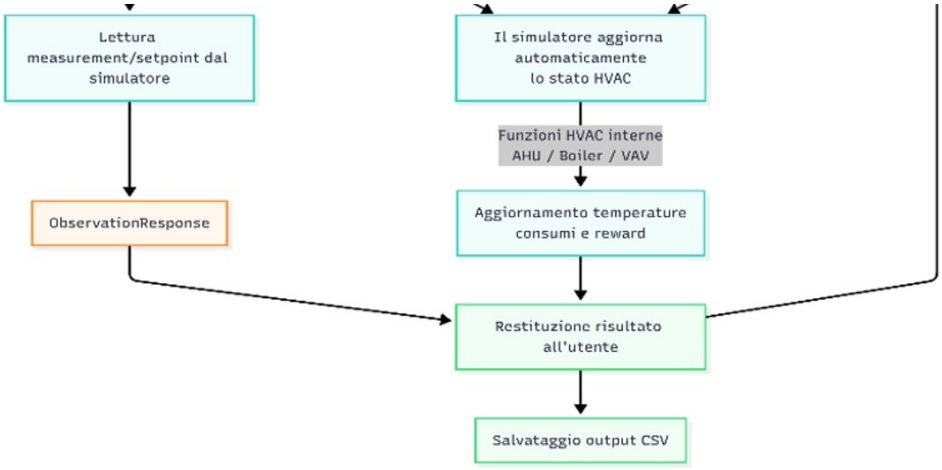


Figura 3.7 Estratto della parte finale del flusso: il simulatore aggiorna le metriche monitorate tramite le funzioni HVAC interne; al termine della simulazione i risultati vengono salvati in un file .csv

3.3. Tools implementati

Per consentire al modello di interagire con l'ambiente simulato SBSim tramite meccanismi di function calling, sono stati implementati tre tools dedicati. Questi strumenti costituiscono l'interfaccia tra il modello linguistico e il simulatore, permettendo all'LLM di esplorare la struttura del sistema, interrogarne lo stato e modificare i parametri di controllo senza accedere direttamente alle logiche interne del simulatore. I tools implementati sono i seguenti:

- **list_devices:** restituisce la struttura gerarchica completa del sistema, comprendente zone, dispositivi, sensori e attuatori disponibili. Viene utilizzato dal modello per comprendere l'organizzazione dell'ambiente e individuare i parametri interrogabili o modificabili;
- **read_point:** consente la lettura del valore di un sensore o di un attuatore associato a uno specifico dispositivo. Il tool viene utilizzato per rispondere a richieste informative dell'utente, come la lettura di temperature e altri parametri monitorati dal simulatore;
- **write_point**: permette di modificare il valore di un attuatore associato a un dispositivo. Il tool viene utilizzato per eseguire azioni di controllo sull'ambiente simulato, ad esempio modificando temperature di riferimento, aperture delle serrande o altri parametri configurabili.

3.4. Gestione degli attuatori

Esplorando l'ambiente SBSim è emerso che gli attuatori non vengono gestiti in modo uniforme. In particolare, l'ambiente è pensato per essere controllato da un agente addestrato tramite Reinforcement Learning, in grado di controllare gli attuatori che rientrano all'interno di un determinato "action space", configurato dai creatori dell'ambiente. Tuttavia, non tutti gli attuatori disponibili rientrano all'interno di questo insieme, in particolare gli unici attuatori inclusi sono **supply_water_setpoint** e **supply_air_heating_temperature_setpoint**. Per consentire quindi al modello di modificare qualsiasi attuatore disponibile nel sistema, al fine di ampliare il margine di manovra nei test, sono stati implementati due canali alternativi di gestione: uno basato sull'action vector dell'agente RL e uno basato su override manuali applicati direttamente ai dispositivi (figura 3.8).

Gli attuatori inclusi nell'action space dell'agente RL vengono gestiti attraverso il normale flusso di esecuzione previsto da SBSim. Quando il modello richiede una modifica, il valore viene convertito nel formato utilizzato dall'agente e inserito all'interno dell'action vector. Durante l'avanzamento, il simulatore interpreta il vettore delle azioni, aggiorna i dispositivi coinvolti e ricalcola lo stato dell'ambiente. Questo approccio rappresenta il meccanismo nativo di controllo del simulatore e garantisce l'applicazione delle logiche interne previste da SBSim, inclusi i controlli di validità e la normalizzazione dei valori all'interno dei range ammessi dall'action space.

Per gli attuatori che non sono inclusi nell'action space, invece, è stato necessario introdurre un meccanismo alternativo basato su override manuali. In questo caso il valore richiesto dal modello viene applicato direttamente al dispositivo e memorizzato in una struttura dedicata. Poiché il simulatore tende a sovrascrivere tali modifiche durante gli step successivi, gli override vengono riapplicati automaticamente ad ogni avanzamento della simulazione. In questo modo il valore impostato dall'utente rimane persistente nel tempo e può influenzare l'evoluzione dell'ambiente. A differenza della gestione tramite action vector, questo approccio elude il processo di normalizzazione utilizzato dall'agente RL e agisce direttamente sul dispositivo interessato.

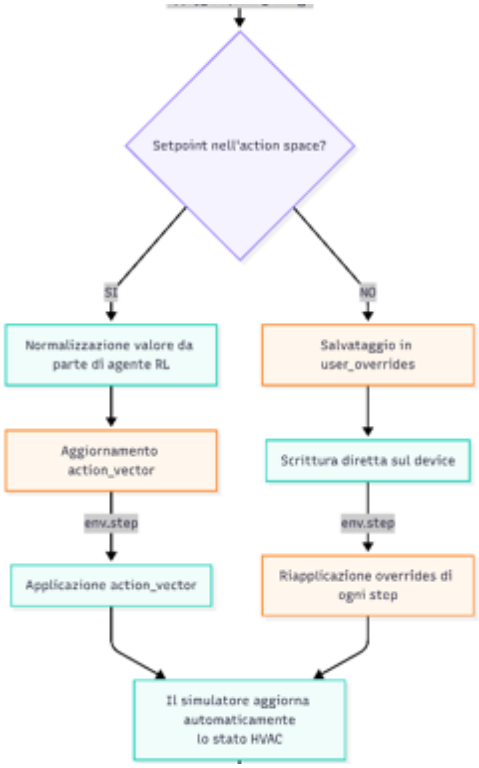


Figura 3.8 Estratto del diagramma in cui viene illustrato il flusso dei due canali paralleli

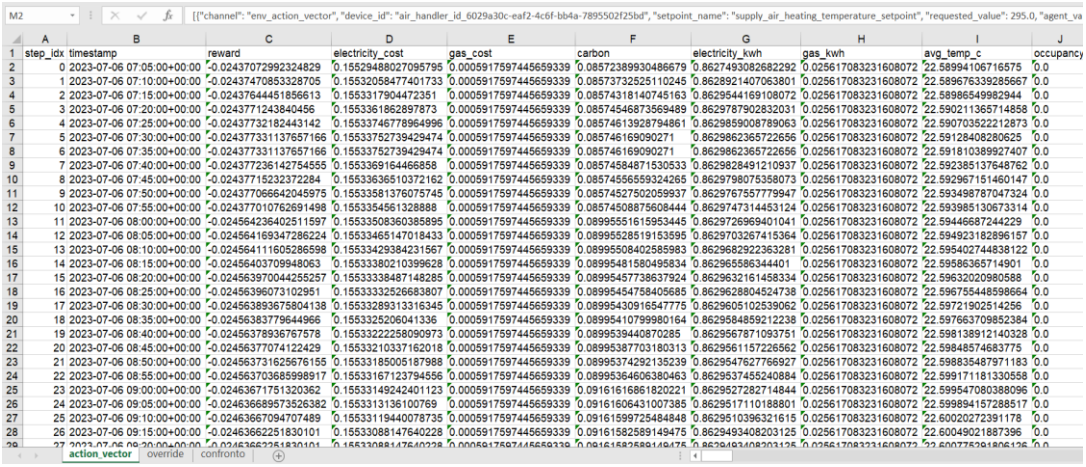
3.4.1. Test action vector vs override

Al fine di verificare che i due meccanismi di gestione degli attuatori producessero gli stessi effetti nella simulazione, è stato effettuato un confronto tra due esecuzioni identiche, nelle quali un singolo attuatore all'interno dell'action space è stato modificato al primo step della simulazione prima tramite action vector e poi tramite override, richiedendo il medesimo valore per entrambe le casistiche.

Nel primo caso, l'attuatore è stato modificato attraverso il flusso standard previsto da SBSim, inserendo il valore all'interno dell'action vector dell'agente RL e aggiornando lo stato durante l'avanzamento. Nel secondo caso, lo stesso attuatore è stato modificato forzando il meccanismo di override, applicando direttamente il valore al dispositivo e riapplicandolo automaticamente ad ogni step della simulazione. In entrambi i casi non sono stati forniti ulteriori input al modello fino al termine della simulazione.

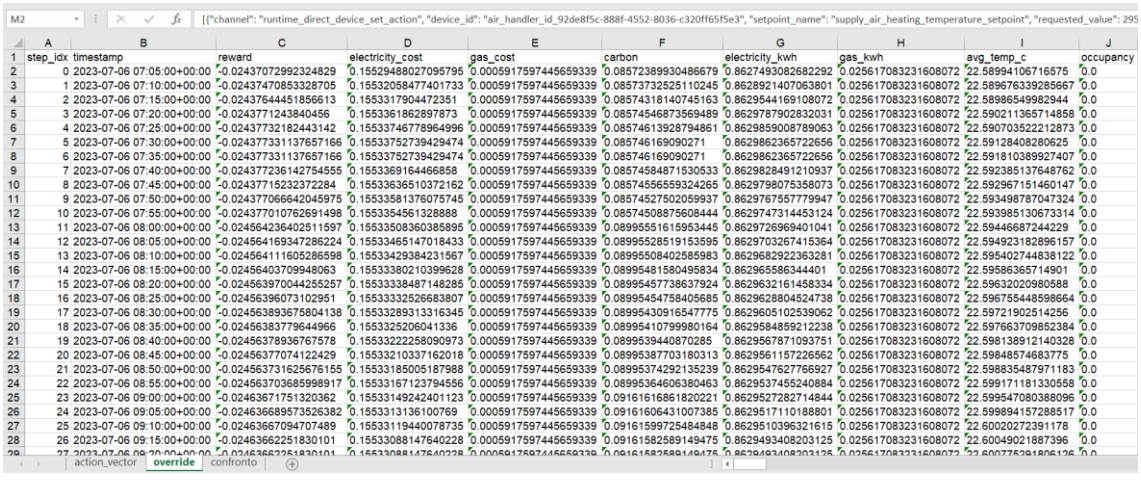
È stato utilizzato come oggetto di test il **supply_air_heating_temperature_setpoint** dell'unità di trattamento dell'aria centrale (AHU), impostandolo inizialmente al valore di 295 K (compreso nel range ammesso 285–300 K). Lo stesso esperimento è stato successivamente ripetuto anche utilizzando i due valori limite dell'intervallo consentito.

I risultati sono stati registrati e confrontati all'interno di un file Excel (figure 3.9 e 3.10), per poi essere confrontati tramite un ulteriore foglio di calcolo (figura 3.11). Tali risultati mostrano come, per ogni step, le principali metriche della simulazione (temperatura media, occupazione, consumi, costi, reward) risultino coincidenti nei due casi. Ciò conferma che, almeno per gli attuatori appartenenti all'action space (ovvero gli unici assoggettabili a questo test), il meccanismo di override diretto produce un comportamento equivalente a quello ottenuto tramite il flusso standard basato sull'action vector, pur seguendo un canale di aggiornamento differente.



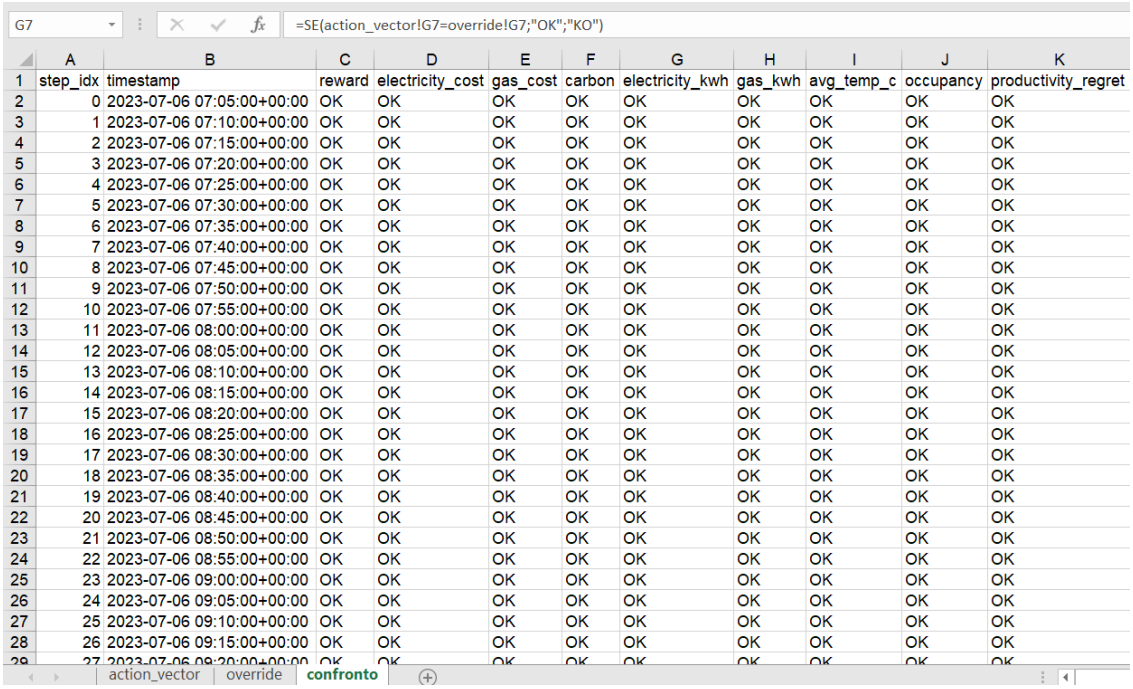
	A	B	C	D	E	F	G	H	I	J
1	step_idx	timestamp	reward	electricity_cost	gas_cost	carbon	electricity_kwh	gas_kwh	avg_temp_c	occupancy
2	0	2023-07-06 07:05:00+00:00	-0.02437072992324829	0.15529488027095795	0.0005917597445659339	0.08572389930486679	0.8627493082682292	0.025617083231608072	22.58994106716575	0.0
3	1	2023-07-06 07:10:00+00:00	-0.02437470853328705	0.15532058477401733	0.0005917597445659339	0.08573732525110245	0.8628921407063801	0.025617083231608072	22.589676339285667	0.0
4	2	2023-07-06 07:15:00+00:00	-0.02437644451856613	0.1553317904472351	0.0005917597445659339	0.08574318140745163	0.8629544169108072	0.025617083231608072	22.58986549982944	0.0
5	3	2023-07-06 07:20:00+00:00	-0.0243771243840456	0.1553361862897873	0.0005917597445659339	0.08574546873569489	0.8629787902832031	0.025617083231608072	22.590211365714858	0.0
6	4	2023-07-06 07:25:00+00:00	-0.02437732182443142	0.15533746778964996	0.0005917597445659339	0.08574613928794861	0.8629859008789063	0.025617083231608072	22.590703522212873	0.0
7	5	2023-07-06 07:30:00+00:00	-0.02437733113767166	0.15533752739429474	0.0005917597445659339	0.085746169090271	0.8629862365722656	0.025617083231608072	22.59128408280625	0.0
8	6	2023-07-06 07:35:00+00:00	-0.02437733113767166	0.15533752739429474	0.0005917597445659339	0.085746169090271	0.8629862365722656	0.025617083231608072	22.591810389927407	0.0
9	7	2023-07-06 07:40:00+00:00	-0.02437723614754555	0.1553369164466858	0.0005917597445659339	0.08574584871530533	0.8629828491210937	0.025617083231608072	22.592385137648762	0.0
10	8	2023-07-06 07:45:00+00:00	-0.02437715232372284	0.15533636510372162	0.0005917597445659339	0.08574556559324265	0.8629798075358073	0.025617083231608072	22.592967151460147	0.0
11	9	2023-07-06 07:50:00+00:00	-0.02437706664204597	0.15533581376075745	0.0005917597445659339	0.08574527502059937	0.8629767557779947	0.025617083231608072	22.593498787047324	0.0
12	10	2023-07-06 07:55:00+00:00	-0.02437701076269148	0.1553354561328888	0.0005917597445659339	0.08574508875608444	0.8629747314453124	0.025617083231608072	22.593985130673314	0.0
13	11	2023-07-06 08:00:00+00:00	-0.02456423640251159	0.15533508360385895	0.0005917597445659339	0.08995551615953445	0.8629726969401041	0.025617083231608072	22.59446687244229	0.0
14	12	2023-07-06 08:05:00+00:00	-0.02456416934728622	0.15533465147018433	0.0005917597445659339	0.08995528519153595	0.8629703267415364	0.025617083231608072	22.594923182896157	0.0
15	13	2023-07-06 08:10:00+00:00	-0.02456411605286598	0.15533429384231567	0.0005917597445659339	0.08995508402585983	0.8629682922363281	0.025617083231608072	22.595402744838122	0.0
16	14	2023-07-06 08:15:00+00:00	-0.02456403709948063	0.15533380210399628	0.0005917597445659339	0.08995481580495834	0.8629655864344401	0.025617083231608072	22.59586365714901	0.0
17	15	2023-07-06 08:20:00+00:00	-0.02456397004255257	0.15533338487148285	0.0005917597445659339	0.08995457738637924	0.8629632161458334	0.025617083231608072	22.59632020980588	0.0
18	16	2023-07-06 08:25:00+00:00	-0.02456396073102951	0.15533332526683807	0.0005917597445659339	0.08995454758405685	0.8629628804524738	0.025617083231608072	22.596755448598664	0.0
19	17	2023-07-06 08:30:00+00:00	-0.02456389367580413	0.15533289313316345	0.0005917597445659339	0.08995430916547775	0.8629605102539062	0.025617083231608072	22.59721902514256	0.0
20	18	2023-07-06 08:35:00+00:00	-0.02456383779644966	0.15533252060041336	0.0005917597445659339	0.08995410799980164	0.8629584859212238	0.025617083231608072	22.597663709852384	0.0
21	19	2023-07-06 08:40:00+00:00	-0.02456378936767578	0.15533222258090973	0.0005917597445659339	0.0899539440870285	0.8629567871093751	0.025617083231608072	22.598138912140328	0.0
22	20	2023-07-06 08:45:00+00:00	-0.02456377074122429	0.15533210337162018	0.0005917597445659339	0.08995387703180313	0.8629561157226562	0.025617083231608072	22.59848574683775	0.0
23	21	2023-07-06 08:50:00+00:00	-0.02456373162567615	0.15533185005187988	0.0005917597445659339	0.08995374292135239	0.8629547627766927	0.025617083231608072	22.598835487971183	0.0
24	22	2023-07-06 08:55:00+00:00	-0.02456370368599817	0.1553316712379556	0.0005917597445659339	0.08995364606380463	0.8629537455240884	0.025617083231608072	22.599171181330558	0.0
25	23	2023-07-06 09:00:00+00:00	-0.02456367073260742	0.15533149242401123	0.0005917597445659339	0.09161616861820221	0.8629527282714844	0.025617083231608072	22.599547080388096	0.0
26	24	2023-07-06 09:05:00+00:00	-0.02456366895735382	0.15533131361007690	0.0005917597445659339	0.09161606431007385	0.8629517110188801	0.025617083231608072	22.599894157288517	0.0
27	25	2023-07-06 09:10:00+00:00	-0.02456366709470749	0.15533119440078735	0.0005917597445659339	0.09161599725484848	0.8629510396321615	0.025617083231608072	22.60020272391178	0.0
28	26	2023-07-06 09:15:00+00:00	-0.02456366225183010	0.15533088147640228	0.0005917597445659339	0.09161582589149475	0.8629493408203125	0.025617083231608072	22.60049021887396	0.0

Figura 3.9 Estratto dei risultati della simulazione ottenuta impostando "supply_air_heating_temperature_setpoint" a 295 K tramite l'action vector dell'agente RL



	A	B	C	D	E	F	G	H	I	J
1	step_idx	timestamp	reward	electricity_cost	gas_cost	carbon	electricity_kwh	gas_kwh	avg_temp_c	occupancy
2	0	2023-07-06 07:05:00+00:00	-0.02437072992324829	0.15529488027095795	0.0005917597445659339	0.08572389930486679	0.8627493082682292	0.025617083231608072	22.58994106716575	0.0
3	1	2023-07-06 07:10:00+00:00	-0.02437470853328705	0.15532058477401733	0.0005917597445659339	0.08573732525110245	0.8628921407063801	0.025617083231608072	22.589676339285667	0.0
4	2	2023-07-06 07:15:00+00:00	-0.02437644451856613	0.1553317904472351	0.0005917597445659339	0.08574318140745163	0.8629544169108072	0.025617083231608072	22.58986549982944	0.0
5	3	2023-07-06 07:20:00+00:00	-0.0243771243840456	0.1553361862897873	0.0005917597445659339	0.08574546873569489	0.8629787902832031	0.025617083231608072	22.590211365714858	0.0
6	4	2023-07-06 07:25:00+00:00	-0.02437732182443142	0.15533746778964996	0.0005917597445659339	0.08574613928794861	0.8629859008789063	0.025617083231608072	22.590703522212873	0.0
7	5	2023-07-06 07:30:00+00:00	-0.02437733113767166	0.15533752739429474	0.0005917597445659339	0.085746169090271	0.8629862365722656	0.025617083231608072	22.59128408280625	0.0
8	6	2023-07-06 07:35:00+00:00	-0.02437733113767166	0.15533752739429474	0.0005917597445659339	0.085746169090271	0.8629862365722656	0.025617083231608072	22.591810389927407	0.0
9	7	2023-07-06 07:40:00+00:00	-0.02437723614754555	0.1553369164466858	0.0005917597445659339	0.08574584871530533	0.8629828491210937	0.025617083231608072	22.592385137648762	0.0
10	8	2023-07-06 07:45:00+00:00	-0.02437715232372284	0.15533636510372162	0.0005917597445659339	0.08574556559324265	0.8629798075358073	0.025617083231608072	22.592967151460147	0.0
11	9	2023-07-06 07:50:00+00:00	-0.02437706664204597	0.15533581376075745	0.0005917597445659339	0.08574527502059937	0.8629767557779947	0.025617083231608072	22.593498787047324	0.0
12	10	2023-07-06 07:55:00+00:00	-0.02437701076269148	0.1553354561328888	0.0005917597445659339	0.08574508875608444	0.8629747314453124	0.025617083231608072	22.593985130673314	0.0
13	11	2023-07-06 08:00:00+00:00	-0.02456423640251159	0.15533508360385895	0.0005917597445659339	0.08995551615953445	0.8629726969401041	0.025617083231608072	22.59446687244229	0.0
14	12	2023-07-06 08:05:00+00:00	-0.02456416934728622	0.15533465147018433	0.0005917597445659339	0.08995528519153595	0.8629703267415364	0.025617083231608072	22.594923182896157	0.0
15	13	2023-07-06 08:10:00+00:00	-0.02456411605286598	0.15533429384231567	0.0005917597445659339	0.08995508402585983	0.8629682922363281	0.025617083231608072	22.595402744838122	0.0
16	14	2023-07-06 08:15:00+00:00	-0.02456403709948063	0.15533380210399628	0.0005917597445659339	0.08995481580495834	0.8629655864344401	0.025617083231608072	22.59586365714901	0.0
17	15	2023-07-06 08:20:00+00:00	-0.02456397004255257	0.15533338487148285	0.0005917597445659339	0.08995457738637924	0.8629632161458334	0.025617083231608072	22.59632020980588	0.0
18	16	2023-07-06 08:25:00+00:00	-0.02456396073102951	0.15533332526683807	0.0005917597445659339	0.08995454758405685	0.8629628804524738	0.025617083231608072	22.596755448598664	0.0
19	17	2023-07-06 08:30:00+00:00	-0.02456389367580413	0.15533289313316345	0.0005917597445659339	0.08995430916547775	0.8629605102539062	0.025617083231608072	22.59721902514256	0.0
20	18	2023-07-06 08:35:00+00:00	-0.02456383779644966	0.15533252060041336	0.0005917597445659339	0.08995410799980164	0.8629584859212238	0.025617083231608072	22.597663709852384	0.0
21	19	2023-07-06 08:40:00+00:00	-0.02456378936767578	0.15533222258090973	0.0005917597445659339	0.0899539440870285	0.8629567871093751	0.025617083231608072	22.598138912140328	0.0
22	20	2023-07-06 08:45:00+00:00	-0.02456377074122429	0.15533210337162018	0.0005917597445659339	0.08995387703180313	0.8629561157226562	0.025617083231608072	22.59848574683775	0.0
23	21	2023-07-06 08:50:00+00:00	-0.02456373162567615	0.15533185005187988	0.0005917597445659339	0.08995374292135239	0.8629547627766927	0.025617083231608072	22.598835487971183	0.0
24	22	2023-07-06 08:55:00+00:00	-0.02456370368599817	0.1553316712379456	0.0005917597445659339	0.08995364606380463	0.8629537455240884	0.025617083231608072	22.599171181330558	0.0
25	23	2023-07-06 09:00:00+00:00	-0.02456367073260742	0.15533149242401123	0.0005917597445659339	0.09161616861820221	0.8629527282714844	0.025617083231608072	22.599547080388096	0.0
26	24	2023-07-06 09:05:00+00:00	-0.02456366895735382	0.1553313136100769	0.0005917597445659339	0.09161606431007385	0.8629517110188801	0.025617083231608072	22.599894157288517	0.0
27	25	2023-07-06 09:10:00+00:00	-0.02456366709470749	0.15533119440078735	0.0005917597445659339	0.09161599725484848	0.8629510396321615	0.025617083231608072	22.60020272391178	0.0
28	26	2023-07-06 09:15:00+00:00	-0.02456366225183010	0.15533088147640228	0.0005917597445659339	0.09161582589149475	0.8629493408203125	0.025617083231608072	22.60049021887396	0.0

Figura 3.10 Estratto dei risultati della simulazione ottenuta impostando "supply_air_heating_temperature_setpoint" a 295 K tramite override diretto sul dispositivo



G7 =SE(action_vector!G7=override!G7;"OK";"KO")

	A	B	C	D	E	F	G	H	I	J	K
1	step_idx	timestamp	reward	electricity_cost	gas_cost	carbon	electricity_kwh	gas_kwh	avg_temp_c	occupancy	productivity_regret
2	0	2023-07-06 07:05:00+00:00	OK	OK	OK	OK	OK	OK	OK	OK	OK
3	1	2023-07-06 07:10:00+00:00	OK	OK	OK	OK	OK	OK	OK	OK	OK
4	2	2023-07-06 07:15:00+00:00	OK	OK	OK	OK	OK	OK	OK	OK	OK
5	3	2023-07-06 07:20:00+00:00	OK	OK	OK	OK	OK	OK	OK	OK	OK
6	4	2023-07-06 07:25:00+00:00	OK	OK	OK	OK	OK	OK	OK	OK	OK
7	5	2023-07-06 07:30:00+00:00	OK	OK	OK	OK	OK	OK	OK	OK	OK
8	6	2023-07-06 07:35:00+00:00	OK	OK	OK	OK	OK	OK	OK	OK	OK
9	7	2023-07-06 07:40:00+00:00	OK	OK	OK	OK	OK	OK	OK	OK	OK
10	8	2023-07-06 07:45:00+00:00	OK	OK	OK	OK	OK	OK	OK	OK	OK
11	9	2023-07-06 07:50:00+00:00	OK	OK	OK	OK	OK	OK	OK	OK	OK
12	10	2023-07-06 07:55:00+00:00	OK	OK	OK	OK	OK	OK	OK	OK	OK
13	11	2023-07-06 08:00:00+00:00	OK	OK	OK	OK	OK	OK	OK	OK	OK
14	12	2023-07-06 08:05:00+00:00	OK	OK	OK	OK	OK	OK	OK	OK	OK
15	13	2023-07-06 08:10:00+00:00	OK	OK	OK	OK	OK	OK	OK	OK	OK
16	14	2023-07-06 08:15:00+00:00	OK	OK	OK	OK	OK	OK	OK	OK	OK
17	15	2023-07-06 08:20:00+00:00	OK	OK	OK	OK	OK	OK	OK	OK	OK
18	16	2023-07-06 08:25:00+00:00	OK	OK	OK	OK	OK	OK	OK	OK	OK
19	17	2023-07-06 08:30:00+00:00	OK	OK	OK	OK	OK	OK	OK	OK	OK
20	18	2023-07-06 08:35:00+00:00	OK	OK	OK	OK	OK	OK	OK	OK	OK
21	19	2023-07-06 08:40:00+00:00	OK	OK	OK	OK	OK	OK	OK	OK	OK
22	20	2023-07-06 08:45:00+00:00	OK	OK	OK	OK	OK	OK	OK	OK	OK
23	21	2023-07-06 08:50:00+00:00	OK	OK	OK	OK	OK	OK	OK	OK	OK
24	22	2023-07-06 08:55:00+00:00	OK	OK	OK	OK	OK	OK	OK	OK	OK
25	23	2023-07-06 09:00:00+00:00	OK	OK	OK	OK	OK	OK	OK	OK	OK
26	24	2023-07-06 09:05:00+00:00	OK	OK	OK	OK	OK	OK	OK	OK	OK
27	25	2023-07-06 09:10:00+00:00	OK	OK	OK	OK	OK	OK	OK	OK	OK
28	26	2023-07-06 09:15:00+00:00	OK	OK	OK	OK	OK	OK	OK	OK	OK
29	27	2023-07-06 09:20:00+00:00	OK	OK	OK	OK	OK	OK	OK	OK	OK

action_vector | override | confronto

Figura 3.11 Estratto del confronto tra i risultati ottenuti: per ogni cella riferita a una metrica e a uno step specifico, l'output "OK" indica che i valori delle due simulazioni coincidono.

Capitolo Quarto

Test Sperimentali

Al fine di valutare la capacità di interpretazione autonoma del modello, sono stati sperimentati diversi prompt con livelli di dettaglio differenti. Ispirandosi alla metodologia "Sasha" implementata dagli studenti dell'Università del Texas[2], tutti i prompt implementati suggeriscono al modello di scomporre il ragionamento derivante dall'input dell'utente in più steps:

0. Comprensione dell'obiettivo implicito;
1. Classificazione della richiesta (lettura o modifica);
2. Filtraggio delle zone e dispositivi rilevanti;
3. Selezione del parametro da leggere/modificare, sulla base delle zone e dei dispositivi coerenti individuati;
4. Selezione del tool da utilizzare coerentemente col tipo di richiesta.

Tale scomposizione permette al modello di migliorare la qualità dei ragionamenti.

Gli esperimenti sono stati organizzati in più fasi. In una prima fase sono stati confrontati diversi prompt su un dataset ridotto di comandi base, con l'obiettivo di verificare se un aumento progressivo del livello di dettaglio delle istruzioni fornite al modello, tramite l'aggiunta di regole generali, esempi e informazioni relative allo schema causale, producesse un miglioramento evidente nelle prestazioni.

Sulla base dei risultati ottenuti nella prima fase, il dataset è stato ampliato includendo richieste di vario genere, anche errate o potenzialmente ambigue, al fine di valutare la risposta del modello su un insieme maggiormente variegato di comandi, mentre il numero di configurazioni confrontate è stato ridotto alle versioni ritenute più significative.

Infine, è stato implementato un tentativo di costruzione auto-incrementale del contesto operativo da parte del modello tramite una chiamata multipla ad Ollama, con l'obiettivo di verificare se il modello fosse in grado di dedurre autonomamente la struttura del sistema.

4.1. Configurazione generale degli esperimenti

Tutti gli esperimenti sono stati eseguiti utilizzando il modello qwen2.5:7b tramite Ollama, integrato con il simulatore SBSim attraverso i tool list_devices, read_point e write_point. Le chiamate al modello sono state effettuate impostando il parametro "**temperature**" pari a 0, al fine di ridurre la variabilità delle risposte e rendere maggiormente comparabili i risultati ottenuti nelle diverse configurazioni.

Per ogni richiesta del dataset è stata valutata in modo qualitativo la correttezza dell'output prodotto dal sistema. Una risposta è stata considerata corretta quando il modello ha selezionato il tool adeguato, individuato il dispositivo corretto, selezionato il sensore o attuatore coerente con la richiesta e, nel caso di una modifica, fornito un valore compatibile con l'operazione richiesta. Gli errori sono stati invece classificati in base alla causa principale osservata, distinguendo tra mancata chiamata del tool, selezione del tool errato, mancata selezione del dispositivo, identificazione errata del dispositivo, scelta non corretta del parametro o generazione di riferimenti non presenti nel simulatore.

Il dataset esteso comprende richieste appartenenti a categorie differenti:

- richieste di esplorazione generale della struttura del sistema (in giallo);
- comandi di lettura/scrittura di sensori e attuatori formulati esplicitamente o in linguaggio naturale (in verde);
- richieste vaghe o ambigue (in viola);
- riferimenti a zone o dispositivi inesistenti e richieste di lettura/scrittura di sensori/attuatori inesistenti o con valori fuori range (in rosso).

Questa varietà di input è stata predisposta al fine di valutare la capacità di interpretazione del modello non solo nei casi più semplici, ma anche rispetto a richieste più vicine a un'interazione naturale con l'utente.

4.2. Prima fase: confronto esplorativo tra prompt

La prima fase sperimentale è stata condotta su un dataset ridotto composto da dieci comandi base. Tali comandi includevano richieste di esplorazione della struttura del sistema, letture di parametri e modifiche di attuatori. Lo scopo di questa fase era valutare

se l'aumento del livello di dettaglio del prompt migliorasse in modo significativo la capacità del modello di selezionare correttamente il tool e generare argomenti coerenti.

4.2.1. System Prompt utilizzati

Sono state confrontate diverse configurazioni di system prompt (ossia i prompt dati in ingresso al sistema prima dell'inizio della simulazione, al fine di costruire il contesto di base e il metodo di ragionamento), a partire da versioni più sintetiche fino a system prompt più dettagliati, contenenti istruzioni operative, descrizioni della struttura del sistema e regole di ragionamento esplicite. In particolare, sono state testate le seguenti configurazioni, riportate in ordine crescente per livello di dettaglio:

- **prompt_funcalling_basic** (figura 4.1): rappresenta la configurazione di base del system prompt. Fornisce al modello una descrizione sintetica del suo compito, gli step logici principali da seguire e una struttura essenziale del sistema, distinguendo tra **Boiler**, **Air Handler** e **VAV**. Include inoltre le regole minime per scegliere tra **read_point** e **write_point**.

```
f"""
 Sei un assistente che controlla un simulatore SbSim tramite function calling.

 Il tuo compito è interpretare richieste in linguaggio naturale e usare i tool disponibili per leggere o modificare lo stato del sistema.

 Segui SEMPRE questo processo mentale prima di usare un tool:

 PASSO 0 - Comprendi l'obiettivo implicito

 PASSO 1 - Classifica la richiesta

 PASSO 2 - Filtra i device rilevanti

 PASSO 3 - Scegli il point corretto

 PASSO 4 - Usa il tool corretto
 - Se l'utente vuole leggere un valore, usa read_point (anche i setpoints possono essere letti)
 - Se l'utente vuole modificare un setpoint, usa write_point
```

Figura 4.1 Testo del system prompt "prompt_funcalling_basic" dato in input al modello

- **prompt_funcalling_withoutrules** (figura 4.2): mantiene la descrizione della struttura del sistema e il processo logico di interpretazione della richiesta, ma rimuove alcune regole operative incluse nelle versioni più dettagliate, relative alla selezione del sensore/attuatore corretto. Questa configurazione consente di valutare quanto incidano le regole esplicite di selezione rispetto alla sola presenza della struttura del sistema.

```
f"""
Sei un assistente che controlla un simulatore SbSim tramite function calling.

Il tuo compito è interpretare richieste in linguaggio naturale e usare i tool disponibili per leggere o modificare lo stato del sistema.

Segui SEMPRE questo processo mentale prima di usare un tool:

PASSO 0 - Comprendi l'obiettivo implicito
Se l'utente non specifica esplicitamente un device, inferisci l'obiettivo (es. temperatura, aria, acqua, comfort).

PASSO 1 - Classifica la richiesta
Decidi se la richiesta dell'utente è:
- una query: l'utente vuole leggere/conoscere un valore o uno stato
- un command: l'utente vuole modificare/impostare qualcosa

PASSO 2 - Filtra i device rilevanti
Seleziona solo i device plausibili per la richiesta dell'utente, in base alla struttura del sistema.

Se l'utente nomina una zona o una stanza specifica:
- prima scegli il device all'interno della zona
- poi scegli il point più vicino tra quelli del device
- non sostituire il device con un altro device globale solo per trovare un point più "perfetto"

PASSO 3 - Scegli il point corretto
Tra i device rilevanti, scegli:
- il measurement corretto se è una query
- il setpoint corretto se è un command

PASSO 3A - Verifica locale sul device scelto
Dopo aver scelto il device rilevante, confronta la richiesta SOLO con i measurement e setpoint realmente disponibili su quel device.

PASSO 4 - Usa il tool corretto
- Se l'utente vuole vedere cosa esiste nel sistema, usa list_devices
- Se l'utente vuole leggere un valore, usa read_point (anche i setpoints possono essere letti)
- Se l'utente vuole modificare un setpoint, usa write_point
```

Figura 4.2 Testo del system prompt “prompt_funcalling_withoutrules” dato in input al modello

- **prompt_funcalling_withoutrelations**: conserva la struttura del sistema, le istruzioni operative (figura 4.3) e la descrizione delle tre famiglie principali di dispositivi, ma rispetto alle versioni successive rimuove le relazioni semantiche tra linguaggio naturale e componenti del simulatore. Questa versione permette di valutare il contributo delle relazioni esplicitate per i vari dispositivi.

```
Regole:
- Non inventare device_id, zone_id, measurement_name o setpoint_name
- Se una zona o un device sono impliciti, inferiscili solo se la struttura del sistema lo rende plausibile
- Usa solo i point realmente disponibili
- Non simulare chiamate ai tool nel testo
- Non restituire pseudo-JSON nel contenuto testuale
- Se la richiesta richiede un'azione o una lettura, usa una tool call reale
- Quando hai già un device plausibile, non usare list_devices solo perché il nome del point non coincide perfettamente
- Scegli sempre il miglior point disponibile sul device rilevante

Regole aggiuntive per SBSim e function calling:
- Non inventare mai device_id completi. Se conosci solo il tipo, usa nomi generici come boiler, caldaia, ahu, air handler o vav: il backend li risolverà.
- Non usare default_zone_id come device_id: le zone non sono device.
- Per leggere un setpoint usa read_point con measurement_name uguale al nome del setpoint.
- Per modificare un setpoint usa write_point.
- Se l'utente dice supply water setpoint del boiler, usa device_id boiler e measurement_name supply_water_setpoint.
- Se l'utente dice damper o serranda di un VAV, usa supply_air_damper_percentage_command.
```

Figura 4.3 Regole aggiuntive specificate all’interno del “prompt_funcalling_withoutrelations”

- **prompt_funcalling_lowlevelwithoutscheme**: contiene un livello elevato di dettaglio nelle istruzioni operative, includendo oltre ai dettagli descritti nei system prompt, precedenti, anche uno schema relazionale utile a indirizzare le richieste

tramite esempi espliciti (figura 4.4). Rispetto alla versione più completa, non include lo schema causale dei concetti ambientali del sistema.

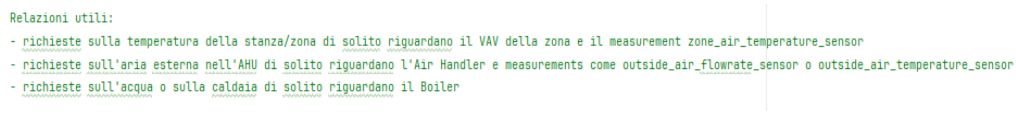


Figura 4.4 Relazioni utili specificate all'interno del "prompt_funcalling_lowlevelewithoutscheme"

- **prompt_funcalling_lowlevelwithcontext**: rappresenta la configurazione più completa. Oltre alle istruzioni operative dettagliate, include una descrizione esplicita della struttura del sistema, dei dispositivi disponibili, dei sensori e attuatori e delle relazioni semantiche utili per collegare richieste naturali a device e parametri tecnici. In aggiunta alle versioni precedenti presenta un breve schema causale utile a guidare il modello alla selezione dei parametri corretti in relazione ai concetti ambientali esplicitati dall'utente (figura 4.5).

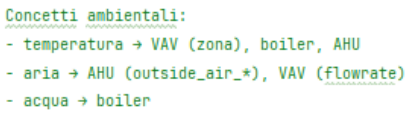


Figura 4.5 Schema causale incluso all'interno del "prompt_funcalling_lowlevelwithcontext"

Al fine di migliorare la capacità interpretativa del modello, tutti i prompt utilizzati in questa fase contengono una descrizione essenziale della struttura del sistema (figura 4.6) e sono stati testati mantenendo attivo l'aiuto semantico fornito dalla funzione **resolve_device_from_context** (di cui al §3.2.2).

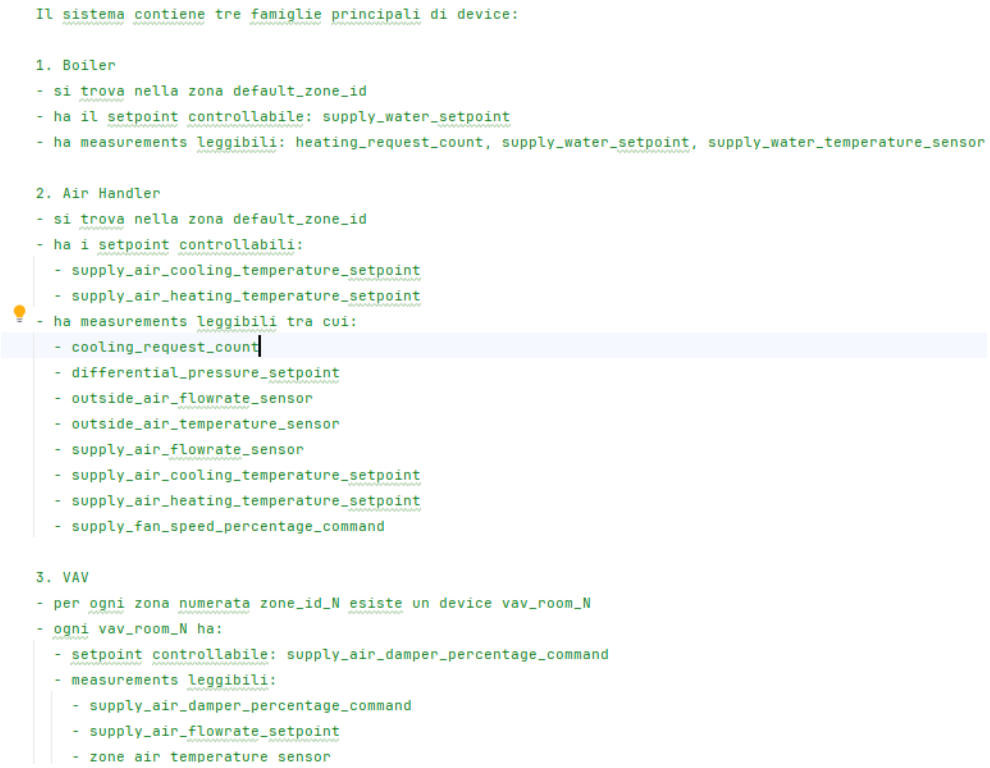


Figura 4.6 Descrizione della struttura integrata in tutti i prompt utilizzati nella prima fase

4.2.2. Dataset ristretto

Il dataset utilizzato nella prima fase sperimentale è composto da 10 istruzioni base, selezionate per coprire le principali tipologie di interazione previste dal sistema. Le richieste includono operazioni di esplorazione della struttura del simulatore, letture di parametri osservabili (formulate in modi differenti) e modifiche di attuatori controllabili, anche con valori fuori range al fine di testare la capacità del sistema di riconoscere e bloccare richieste che non possono essere soddisfatte.

L'esito di ciascuna richiesta è stato valutato come "OK" se rispettava la risposta attesa; in caso contrario è stato valutato "KO". La percentuale di successo di ogni prompt, relativamente a una singola run, è stata valutata come la percentuale di "OK" sul numero totale di richieste. Nella seguente tabella viene riportata la lista degli input utilizzati e la risposta attesa per ciascuno di essi.

RICHIESTA	RISPOSTA ATTESA
Mostrami tutti i device disponibili	Utilizzo del tool list_devices
Quali sensori sono disponibili nel sistema?	Utilizzo del tool list_devices
Dimmi temperatura zona 10	Restituzione valore di zone_air_temperature_sensor di vav_room_10
Qual è la temperatura nella stanza 25?	Restituzione valore di zone_air_temperature_sensor di vav_room_25
Dimmi il valore del sensore di temperatura in zona 3	Restituzione valore di zone_air_temperature_sensor di vav_room_3
Leggi supply_air_flowrate_setpoint del device vav_room_7	Restituzione valore di zone_air_temperature_sensor di vav_room_7
Imposta apertura serranda della stanza 10 a 0.5	Imposta supply_air_damper_percentage_command di vav_room_10 a 0.5
Apri la serranda della zona 10 al 70 percento	Imposta supply_air_damper_percentage_command di vav_room_10 a 0.7
Imposta la temperatura di mandata dell'aria a 30 gradi	Non applica la richiesta in quanto il valore è fuori range
Aumenta la temperatura dell'acqua a 320	Imposta supply_water_setpoint del boiler a 320

4.2.3. Risultati prima fase

Ciascun prompt è stato testato 10 volte al fine di valutare con maggiore stabilità la capacità di interpretazione del modello, riducendo l'incidenza di eventuali variazioni nelle risposte del modello e ottenendo in questo modo una prima indicazione sulla stabilità del comportamento osservato nelle diverse configurazioni sperimentali. La prestazione di ciascun prompt è stata valutata come la media delle prestazioni ottenute in ogni run. I risultati complessivi sono riportati nel paragrafo successivo.

I risultati ottenuti mostrano prestazioni complessivamente simili tra le diverse configurazioni di prompt. Sebbene il prompt più completo ottenga il risultato migliore (pari all'80%), il margine rispetto alle configurazioni più semplici rimane contenuto.

Le principali criticità osservate riguardano invece alcune richieste di lettura e la gestione di comandi formulati in modo meno aderente ai parametri tecnici attesi dal simulatore. A tale scopo, nella fase successiva il dataset è stato ampliato includendo un numero maggiore di richieste semantiche, ambigue o potenzialmente problematiche, così da valutare in modo più preciso il ruolo della struttura del sistema e dei meccanismi di risoluzione dei riferimenti.

4.2.4. Analisi del confronto esplorativo

Di seguito è riportata la tabella dei risultati ottenuti nella prima fase:

funcalling_basic	77,00%
funcalling_withoutrules	74,00%
funcalling_withoutrelations	74,00%
funcalling_lowlevelwithoutscheme	78,00%
funcalling_lowlevelwithcontext	80,00%

I risultati della prima fase hanno mostrato prestazioni complessivamente simili tra diverse configurazioni. Dai risultati della prima fase è emerso che tutte le configurazioni di prompt testate sono state in grado di gestire correttamente, nelle diverse run, le richieste esplorative e i comandi di modifica esplicita degli attuatori. In particolare, il modello ha interpretato correttamente le istruzioni finalizzate alla visualizzazione della struttura del sistema e alla modifica di parametri chiaramente identificabili, selezionando il tool adeguato, generando argomenti coerenti con la richiesta dell'utente e bloccando le richieste inesaudibili.

Le principali criticità sono invece emerse in corrispondenza di alcune richieste di lettura e di comandi formulati in modo meno aderente rispetto ai parametri attesi dal simulatore. In particolare, tutti i prompt hanno mostrato difficoltà nella gestione della richiesta relativa all'apertura della serranda espressa in formato percentuale, evidenziando un problema di interpretazione del valore e di conversione rispetto al formato atteso dall'attuatore. Inoltre, in alcune run, le richieste di lettura dei sensori/attuatori non sono state gestite in modo stabile: il modello ha occasionalmente fallito, talvolta non

richiamando alcun tool tramite function calling, in altri casi restituendo list_devices o infine individuando sensori non coerenti con la richiesta dell'utente.

Dai risultati ottenuti emerge quindi che l'aumento del dettaglio del prompt non produce un miglioramento significativo delle prestazioni. Questo risultato suggerisce che, almeno per richieste semplici, la quantità di istruzioni di ragionamento fornite al modello non costituisce un fattore determinante per il miglioramento delle prestazioni.

4.3. Seconda fase: confronto con dataset esteso

I risultati della prima fase sembrano mostrare che, su un dataset ristretto composto da richieste relativamente semplici, l'aumento del livello di dettaglio del prompt non produce un miglioramento particolarmente marcato delle prestazioni. Per questo motivo, nella seconda fase sperimentale, l'attenzione è stata spostata dal semplice confronto tra system prompt progressivamente più dettagliati alla valutazione del ruolo svolto dalla struttura del sistema e dai meccanismi di supporto implementati nel layer Python.

4.3.1. System Prompt utilizzati

Poiché nella prima fase sperimentale le configurazioni di system prompt avevano restituito risultati complessivamente simili, nella seconda fase non sono state confrontate nuovamente tutte le configurazioni, selezionando invece i due system prompt posti agli estremi del livello di complessità, ossia **prompt_funcalling_basic** e **prompt_funcalling_lowlevelwithcontext** (paragrafo §4.2.1). A questi sono state aggiunte due ulteriori varianti del prompt_funcalling_basic, al fine di confrontare e valutare separatamente il contributo della conoscenza strutturale e del supporto semantico fornito dal layer Python. In particolare, le due configurazioni aggiuntive sono le seguenti:

- **prompt_basic_withoutstructure**, nel quale, a partire dal prompt_funcalling_basic, viene rimossa la descrizione esplicita della struttura del sistema, lasciando al modello un livello minore di conoscenza iniziale sull'ambiente;
- prompt_funcalling_basic **senza resolve_device_from_context**: in questa configurazione viene impiegato il prompt_funcalling_basic, ma viene disattivato

il supporto offerto dalla funzione resolve_device_from_context. A livello tecnico, questa operazione è stata effettuata riducendo l'output della funzione resolve_device_from_context alla sola restituzione del device_id generato dal modello, togliendo quindi la possibilità di correggere o completare riferimenti parziali, sinonimici o non perfettamente aderenti agli identificativi tecnici presenti nel simulatore.

4.3.2. Dataset esteso

Gli esperimenti della seconda fase sono stati condotti stilando un dataset più esteso, composto da 56 richieste. Oltre a un aumento quantitativo degli user input, rispetto al dataset ristretto utilizzato nella prima fase, il nuovo insieme di input includeva una maggiore variabilità nelle tipologie di richiesta. L'obiettivo era valutare il comportamento del modello in condizioni più vicine a un'interazione reale (nella quale l'utente può formulare in vario modo richieste articolate, incomplete o imprecise), verificando il contributo differenziale che il layer intermedio apporta a supporto del modello.

L'esito di ciascuna richiesta è stato valutato come "OK" se rispettante la risposta attesa; in caso contrario è stato valutato "KO". La percentuale di successo di ogni prompt, relativamente a una singola run, è stata valutata come la percentuale di "OK" sul numero totale di richieste. Nella seguente tabella viene riportata la lista di input utilizzati e la risposta attesa per ciascuno di essi.

RICHIESTA	RISPOSTA ATTESA
Quali device sono disponibili nel sistema?	Utilizzo del tool list_devices
Quali sensori sono disponibili nel sistema?	Utilizzo del tool list_devices
Mostrami zone, device, setpoint e measurement disponibili	Utilizzo del tool list_devices
Fammi vedere la struttura dell'edificio simulato	Utilizzo del tool list_devices
Quali attuatori posso controllare?	Utilizzo del tool list_devices
Dimmi supply_water_setpoint del boiler	Restituzione valore di supply_water_setpoint del boiler
Leggi supply_water_temperature_sensor del boiler	Restituzione valore di supply_water_temperature_sensor del boiler
Leggi supply_air_flowrate_sensor dell'AHU	Restituzione valore di supply_air_flowrate_sensor dell'AHU
Dimmi outside_air_flowrate_sensor della centrale aria	Restituzione valore di outside_air_flowrate_sensor dell'AHU

Leggi cooling_request_count dell'air handler	Restituzione valore di cooling_request_count dell'AHU
Dimmi zone_air_temperature_sensor di vav_room_1	Restituzione valore di zone_air_temperature_sensor di vav_room_1
Leggi supply_air_damper_percentage_command di vav_room_10	Restituzione valore di supply_air_damper_percentage_command di vav_room_10
Qual è la temperatura della stanza 3?	Restituzione valore di zone_air_temperature_sensor di vav_room_3
Dimmi la temperatura della zona 10	Restituzione valore di zone_air_temperature_sensor di vav_room_10
Quanto è calda la stanza 25?	Restituzione valore di zone_air_temperature_sensor di vav_room_25
Leggi la temperatura ambiente della zona 7	Restituzione valore di zone_air_temperature_sensor di vav_room_7
Dimmi il valore della serranda della stanza 12	Restituzione valore di supply_air_damper_percentage_command di vav_room_12
Qual è l'apertura della serranda in zona 4?	Restituzione valore di supply_air_damper_percentage_command di vav_room_4
Dimmi il setpoint dell'acqua calda prodotta dal boiler	Restituzione valore di supply_water_setpoint del boiler
Qual è il target di raffreddamento dell'aria centrale?	Restituzione valore di supply_air_cooling_temperature_setpoint dell'AHU
Qual è il target di riscaldamento dell'aria immessa?	Restituzione valore di supply_air_heating_temperature_setpoint dell'AHU
Dimmi quante richieste di raffreddamento ci sono nell'AHU	Restituzione valore di cooling_request_count dell'AHU
Imposta supply_water_setpoint del boiler a 330	Imposta supply_water_setpoint del boiler a 330
Imposta supply_air_cooling_temperature_setpoint della centrale aria a 305	Imposta supply_air_cooling_temperature_setpoint dell'AHU a 305
Imposta supply_air_damper_percentage_command di vav_room_5 a 0.5	Imposta supply_air_damper_percentage_command di vav_room_5 a 0.5
Metti supply_air_damper_percentage_command di vav_room_20 a 0.8	Imposta supply_air_damper_percentage_command di vav_room_20 a 0.8
Aumenta il target dell'acqua calda del boiler a 335	Imposta supply_water_setpoint del boiler a 335
Porta la temperatura target di raffreddamento dell'AHU a 300	Imposta supply_air_cooling_temperature_setpoint dell'AHU a 300
Apri la serranda della stanza 8 al 60%	Imposta supply_air_damper_percentage_command di vav_room_8 a 0.6
Chiudi quasi del tutto la serranda della zona 9	Imposta supply_air_damper_percentage_command di vav_room_9 a un valore basso, ad esempio 0.1

Metti la serranda della stanza 15 a metà	Imposta supply_air_damper_percentage_command di vav_room_15 a 0.5
Alza il setpoint dell'acqua prodotta dalla caldaia	Richiesta incompleta: manca il valore da impostare
Abbassa il setpoint di raffreddamento della centrale aria a 295	Imposta supply_air_cooling_temperature_setpoint dell'AHU a 295
Fa troppo caldo nella stanza 6	Richiesta ambigua: non eseguire modifiche senza specificare parametro e valore
Aumenta il comfort nella stanza 2	Richiesta ambigua: non eseguire modifiche senza specificare parametro e valore
Riduci i consumi dell'aria centrale	Richiesta ambigua: non eseguire modifiche senza specificare parametro e valore
Migliora la situazione termica dell'edificio	Richiesta ambigua: non eseguire modifiche senza specificare parametro e valore
Ottimizza il sistema HVAC	Richiesta ambigua: non eseguire modifiche senza specificare parametro e valore
Abbassa un po' la temperatura	Richiesta ambigua: non eseguire modifiche senza specificare zona/dispositivo, parametro e valore
Aumenta un po' l'aria	Richiesta ambigua: non eseguire modifiche senza specificare zona/dispositivo, parametro e valore
Dimmi la temperatura della zona 999	Errore: zona 999 non esistente
Qual è la temperatura della stanza 0?	Errore: zona 0 non esistente
Leggi zone_air_temperature_sensor di vav_room_999	Errore: dispositivo vav_room_999 non esistente
Imposta supply_air_damper_percentage_command di vav_room_999 a 0.5	Errore: dispositivo vav_room_999 non esistente
Leggi humidity_sensor di vav_room_1	Errore: sensore humidity_sensor non disponibile per vav_room_1
Dimmi pressure_sensor del boiler	Errore: sensore pressure_sensor non disponibile per il boiler
Leggi co2_sensor della stanza 5	Errore: sensore co2_sensor non disponibile per vav_room_5
Qual è il livello di umidità della zona 10?	Errore: sensore di umidità non disponibile per vav_room_10
Imposta humidity_setpoint della stanza 4 a 50	Errore: setpoint humidity_setpoint non disponibile per vav_room_4
Imposta fan_speed_setpoint dell'AHU a 70	Errore: setpoint fan_speed_setpoint non disponibile per l'AHU
Imposta pressure_setpoint del boiler a 2	Errore: setpoint pressure_setpoint non disponibile per il boiler
Accendi la luce della stanza 3	Errore: attuatore relativo all'illuminazione non disponibile
Imposta supply_air_damper_percentage_command di vav_room_1 a 2	Errore: valore non coerente per supply_air_damper_percentage_command, atteso tra 0 e 1 supply_water_setpoint Errore: valore non numerico per supply_air_heating_temperature_setpoint

Imposta supply_air_damper_percentage_command di vav_room_1 a -0.5	Errore: valore non coerente per supply_air_damper_percentage_command, atteso tra 0 e 1
Imposta supply_water_setpoint del boiler a -10	Errore: valore non coerente per per supply_water_setpoint
Imposta supply_air_heating_temperature_setpoint dell'AHU a abc	Errore: valore non numerico per supply_air_heating_temperature_setpoint

4.3.3. Risultati seconda fase

Come per la prima fase, ciascun prompt è stato testato 10 volte per valutare con maggiore stabilità la capacità di interpretazione del modello, riducendo l'incidenza di eventuali variazioni. I risultati ottenuti sono riportati nel paragrafo successivo.

I risultati della seconda fase mostrano una differenza più marcata tra le configurazioni rispetto alla prima fase. I due prompt dotati di struttura del sistema, **funcalling_basic** e **funcalling_lowlevelwithcontext**, ottengono entrambi una percentuale media di successo pari al 69,64%. La rimozione della struttura dal prompt porta invece il risultato al 39,29%, mentre la disattivazione del supporto semantico fornito da **resolve_device_from_context** riduce ulteriormente la prestazione al 32,14%.

Data la rilevanza della struttura e del resolver suggerita dai risultati ottenuti in questa fase, è stato introdotto un ulteriore esperimento volto a verificare se il modello potesse costruire autonomamente una rappresentazione del contesto operativo attraverso l'esplorazione progressiva dell'ambiente.

4.3.4. Analisi del ruolo della struttura e del resolver

Di seguito sono riportati i risultati complessivi ottenuti per le configurazioni testate nella seconda fase:

funcalling_basic + resolver disattivato	32,14%
basic_withoutstructure	39,29%
funcalling_basic	69,64%
funcalling_lowlevelwithcontext	69,64%

I risultati della seconda fase sembrano confermare quanto emerso nella prima fase. Le due configurazioni dotate di struttura del sistema (**funcalling_lowlevelwithcontext** e

prompt_funcalling_basic) ottengono lo stesso risultato medio complessivo, pari a 39 richieste corrette su 56. Questo dato suggerisce che, sul dataset esteso, l'aggiunta di regole operative più dettagliate, relazioni semantiche e schema causale non produce un miglioramento evidente rispetto a un prompt più sintetico, purché quest'ultimo contenga comunque una descrizione essenziale della struttura del sistema.

Il calo più significativo si osserva invece quando la struttura del sistema viene rimossa dal prompt. In questa configurazione, il numero di richieste corrette scende da 39 a 22 su 56. Tale risultato indica che la conoscenza preliminare della struttura dell'ambiente costituisce un elemento rilevante per la corretta interpretazione delle richieste. Senza informazioni esplicite su dispositivi, sensori e attuatori disponibili, il modello tende più frequentemente a produrre risposte non operative, a selezionare parametri non coerenti o a non generare la tool call necessaria.

Il risultato peggiora ulteriormente quando viene disattivato l'aiuto semantico del layer Python. In questo caso le richieste corrette scendono a 18 su 56. Questo dato evidenzia il ruolo svolto dalla funzione resolve_device_from_context nel compensare alcune imprecisioni del modello, soprattutto nei casi in cui l'utente utilizza riferimenti naturali o parziali ai dispositivi. Ad esempio, una richiesta che menziona genericamente "la caldaia" o "il boiler" deve essere ricondotta all'identificativo tecnico effettivamente presente nel simulatore, che può variare dinamicamente a ogni esecuzione. In assenza di un meccanismo di risoluzione, il modello deve produrre direttamente l'identificativo corretto, ma non sempre dispone delle informazioni necessarie per farlo.

Dal confronto tra le quattro configurazioni emerge quindi come il fattore più rilevante non sia tanto la quantità complessiva di istruzioni inserite nel prompt, quanto la disponibilità di una rappresentazione utile della struttura del sistema e di un meccanismo di supporto per il mapping tra linguaggio naturale e identificativi tecnici. Il prompt più completo e quello di base ottengono infatti la stessa prestazione, mentre la rimozione della struttura produce un calo marcato. Questo suggerisce che, nel contesto analizzato, l'LLM è in grado di seguire istruzioni operative anche relativamente sintetiche, ma incontra maggiori difficoltà quando deve dedurre autonomamente la struttura tecnica dell'ambiente.

Le principali criticità osservate nella seconda fase riguardano tre aspetti. Il primo è l'identificazione dei dispositivi, soprattutto quando il riferimento dell'utente non coincide

con il device_id tecnico previsto dal simulatore. Il secondo riguarda la selezione del parametro corretto, in particolare nei casi in cui più sensori o attuatori sono semanticamente vicini tra loro. Infine, il terzo riguarda la gestione delle richieste ambigue o non eseguibili, nelle quali il modello deve decidere se produrre una tool call, chiedere chiarimenti o restituire un errore coerente.

Nel complesso, la seconda fase mostra che il modello può interagire con il simulatore in modo efficace quando dispone di una struttura minima del sistema e quando il backend fornisce un supporto alla risoluzione dei riferimenti. Tuttavia, la riduzione delle prestazioni nelle configurazioni prive di struttura e di supporto semantico indica che il solo modello linguistico non è sufficiente a garantire un grounding affidabile tra richiesta naturale e componenti tecnici del simulatore.

4.4. Terza fase: costruzione incrementale del contesto

Dopo aver valutato l'effetto di diverse configurazioni di prompt, è stato condotto un ulteriore esperimento con l'obiettivo di verificare se il modello fosse in grado di costruire progressivamente una rappresentazione utile della struttura del simulatore. L'idea alla base di questa prova era ridurre la quantità di conoscenza strutturale fornita inizialmente nel system prompt e osservare se Qwen fosse in grado di ricavare autonomamente le informazioni necessarie attraverso l'interazione con i tool disponibili.

In particolare, l'esperimento è stato impostato cercando di far utilizzare al modello il tool **list_devices** come fonte informativa iniziale. L'output prodotto da tale tool contiene infatti informazioni relative a zone, dispositivi, sensori e attuatori disponibili nel simulatore. L'obiettivo era quello di far analizzare al modello la struttura ricavata da tale output sintetizzandola in un contesto più leggibile, allo scopo di utilizzare successivamente tale contesto per interpretare in modo più corretto le richieste dell'utente.

L'esperimento aveva quindi lo scopo di valutare una forma di costruzione incrementale del contesto: invece di fornire al modello una descrizione completa e predefinita della struttura SBSim, si è provato a far emergere tale conoscenza nel corso dell'interazione. Questo approccio avrebbe potuto rendere il sistema più flessibile rispetto a configurazioni

diverse del simulatore, poiché il modello avrebbe potuto adattarsi alla struttura osservata senza richiedere una descrizione manuale dettagliata all'interno del prompt.

Tuttavia, i risultati ottenuti non hanno evidenziato un miglioramento significativo. Il modello è stato effettivamente in grado di produrre descrizioni testuali della struttura osservata, ma tali descrizioni non si sono tradotte in una rappresentazione sufficientemente precisa e operativa per guidare correttamente le successive chiamate ai tool. In alcuni casi, il modello ha generato una descrizione generica di un sistema HVAC, introducendo elementi non coerenti con la struttura effettiva del simulatore, come un numero errato di zone o componenti non presenti nell'ambiente considerato. In altri casi, il modello ha riconosciuto alcune informazioni corrette relative ai VAV, ai setpoint e alle measurement disponibili, ma la rappresentazione prodotta è rimasta parziale e non sufficiente per risolvere in modo affidabile il problema del mapping tra linguaggio naturale e identificativi tecnici. Maggiori dettagli sono riportati nell'analisi dedicata all'interno del paragrafo §4.4.1.

La criticità principale emersa riguarda quindi la differenza tra comprendere descrittivamente una struttura e utilizzarla operativamente all'interno di una tool call. Il modello può produrre una sintesi apparentemente coerente dell'output di **list_devices**, ma ciò non garantisce che sia poi in grado di selezionare correttamente il **device_id**, il **measurement_name** o il **setpoint_name** da utilizzare nella chiamata successiva. Questo limite è particolarmente rilevante nel caso di SBSim, dove l'interazione con il simulatore richiede identificativi tecnici esatti e parametri coerenti con le interfacce previste.

4.4.1. Analisi della costruzione incrementale del contesto

Le criticità dei prompt incrementali generati dal modello sono riassunte di seguito. Per ciascun caso viene sintetizzato il contenuto prodotto e viene evidenziata la principale criticità osservata rispetto all'obiettivo di costruire un contesto operativo utilizzabile nelle successive chiamate ai tool.

- **Prompt generato 1** (figura 4.7)**:**
    - **Sintesi del contenuto prodotto dal modello:** il modello descrive il sistema come una configurazione HVAC composta da 20 zone, ciascuna associata a componenti come serrande, ventole di ritorno e sensori di temperatura.

- **Criticità osservata:** La descrizione non risulta coerente con la struttura effettiva del simulatore, che comprende 128 zone, di cui 127 numerate e una default_zone. Inoltre, il modello introduce componenti e valori di esempio non presenti nella rappresentazione operativa utilizzata dal sistema (return air fan, sensori supply/return, timestamp fittizi). La mancata menzione della default_zone è particolarmente rilevante. In essa sono infatti collocati boiler e AHU.

```
1. **System Configuration**:

   * The system is configured with 20 zones.
   * Each zone has multiple components: supply air dampers, return air fans, and temperature sensors.
2. **Components per Zone**:

   * Each zone (e.g., `vav\\\_room\\\_1`, `vav\\\_room\\\_2`) contains:

     * A supply air damper with a unique ID.
     * A return air fan with a unique ID.
     * Temperature sensors for both the supply and return air.
3. **Component Details**:

   * Each component has an ID, type (e.g., "supply\_air\_damper"), and possibly other attributes like `name` or `description`.
   * The state of each component is represented by its current value (`value`) and a timestamp (`timestamp`).
4. **State Representation**:

   * The system tracks the state of each component, including whether it's open/closed (for dampers) or running/stopped (for fans)
   * Temperature values are recorded for both supply and return air.
5. **Write Operations**:

   * There are no write operations applied in this snapshot (`writes\\\ applied` is an empty list).
```

Figura 4.7 Estratto del primo prompt generato

- **Prompt generato 2** (figura 4.8)**:**
  - **Sintesi del contenuto prodotto dal modello:** il modello riconosce una struttura composta da VAV room, associando a ciascun elemento uno zone_id, un attuatore e alcuni sensori disponibili.
  - **Criticità osservata:** l'output è più vicino alla struttura reale rispetto al primo caso, ma rimane parziale. Il modello si concentra sui dispositivi VAV e non ricostruisce il ruolo della default_zone, del boiler e dell'AHU. Inoltre, il valore **device_type = 4** viene trattato come informazione centrale, e al tempo stesso non viene esplicitata una regola operativa per ricavare il device_id effettivo da usare nelle tool call (come vav_room_1, vav_room_2, e così via).

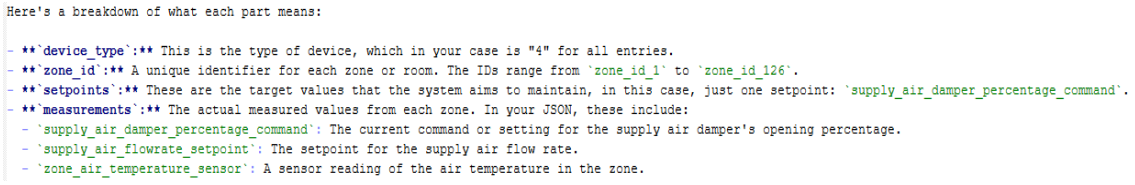


Figura 4.8 Estratto del secondo prompt generato

- **Prompt generato 3** (figura 4.9)**:**
    - **Sintesi del contenuto prodotto dal modello:** il modello interpreta gli elementi osservati come configurazioni di stanze VAV, indicando per ciascuna voce device_type, zone_id, setpoint e measurement associate.
    - **Criticità osservata:** La rappresentazione individua alcuni campi corretti, ma non risolve il problema principale dell'esperimento: trasformare una descrizione della struttura in un contesto operativo utilizzabile. In particolare, il modello non chiarisce come collegare richieste in linguaggio naturale (ad es. "temperatura della stanza 7"), agli identificativi tecnici richiesti dai tool.

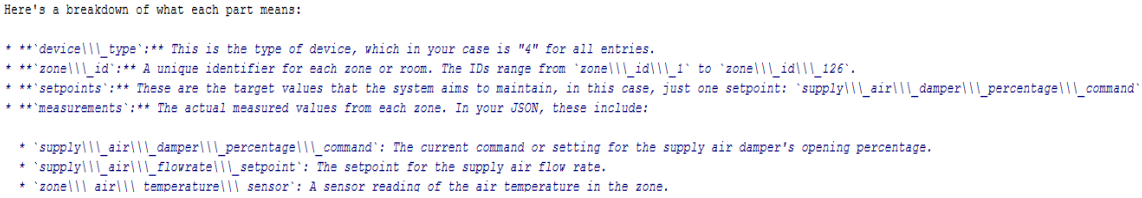


Figura 4.9 Estratto del terzo prompt generato

- **Prompt generato 4** (figura 4.10)**:**
    - **Sintesi del contenuto prodotto dal modello:** il modello produce nuovamente una descrizione delle VAV room, specificando che ciascun elemento contiene un zone_id, un setpoint e alcune measurement disponibili.
    - **Criticità osservata:** anche in questo caso il modello mostra una capacità di sintesi descrittiva, ma non è in grado di determinare stabilmente il contesto operativo. L'output, infatti, non integra la struttura completa del simulatore, non considera la default_zone e non fornisce istruzioni affidabili per

distinguere tra richieste sui VAV, sul boiler e sull'AHU. Inoltre, la parte finale assume la forma di una spiegazione generale di possibili usi del sistema HVAC, invece di diventare una guida concreta per le successive chiamate ai tool.

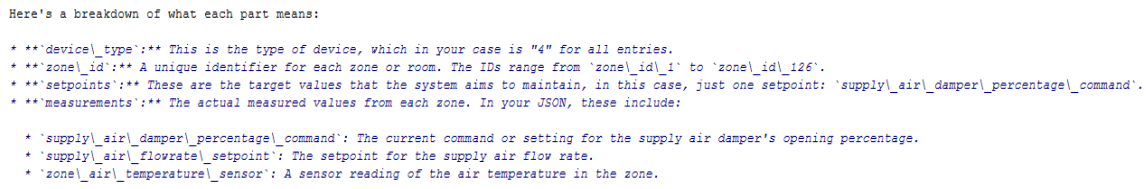


Figura 4.10 Estratto del quarto prompt generato

L'esperimento mostra quindi che la costruzione automatica del contesto, almeno nella configurazione testata, non è sufficiente a sostituire una descrizione strutturata fornita nel prompt o una logica di supporto implementata nel layer Python. La conoscenza ottenuta tramite **list_devices**, infatti, viene trattata dal modello soprattutto come informazione descrittiva, mentre l'esecuzione corretta delle richieste richiede una rappresentazione più vincolata e direttamente utilizzabile.

Questo risultato conferma l'importanza del layer applicativo come componente di mediazione tra l'output del modello e l'ambiente simulato. In particolare, funzioni come **resolve_device_from_context** e i controlli implementati in **esegui_intent** risultano necessarie per trasformare richieste formulate in linguaggio naturale in operazioni coerenti con la struttura tecnica di SBSim.

4.5. Quarta fase: esperimento ReAct-Like

4.5.1. Configurazione dell'esperimento

La quarta fase sperimentale ha riguardato l'introduzione di una modalità di interazione ispirata al paradigma ReAct (paragrafo §2.4). L'obiettivo di questa prova non era misurare in modo quantitativo le prestazioni complessive della configurazione ReAct-like sull'intero dataset sperimentale, ma analizzare in modo mirato se il modello fosse in

grado di costruire un ragionamento iterativo utile a ricondurre una richiesta in linguaggio naturale all'identificativo tecnico corretto del dispositivo. Per questo motivo è stato mantenuto costante il system prompt ed è stato utilizzato un singolo input rappresentativo di una criticità già emersa negli esperimenti precedenti. Tale scelta consente di osservare più chiaramente se l'alternanza tra tool call, osservazione e nuova generazione del modello sia sufficiente a correggere un riferimento descrittivo non direttamente compatibile con gli identificativi richiesti dal simulatore.

In questa fase non è stata implementata una versione ReAct classica, basata su template testuali delimitati da opportune **stopwords** quali "Thought", "Action", "Action Input" e "Observation". Tale scelta è stata motivata dal fatto che le linee guida operative consultate per Qwen2.5 sconsigliano l'utilizzo di template ReAct basati su parole chiave e parsing testuale, poiché questo approccio può introdurre errori nella ricostruzione delle chiamate ai tool[31]. Per questo motivo, invece di richiedere al modello di produrre azioni in forma testuale, si è scelto di mantenere il meccanismo di function calling strutturato già utilizzato negli esperimenti precedenti, introducendo però un ciclo iterativo che reinserisce nel contesto del modello le osservazioni prodotte dai tool.

La modalità sperimentata può essere definita "ReAct-like": essa non replica formalmente il template originale del paradigma ReAct, ma ne riprende la logica operativa principale, cioè l'alternanza tra azione, osservazione e successiva decisione del modello (figura 4.11). In questa configurazione, Qwen riceve a ogni iterazione la richiesta iniziale dell'utente, la cronologia dei messaggi precedenti e l'eventuale risultato restituito dal tool nell'iterazione precedente. Sulla base di queste informazioni, il modello può decidere se generare una nuova tool call, correggendo eventualmente gli argomenti utilizzati in precedenza, oppure se produrre una risposta testuale finale.

Dal punto di vista implementativo, è stata realizzata una funzione alternativa, denominata **interpreta_prompt_react**, che sostituisce la chiamata one-shot al modello con un ciclo composto da più iterazioni. A ogni iterazione il modello può generare una tool call oppure una risposta finale. Se viene prodotta una tool call, questa viene eseguita tramite la funzione **esegui_tools**; il risultato viene poi trasformato in osservazione e reinserito nella lista dei messaggi, in modo che il modello possa utilizzarlo nell'iterazione successiva.

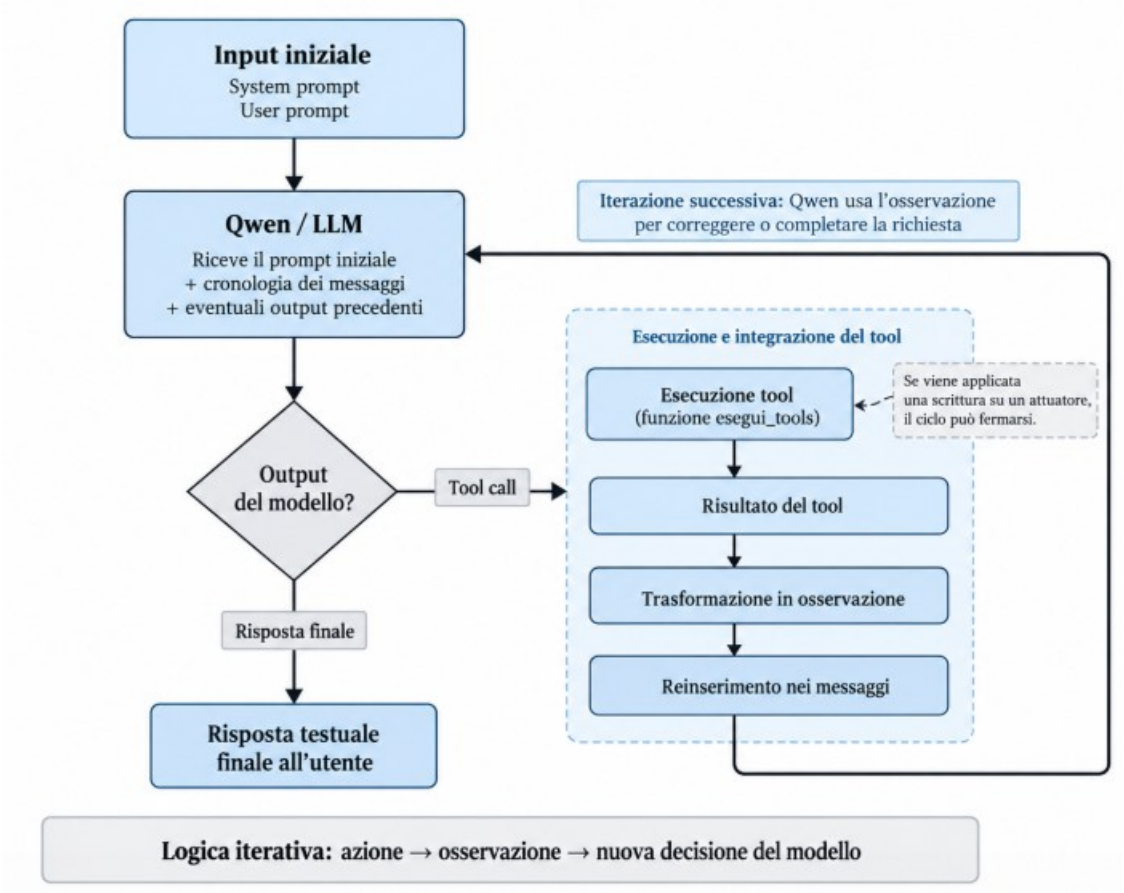


Figura 4.11 Diagramma di flusso della modalità ReAct-Like implementata

L'esperimento è stato condotto mantenendo disattivato l'aiuto semantico fornito dalla funzione **resolve_device_from_context**. In questa configurazione, il layer Python non corregge automaticamente riferimenti generici come "boiler" o "centrale aria", ma utilizza il **device_id** prodotto dal modello senza modificarlo. Questa scelta ha permesso di isolare il comportamento del modello e di verificare se, attraverso più iterazioni e attraverso l'osservazione dei risultati restituiti dai tool, Qwen fosse in grado di risalire autonomamente all'identificativo tecnico corretto.

Come input principale è stato utilizzato lo user prompt "dimmi **supply_water_setpoint** del **boiler**". Questa richiesta indica in modo esplicito il parametro da leggere, al contrario del dispositivo che viene espresso tramite un riferimento descrittivo (boiler) che non coincide con l'identificativo tecnico effettivo presente nel simulatore. Tale caso risulta quindi particolarmente significativo, in quanto il dispositivo del boiler presenta un identificativo dinamico (nella forma "boiler_id_<sequenza alfanumerica>") che varia per ogni simulazione, richiedendo quindi una corretta associazione tra nome descrittivo e

identificativo tecnico. Lo user prompt selezionato risultava quindi particolarmente coerente con l'obiettivo del test, ossia quello di verificare se il ciclo ReAct-like, pur in assenza della risoluzione semantica implementata nel layer Python, fosse in grado di guidare il modello verso una nuova chiamata corretta dopo aver ricevuto un errore o dopo aver osservato la struttura del sistema tramite i tool disponibili.

4.5.2. Risultati ottenuti

I tentativi effettuati hanno mostrato che la modalità ReAct-like è stata integrata correttamente dal punto di vista del flusso operativo. Il sistema è infatti in grado di alternare chiamate ai tool, osservazioni e nuove interrogazioni al modello, seguendo una logica iterativa coerente con l'obiettivo dell'esperimento.

Tuttavia, la configurazione testata non è risultata sufficiente a risolvere autonomamente il problema dell'identificazione degli identificativi tecnici dei dispositivi. Nei diversi tentativi, il modello non è riuscito a trasformare stabilmente le osservazioni ricevute dai tool in una nuova chiamata corretta. In alcuni casi ha utilizzato riferimenti descrittivi non validi, come "boiler", al posto del relativo identificativo tecnico; in altri ha richiamato **list_devices**, ma si è limitato a produrre una descrizione della struttura osservata senza poi completare tramite il tool **read_point** la richiesta espressa nello user prompt.

Un risultato parzialmente positivo è emerso quando l'input conteneva già un identificativo tecnico corretto, come nel caso delle richieste di lettura "dimmi **supply_air_flowrate_setpoint** di **vav_room_7**" e di scrittura "imposta **supply_percentage_damper** di **vav_room_8** a **0.6**". In tale situazione, il modello ha generato correttamente la chiamata al tool e ha utilizzato il risultato ottenuto per produrre una risposta testuale più leggibile per l'utente.

Nel complesso, l'esperimento suggerisce quindi che la modalità ReAct-like possa essere utile per organizzare l'interazione in più passaggi e per riformulare in modo naturale i risultati restituiti dai tool, ma non sia sufficiente, nella configurazione testata, a sostituire il supporto semantico fornito dal layer Python nella risoluzione dei riferimenti ai dispositivi. L'analisi dettagliata dei singoli tentativi è riportata di seguito.

4.5.3. Analisi dell'implementazione iterativa ReAct-like

Di seguito sono riportati i vari tentativi effettuati e i relativi risultati ottenuti:

- **Tentativo 1**
    - **Modifica introdotta:** nessuna modifica aggiuntiva rispetto al prompt di base (prompt_funcalling_basic).
    - **Risultato osservato** (figura 4.12)**:** il modello ha utilizzato boiler come device_id, senza riuscire a ricondurre il riferimento descrittivo all'identificativo tecnico effettivo del dispositivo. Dopo l'errore restituito dal sistema, non è stata generata una nuova tool call corretta per completare la richiesta.

```
Risultato function calling:
final_text: Il device 'boiler' non esiste nel sistema. I dispositivi disponibili sono:

- boiler_id_0a1592f4-f427-4bf6-90a7-6c0ef6b26bd0 (appartiene alla zona default_zone_id)
- air_handler_id_684e24c0-9f9b-4b7c-8e85-b7cd2a22b93e (appartiene alla zona default_zone_id)
```

Figura 4.12 Estratto del risultato ottenuto dal Tentativo 1

- **Tentativo 2**
    - **Modifica introdotta:** sono state aggiunte regole esplicite per consentire l'uso dei tool in più passaggi, utilizzare list_devices quando il device_id non è noto, sfruttare le osservazioni precedenti e non inventare dispositivi, sensori o attuatori.
    - **Risultato osservato** (figura 4.13)**:** il modello non è comunque riuscito a individuare correttamente il dispositivo richiesto. Inoltre, invece di correggere operativamente la chiamata, ha prodotto una risposta testuale non risolutiva.

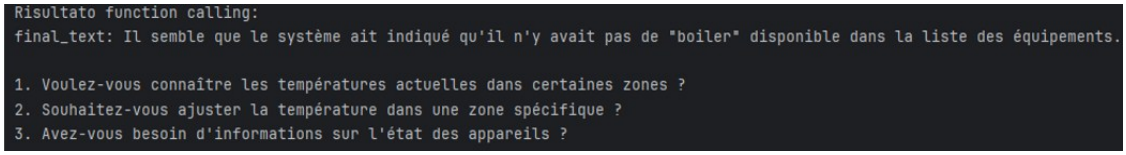


Figura 4.13 Estratto del risultato ottenuto dal Tentativo 2

- **Tentativo 3**
    - **Modifica introdotta:** sono state aggiunte ulteriori regole, tra cui l'obbligo di rispondere in italiano e l'indicazione di utilizzare identificativi con prefisso **boiler_id** o **air_handler_id** quando presenti nelle osservazioni.
    - **Risultato osservato** (figura 4.14)**:** anche questa configurazione non ha prodotto il risultato atteso. Il modello ha ignorato in parte le istruzioni ricevute, producendo una risposta in lingua diversa dall'italiano e una descrizione generica del sistema, senza arrivare alla tool call corretta.

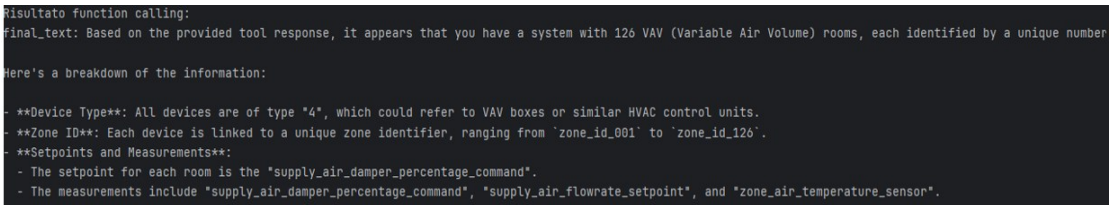


Figura 4.14 Estratto del risultato ottenuto dal Tentativo 3

- **Tentativo 4**
    - **Modifica introdotta:** è stato modificato il ciclo della funzione interpreta_prompt_react per evitare che il processo si interrompesse dopo un errore. In questi casi veniva aggiunto un nuovo messaggio al modello, richiedendo di usare l'osservazione precedente per generare una nuova tool call operativa. Il numero massimo di iterazioni è stato inoltre aumentato a quattro.
    - **Risultato osservato** (figura 4.15)**:** il modello ha richiamato il tool list_devices, ma si è limitato a produrre una descrizione della struttura del

sistema. Il ciclo si è quindi fermato prima che venisse generata la chiamata read_point necessaria per completare la richiesta iniziale.

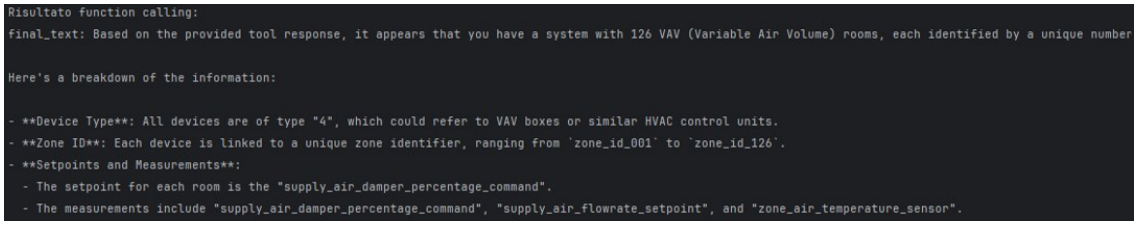


Figura 4.15 Estratto del risultato ottenuto dal Tentativo 4

- **Tentativo 5**
  - **Modifica introdotta:** è stata aggiunta una condizione specifica per impedire la chiusura del ciclo dopo l'utilizzo di list_devices, nei casi in cui la richiesta originale richiedesse ancora una lettura o una modifica. Il modello è stato quindi istruito a usare le informazioni osservate per generare una nuova tool call operativa.
  - **Risultato osservato** (figura 4.16)**:** Il modello ha effettivamente proseguito nelle iterazioni successive, utilizzando tutti i cicli disponibili. Tuttavia, pur arrivando a indicare nella risposta finale l'identificativo tecnico del boiler, non è riuscito a utilizzare tale informazione per generare una nuova chiamata **read_point** corretta e restituire il valore richiesto di **supply_water_setpoint**, evidenziando una difficoltà nel trasformare l'osservazione della struttura in un'azione operativa coerente.

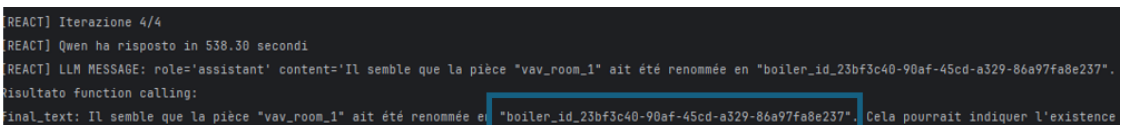


Figura 4.16 Estratto del risultato ottenuto dal Tentativo 5

- **Tentativo 6**
    - **Modifica introdotta:** è stato modificato il system prompt per chiarire che termini come "**boiler**", "**air handler**" e "**VAV**" indicano famiglie o tipologie di dispositivi, e non identificativi tecnici da utilizzare direttamente nelle tool call.
    - **Risultato osservato** (figura 4.17)**:** il comportamento del modello rimane sostanzialmente analogo ai tentativi precedenti: anche chiarendo la distinzione tra categoria logica e identificativo tecnico, il modello non è riuscito a risalire in modo affidabile al device_id corretto.

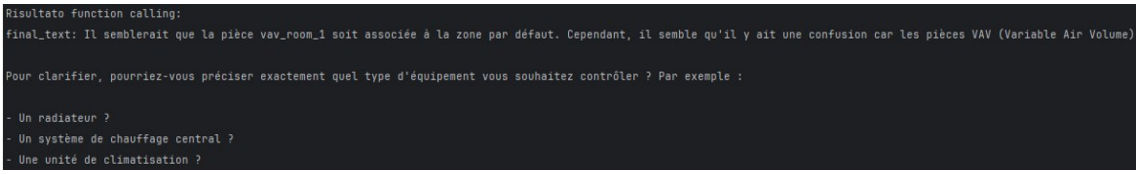


Figura 4.17 Estratto del risultato ottenuto dal Tentativo 6

Nel complesso, i tentativi effettuati mostrano che la modalità ReAct-like è stata integrata correttamente dal punto di vista del flusso operativo, poiché il sistema riesce ad alternare chiamate ai tool, osservazioni e nuove interrogazioni al modello. Tuttavia, tale modalità non è risultata sufficiente a risolvere autonomamente il problema del riconoscimento degli identificativi tecnici dei dispositivi, confermando la necessità del supporto semantico fornito dal layer Python.

In particolare, anche quando il modello riceveva l'osservazione prodotta da `list_devices`, non riusciva sempre a trasformare tale informazione in una nuova chiamata corretta. In alcuni casi tendeva a produrre una descrizione generale della struttura del sistema, fermando il ciclo prima dell'esecuzione della lettura richiesta. In altri casi proseguiva nelle iterazioni successive, ma selezionava identificativi non validi o non coerenti con la struttura effettiva del simulatore.

Successivamente è stato testato l'input "leggi **supply_air_flowrate_setpoint** del device vav_room_7", avente device_id esplicitato correttamente. In questo caso il modello ha generato correttamente la chiamata **read_point** e ha utilizzato il risultato restituito dal

tool per produrre, nell'iterazione successiva, una risposta testuale più leggibile per l'utente (figura 4.18). Analogamente, è stato testato l'input "imposta **supply_air_percetnage_command** di vav_room_8 a 0.6", allo scopo di verificare il comportamento del modello in caso di una richiesta di scrittura contenente tutti gli argomenti necessari. Anche in questo caso, il modello ha interpretato correttamente l'input, individuando l'attuatore corretto e applicando la sovrascrittura richiesta con il valore esplicitato (figura 4.19).



Figura 4.18 Il modello utilizza il tool "read_point" per rispondere correttamente alla richiesta di lettura dell'utente

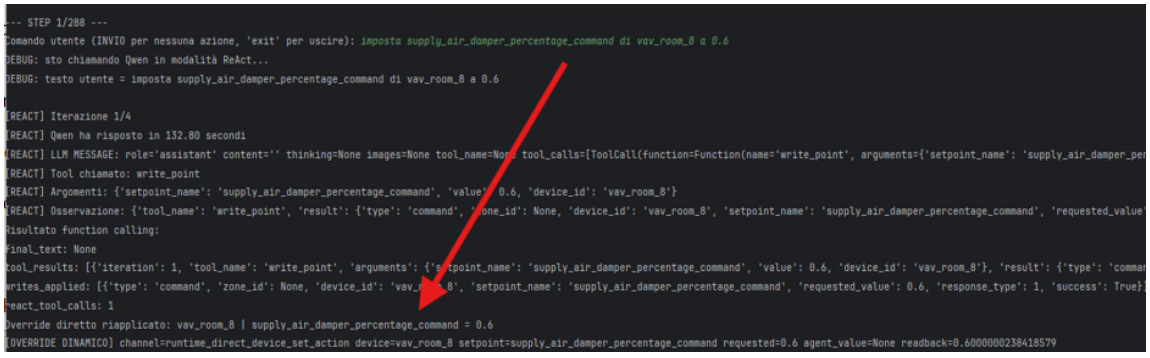


Figura 4.19 Il modello utilizza il tool "write_point" per rispondere correttamente alla richiesta di scrittura dell'utente

Questo risultato suggerisce che la modalità ReAct-like possa quindi essere considerata una soluzione utile per rendere l'interazione con l'utente più flessibile e per permettere al modello di trasformare un'osservazione tecnica restituita dal simulatore in una risposta finale maggiormente naturale e comprensibile.

Nel complesso, tuttavia, l'esperimento evidenzia il fatto che l'interazione iterativa, da sola, non riesce a garantire la corretta interpretazione della struttura tecnica dell'ambiente. Anche fornendo al modello la possibilità di osservare la lista dei dispositivi e di correggere le chiamate precedenti, rimane necessaria una forma di supporto esterno per il mapping tra riferimenti linguistici e identificativi tecnici. Questo conferma il ruolo centrale del layer Python non solo come componente esecutivo, ma anche come meccanismo di validazione e risoluzione semantica.

4.6. Quinta fase: esperimento ReAct testuale

4.6.1. Configurazione dell'esperimento

A seguito dei risultati ottenuti con l'implementazione ReAct-like descritta nella fase precedente, è stata introdotta un'ulteriore configurazione sperimentale finalizzata a verificare il comportamento del modello in una modalità più aderente al paradigma ReAct originale. Tale configurazione, pur non corrispondendo alla modalità operativa suggerita per Qwen2.5 a causa della sua difficoltà nel parsing degli argomenti, è stata testata come esperimento di confronto, con l'obiettivo di valutare se l'in-context learning e l'inserimento di esempi contestuali nel formato richiesto dal paradigma ReAct potessero favorire una migliore generalizzazione del comportamento atteso e un riconoscimento più affidabile degli identificativi tecnici necessari alla chiamata dei tool.

Questa fase assume quindi un ruolo di confronto rispetto all'implementazione ReAct-like precedente. Mentre quest'ultima era basata su un ciclo iterativo esterno, controllato dal codice Python, e sull'utilizzo del function calling strutturato, la configurazione testuale richiede al modello di produrre esplicitamente una sequenza nel formato richiesto dal paradigma ReAct, generalizzando gli esempi contestuali forniti nel few-shot prompting.

L'esperimento è stato configurato introducendo un prompt di quattro esempi few-shot nel formato ReAct. In particolare, sono stati inseriti due esempi relativi a operazioni di lettura (figura 4.20) e due esempi relativi a operazioni di scrittura (figura 4.21). L'esempio 1 costituisce il primo esempio di lettura e riguarda una richiesta formulata tramite riferimento descrittivo al boiler. In questo caso, il modello deve prima utilizzare

list_devices per individuare l'identificativo tecnico del dispositivo e successivamente richiamare read_point sul parametro richiesto. Questo esempio serve quindi a rappresentare il caso in cui la richiesta dell'utente non contiene direttamente l'identificativo del dispositivo, ma richiede un passaggio preliminare di esplorazione della struttura. Il secondo esempio di lettura (esempio 2) riguarda invece una richiesta riferita a una stanza numerata, nella quale il modello deve associare la stanza al corrispondente dispositivo VAV e leggere il sensore richiesto. Questo esempio è stato inserito per rappresentare i casi in cui il riferimento dell'utente è più vicino alla struttura ricorrente del simulatore.

```
ESEMPIO 1
Utente: dimmi supply_water_setpoint del boiler
Thought: La richiesta è una lettura, ma il dispositivo è indicato con un riferimento descrittivo. Devo prima osservare la struttura del sistema per individuare il device_id tecnico corretto.
Action: list_devices({})
Observation: {"devices": {"boiler_id_ESEMPIO": {"measurements": ["supply_water_setpoint", "supply_water_temperature_sensor"], "setpoints": ["supply_water_setpoint"]}}}
Thought: Ora posso usare il device_id tecnico osservato e leggere il valore richiesto.
Action: read_point({"device_id": "boiler_id_ESEMPIO", "measurement_name": "supply_water_setpoint"})
Observation: {"type": "query", "device_id": "boiler_id_ESEMPIO", "measurement_name": "supply_water_setpoint", "value": 320.0}
Thought: Ho ottenuto il valore richiesto.
Final Answer: Il valore di supply_water_setpoint del boiler è 320.0.

ESEMPIO 2
Utente: qual è la temperatura della stanza 3?
Thought: La richiesta è una lettura riferita a una stanza numerata. Per le zone numerate posso ricavare direttamente il device tecnico VAV corrispondente e leggere il sensore di temperatura.
Action: read_point({"device_id": "vav_room_3", "measurement_name": "zone_air_temperature_sensor"})
Observation: {"type": "query", "device_id": "vav_room_3", "measurement_name": "zone_air_temperature_sensor", "value": 295.15}
Thought: Ho ottenuto il valore richiesto.
Final Answer: La temperatura della stanza 3 è 295.15 K.
```

Figura 4.20 Esempi few-shot di lettura per l'in-context learning del modello

Analogamente, per le operazioni di scrittura, l'esempio 3 riguarda un comando esplicito su un dispositivo VAV, nel quale sono già presenti il device_id tecnico, il setpoint da modificare e il valore numerico da impostare. Tale esempio serve a mostrare il comportamento atteso quando la richiesta contiene già tutti gli argomenti necessari alla chiamata di write_point. Il secondo esempio di scrittura (esempio 4) riguarda invece una richiesta descrittiva sul boiler, nella quale il modello deve prima utilizzare list_devices per individuare l'identificativo corretto del dispositivo, per poi generare una chiamata write_point utilizzando l'attuatore e il valore richiesti. Questo esempio completa il caso precedente, mostrando il comportamento atteso quando anche un comando di modifica richiede una fase preliminare di esplorazione rispetto alla struttura del simulatore.

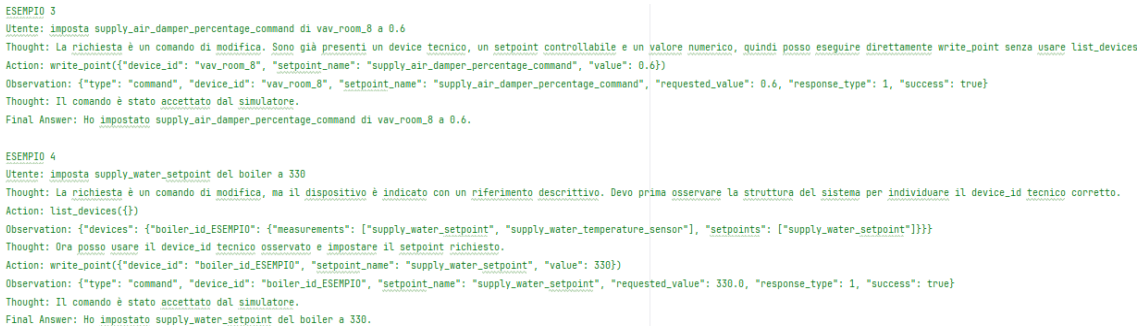


Figura 4.21 Esempi few-shot di scrittura per l'in-context learning del modello

La scelta di utilizzare quattro esempi deriva dalla volontà di coprire due dimensioni principali: da un lato la distinzione tra lettura e scrittura, dall'altro la distinzione tra richieste con identificativi tecnici già espliciti e richieste con riferimenti descrittivi o semantici da risolvere. In questo modo, il prompt fornisce al modello esempi rappresentativi del comportamento atteso, mantenendo però un'impostazione "few-shot", quindi senza includere un numero eccessivo di casi specifici.

Gli esempi mostrano al modello la struttura attesa dell'interazione, articolata in quattro componenti principali:

- **Thought**, in cui il modello esplicita il ragionamento necessario per interpretare la richiesta;
- **Action**, in cui viene indicato il tool da utilizzare e i relativi argomenti;
- **Observation**, in cui viene riportato il risultato restituito dal tool;
- **Final Answer**, in cui il modello produce la risposta finale all'utente.

Anche in questa configurazione è stato disattivato il resolver semantico, in modo da evitare che il layer Python correggesse automaticamente i riferimenti descrittivi indicati dall'utente verso gli identificativi tecnici effettivi del simulatore.

4.6.2. Risultati ottenuti

Mentre la versione ReAct-like, purché la richiesta dell'utente fosse sufficientemente esplicita, è risultata operativamente funzionante sia in casi di lettura (tramite read_point) che in casi di modifica (tramite write_point), la configurazione testuale del ReAct, seppur supportato dal few-shot prompting, ha prodotto risultati più deboli.

In particolare, nei casi in cui il modello doveva ricavare l'identificativo tecnico del boiler a partire da un riferimento descrittivo, l'interazione non è arrivata stabilmente alla chiamata operativa corretta. Il modello tendeva a utilizzare list_devices oppure a ricevere l'osservazione contenente la struttura del simulatore, ma non riusciva poi a trasformare tale osservazione in una successiva chiamata read_point o write_point corretta.

Un comportamento analogo è stato osservato anche nelle richieste di lettura e scrittura, anche quando gli argomenti della richiesta erano ben esplicitati. Pur avendo a disposizione esempi few-shot che mostravano il formato ReAct e l'utilizzo di read_point e write_point, il modello non è riuscito a generalizzare stabilmente il comportamento atteso su richieste diverse da quelle mostrate negli esempi.

L'unico risultato positivo osservato in questa configurazione è stato ottenuto quando la richiesta di test era sostanzialmente coincidente con uno degli esempi inseriti nel prompt few-shot. In quel caso il modello è riuscito a riprodurre il meccanismo ReAct, generando una sequenza coerente di ragionamento, azione, osservazione e risposta finale.

Questo risultato suggerisce che il modello recepisca almeno in parte il formato fornito tramite in-context learning, ma non riesca necessariamente a generalizzarlo quando deve applicarlo a nuovi sensori, nuovi attuatori o riferimenti descrittivi non direttamente coincidenti con gli esempi. In particolare, sembra emergere una difficoltà nell'utilizzare l'osservazione restituita da list_devices come base per costruire una nuova azione operativa. Per un'analisi più dettagliata dei risultati ottenuti per le singole richieste si rimanda al paragrafo successivo.

4.6.3. Analisi dell'implementazione ReAct testuale few-shot

Al fine di valutare la capacità di ragionamento del modello guidato tramite few-shot prompting e di chiarire ulteriormente il ruolo del supporto applicativo nella costruzione di un sistema di interazione tra LLM e ambiente simulato, sono state testate tre richieste relative a tre tipologie differenti di operazioni:

- Richiesta di lettura tramite read_point, con device_id e measurement_name già espliciti ("dimmi **supply_air_flowrate_setpoint** di **vav_room_7**");

- Richiesta di scrittura tramite write_point, con device_id, setpoint_name e valore già espliciti (“imposta **supply_air_percetnage_command** di **vav_room_8** a **0.6**”);
- Richiesta contenente un riferimento descrittivo, che richiede prima l’utilizzo di **list_devices** e poi la generazione di una successiva azione operativa (“dimmi **outside_air_flowrate_sensor** della centrale aria”).

In tutti e tre i casi il comportamento osservato è sostanzialmente analogo: il modello ha avviato correttamente il ciclo ReAct generando una prima Action verso list_devices, ma dopo aver ricevuto l’Observation contenente la struttura del simulatore non ha prodotto la successiva chiamata operativa read_point o write_point. Al contrario, ha generato una risposta descrittiva della struttura osservata, concentrandosi in particolare sui dispositivi VAV e sui relativi campi disponibili. In figura 4.22 viene illustrata come esempio esplicativo la risposta ottenuta al comando di lettura.

```
--- STEP 1/288 ---
Comando utente (INVIO per nessuna azione, 'exit' per uscire): dimmi supply_air_flowrate_setpoint di vav_room_7
DEBUG: sto chiamando Qwen in modalità ReAct testuale canonica...
DEBUG: testo utente = dimmi supply_air_flowrate_setpoint di vav_room_7

[REACT-TEXT] Iterazione 1/4
[REACT-TEXT] Qwen ha risposto in 397.78 secondi
[REACT-TEXT] LLM CONTENT:
Thought: La richiesta è una lettura del setpoint di flusso d'aria di un dispositivo VAV specifico. Devo prima verificare se il device_id tecnico per vav_room_7 è noto.
Action: list_devices({})
[REACT-TEXT] Tool chiamato: list_devices
[REACT-TEXT] Argomenti: {}
[REACT-TEXT] Observation: {'zones': {'zone_id_1': ['vav_room_1'], 'zone_id_2': ['vav_room_2'], 'zone_id_3': ['vav_room_3'], 'zone_id_4': ['vav_room_4'], 'zone_id_5': ['vav_room_5'], 'zone

[REACT-TEXT] Iterazione 2/4
[REACT-TEXT] Qwen ha risposto in 948.37 secondi
[REACT-TEXT] LLM CONTENT:
It looks like you have a JSON-like structure representing various VAV (Variable Air Volume) rooms, each with specific setpoints and measurements. Here's a breakdown of the key components:

1. **Device Type**: All devices are of type `4`, which could represent a specific kind of HVAC system or control mechanism.
2. **Zone ID**: Each room has a unique zone identifier (`zone_id`), ranging from `zone_id_1` to `zone_id_126`.
3. **Setpoints and Measurements**:
   - `supply_air_damper_percentage_command`: This likely represents the command sent to control the damper opening percentage in the supply air duct.
   - `supply_air_flowrate_setpoint`: This is probably a setpoint for the flow rate of air supplied to the room.
   - `zone_air_temperature_sensor`: This measures the temperature within the zone.

### Example Analysis

Let's take an example from your data:

```json
{
  "vav_room_1": {
    "device_type": 4,
    "zone_id": "zone_id_1",
    "setpoints": {
      "supply_air_damper_percentage_command": 50.0,
      "supply_air_flowrate_setpoint": 200
    },
    "measurements": {
      "zone_air_temperature_sensor": 22.5
    }
  }
}
```

Figura 4.22 Estratto della risposta ottenuta per la richiesta “dimmi supply_air_flowrate_setpoint di vav_room_7”
```

Lo stesso user prompt è stato successivamente inserito all'interno degli esempi contestuali forniti all'interno del few-shot prompting, per verificare se il modello fosse in grado di recepire il formato ReAct quando la richiesta da gestire era già rappresentata in modo esplicito nel contesto fornito. In questo caso, il modello riesce a riprodurre correttamente la sequenza ReAct e a completare il task (figura 4.23). Questo aspetto indica che l'in-context learning è effettivamente recepito dal modello almeno a livello locale. Tuttavia, quando la richiesta richiede di applicare lo stesso schema a un dispositivo, un sensore o un attuatore diverso, il comportamento diventa instabile.

```
--- STEP 2/288 ---
Comando utente (INVIO per nessuna azione, 'exit' per uscire): dimmi supply_air_flowrate_setpoint di vav_room_7
DEBUG: sto chiamando Qwen in modalità ReAct testuale canonica...
DEBUG: testo utente = dimmi supply_air_flowrate_setpoint di vav_room_7

[REACT-TEXT] Iterazione 1/4
[REACT-TEXT] Qwen ha risposto in 198.77 secondi
[REACT-TEXT] LLM CONTENT:
Thought: L'utente chiede un setpoint del dispositivo VAV corrispondente alla stanza 7. Ho già il device_id tecnico (vav_room_7)
Action: read_point({"device_id": "vav_room_7", "measurement_name": "supply_air_flowrate_setpoint"})
[REACT-TEXT] Tool chiamato: read_point
[REACT-TEXT] Argomenti: {'device_id': 'vav_room_7', 'measurement_name': 'supply_air_flowrate_setpoint'}
[REACT-TEXT] Observation: {'type': 'query', 'zone_id': None, 'device_id': 'vav_room_7', 'measurement_name': 'supply_air_flowrate

[REACT-TEXT] Iterazione 2/4
[REACT-TEXT] Qwen ha risposto in 15.36 secondi
[REACT-TEXT] LLM CONTENT:
Thought: Ho ottenuto il valore richiesto.
Final Answer: Il setpoint di supply_air_flowrate_setpoint per vav_room_7 è 0.035.
```

Figura 4.23 Estratto della risposta ottenuta dopo aver inserito la richiesta come esempio nel few-shot prompting. Il sistema restituisce all'utente la risposta corretta nella final answer.

Il risultato osservato è particolarmente rilevante perché l'obiettivo dell'esperimento non era verificare se il modello potesse memorizzare o imitare un singolo esempio, ma se potesse utilizzare il formato ReAct per costruire un percorso operativo generalizzabile. I risultati suggeriscono che, nella configurazione testata, tale generalizzazione non avviene in modo affidabile.

La criticità più evidente riguarda l'utilizzo di **list_devices**. In linea teorica, il modello dovrebbe usare questo tool per osservare la struttura del simulatore, individuare il device_id tecnico corretto e poi generare una nuova azione **read_point** o **write_point**. In pratica, il modello tende invece a fermarsi a una descrizione della struttura osservata, senza completare il passaggio operativo successivo. Questo comportamento era già

emerso, in forma diversa, negli esperimenti sulla costruzione incrementale del contesto: l'informazione strutturale viene trattata dal modello come contenuto descrittivo, ma non viene trasformata stabilmente in una rappresentazione direttamente utilizzabile per l'esecuzione.

Il confronto con la versione ReAct-like è quindi significativo. La versione ReAct-like risulta più efficace dal punto di vista operativo, ma anche più guidata: il modello utilizza il function calling strutturato e il layer Python può reinserire osservazioni o messaggi correttivi quando il task non viene completato. La versione canonica, invece, riduce questi aiuti e mostra più chiaramente il limite del modello. Tale limite non riguarda soltanto la scelta del tool, ma soprattutto la capacità di mantenere il task originale dopo una osservazione ampia e di estrarre da essa gli argomenti tecnici necessari.

Una possibile spiegazione, da considerare in forma prudente, riguarda la natura dell'osservazione restituita da list_devices. Tale osservazione contiene infatti l'intera struttura del simulatore, inclusi tutti i dispositivi VAV, i relativi sensori e attuatori disponibili e i dispositivi globali associati alla zona di default. Questa rappresentazione, pur completa, risulta molto ampia e ripetitiva. È quindi possibile che il modello interpreti tale output soprattutto come una descrizione generale della struttura del sistema, senza riuscire a estrarre in modo affidabile i dettagli tecnici necessari per generare la successiva Action, forse a causa dell'ampiezza dell'osservazione o della sua complessità.

Nel complesso, l'esperimento fornisce quindi un risultato negativo ma informativo. La configurazione ReAct testuale mostra che l'introduzione del formato "Thought - Action - Observation - Final Answer", seppure supportata da alcuni esempi contestuali, non è sufficiente, nel caso testato, a rendere il modello autonomo nella risoluzione dei riferimenti tecnici del simulatore.

Capitolo Quinto

Analisi complessiva dei risultati

Il presente capitolo analizza complessivamente i risultati ottenuti nelle diverse fasi sperimentali descritte nel Capitolo 4, con l'obiettivo di valutare in che misura i risultati ottenuti rispondano agli obiettivi del lavoro. L'analisi considera tre aspetti principali: la capacità del modello di selezionare il tool corretto, la capacità di produrre argomenti coerenti con la struttura tecnica di SBSim e il ruolo svolto dal prompt e dal layer Python nel supportare il mapping tra linguaggio naturale e identificativi tecnici.

L'insieme degli esperimenti condotti consente di formulare alcune considerazioni complessive sul ruolo degli LLM nell'interazione con un ambiente HVAC simulato. I risultati non indicano una progressione lineare in cui l'aumento del dettaglio del prompt o l'introduzione di strategie più complesse producono automaticamente prestazioni migliori. Al contrario, emerge un quadro più articolato, nel quale la correttezza dell'interazione dipende dall'equilibrio tra informazioni fornite al modello, struttura dei tool e supporto applicativo esterno.

La prima fase sperimentale mostra che, su un dataset ristretto e composto da richieste relativamente semplici, il livello di dettaglio del prompt non produce differenze marcate. Tutte le configurazioni testate riescono a gestire una parte significativa delle richieste, ma presentano comunque errori nella selezione dei parametri, nella conversione di istruzioni formulate in linguaggio naturale e nella scelta del tool corretto. Da questa fase emerge quindi che un prompt più lungo o più dettagliato non garantisce, da solo, un miglioramento sostanziale.

La seconda fase evidenzia invece in modo più netto l'importanza della struttura del sistema e del resolver semantico. Quando le richieste diventano più numerose e più varie, la presenza di informazioni strutturali sui dispositivi, sui sensori e sugli attuatori diventa determinante. In particolare, il modello fatica a ricondurre riferimenti descrittivi come "boiler", "caldaia", "centrale aria" o "stanza" agli identificativi tecnici effettivi del simulatore. Il resolver svolge quindi un ruolo centrale nel rendere eseguibili richieste che il modello, da solo, non riesce a tradurre in modo stabile.

La terza fase, dedicata alla costruzione incrementale del contesto, mostra un ulteriore limite. Il modello è in grado di leggere e sintetizzare la struttura restituita da list_devices,

ma tende a produrre una rappresentazione descrittiva, non una conoscenza operativa affidabile. L'informazione ottenuta dal simulatore non viene trasformata stabilmente in una guida concreta per successive chiamate ai tool. Questo risultato suggerisce che la semplice disponibilità di informazioni strutturali non è sufficiente: tali informazioni devono essere rappresentate in modo vincolato e direttamente utilizzabile.

La quarta fase, basata su un'implementazione ReAct-like, mostra che l'interazione iterativa può migliorare la gestione del ciclo modello-tool-osservazione. Il modello riesce a utilizzare i tool in più passaggi e a produrre risposte più coerenti quando il task è sufficientemente esplicito. Tuttavia, questa configurazione rimane fortemente dipendente dal supporto del layer Python e dal function calling strutturato. Il miglioramento non deriva quindi solo dal paradigma ReAct, ma dall'integrazione tra modello, tool, osservazioni e logica applicativa.

La quinta fase, dedicata al ReAct canonico, conferma questo aspetto da una prospettiva opposta. Riducendo il supporto applicativo e affidando maggiormente al modello la produzione testuale di Thought, Action, Observation e Final Answer, le prestazioni peggiorano. Il modello riesce a riprodurre il meccanismo quando la richiesta è molto vicina a un esempio few-shot, ma non generalizza in modo affidabile su richieste diverse. Questo risultato suggerisce che il formato ReAct, da solo, non è sufficiente a risolvere il problema del grounding operativo in un ambiente tecnico complesso.

Nel complesso, il principale insegnamento emerso dalla sperimentazione è che l'LLM non può essere considerato, in questa configurazione, un agente autonomo capace di controllare direttamente il simulatore. Il modello funziona meglio come componente linguistica inserita in un'architettura più ampia, nella quale il layer applicativo svolge funzioni essenziali di mediazione, validazione e supporto semantico. L'interazione corretta con SBSim richiede infatti non solo la comprensione della richiesta utente, ma anche la capacità di produrre identificativi tecnici, nomi di measurement e setpoint coerenti con la struttura effettiva dell'ambiente.

Pertanto, il contributo principale del lavoro non consiste nella realizzazione di un sistema completamente autonomo, ma nell'analisi sperimentale dei fattori che rendono possibile o fragile l'interazione tra un LLM locale e un simulatore HVAC. I risultati mostrano che prompt engineering, function calling, costruzione autonoma del contesto e ReAct sono

strumenti utili, ma non sufficienti se non accompagnati da una rappresentazione strutturata del dominio e da un layer applicativo robusto.

Capitolo Sesto

Conclusioni e sviluppi futuri

6.1. Conclusioni

La presente tesi ha analizzato l'utilizzo di un Large Language Model come interfaccia in linguaggio naturale per l'interazione con un ambiente HVAC simulato. Il lavoro si è concentrato sull'integrazione tra Smart Building Simulator, il modello Qwen2.5:7B eseguito localmente tramite Ollama e un layer Python incaricato di collegare le richieste dell'utente ai tool disponibili per leggere sensori, modificare setpoint ed esplorare la struttura del sistema.

L'obiettivo principale era valutare in modo sperimentale capacità e limiti del modello nell'interazione con un ambiente tecnico strutturato. In particolare, sono stati analizzati tre aspetti: la capacità del modello di selezionare il tool corretto, la capacità di produrre argomenti coerenti con la struttura di SBSim e i ruoli del prompt e del layer applicativo nel supportare il mapping tra linguaggio naturale e identificativi tecnici.

I risultati ottenuti mostrano che l'LLM è in grado di gestire alcune richieste semplici, soprattutto quando il device_id, il measurement_name o il setpoint_name sono già espliciti nella richiesta dell'utente. In questi casi il modello può generare correttamente una chiamata ai tool e restituire una risposta comprensibile. Tuttavia, le prestazioni peggiorano quando la richiesta contiene riferimenti descrittivi o ambigui, che richiedono di ricondurre concetti naturali come "boiler", "centrale aria", "stanza" o "serranda" agli identificativi tecnici del simulatore.

Una delle evidenze principali riguarda quindi il ruolo del layer Python. Le funzioni di validazione, normalizzazione e risoluzione semantica si sono dimostrate necessarie per evitare che l'output del modello venga applicato direttamente al simulatore in modo non coerente. Il layer applicativo non svolge soltanto una funzione esecutiva, ma rappresenta un componente di mediazione indispensabile tra linguaggio naturale e struttura tecnica del sistema.

Gli esperimenti sulla costruzione incrementale del contesto e sul paradigma ReAct hanno confermato questa conclusione. Il modello è in grado di ricevere informazioni sulla struttura del sistema, ma tende a trattarle come contenuto descrittivo più che come

conoscenza operativa. Anche il formato ReAct, pur utile per organizzare l'interazione in più passaggi, non è risultato sufficiente a sostituire il supporto applicativo esterno. In particolare, la versione ReAct canonica ha mostrato difficoltà nel generalizzare gli esempi few-shot e nel trasformare l'osservazione restituita da list_devices in una successiva azione read_point o write_point corretta.

Questi risultati indicano che, nella configurazione sperimentata, un LLM locale può essere impiegato efficacemente come interfaccia linguistica verso un sistema IoT o cyber-fisico, ma non può essere considerato autonomamente affidabile per il controllo diretto dell'ambiente. L'affidabilità deriva dall'integrazione tra modello, tool e layer applicativo, più che dal solo modello linguistico.

6.2. Sviluppi futuri

A partire dai limiti osservati durante la sperimentazione, è possibile individuare alcune direzioni di sviluppo futuro.

Dal punto di vista degli sviluppi futuri, una prima direzione riguarda la rappresentazione della struttura del sistema. L'esperimento con list_devices ha mostrato che restituire al modello l'intera struttura del simulatore può generare osservazioni molto ampie e difficili da utilizzare. Sarebbe quindi utile progettare rappresentazioni più compatte e gerarchiche, ad esempio distinguendo tra dispositivi globali, come boiler e AHU, e dispositivi ripetitivi, come i VAV associati alle singole zone. Una rappresentazione più operativa potrebbe facilitare il grounding tra richiesta utente e tool call successiva.

Una seconda direzione riguarda l'introduzione di tecniche di retrieval o di ricerca mirata all'interno della struttura del simulatore. Invece di fornire al modello l'intera lista dei device, il sistema potrebbe recuperare soltanto i dispositivi e i campi potenzialmente rilevanti per la richiesta corrente. Questo permetterebbe di ridurre il carico informativo sul modello e di migliorare la precisione nella selezione di device_id, measurement_name e setpoint_name.

Una terza possibilità consiste nel confrontare modelli diversi. In questa tesi è stato utilizzato Qwen2.5:7B in esecuzione locale, ma sarebbe interessante ripetere gli stessi esperimenti con modelli di dimensione maggiore, modelli specializzati nel tool use o

modelli commerciali, al fine di valutare quanto i limiti osservati dipendano dal modello specifico e quanto invece dalla natura del task.

Un ulteriore sviluppo riguarda la valutazione delle modifiche di controllo sull'andamento del simulatore. Il lavoro ha verificato la possibilità di leggere sensori e impostare setpoint, ma non ha approfondito in modo sistematico l'effetto delle azioni generate su comfort, consumi, costi energetici ed emissioni. Una prosecuzione naturale consisterebbe quindi nell'integrare una valutazione più ampia delle conseguenze delle azioni prodotte dal sistema.

Infine, sarebbe possibile estendere l'architettura verso scenari più vicini a un sistema multi-agente o a un assistente operativo per smart building. In tale prospettiva, l'LLM potrebbe essere affiancato da moduli specializzati per la pianificazione, la verifica dei vincoli, la gestione della memoria e la validazione delle azioni prima della loro applicazione all'ambiente. Questo consentirebbe di superare l'approccio sperimentale adottato nella tesi e di avvicinarsi a una soluzione più robusta e scalabile.

Questa tesi sperimentale è stata svolta con l'obiettivo di analizzare capacità e limiti di un Large Language Model come interfaccia in linguaggio naturale per il controllo e l'interrogazione di sistemi IoT, assumendo come caso di studio un ambiente HVAC simulato. Il lavoro si inserisce quindi nel più ampio ambito dell'utilizzo degli LLM e degli agenti AI come strumenti di interazione con sistemi cyber-fisici complessi, evidenziando come tali modelli possano rappresentare un supporto promettente per rendere più naturale l'accesso a sistemi tecnici strutturati, seppur richiedendo al tempo stesso architetture di integrazione robuste, capaci di collegare in modo affidabile linguaggio naturale, tool software e struttura effettiva dell'ambiente controllato.

Bibliografia

[1] François Michelon, Yihong Zhou, Thomas Morstyn, *"Large Language Model Interface for Home Energy Management Systems"*, Proc. 16th ACM Int. Conf. on Future and Sustainable Energy Systems, (2025)

[2] Evan King, Haoxiang Yu, Sangsu Lee, Christine Julien, *"Sasha: Creative Goal-Oriented Reasoning in Smart Homes with Large Language Models"*, Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies, (2024)

[3] Chien-fei Chen, Xiaojing Xu, Jacqueline Adams, James Brannon, Fangxing Li, Allen Walzem, *"When East meets West: Understanding residents' home energy management system adoption intention and willingness to pay in Japan and the United States"*, Energy Research & Social Science, (2020)

[4] *"Smart Control Project Documentation"*, documentazione tecnica online, https://gitwyd.github.io/sbsim_documentation/#smart-control-project-documentation

[5] An Yang et al., *"Qwen Technical Report"*, arXiv preprint arXiv:2309.16609, (2023)

[6] J. Goldfeder, J. Sipple, *"A Lightweight Calibrated Simulation Enabling Efficient Offline Learning for Optimal Control of Real Buildings"*, BuildSys '23, (2023)

[7] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, L. Zettlemoyer, N. Cancedda, T. Scialom, *"Toolformer: Language Models Can Teach Themselves to Use Tools"*, NIPS '23: Proceedings of the 37th International Conference on Neural Information Processing Systems, (2023)

[8] S.S. Chowa, R. Alvi, S.S. Rahman et al., *"From language to action: a review of large language models as autonomous agents and tool users."* Artif Intell Rev 59, 71, (2026)

[9] A. Vaswani et al., "*Attention Is All You Need*", Proceedings of the 31st International Conference on Neural Information Processing Systems, (2017)

[10] T. B. Brown et al., "*Language Models Are Few-Shot Learners*", Proceedings of the 34th International Conference on Neural Information Processing Systems, (2020)

[11] A. Yang et al., "*Qwen2.5 Technical Report*", arXiv preprint arXiv:2412.15115, (2024)

[12] Stan Franklin, Arthur Graesser, "*Is it an Agent, or Just a Program?: A Taxonomy for Autonomous Agents*", (1996)

[13] L. Wang et al., "*A Survey on Large Language Model Based Autonomous Agents*", Front. Comput. Sci. 18, 186345, (2024)

[14] Y. Zheng, H. Wei, J. Lu et al., "*Investigating effective LLM-based in-context tool use: what matters and how to improve*", Front. Comput. Sci. 20, 2007323, (2026)

[15] "*Function Calling*", documentazione tecnica online, https://qwen.readthedocs.io/en/v2.0/framework/function_call.html

[16] S. Radzymiński, *"How does Playwright MCP work?"*, articolo online, https://www.awesome-testing.com/2025/07/playwright-mcp

[17] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, Y. Cao, "*ReAct: Synergizing Reasoning and Acting in Language Models*", arXiv preprint arXiv:2210.03629, (2023)

[18] "*ReAct Prompting*", documentazione tecnica online, https://www.promptingguide.ai/techniques/react

[19] "*Ollama GitHub Repository – README*", repository GitHub, https://github.com/ollama/ollama/blob/main/README.md

[20] "*API Reference Introduction*", documentazione tecnica online, https://docs.ollama.com/api

[21] "*Tool support*", documentazione online, https://ollama.com/blog/tool-support

[22] D. Rivkin, F. Hogan, A. Feriani, A. Konar, A. Sigal, S. Liu, G. Dudek, "*SAGE: Smart Home Agent with Grounded Execution*", arXiv preprint arXiv:2311.00772, (2023)

[23] Yu Chaerin et al., "*Leveraging llms for efficient and personalized smart home automation.*", arXiv preprint arXiv:2601.04680, (2026)

[24] B. R. Barricelli, D. Fogli, S. Gallo, D. Guizzardi, S. Maenza, A. Mattioli, F. Paternò, "*A smart home platform integrating an energy-aware digital twin and an LLM-enabled conversational agent*", Behaviour & Information Technology, 1–24, (2026)

[25] S. Li, Y. Guo, J. Yao, Z. Liu, H. Wang, "*HomeBench: Evaluating LLMs in Smart Homes with Valid and Invalid Instructions Across Single and Multiple Devices*", ACL Anthology, (2025)

[26] J. Liang et al., "*Code as policies: Language model programs for embodied control.*", IEEE International conference on robotics and automation (ICRA), (2023)

[27] A. Brohan et al., "*RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control*", arXiv preprint arXiv:2307.15818, (2023)

[28] L. Zhang, Z. Chen**,** "*Large language model-based interpretable machine learning control in building energy systems*", Energy and Buildings 313, 114278, (2024)

[29] Y. Xia et al., "*Control industrial automation system with large language model agents.*", 30th International Conference on Emerging Technologies and Factory Automation (ETFA), (2025)

[30] "*llm_to_sbsim*", repository GitHub, https://github.com/suraci00/llm_to_sbsim

[31] "*Qwen 2.5 Best Practices for Tool Calling*", repository GitHub, https://github.com/allierays/ollama-wellness/blob/master/docs/08_QWEN_BEST_PRACTICES.md

Appendici

Appendice A – Stralci di codice del layer applicativo

Questa appendice riporta alcuni estratti del codice Python sviluppato per collegare il modello linguistico Qwen al simulatore SBSim, con l'obiettivo di mostrare le parti più rilevanti per comprendere il funzionamento del layer applicativo descritto nel Capitolo 3.

A.1 Definizione dei tool disponibili

Il primo estratto mostra la definizione della variabile **TOOLS** (figura A.1), utilizzata per descrivere al modello le funzioni disponibili. Per ciascun tool vengono specificati il nome della funzione, una descrizione e lo schema dei parametri accettati.

```
TOOLS = [
    {
        "type": "function",
        "function": {
            "name": "list_devices",
            "description": "Restituisce zone, device, campi osservabili e campi controllabili del simulatore.",
            "parameters": {
                "type": "object",
                "properties": {}
            }
        }
    },
    {
        "type": "function",
        "function": {
            "name": "read_point",
            "description": "Legge un measurement o un setpoint leggibile da un device.",
            "parameters": {
                "type": "object",
                "properties": {
                    "device_id": {"type": "string"},
                    "measurement_name": {"type": "string"}
                },
                "required": ["device_id", "measurement_name"]
            }
        }
    },
    {
        "type": "function",
        "function": {
            "name": "write_point",
            "description": "Imposta un setpoint controllabile su un device.",
            "parameters": {
                "type": "object",
                "properties": {
                    "device_id": {"type": "string"},
                    "setpoint_name": {"type": "string"},
                    "value": {"type": "number"}
                },
                "required": ["device_id", "setpoint_name", "value"]
            }
        }
    }
]
```

Figura A.1 Definizione dei tool disponibili

A.2 Gestione delle tool call generate dal modello

La funzione **esegui_tools** (figura A.2) riceve il nome del tool e gli argomenti generati dal modello tramite function calling. Il layer applicativo normalizza gli argomenti, risolve eventuali riferimenti ai dispositivi e costruisce l'oggetto **Intent** da inoltrare al simulatore.

```python
def esegui_tools(building, devices, zone_map, tool_name: str, args: dict) -> dict:  2 usages new *
    #esegue le funzioni del tool richiamato dal function calling
    args = clean_tool_args(args)

    if tool_name == "list_devices":
        return {
            "zones": zone_map,
            "devices": devices,
        }

    point_name = args.get("measurement_name") or args.get("setpoint_name")
    device_id = resolve_device_from_context(args.get("device_id"), point_name, devices)

    if tool_name == "read_point":
        intent = Intent(
            type="query",
            device_id=device_id,
            measurement_name=point_name,
        )
        result = esegui_intent(building, intent, devices, zone_map)
        if isinstance(result, dict) and "error" not in result:
            result["success"] = True
        else:
            result["success"] = False
        return result

    if tool_name == "write_point":
        intent = Intent(
            type="command",
            device_id=device_id,
            setpoint_name=point_name,
            value=args.get("value"),
        )
        result = esegui_intent(building, intent, devices, zone_map)
        if isinstance(result, dict) and "error" not in result:
            result["success"] = result.get("response_type") == 1
        return result

    return {"error": f"Tool sconosciuto: {tool_name}"}
```

Figura A.2 Estratto della funzione esegui_tools

A.3 Normalizzazione degli argomenti

La funzione **clean_tool_args** (figura A.3) gestisce alcuni casi ricorrenti negli argomenti prodotti dal modello, quali campi vuoti o valori numerici restituiti in forma testuale. Questo passaggio contribuisce a rendere più robusta l'esecuzione delle chiamate.

```python
def clean_tool_args(args: dict) -> dict:  1 usage  new *
    cleaned = {}
    for k, v in (args or {}).items():
        if v == "":
            cleaned[k] = None
        elif k == "value" and isinstance(v, str):
            try:
                cleaned[k] = float(v)
            except ValueError:
                cleaned[k] = v
        else:
            cleaned[k] = v
    return cleaned
```

Figura A.3 Estratto della funzione clean_tool_args

A.4 Risoluzione dei riferimenti ai dispositivi

La funzione **resolve_device_from_context** (figura A.4) rappresenta uno degli elementi più rilevanti del layer Python, in quanto affronta il problema del grounding tra linguaggio naturale e identificativi tecnici del simulatore. La funzione gestisce eventuali riferimenti generici, provando ad associarli ai **device_id** effettivamente presenti in SBSim.

```python
def resolve_device_from_context(device_id, point_name, devices):  1 usage  new *
    #Risolve riferimenti generici come 'boiler' o 'ahu' verso i device_id reali
    if device_id in devices: # se ha già trovato il device -> OK
        return device_id

    raw_device = str(device_id or "").strip().lower()
    point_name = point_name or ""

    if point_name: #cerca i devices con il setpoint/measurement individuato
        candidates = [
            d for d, info in devices.items()
            if point_name in info.get("measurements", [])
            or point_name in info.get("setpoints", [])
        ]
        if len(candidates) == 1:
            return candidates[0]

    alias_prefixes = [] #mappatura sinonimi
    if any(x in raw_device for x in ["boiler", "caldaia"]):
        alias_prefixes = ["boiler"]
    elif any(x in raw_device for x in ["ahu", "air handler", "centrale aria", "air_handler"]):
        alias_prefixes = ["air_handler"]
    elif "vav" in raw_device:
        alias_prefixes = ["vav"]

    if alias_prefixes:  #per evitare che non riconosca i device che hanno un suffisso
        candidates = [d for d in devices if any(d.lower().startswith(prefix) for prefix in alias_prefixes)]
        if point_name:
            candidates = [
                d for d in candidates
                if point_name in devices[d].get("measurements", [])
                or point_name in devices[d].get("setpoints", [])
            ]
        if len(candidates) == 1:
            return candidates[0]

    if raw_device: #per evitare che non riconosca device scritti in maiuscolo/camel case/ecc.
        candidates = [d for d in devices if raw_device in d.lower()]
        if point_name:
            candidates = [
                d for d in candidates
                if point_name in devices[d].get("measurements", [])
                or point_name in devices[d].get("setpoints", [])
            ]
        if len(candidates) == 1:
            return candidates[0]

    return device_id
```

Figura A.4 Estratto della funzione resolve_device_from_context

A.5 Esecuzione delle query e dei comandi verso SBSim

La funzione **esegui_intent** (figura A.5) riceve l'oggetto **Intent** costruito dal layer Python e lo traduce nel formato richiesto da SBSim. Le richieste vengono distinte tra query, gestite tramite **ObservationRequest**, e comandi, gestiti tramite **ActionRequest**. La funzione include inoltre alcuni controlli sugli argomenti ricevuti, al fine di evitare l'inoltro al simulatore di richieste non coerenti con la struttura effettiva dell'ambiente. Nel caso in cui venga rilevata un'incoerenza, la funzione restituisce un messaggio di errore esplicativo, senza intervenire direttamente sul contenuto della risposta generata dal modello.

```
def esegui_intent(building, intent: Intent, devices: dict, zone_map: dict) -> dict:  2 usages new *
        resp = building.request_observations(req)
        v = resp.single_observation_responses[0].continuous_value #0 perchè al momento processiamo una richiesta alla volta quindi prendiamo il primo elemento
        return {
            "type": "query",
            "zone_id": intent.zone_id,
            "device_id": intent.device_id,
            "measurement_name": intent.measurement_name,
            "value": float(v),
        }

    # COMMAND
    if not intent.setpoint_name or intent.value is None: #errore se setpoint_name o valore = none
        return {
            "error": "Command senza setpoint_name o value.",
            "valid_setpoints": devices[intent.device_id]["setpoints"],
        }

    # validazione setpoints (controllo che il setpoint sia disponibile all'interno dell'intent_device)
    valid_setpoints = set(devices[intent.device_id]["setpoints"])
    if intent.setpoint_name not in valid_setpoints:
        return {
            "error": f"Setpoint '{intent.setpoint_name}' non valido per {intent.device_id}.",
            "valid_setpoints": sorted(valid_setpoints),
        }

    # creo un'istanza specifica per la richiesta dell'utente
    req = smart_control_building_pb2.ActionRequest()
    req.single_action_requests.append(
        smart_control_building_pb2.SingleActionRequest(
            device_id=intent.device_id,
            setpoint_name=intent.setpoint_name,
            continuous_value=float(intent.value),
        )
    )

    # restituisco la risposta
    resp = building.request_action(req)
    # 1 = ACCEPTED
    response_type = int(resp.single_action_responses[0].response_type) #0 perchè al momento processiamo una richiesta alla volta quindi prendiamo il primo elemento
    return {
        "type": "command",
        "zone_id": intent.zone_id,
        "device_id": intent.device_id,
        "setpoint_name": intent.setpoint_name,
        "requested_value": float(intent.value),
        "response_type": response_type,
    }
```

Figura A.5 Estratto semplificato della funzione esegui_intent

Ringraziamenti

In questo breve spazio desidero ringraziare tutte le persone senza le quali non sarei riuscito a portare a termine questo percorso.

Ringrazio innanzitutto i Professori Franco Zambonelli e Stefano Mariani, per avermi dato la possibilità di approfondire un tema così attuale e stimolante, e allo stesso tempo per i suggerimenti e il supporto fornito durante lo sviluppo del lavoro.

Desidero inoltre ringraziare i miei genitori, mia sorella, i miei nonni e tutti i miei parenti, per il sostegno e l'affetto ricevuto anche a distanza in questi anni.

Un ringraziamento speciale va a Melissa, che nei momenti di difficoltà mi ha sempre accompagnato con affetto e pazienza in questa fase. Grazie per avermi ascoltato, incoraggiato e supportato nei momenti di stanchezza, e per aver condiviso con me l'attesa di questo risultato.

Infine, ringrazio anche gli amici e tutte le persone che, in modi diversi, hanno fatto parte del mio percorso di vita, a partire dai miei compagni di Università con cui ho stretto legami indelebili, fino a quelli di sempre e quelli che si sono aggiunti durante il cammino. A ognuno di loro auguro di trovare la propria strada e la forza di percorrerla, anche nei momenti più difficili.

Grazie di cuore,

Simone