Report LLM su Server Lenovo

Framework metodologico per sistemi conversazionali enterprise

Una collaborazione Memori, Lenovo e Araneum e i loro team

a cura di: Nunzio Fiore, Fabio Lecca, Mario Sebastiani, Massimo Chiriatti

15/01/2025

Prefazione

Verso l'autonomia dell'AI conversazionale: Un progetto di ricerca congiunto

In un'epoca in cui l'intelligenza artificiale sta ridefinendo il modo in cui interagiamo con la tecnologia, tre realtà italiane d'eccellenza - Memori, Lenovo e Araneum - hanno unito le proprie forze per esplorare nuove frontiere nell'ambito degli LLM (Large Language Models) on-premise. Sotto la guida di Nunzio Fiore, CEO di Memori, Massimo Chiriatti, CTIO di Lenovo Italia, e Fabio Lecca, CTO di Araneum, questo progetto di ricerca rappresenta un punto di svolta nell'evoluzione dei sistemi conversazionali enterprise.

La piattaforma AIsuru di Memori, già affermata nel panorama dell'AI conversazionale, è stata al centro di questa sperimentazione, mettendo alla prova la sua capacità di adattarsi e performare con modelli linguistici locali su infrastrutture dedicate. L'expertise di Lenovo nell'hardware di alta performance e la profonda conoscenza di Araneum nel campo dell'intelligenza artificiale hanno creato le condizioni ideali per questo studio pionieristico.

Obiettivo della ricerca

Questo report non si limita a presentare un'analisi comparativa di modelli LLM o specifiche hardware - elementi destinati a evolversi rapidamente nel dinamico mondo dell'AI. Il suo vero valore risiede nel delineare una roadmap metodologica per le aziende che desiderano implementare soluzioni di AI conversazionale on-premise mantenendo gli stessi standard qualitativi dei servizi cloud proprietari.

La ricerca si concentra su aspetti fondamentali e duraturi:

- L'identificazione di pattern di ottimizzazione per massimizzare le performance dei modelli locali
- La definizione di metodologie scalabili per l'implementazione di sistemi conversazionali enterprise
- L'elaborazione di strategie per bilanciare qualità delle risposte e efficienza computazionale
- La creazione di framework decisionali per la scelta dei modelli più adatti alle specifiche esigenze aziendali.

Questo lavoro rappresenta non solo un punto di riferimento tecnico, ma anche una guida strategica per le organizzazioni che intendono intraprendere un percorso verso l'autonomia tecnologica nell'ambito dell'AI conversazionale, garantendo privacy, controllo e personalizzazione senza compromettere la qualità del servizio.

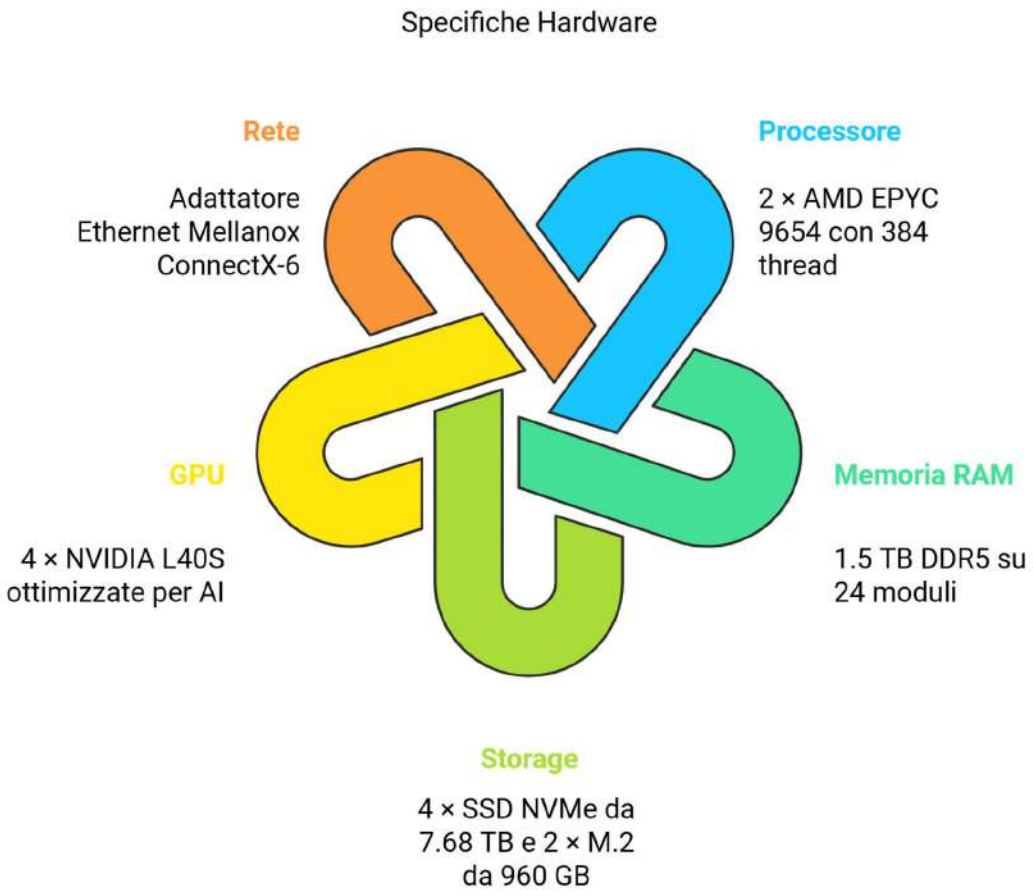
Indice

1. Il Server Lenovo

Specifiche hardware

Il server Lenovo utilizzato per i test è configurato con hardware ad alte prestazioni per garantire un'analisi approfondita delle capacità dei modelli di linguaggio di grandi dimensioni (LLM). Le principali specifiche sono le seguenti:

- **Processore**: 2 × AMD EPYC 9654 96-Core Processor, per un totale di 384 thread, con una frequenza base di 2.4 GHz.
- **Memoria RAM**: 1.5 TB, distribuita su 24 moduli DDR5 da 64 GB con frequenza di 4800 MHz.
- **Storage**:
    - 4 × SSD NVMe PCIe 4.0 da 7.68 TB (intensivo in lettura), per garantire alta velocità di accesso ai dati.
    - 2 × SSD M.2 NVMe PCIe 4.0 da 960 GB per il sistema operativo e i file di configurazione.
- **GPU**: 4 × NVIDIA L40S con 48 GB di memoria ciascuna, basate su architettura Ampere. Queste GPU sono ottimizzate per carichi AI intensivi e supportano la tecnologia CUDA 12.4.
- **Rete**: Adattatore Ethernet Mellanox ConnectX-6 Lx con supporto per velocità di 10/25 GbE.



Considerazioni sulle risorse hardware

Le risorse della macchina si sono rivelate adeguate per la maggior parte dei test, ma sono emersi alcuni aspetti critici:

- **Utilizzo delle GPU**: Alcuni modelli LLM con alto numero di parametri (>32B) hanno saturato quasi completamente una singola GPU (90% di utilizzo) durante richieste complesse, con tempi di elaborazione fino a 6 secondi per risposta. Questo suggerisce che modelli più grandi o carichi più intensi potrebbero richiedere un'infrastruttura GPU ancora più scalabile. Tali modelli sono stati comunque caricati utilizzando il “tensor parallellism” di vllm distribuendoli sulla memoria di più GPU, ma questo ha inevitabilmente diminuito il grado di parallelismo.
- **Consumo di memoria**: la RAM disponibile non hanno rappresentato un collo di bottiglia durante i test, comunque l'alto consumo di memoria durante il caricamento dei modelli lascia spazio a considerazioni sulla possibilità di ottimizzare ulteriormente l'utilizzo delle risorse.

- **Capacità di rete**: La configurazione di rete si è dimostrata adeguata per la simulazione di carichi concorrenti, con nessun impatto significativo sulla latenza.

Questa infrastruttura fornisce un ambiente solido per testare modelli LLM avanzati, consentendo un'analisi completa delle loro prestazioni in scenari simulati. I dettagli raccolti costituiscono una base per eventuali ottimizzazioni future e per la definizione di requisiti in contesti produttivi.

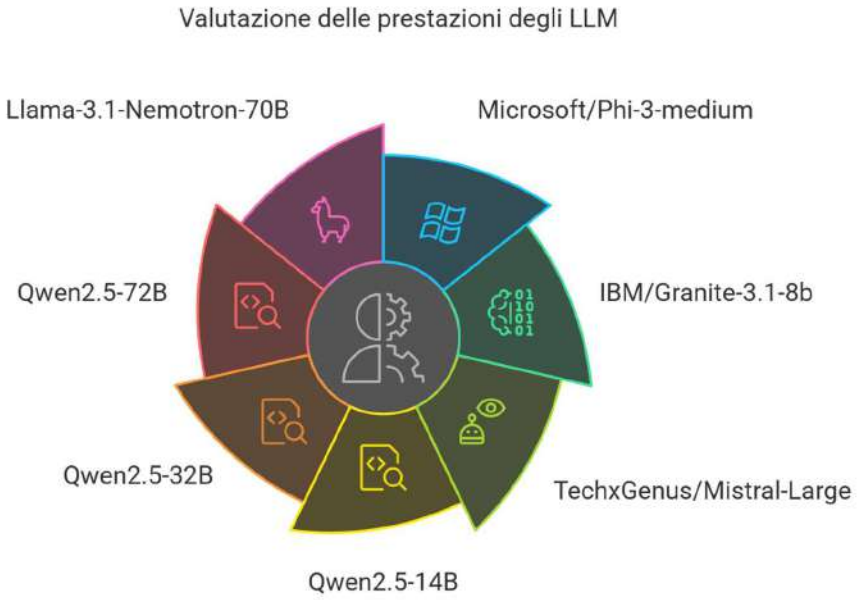
2. Modelli LLM

Modelli individuati per i test

Nel corso dell'analisi, sono stati testati diversi modelli di linguaggio di grandi dimensioni, con l'obiettivo di identificare quello più adatto per rispondere a richieste complesse e operare in ambienti multiutente ad alta intensità.

Modello	Azienda	Dimensione	Context window	Memoria occupata (approx)
gemma-2b	Google	2B	8k ⚠	~1.6 GB
gemma-7b	Google	7B	8k ⚠	~5.6 GB
gemma-2-27b-it	Google	27B	8k ⚠	~21.5 GB
Llama-3.1-405B-FP8	Meta-Llama	405B	128k	~243 GB
Llama-3.1-8B	Meta-Llama	8B	128k	~4.9 GB
Llama-3.1-8B-Instruct	Meta-Llama	8B	128k	~33 GB
Llama-3.1-Nemotron-70B-Instruct	Meta-Llama	70B	128k	~43 GB
Llama-3.3-70B-Instruct	Meta-Llama	70B	128k	~43 GB
Meta-Llama-3.3-70B-Instruct-AWQ-INT4	Meta-Llama	70B	128k	~39 GB
Mistral-Nemo-Instruct-2407	Mistral	12.2B	128k	~51 GB
Mistral-Large-Instruct-2407-AWQ	Mistral	123B	128k	~56 GB
Mistral-Large-Instruct-2411	Mistral	123B	128k	~56 GB
Mistral-Large-Instruct-2411-AWQ	Mistral	123B	128k	~56 GB
Phi-3.5-MoE-instruct	Microsoft	6.6B	128k	~23 GB
Phi-3-medium-128k-instruct	Microsoft	14B	128k	~62 GB
Qwen2.5-1.5B-Instruct	Qwen	1.54B	128k	~0.986 GB

Qwen2.5-Coder-32B-Instruct	Qwen	32.8B	128k	~18 GB
Qwen2.5-32B-Instruct-GPTQ-Int4	Qwen	32.5B	128k	~18 GB
Qwen2.5-72B-Instruct	Qwen	72.7B	128k	~40 GB
Qwen2.5-72B-Instruct-AWQ	Qwen	72.7B	128k	~40 GB
Granite-3.1-8b-instruct	Ibm	8B	128k	~9 GB



Per determinare la memoria realmente utilizzata dai modelli è stato realizzato uno script phyton **mem.py,** che per ogni modello forniva il valore della memoria totale allocata

python mem.py "google/gemma-2b"
Used GPU memory: 9560.2978515625 MB

Se un modello non riesce ad essere caricato in memoria in una singola GPU, lo script fallisce ed esso viene caricato tramite vllm utilizzando un numero crescente di GPU utilizzando il parametro --tensor-parallel-size.

Alcuni modelli sono così grandi da saturare lo spazio complessivo della memoria delle 4 GPU e pur essendo disponibile la funzionalità di offloading, non è stata ritenuta valida ai fini delle performance di esercizio (ad es. DeepSeek-V3 caricato in 750GB della RAM tradizionale ha evidenziato una velocità di soli 3 token/secondo, inutilizzabile per i nostri scopi).

Oltre alla valutazione qualitativa e quantitativa delle prestazioni, è fondamentale analizzare alcune caratteristiche tecniche chiave dei modelli testati per comprenderne le implicazioni

pratiche nell'adozione. Questi parametri, infatti, influenzano direttamente la scelta del modello in base alle esigenze specifiche di utilizzo.

Dimensione del modello e utilizzo della memoria

La dimensione del modello, espressa in termini di parametri (es. 8B, 70B, ecc.), è uno dei fattori principali che determinano la complessità computazionale e il consumo di risorse. Modelli più grandi come **Llama-3.1-Nemotron-70B-Instruct** e **Qwen2.5-72B-Instruct** offrono generalmente una maggiore capacità di generalizzazione e comprensione del contesto, ma richiedono risorse hardware più elevate, come memoria GPU e RAM. Ad esempio:

- **Qwen2.5-72B-Instruct** occupa circa **43 GB di memoria**, il che lo rende impegnativo per l'esecuzione su macchine con risorse limitate.
- Modelli più piccoli, come **gemma-2b**, richiedono circa **1.6 GB di memoria**, ma le loro prestazioni sono limitate a scenari meno complessi.

Questa correlazione tra dimensione e risorse necessarie sottolinea la necessità di un'attenta pianificazione dell'infrastruttura per garantire l'esecuzione fluida del modello scelto.

Finestra di contesto e capacità di elaborazione

Un altro elemento cruciale è la dimensione della **finestra di contesto**, che definisce la quantità di informazioni che un modello può elaborare contemporaneamente. I modelli selezionati per il test come da requisito presentano tutti una finestra di contesto estesa (**128k token**), adatta per elaborare testi complessi e richieste che richiedono una comprensione del contesto a lungo termine.

Tuttavia, l'utilizzo di una finestra così ampia può comportare un aumento dei costi computazionali, in particolare per modelli di grandi dimensioni. Inoltre, un contesto eccessivamente dettagliato può introdurre rumore, rendendo più difficile per il modello estrarre le informazioni più rilevanti. Questo problema è stato particolarmente evidente in modelli come **Phi-3-medium-128k-instruct**, che ha mostrato una riduzione della coerenza delle risposte in presenza di input molto estesi (phi-4 è stato testato la mattina del 9 gennaio ma ha mostrato limiti analoghi).

Impatto degli algoritmi di ottimizzazione

Alcuni modelli, come **Meta-Llama-3.3-70B-Instruct-AWQ-INT4**, sono stati utilizzati nella versione quantizzata in INT4. Queste tecniche riducono significativamente l'utilizzo della memoria della GPU, rendendo modelli di grandi dimensioni più accessibili per sistemi con

risorse limitate. Tuttavia, tali ottimizzazioni possono comportare una leggera perdita di precisione, che potrebbe non essere accettabile per applicazioni critiche.

3. Criteri di valutazione delle prestazioni mediante gli Agenti di AiSuru

Qualità delle risposte e affidabilità

Per valutare la qualità delle risposte fornite dai modelli LLM testati, è stata implementata una metodologia qualitativa che ha coinvolto i seguenti modelli selezionati in base ai criteri indicati (128k di contesto, possibilità di caricare interamente nella GPU, utilizzo del function calling):

- **microsoft/Phi-3-medium-128k-instruct**
- **ibm-granite/granite-3.1-8b-instruct**
- **TechxGenus/Mistral-Large-Instruct-2411-AWQ**
- **Qwen2.5-14B-Instruct-AWQ**
- **Qwen2.5-32B-Instruct-AWQ**
- **Qwen2.5-72B-Instruct-AWQ**
- **Llama-3.1-Nemotron-70B-Instruct**

Ogni modello LLM presente sulla macchina è stato esposto tramite reverse proxy (in maniera seriale per limiti della RAM delle GPU) e collegato con la piattaforma AISuru, sulla quale sono stati configurati tre assistenti virtuali, denominati **Manuela**, **Cloe** e **SquadraCalcio* (***un agente reale ma anonimizzato per non rivelare il cliente per cui è stato creato), ciascuno dotato di un diverso prompt introduttivo, progettato per guidare il comportamento e il tono delle risposte.

Tutti gli agenti vengono dall'utilizzo reale di clienti di Memori che adoperano questi agenti per lavoro quotidianamente. Il tool Aisuru permette di farli funzionare agganciandoli a LLM come Claude Sonnet 3.5 o GPT4o.

L'utilizzo reale degli agenti prevede di sfruttare al meglio gli LLM online ma lo scopo di questo esperimento era di vedere a parità di condizioni come si comporta un modello locale su una buona macchina locale.

Aisuru permette l'ottimizzazione dei prompt e dei dati e di applicare ogni strategia utile a un funzionamento migliore una volta che si sceglie di andare in locale.

A ciascun assistente è stata sottoposta una serie di domande, distribuite in ambiti specifici di applicazione, e le risposte sono state archiviate in una tabella per l'analisi.

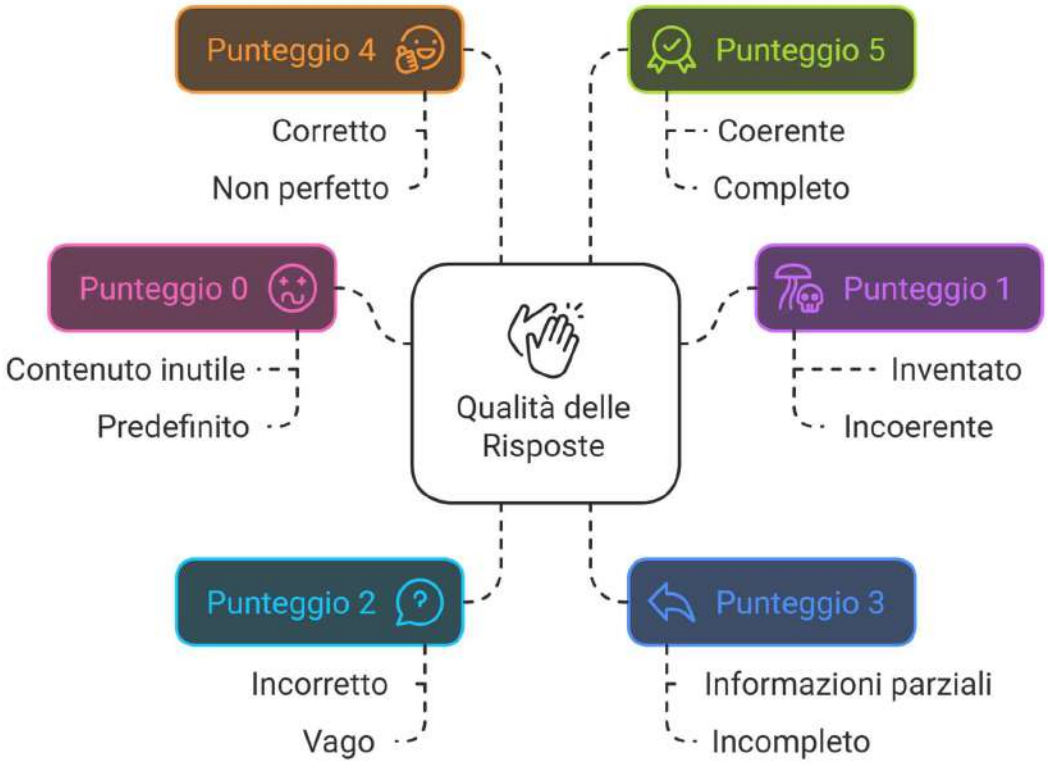
E' stato realizzato uno script che utilizza le API di AISuru per aprire una sessione, sottoporre le domande concordate e salvare le relative risposte in un file di testo, anche se è stato utilizzato su un sottoinsieme di casi in quanto l'Agente Cloe richiedeva l'apertura della chat da contesti specifici che erano già preimpostati. In altri casi il tempo di elaborazione della risposta

da parte dei modelli più grandi eccedeva il tempo di timeout del tunnel e quindi in questi casi le risposte sono state catturate manualmente.

Le risposte sono state quindi valutate da un operatore umano sulla base di una scala predefinita, con una leggenda che assegna un punteggio compreso tra 0 e 5:

- **0**: Risposta non valida o predefinita, priva di contenuto utile.
- **1**: Allucinazione evidente, con contenuti inventati o incoerenti.
- **2**: Risposta errata, vaga o non congruente con la domanda.
- **3**: Risposta parziale, contenente solo una parte delle informazioni richieste.
- **4**: Risposta corretta, sebbene non perfettamente aderente a quanto atteso.
- **5**: Risposta attesa, pienamente coerente con la domanda e completa nelle informazioni.

Per ogni domanda, nella stessa riga della tabella di valutazione erano riportati la domanda, la risposta attesa e la risposta generata dal modello. Questa configurazione ha facilitato l'analisi comparativa tra i modelli e i loro rispettivi assistenti.



Il risultato di tutti i singoli test è stato caricato in un Google Doc che complementa il presente documento, per comodità riportiamo la tabella con la media delle votazioni per i modelli, dai quali si evince che modelli con un maggior numero di parametri quasi sempre forniscono risposte migliori, a scapito della velocità di elaborazione, quindi il consiglio è quello di selezionare il modello che risponde in maniera ottimale anche rispetto alle prestazioni attese (throughput).

	Manuela	Cloe	Squadra Calcio
microsoft/Phi-3-medium-128k-instruct	2.25	3.05	2.3
ibm-granite/granite-3.1-8b-instruct	2.95	3.7	2.25
TechxGenus/Mistral-Large-Instruct-2411-AWQ	3.35	3.7	2.55
Qwen2.5-14B-Instruct-AWQ	3.95	3.25	2.45
Qwen2.5-32B-Instruct-AWQ	4	3.55	2.55
Qwen2.5-72B-Instruct-AWQ	4.45	3.75	3.25
Llama-3.1-Nemotron-70B-Instruct	n/a	3.75	3.25

Nota: il modello llama3.1 ha presentato incongruenze con le funzioni dell'Agente "Manuela" quindi non è stato possibile effettuare il test.

Performance sotto carico

Durante i test qualitativi, è stata valutata anche la capacità dei modelli di mantenere una qualità costante delle risposte in condizioni controllate, adottando un approccio uniforme per garantire la comparabilità. A tutti i modelli è stata assegnata la stessa temperatura durante i test, un parametro che influenza il livello di casualità e creatività delle risposte. Tuttavia, è importante notare che il valore di temperatura non ha la stessa scala e quindi non ha un effetto identico su tutti i modelli, a causa delle differenze nelle rispettive architetture e nei processi di ottimizzazione. Ciò significa che una stessa impostazione di temperatura potrebbe generare risposte più creative in un modello e più conservative in un altro.

Analisi comparativa delle performance

L'analisi delle performance sotto carico ha evidenziato significative differenze tra i modelli in termini di qualità delle risposte fornite. I risultati medi per ciascun modello, calcolati sulla base dei punteggi assegnati ai tre assistenti (Manuela, Cloe e SquadraCalcio), sono riassunti di seguito:

- **microsoft/Phi-3-medium-128k-instruct**: ha ottenuto un punteggio medio di 2.53, evidenziando difficoltà nella generazione di risposte coerenti, con una tendenza a produrre contenuti vaghi o non pertinenti.
- **ibm-granite/granite-3.1-8b-instruct**: il modello ha registrato una media di 2.96, dimostrando una qualità leggermente superiore, sebbene ancora non sufficiente per scenari applicativi complessi.
- **TechxGenus/Mistral-Large-Instruct-2411-AWQ**: con una media di 3.2, il modello ha fornito prestazioni accettabili, ma con una tendenza a risposte parziali o incomplete.
- **Qwen2.5-14B-Instruct-AWQ**: ha ottenuto un punteggio medio di 3.21, posizionandosi nella fascia media, con un livello di qualità variabile tra i tre assistenti testati.
- **Qwen2.5-32B-Instruct-AWQ**: con una media di 3.36, il modello ha mostrato una buona capacità di fornire risposte corrette, sebbene non sempre perfettamente aderenti alle attese.
- **Qwen2.5-72B-Instruct-AWQ**: ha registrato una media di 3.81, dimostrando una notevole capacità di generare risposte di alta qualità, con un'elevata coerenza tra i tre assistenti.
- **Llama-3.1-Nemotron-70B-Instruct**: con un punteggio medio di 3.5, questo modello si è distinto per una buona affidabilità nelle risposte, sebbene con alcune oscillazioni nella qualità tra i diversi assistenti.

Osservazioni sui risultati qualitativi degli Agenti di AISuru

Il modello **Qwen2.5-72B-Instruct-AWQ** ha ottenuto i migliori risultati medi, evidenziando una superiorità nella capacità di generare risposte attese e aderenti al contesto. Tuttavia, le differenze tra i tre assistenti suggeriscono che il prompt introduttivo può influenzare significativamente la qualità delle risposte, anche per i modelli più avanzati.

I modelli della famiglia Qwen hanno mostrato un miglioramento progressivo delle performance all'aumentare delle dimensioni del modello (14B, 32B, 72B), confermando l'importanza della scalabilità per ottenere risposte di qualità superiore. Al contrario, modelli come **microsoft/Phi-3-medium-128k-instruct** e **ibm-granite/granite-3.1-8b-instruct** si sono rivelati meno adatti a scenari che richiedono coerenza e accuratezza elevate.

Infine, il modello **Llama-3.1-Nemotron-70B-Instruct** ha dimostrato una buona affidabilità complessiva, pur non eccellendo rispetto a **Qwen2.5-72B**. Questo risultato conferma che, oltre alle dimensioni del modello, altri fattori architetturali e di configurazione possono influire significativamente sulla performance sotto carico.

Questi risultati sottolineano l'importanza di adattare i parametri di test e la configurazione del prompt alle specificità di ciascun modello, per massimizzarne le capacità e garantire risultati ottimali in contesti applicativi reali.

Il recente modello llama3.3 si è rivelato problematico con la gestione del function calling e quindi è stato scartato nei test sulla qualità delle risposte dell'Agente.

4. Telemetria

Prima di avviare i test con vllm è stata abilitata la telemetria grazie ad una funzione incorporata per utilizzare il protocollo OTLP (Open Telemetry Protocol) ed è stata avviata una instanza del prodotto open source Jaeger (www.jaegertracing.io) per poter raccogliere il dettaglio dei dati di performance delle varie fasi di gestione delle richieste da parte di vllm per poter distinguere ad es. i tempi di accodamento e quelli di processamento del modello.

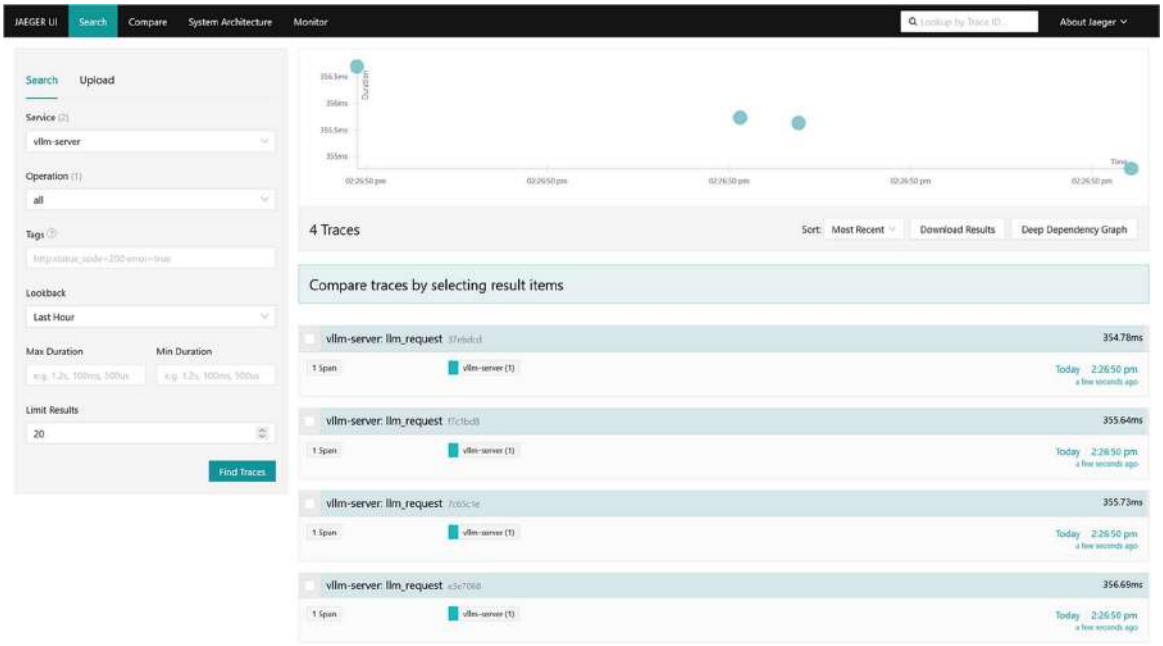
```
docker run -d --name jaeger \
  -e COLLECTOR_ZIPKIN_HTTP_PORT=9411 \
  -p 5775:5775/udp \
  -p 6831:6831/udp \
  -p 6832:6832/udp \
  -p 5778:5778 \
  -p 16686:16686 \
  -p 14268:14268 \
  -p 9411:9411 \
  jaegertracing/all-in-one:1.6.0
```

dopo aver avviato il container e impostato le seguenti variabili di ambiente:

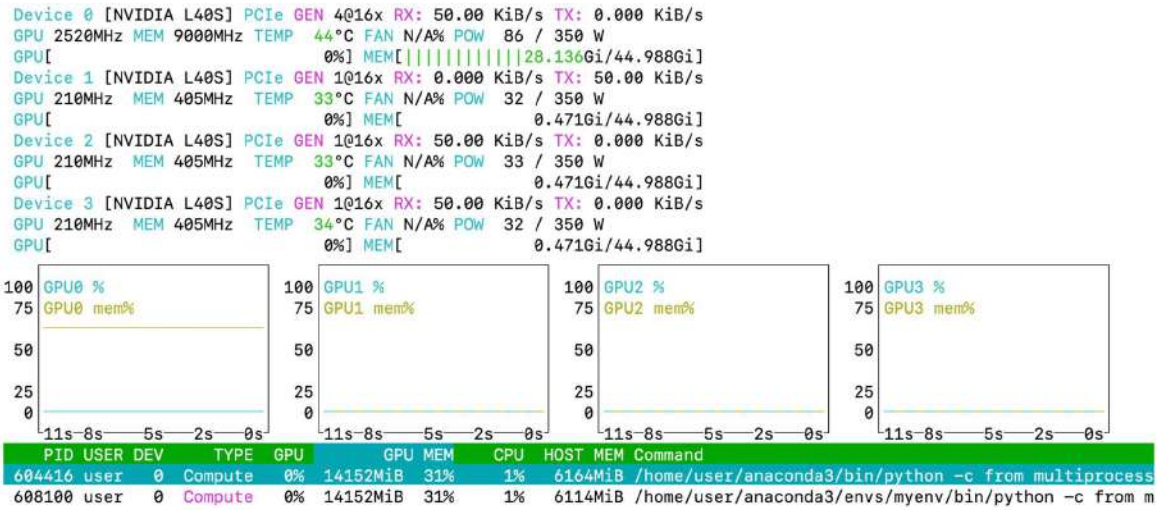
```
export OTEL_EXPORTER_OTLP_TRACES_ENDPOINT=grpc://$JAEGER_IP:4317
echo $OTEL_EXPORTER_OTLP_TRACES_ENDPOINT
export OTEL_SERVICE_NAME="vllm-server"
export OTEL_EXPORTER_OTLP_TRACES_INSECURE=true
```

vllm viene quindi avviato con il parametro:

```
 --otlp-traces-endpoint="$OTEL_EXPORTER_OTLP_TRACES_ENDPOINT"
```



Per la verifica dello stato della memoria e della CPU delle GPU è stato usato il comando nvtop.



5. Test di prestazioni

Nell'ambito della valutazione delle capacità dei modelli linguistici (LLM) a nostra disposizione, abbiamo condotto un'analisi approfondita delle loro performance utilizzando due strumenti principali: Apache JMeter e un sistema di benchmark interno a vLLM. Questi strumenti ci hanno permesso di esaminare e confrontare in modo sistematico la risposta dei modelli in scenari di carico differenti, evidenziando sia i punti di forza che le aree di miglioramento.

Strumenti utilizzati

Apache JMeter è un'applicazione open source scritta in Java, progettata per eseguire test di carico e misurare le prestazioni di applicazioni, server e protocolli. Inizialmente sviluppato per testare applicazioni web, si è evoluto per supportare numerosi tipi di test, tra cui l'invio di richieste a servizi REST API, come quelle utilizzate nei nostri esperimenti.

Grazie alla sua capacità di simulare carichi elevati con richieste concorrenti, JMeter ci ha permesso di:

- **Inviare molteplici chiamate API** ai modelli con diverse domande.
- **Misurare il throughput**, ovvero il numero di richieste completate con successo in un determinato intervallo di tempo.
- **Valutare le prestazioni** dei modelli in base a parametri variabili, come il numero di thread (utenti simulati) e il carico complessivo.

Questa configurazione ci ha consentito di analizzare il comportamento dei modelli sotto stress e di confrontare la loro efficienza nel rispondere alle richieste, senza valutare effettivamente il contenuto qualitativo della risposta. Abbiamo scelto di utilizzare prompt fissi e di utilizzare come metrica il numero di richieste completate al secondo (throughput) invece dei token al secondo che sono meno indicativi del numero di utenti simultanei che possono essere gestiti. La misura dei token/sec è stata ottenuta con lo script di vllm e viene descritta successivamente.

Prompt usati

Per analizzare le prestazioni dei modelli linguistici, sono stati definiti **cinque prompt di test**, ciascuno progettato per simulare scenari di utilizzo specifici con difficoltà differenti.

Prompt 1: Domanda semplice

Il primo test ha previsto un prompt molto diretto e immediato:
"Parlami della città di Roma."

L’obiettivo era valutare come il modello si comportasse a rispondere ad una domanda banale, senza contesto complesso o richieste particolari. Le impostazioni utilizzate erano le seguenti:

- **Temperatura:** 0 (risposte deterministiche e prive di variabilità).
- **Lunghezza massima dell'output (max_length):** 100 token.

Prompt 2: Simulazione di un comportamento di un’impiegata e valutazione dei sentimenti delle risposte

Il secondo test ha simulato un contesto molto più articolato (circa 1400 tokens), dove il modello doveva interpretare un ruolo specifico e interagire in base a una personalità dettagliata: **Ruolo:** L'assistente interpreta Giovanna, un’impiegata ansiosa e difensiva. **Scenario:** L'utente (nel ruolo di manager) cerca di fornire feedback per migliorare le performance lavorative di Giovanna, affrontando il suo atteggiamento ostile e chiuso.

Il prompt richiedeva una risposta in linea con il personaggio e includeva anche un feedback finale sulla qualità della comunicazione dell’utente. Le impostazioni per questo test erano:

- **Max tokens:** 8192
- **Temperature:** 0.8
- **Top_p:** 0.95
- **Frequency penalty:** 0
- **Presence penalty:** 0

Prompt 3: Risposte su un progetto europeo specifico

Il terzo test ha previsto un contesto ancora più esteso (circa 5400 tokens) legato alla descrizione del progetto **TRACE4EU**, un’iniziativa europea focalizzata sulla tracciabilità dei prodotti nel settore agroalimentare e della pesca tramite l’utilizzo della tecnologia blockchain. L'obiettivo del modello era rispondere in italiano a domande specifiche sul progetto, mantenendo un linguaggio preciso, formale e limitandosi alle informazioni fornite e con un massimo di 200 parole.

Specifiche tecniche del test:

- **Max tokens:** 8192
- **Temperature:** 0.7
- **Frequency penalty:** 0
- **Presence penalty:** 0

Prompt 4: Riassunto di un testo lungo sulla storia della mafia

Il quarto test si è basato sulla capacità del modello di analizzare e sintetizzare un testo esteso (circa 4000 token) sulla storia della mafia. Il prompt richiedeva di produrre un riassunto che fosse accurato, conciso e che mantenesse la coerenza e i punti salienti del contenuto originale.

Specifiche tecniche del test:

- **Max tokens:** 8192.
- **Temperature:** 0.7
- **Frequency penalty:** 0
- **Presence penalty:** 0

L'obiettivo di questi tre prompt era quello di capire le performance dei modelli in base a domande e contesti di dimensioni molto elevate.

Prompt 5: Simulazione di una Function Call

Il quinto test si concentrava sulla capacità del modello di riconoscere la necessità di una chiamata di funzione e gestirla correttamente attraverso due fasi distinte:

1. **Prima fase:** Il modello riceveva un prompt iniziale contenente una richiesta strutturata che implicava l'invocazione di una funzione per ottenere un risultato. L'obiettivo era verificare se il modello riconoscesse che non poteva fornire una risposta diretta senza simulare la chiamata di una funzione.
2. **Seconda fase:** Una volta riconosciuta la necessità della function call, al modello veniva inviato un secondo prompt, contenente:
    - La domanda iniziale.
    - Le informazioni che la funzione avrebbe dovuto restituire.

L'obiettivo di questo test era valutare la capacità del modello di simulare uno scenario dinamico, in cui il completamento della risposta richiedeva una chiamata esterna.

Modalità di test

Per ogni prompt, sono state adottate due modalità di test, mirate a valutare sia la capacità di gestire richieste seriali che quella di supportare richieste parallele:

1. **Richieste seriali:**

    Un singolo utente inviava il prompt al modello **100 volte consecutivamente**. Questo test mirava a misurare il throughput in uno scenario seriale, ossia quante richieste al secondo il modello fosse in grado di gestire in una sequenza non parallela.

2. **Richieste parallele:**

    10 utenti simulati inviavano **50 richieste ciascuno**, per un totale di 500 richieste parallele. Questo test era progettato per valutare la capacità del sistema di gestire richieste contemporanee, analizzando il livello di parallelizzazione supportato.

Implementazione del test

I test sono stati eseguiti tramite un comando CLI configurato dinamicamente per invocare jmeter su un piano di test contenente il relativo prompt. La struttura del comando era la seguente:

```
command = [
   jmeter_path, "-n", "-t", jmx_file,"-Jmodel", model, "-JNUM_THREADS", str(NUM_THREADS_1),
   "-JNUM_LOOPS", str(NUM_LOOPS_1),"-l", results_file_1, "-e", "-o", report_path_1
]
```

Parametri principali del comando:

- **jmx_file:** File JMeter (.jmx) configurato per il prompt specifico.
- **model:** Il Large Language Model da testare.
- **NUM_THREADS:** Numero di utenti simulati (1 per test seriali, 10 per test paralleli).
- **NUM_LOOPS:** Numero di iterazioni per utente (100 per seriali, 50 per paralleli).

Output generato:

I risultati del test venivano salvati in una cartella designata, contenente il report generato da JMeter. I valori principali estratti per l'analisi erano:

- **Transaction/s** (richieste gestite al secondo).
- **Tempo minimo di risposta.**
- **Tempo massimo di risposta.**
- **Tempo medio di risposta.**

Automazione dei test

L'intero processo è stato automatizzato mediante uno script che costruiva dinamicamente il comando CLI per ogni combinazione di modello e prompt.

1. **File JMX preconfigurati:** Per ogni prompt era stato creato un file `.jmx`, che includeva la configurazione necessaria per inviare le richieste.
2. **Script di automazione:** Lo script iterava su ogni modello, generava i comandi appropriati (seriali e paralleli) e li eseguiva automaticamente. Questo ha permesso di eseguire i 10 test (5 seriali e 5 paralleli) in sequenza per ogni modello, senza intervento manuale, salvando i risultati al termine di ciascun ciclo.
3. **Cambio di modello:** L'unico intervento richiesto era cambiare il modello nel parametro dello script per avviare un nuovo set di test.

Risultati

I risultati ottenuti hanno confermato alcune tendenze chiave legate alle dimensioni dei modelli, alla complessità dei prompt e alla modalità di invio delle richieste.

1. Performance in funzione della dimensione del modello

È emerso chiaramente che all'aumentare delle dimensioni del modello (e quindi del numero di parametri), le performance degradano in termini di tempo di risposta e numero di transazioni al secondo (transaction/s).

Ad esempio, nel caso del **Prompt 1**:

- Qwen2.5-14B-Instruct-AWQ: Modello con **14B parametri**: **1.10 t/s**
- Qwen2.5-72B-Instruct-AWQ: Modello con **72B parametri**: **0.47 t/s**

Questo comportamento è prevedibile, poiché modelli più grandi richiedono maggiore potenza computazionale e tempo per generare una risposta.

2. Impatto della complessità del prompt

La complessità del prompt influisce significativamente sulle performance del modello. Prompt più articolati e con un maggiore carico di elaborazione comportano una riduzione delle transazioni al secondo.

Per esempio, utilizzando lo stesso modello da **14B parametri**, si osserva:

- **Prompt 1** (semplice): **1.10 t/s**
- **Prompt 4** (riassunto lungo)**: 0.31 t/s**

Questo trend è stato rilevato per tutti i modelli, evidenziando che la complessità delle richieste aumenta il tempo di elaborazione necessario.

3. Confronto tra richieste seriali e parallele

Nel contesto delle richieste parallele (10 utenti simultanei), non è stato osservato un degrado diretto delle performance simile a quello dei test seriali. Al contrario, per gli stessi modelli e prompt, il numero di transazioni al secondo è risultato più alto rispetto ai test seriali.

Ad esempio, per lo stesso **Prompt 1 e modello da 8B parametri**:

- **Richieste seriali: 1.10 t/s**
- **Richieste parallele: 9.23 t/s**

Questo risultato suggerisce che il sistema VLLM sfrutti efficacemente la parallelizzazione delle richieste, permettendo al modello di processare simultaneamente più richieste senza un calo significativo delle performance complessive.

Di seguito vengono riportate due tabelle che sintetizzano i risultati dei test:

La seguente tabella mostra le performance registrate per richieste seriali da un singolo utente.

Modello	Prompt 1				Prompt 2				Prompt 3			
	T/s	Avg	Min	Max	T/s	Avg	Min	Max	T/s	Avg	Min	Max
Qwen2.5-14B-Instruct-AWQ	1,10	903,99	851,00	1225,00	0,82	1214,55	652,00	2701,00	0,35	2850,33	2082,00	3533,00
Qwen2.5-32B-Instruct-AWQ	0,76	1313,92	1302,00	1469,00	0,55	1822,22	417,00	3805,00	0,18	5540,67	4484,00	6710,00
Qwen2.5-72B-Instruct-AWQ	0,47	2148,07	2135,00	2307,00	0,25	3947,26	1647,00	6615,00	0,07	14965,00	10476,00	22032,00
microsoft/Phi-3-medium-128k-instruct	0,42	2371,90	2325,00	2556,00	0,23	4309,26	957,00	195259,00	0,10	9956,61	1752,00	24260,00
ibm-granite/granite-3.1-8b-instruct	0,99	1006,15	995,00	1129,00	0,42	2380,37	458,00	5791,00	0,18	5599,75	2802,00	11512,00
TechxGenus/Mistral-Large-Instruct-2411-AWQ	0,32	3163,93	3088,00	3480,00	0,22	4601,50	1636,00	8957,00	0,06	16562,86	12792,00	22802,00
Llama-3.1-Nemotron-70B-Instruct	0,18	5455,70	5433,00	5631,00	0,04	22407,26	10122,00	31589,00	0,03	36521,25	15888,00	53375,00

Modello	Prompt 4				Prompt 5			
	T/s	Avg	Min	Max	T/s	Avg	Min	Max
Qwen2.5-14B-Instruct-AWQ	0,31	3186,30	2404,00	4870,00	0,93	1070,32	413,00	3056,00
Qwen2.5-32B-Instruct-AWQ	0,15	6582,57	4574,00	11134,00	0,60	1665,33	563,00	9389,00
Qwen2.5-72B-Instruct-AWQ	0,07	15175,74	8692,00	34572,00	0,39	2552,16	870,00	6076,00

microsoft/Phi-3-medium-128k-instruct	0,08	12184,08	6474,00	24185,00	0,51	1949,81	54,00	10189,00
ibm-granite/granite-3.1-8b-instruct	0,12	8188,35	4927,00	12617,00	0,95	1056,56	351,00	3106,00
TechxGenus/Mistral-Large-Instruct-2411-AWQ	0,04	25059,57	17387,00	33393,00	0,23	4321,30	1832,00	10738,00
Llama-3.1-Nemotron-70B-Instruct	0,04	27509,14	23465,00	32747,00	0,07	14004,52	6280,00	41352,00

La seguente tabella riassume le performance ottenute con richieste parallele da parte di 10 utenti simultanei.

Modello	Prompt 1				Prompt 2				Prompt 3			
	T/s	Avg	Min	Max	T/s	Avg	Min	Max	T/s	Avg	Min	Max
Qwen2.5-14B-Instruct-AWQ	9,23	1078,66	1006,00	1437,00	3,22	3030,58	859,00	7877,00	0,96	10275,42	2885,00	16616,00
Qwen2.5-32B-Instruct-AWQ	6,27	1591,20	1545,00	1896,00	2,04	4755,25	449,00	11392,00	0,57	17559,95	3594,00	30970,00
Qwen2.5-72B-Instruct-AWQ	10,53	4733,14	4501,00	4974,00	0,74	9897,97	1863,00	324359,00	0,24	41130,09	13572,00	65427,00
microsoft/Phi-3-medium-128k-instruct	14,06	3546,47	3441,00	4259,00	1,90	3795,82	685,00	122260,00	0,39	22336,44	3657,00	448718,00
ibm-granite/granite-3.1-8b-instruct	27,09	1830,54	1619,00	2214,00	1,51	4767,21	790,00	112571,00	0,65	13113,52	3473,00	209952,00
TechxGenus/Mistral-Large-Instruct-2411-AWQ	7,67	6459,43	4754,00	9600,00	0,65	14725,21	3281,00	34332,00	0,17	59712,45	14790,00	96883,00
Llama-3.1-Nemotron-70B-Instruct	5,96	8375,23	8079,00	8589,00	0,32	29754,12	9822,00	46183,00	0,15	65615,25	23289,00	151051,00

Modello	Prompt 4				Prompt 5			
	T/s	Avg	Min	Max	T/s	Avg	Min	Max
Qwen2.5-14B-Instruct-AWQ	2,63	3632,02	2458,00	8885,00	5,78	1701,76	438,00	6142,00

Qwen2.5-32B-Instruct-AWQ	0,57	16971,21	5521,00	29110,00	3,44	2696,38	572,00	82860,00
Qwen2.5-72B-Instruct-AWQ	0,25	37825,01	10023,00	87932,00	2,21	4380,28	875,00	12411,00
microsoft/Phi-3-medium-128k-instruct	0,40	23949,32	11538,00	45724,00	3,57	2621,90	53,00	18791,00
ibm-granite/granite-3.1-8b-instruct	0,64	15459,98	7378,00	23898,00	6,36	1515,97	461,00	5174,00
TechxGenus/Mistral-Large-Instruct-2411-AWQ	0,15	65655,09	21059,00	110508,00	1,35	7119,26	1534,00	20358,00
Llama-3.1-Nemotron-70B-Instruct	0,20	49138,81	31899,00	67077,00	0,55	16954,29	2038,00	51362,00

6. Funzionamento del function calling

Requisiti e supporto nei modelli testati

Il function calling rappresenta una funzionalità avanzata che consente ai modelli LLM di interagire con sistemi esterni attraverso l'esecuzione di funzioni definite in un contesto predefinito. La configurazione implementata durante i test ha previsto l'utilizzo di un intermediario per orchestrare le chiamate. Il processo seguito è stato il seguente:

. Inizializzazione del contesto: al modello è stata fornita una lista di funzioni disponibili, con relative descrizioni e parametri, inserita nella finestra di contesto.

. Valutazione del modello: durante l'elaborazione della richiesta, il modello decideva se invocare una funzione in base alle informazioni fornite.

. Interazione con la funzione: l'intermediario gestiva l'esecuzione della funzione invocata, passando poi il risultato nuovamente al modello per completare l'elaborazione della risposta finale.

Tutti i modelli selezionati sono compatibili con il function calling, purché configurati correttamente. Tuttavia, la qualità dell'interazione tra il modello e il sistema dipende da diversi fattori, tra cui la capacità del modello di comprendere e selezionare la funzione più appropriata e la gestione efficiente della context windows.

Risultati sui test specifici

Durante i test, sono emerse criticità legate principalmente alla dimensione del contesto e all'affidabilità delle interazioni con le funzioni invocate.

La quantità di informazioni passate al modello, inclusi i dettagli delle funzioni disponibili e i dati della richiesta, si è dimostrata un fattore determinante per il successo del function calling. Modelli con una finestra di contesto limitata hanno riscontrato difficoltà nell'analizzare correttamente le informazioni e nel prendere decisioni coerenti. Quando il contesto superava una certa soglia, le prestazioni del modello peggioravano sensibilmente, con risposte incoerenti o mancata selezione della funzione corretta.

Sebbene **Llama 3.3** si sia dimostrato valido nella generazione di risposte coerenti, ha mostrato difficoltà significative nell'interagire correttamente con il sistema di function calling. In diverse occasioni, il modello non è riuscito a selezionare la funzione corretta. Per mitigare questa limitazione, si è optato per l'utilizzo del modello **Llama 3.1**, che, pur essendo leggermente meno avanzato in termini di qualità generale, ha garantito una maggiore affidabilità per i test delle funzioni.

7. Benchmark di vllm

Questo capitolo fornisce un'analisi quantitativa delle performance dei modelli testati, utilizzando un dataset di circa **53k conversazioni ShareGPT** derivate da un dataset iniziale di ~100k interazioni presente su Huggingface:

https://huggingface.co/datasets/anon8231489123/ShareGPT_Vicuna_unfiltered/blob/main/ShareGPT_V3_unfiltered_cleaned_split.json

vLLM

vLLM è una libreria open-source progettata per l'inferenza e il servizio di modelli di linguaggio di grandi dimensioni (LLM) in modo efficiente e ad alte prestazioni. Con il comando *serve* si possono istanziare i LLM installati per interrogarli e valutarne le prestazioni.

Inoltre, disponendo di 4 GPU, vLLM ha distribuito il carico del modello su tutte le GPU utilizzando il parametro `--tensor-parallel-size 4`, ottimizzando così l'uso delle risorse hardware.

L'obiettivo principale è stato misurare l'efficienza computazionale dei modelli, valutando metriche fondamentali come throughput, velocità di elaborazione dei token e tempi di completamento delle richieste.

Metodologia di benchmark

I test sono stati condotti utilizzando input standardizzati e un limite massimo di token per la risposta (**Output-len**) impostato rispettivamente a 128 e 512 token. Per ogni modello, sono stati raccolti i seguenti indicatori di performance:

- **throughput (requests/s)**: il numero di richieste completate al secondo, che rappresenta un indicatore diretto della capacità del modello di gestire carichi multiutente.
- **total tokens/s**: il totale dei token processati al secondo, inclusi input ed elaborazione interna.
- **output tokens/s**: il numero di token generati dal modello al secondo, un indicatore dell'efficienza del modello nella produzione di risposte.
- **tempo di completamento**: il tempo necessario per processare l'intero dataset dato in input.
- **output-len**: il limite massimo di token per le risposte generate.

I test sono stati condotti mantenendo una configurazione uniforme, garantendo una temperatura costante per tutti i modelli. Tuttavia, come già accennato in capitoli precedenti, lo stesso valore di temperatura può avere effetti diversi su modelli differenti, a causa delle variazioni nelle loro architetture e implementazioni.

Modello	Throughput requests/s	total tokens/s	output tokens/s	Tempo	Output-len
ibm-granite/granite-3.1-8b-instruct	8.8	6732.69	4503.63	1:53	512
DiTy/gemma-2-27b-it-function-calling-GGUF	4.38	3303.75	2244.2	3:47	512
google/gemma-2-27b	4.39	3305.37	2245.3	3:47	512
google/gemma-2-27b-it	4.41	3321.08	2255.98	3:46	512
casperhansen/llama-3.3-70b-instruct-awq	2.71	1993.39	1386.09	6:08	512
google/gemma-2-9b-it	6.12	4615.41	3135.2	2:42	512
google/gemma-2-9b-it	14.34	10811.68	7344.11	1:09	512
google/gemma-7b	7.26	5472.21	3717.14	2:17	512
google/gemma-2b	14.24	10734.51	7291.69	1:09	512

casperhansen/llama-3.3-70b-instruct-awq	6.58	2316.96	841.74	2:31	128
ibnzterrell/Meta-Llama-3.3-70B-Instruct-AWQ-INT4	6.66	2345.59	852.14	2:29	128
kosbu/Llama-3.3-70B-Instruct-AWQ	6.65	2344.34	851.69	2:29	128
meta-llama/Llama-3.1-8B	28.38	9999.17	3632.66	0:34	128
meta-llama/Llama-3.1-8B-Instruct	28.17	9925.17	3605.77	0:35	128
microsoft/Phi-3-medium-128k-instruct	13.96	5384.64	1786.9	1:11	128
microsoft/Phi-3.5-MoE-instruct	18.36	7080.21	2349.58	0:54	128
mistralai/Mistral-Nemo-Instruct-2407	19.61	7102.14	2509.71	0:50	128
Nexusflow/Athene-V2-Chat	4.49	1604.69	574.81	3:42	512
Qwen/Qwen2.5-1.5B-Instruct	55.5	19831.01	7103.63	0:17	512
Qwen/Qwen2.5-14B-Instruct-AWQ	17.13	6122.48	2193.12	0:57	128
Qwen/Qwen2.5-14B-Instruct-AWQ	6.23	4620.94	3191.44	2:39	512
Qwen/Qwen2.5-32B-Instruct-AWQ	4.5	3332.53	2301.6	3:41	512
Qwen/Qwen2.5-32B-Instruct-GPTQ-Int4	4.51	3345.93	2310.86	3:41	512
Qwen/Qwen2.5-72B-Instruct	1.24	918.42	634.31	13:26	512
Qwen/Qwen2.5-72B-Instruct-AWQ	2.64	1953.93	1349.48	6:18	512
TechxGenus/Mistral-Large-Instruct-2407-AWQ	1.68	1285.95	861.22	9:54	512
TechxGenus/Mistral-Large-Instruct-2411-AWQ	1.68	1286.33	861.48	9:53	512

Analisi dei risultati

I dati raccolti rivelano notevoli differenze tra i modelli in termini di efficienza computazionale e capacità di rispondere a richieste in modo rapido e scalabile. Di seguito, una sintesi dei risultati più rilevanti:

Il modello **Qwen/Qwen2.5-1.5B-Instruct** si è distinto per il throughput più alto, raggiungendo **55.5 richieste al secondo** con un totale di **19,831 token elaborati al secondo**. Questo risultato dimostra che modelli di dimensioni contenute possono eccellere in scenari che richiedono velocità e scalabilità elevate.
Anche i modelli della famiglia **Meta-Llama-3.1-8B** hanno mostrato performance molto competitive, con un throughput di **28.38 richieste al secondo** e tempi di elaborazione significativamente ridotti (34-35 secondi per l'intero dataset).

Modelli come **google/gemma-2b** e **google/gemma-2-9b-it** hanno dimostrato un eccellente bilanciamento tra throughput e velocità di generazione dei token, elaborando rispettivamente **10,734** e **10,811 token al secondo** e producendo circa **7,300 token di output al secondo**. Questi modelli si sono rivelati particolarmente adatti per scenari che richiedono sia una buona qualità delle risposte (con domande semplici) sia un'elevata efficienza computazionale.

Modelli di grandi dimensioni come **Qwen2.5-72B-Instruct** e **Llama-3.3-70B-Instruct-AWQ** hanno mostrato una chiara penalizzazione in termini di throughput e tempi di completamento. Ad esempio, **Qwen2.5-72B-Instruct** ha completato appena **1.24 richieste al secondo**, con un tempo di elaborazione di oltre 13 minuti per il dataset completo. Questi risultati evidenziano le sfide legate alla scalabilità dei modelli più grandi, soprattutto in contesti multiutente.

Modelli ottimizzati tramite tecniche come **AWQ** e **INT4**, come **Meta-Llama-3.3-70B-Instruct-AWQ-INT4**, hanno dimostrato un miglioramento nell'efficienza rispetto alle loro versioni non ottimizzate. Ad esempio, la versione **AWQ-INT4** ha elaborato più richieste e token rispetto alla versione standard, con una riduzione minima nella qualità delle risposte.

Osservazioni e implicazioni pratiche

I risultati dei benchmark evidenziano che la scelta del modello ottimale dipende fortemente dal caso d'uso e dai requisiti di scalabilità. I modelli più piccoli e ottimizzati, come **Qwen2.5-1.5B-Instruct** e **Meta-Llama-3.1-8B**, sono ideali per scenari che richiedono throughput elevato e tempi di risposta rapidi. Al contrario, modelli di dimensioni maggiori, come **Qwen2.5-72B-Instruct**, offrono maggiore capacità di comprensione e accuratezza, ma con costi computazionali significativamente più elevati.

Infine, l'uso di tecniche di ottimizzazione come AWQ e INT4 può rappresentare un compromesso efficace per migliorare l'efficienza senza sacrificare significativamente la qualità delle risposte. Questo approccio potrebbe essere particolarmente utile per modelli di grandi dimensioni, garantendo una migliore scalabilità in ambienti produttivi.

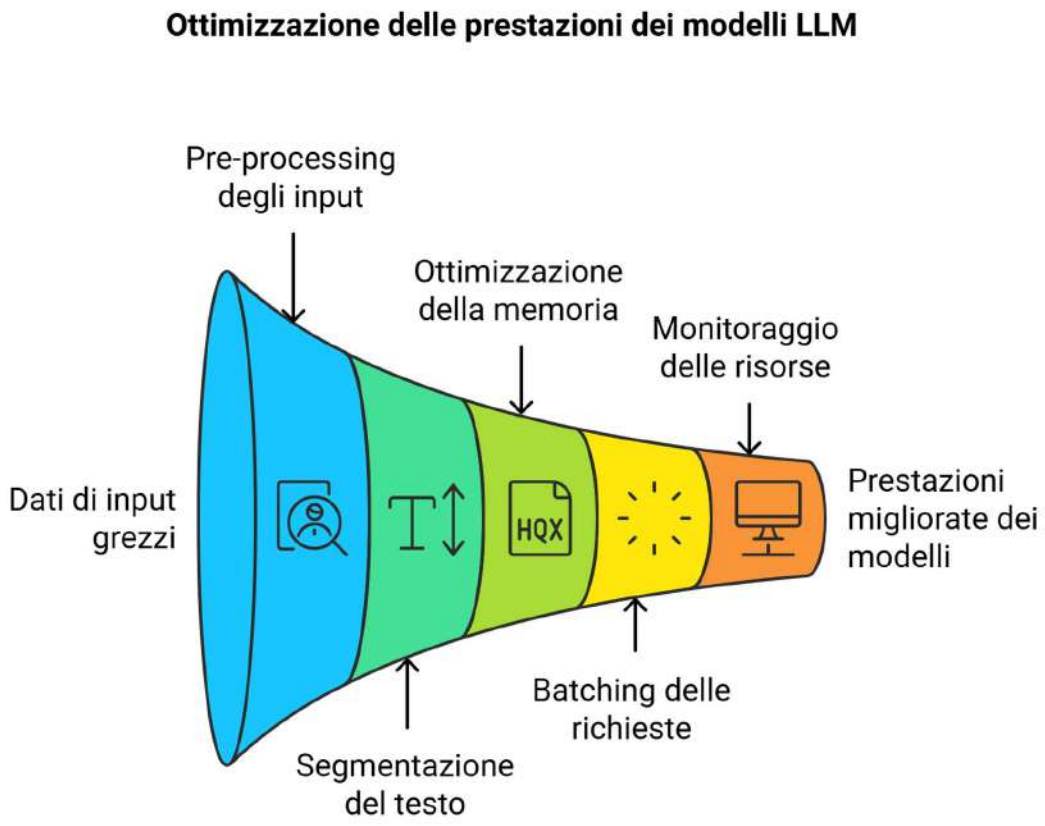
8. Conclusioni

Soluzioni per migliorare le performance

Per migliorare le prestazioni nell'utilizzo dei modelli LLM locali, è necessario intervenire sia a livello software che hardware. Uno degli interventi più efficaci è il pre-processing degli input. Quando i modelli vengono utilizzati per analizzare grandi quantità di testo, come articoli o interi blog, diventa essenziale estrarre preventivamente solo le informazioni rilevanti. Ad esempio, si possono rimuovere metadati, intestazioni e contenuti irrilevanti, riducendo così il volume dei dati elaborati senza sacrificare la qualità delle risposte. Inoltre, la segmentazione dei testi più lunghi in blocchi gestibili consente di ridurre la pressione sulle GPU, migliorando la coerenza dei risultati.

Dal punto di vista della pipeline di inferenza, l'adozione di tecniche come la quantizzazione del modello può ridurre significativamente il consumo di memoria, passando da rappresentazioni in FP32 a INT8. Questo tipo di ottimizzazione comporta un minore utilizzo delle risorse senza un impatto evidente sull'accuratezza delle risposte. Un ulteriore miglioramento può essere ottenuto implementando il batching delle richieste, che permette di processare più input contemporaneamente, sfruttando al massimo le capacità parallele delle GPU.

La gestione e il monitoraggio delle risorse sono altri aspetti critici. Strumenti avanzati possono fornire una visione in tempo reale dell'utilizzo di GPU, CPU e RAM, aiutando a identificare i colli di bottiglia e a distribuire dinamicamente il carico. Inoltre, un'infrastruttura hardware aggiornata, con GPU di ultima generazione e soluzioni di storage NVMe ad alte prestazioni, può ridurre ulteriormente i tempi di caricamento dei modelli e migliorare l'efficienza generale del sistema.



Raccomandazioni per la scalabilità

Per supportare un aumento del carico e garantire una crescita sostenibile del sistema, è fondamentale adottare strategie mirate. Una configurazione distribuita rappresenta una soluzione efficace, in cui i modelli sono distribuiti su più macchine, bilanciando il carico tra GPU diverse. Questo approccio consente di scalare orizzontalmente e di gestire un numero crescente di utenti concorrenti.

Un'altra tecnica utile è l'implementazione di un sistema di caching per richieste ricorrenti, tipiche dell'interazione di un chatbot, essendo il sottosistema LLM stateless. Vllm offre questa funzionalità chiamata prefix-caching, che permette di velocizzare l'elaborazione dell'input nel caso di richieste consecutive sull'endpoint /chat/completions.

Questo permette di evitare di ricalcolare risposte a conversazioni progressive, riducendo il carico computazionale e migliorando i tempi di risposta. Parallelamente, può essere vantaggioso integrare modelli più leggeri per gestire richieste semplici, riservando l'uso dei modelli più avanzati per scenari complessi o critici.

Sintesi dei risultati principali

I risultati dell'analisi mostrano che i modelli di linguaggio di grandi dimensioni (LLM) offrono un'ampia gamma di prestazioni, ma il loro utilizzo in ambienti produttivi richiede un'attenta valutazione delle risorse necessarie e dei vantaggi operativi. Modelli di grandi dimensioni, come **Qwen 2.5-72B-Instruct**, si distinguono per la capacità di generare risposte dettagliate e altamente contestuali, ma richiedono infrastrutture hardware di elevata potenza, con GPU di fascia alta e ampie capacità di memoria. Al contrario, modelli più piccoli e ottimizzati, come **Qwen 2.5-1.5B-Instruct**, hanno dimostrato di essere altamente scalabili in termini di throughput e velocità, pur sacrificando in parte la complessità delle risposte.

Un punto critico emerso riguarda la capacità di gestire elevati carichi simultanei. Modelli più grandi, pur eccellendo in qualità, soffrono in termini di throughput, rendendoli poco adatti a organizzazioni con un numero elevato di utenti concorrenti.

In questo contesto, l'elemento di privacy rimane uno dei principali vantaggi dell'adozione di LLM locali, garantendo che i dati sensibili degli utenti non escano dall'infrastruttura aziendale.

Suggerimenti per l'adozione del modello

L'implementazione di LLM locali richiede una pianificazione precisa, soprattutto per aziende che prevedono di gestire volumi elevati di richieste simultanee. Per ambienti a basso o moderato carico, modelli ottimizzati come **Meta-Llama-3.1-8B** offrono un eccellente compromesso tra qualità e throughput, mantenendo bassi i requisiti hardware e i costi operativi. Tali modelli risultano particolarmente adatti per applicazioni in cui la velocità di risposta è critica, ma la complessità delle richieste è limitata.

D'altra parte, per scenari che richiedono risposte complesse o contesti estesi, modelli come **Qwen 2.5-72B-Instruct** possono essere utilizzati con successo, a condizione che l'infrastruttura hardware sia adeguata e che il volume di richieste simultanee sia contenuto. In questi casi, è essenziale valutare con attenzione il rapporto costo-beneficio, considerando che il risparmio economico derivante dall'adozione di LLM locali rispetto ai servizi cloud può essere compromesso da costi operativi elevati per mantenere tali modelli attivi.

L'ultima considerazione riguarda la necessità di valutare anche la licenza del modello che si desidera adottare in caso di utilizzo per fini commerciali.

Ottimizzazione delle performance

Per migliorare le performance, si consiglia di adottare le seguenti strategie:

- **Filtro a monte del prompt,** in particolare i prompt che scaturiscono dall'esito di function quali web_scraper che possono fornire in input dei contenuti non rilevanti quali tag html, loghi in formato svg, o altri contenuti che richiedono un lungo tempo di elaborazione in input (per un contesto di 128k parliamo anche di decine di secondi), senza aggiungere valore alla risposta ottenibile
- **Ottimizzazioni dei modelli**, come la quantizzazione (INT4 o INT8) per ridurre il consumo di risorse, migliorando l'efficienza e aumentando il parallelismo senza sacrificare in modo significativo la qualità delle risposte.
- **Caricamento di una istanza di modello per ogni GPU disponibile** (quando possibile anche più istanze, fino a saturare la memoria disponibile di ogni singola GPU)
- **Utilizzo di haproxy o di load balancer** equivalente per poter raggiungere più istanze di vllm in parallelo
- **Caching delle richieste**, tramite prefix-caching per ridurre il carico computazionale in caso di interazioni ricorrenti o prevedibili (tipiche del caso dei chatbot)
- **Selezione dinamica dei modelli**, tramite un router che possa assegnare richieste "semplici" a modelli più leggeri e conservi quelli più avanzati per richieste critiche, ad esempio dove è presente il tag json relativo al function calling
- **Effettuare un finetuning di modelli più semplici** con i contenuti specifici del cliente, per poter rispondere in maniera più accurata e aumentare il punteggio delle risposte

Nota: Nel caso degli Agenti di AISuru, a causa del poco tempo a disposizione per effettuare i test, gli agenti testati non erano ottimizzati su gran parte delle strategie indicate, ma sul fronte prompt e function le ottimizzazioni sono funzionalità già previste dalla piattaforma.

Per fare un esempio concreto, adottando un modello che ha un throughput di 10 richieste al secondo, che può essere parallelizzato con due istanze su ognuna delle 4 GPU (8x parallelismo), e immaginando che un utente tipico possa inviare una richiesta ogni 60 secondi (valutando il tempo per leggere la risposta precedente e digitare la successiva), abbiamo:

10 req x 4 gpu x 2 (modelli per gpu) = 80 richieste al secondo

80 richieste al secondo x 60 secondi = 4800 utenti simultanei

La formula interattiva per il calcolo del numero di utenti a partire dalla configurazione e del throughput ottenuto è stata riportata nel Google Spreadsheet nel foglio apposito (Calcolo utenti concorrenti):

https://docs.google.com/spreadsheets/d/1gW9xHn5Gi63_hsf6QtBrmdOrc_klHi0VoSXCJxkOmXE/edit?usp=sharing

Ovviamente il numero di GPU e il numero dei modelli per GPU sono i moltiplicatori più importanti che consentono di scalare per raggiungere gli obiettivi dei clienti.

Appendice e Riferimenti

Calculate: How much GPU Memory you need to serve any LLM?

https://www.substratus.ai/blog/calculating-gpu-memory-for-llm

Optimizing LLMs for Speed and Memory

https://huggingface.co/docs/transformers/v4.35.0/llm_tutorial_optimization

Can you run it? LLM version

https://huggingface.co/spaces/Vokturz/can-it-run-llm

vLLM v0.6.0: 2.7x Throughput Improvement and 5x Latency Reduction

https://blog.vllm.ai/2024/09/05/perf-update.html

Follow the Trail: Supercharging vLLM with OpenTelemetry Distributed Tracing

https://medium.com/@ronen.schaffer/follow-the-trail-supercharging-vllm-with-opentelemetry-distributed-tracing-aa655229b46f

Lenovo Server SR675 v3:

https://lenovopress.lenovo.com/lp1611-thinksystem-sr675-v3-server