ROMA TRE
UNIVERSITÀ DEGLI STUDI

Dipartimento di Ingegneria Civile, Informatica
e delle Tecnologie Aeronautiche

Corso di Laurea in Ingegneria Informatica

Potenziare i chatbot basati su Large Language Models (LLM) attraverso Model Context Protocol (MCP)

Anno Accademico 2024/2025

Laureando

Del Prete Andrea

Matricola 589453

Relatore

Prof. Merialdo Paolo

Tutor

Dott. Di Nardo Giorgio

Ringraziamenti

Vorrei sinceramente esprimere gratitudine ai miei docenti, per la dedizione e la passione con cui mi hanno trasmesso le loro conoscenze, contribuendo in modo fondamentale alla mia crescita accademica; ai colleghi di Proge-Software, per la professionalità e la disponibilità con la quale mi hanno accolto, guidato e supportato durante il mio percorso di tirocinio, permettendomi di mettere in pratica quanto appreso e di acquisire nuove competenze; ai miei amici e colleghi di università, per il costante sostegno, la collaborazione e la condivisione di momenti indimenticabili, che hanno reso questo percorso più ricco e stimolante; ed infine alla mia famiglia, per il supporto incondizionato che mi hanno sempre dimostrato e senza la quale nulla di tutto questo sarebbe stato possibile.

Indice

Indice

Indice

Introduzione

Siamo oramai in un'era in cui l'intelligenza artificiale è presente in ogni aspetto della nostra vita digitale, ancora di più dopo l'uscita di ChatGPT, che infranse il record di applicazione con la più rapida crescita nella storia di Internet, ben 100 milioni di utenti in soli 2 mesi [1]. Gli ultimi anni in particolare sono stati caratterizzati dall'intelligenza artificiale generativa, capace cioè di generare testi, audio, immagini o video a partire da un semplice "prompt" dell'utente, come ad esempio i modelli GPT alla base di ChatGPT stesso.

Di particolare interesse sono le IA capaci di generare testo, i cosiddetti Large Language Model (LLM), letteralmente modelli linguistici di grandi dimensioni, in grado di emulare con sorprendente accuratezza le capacità conversazionali di un essere umano. Un'interessante conseguenza del modo in cui questi algoritmi sono stati creati, ma soprattutto della mole di dati usata per addestrarli, è stato l'emergere di comportamenti che potremmo considerare intelligenti, con un livello quasi pari a quello di un essere umano. Ad esempio, gli LLM sono in grado di ricordare informazioni, generare risposte articolate basate su un ampio contesto e ragionare su problemi complessi derivanti dalle branche più disparate, dalla meccanica quantistica alla biologia evolutiva.

Negli anni è perciò nato il desiderio di rendere ancora più sofisticate le abilità di questi modelli linguistici, finora relegate al "semplice" rispondere ai messaggi degli utenti. Ed è per questo che Anthropic, una delle maggiori aziende nel settore della ricerca sugli LLM e creatrice dei molto diffusi modelli Claude, ha ideato un meccanismo che facilita questo obiettivo: il Model Context Protocol (MCP). Questo

protocollo open-source definisce una procedura standard che consente alle applicazioni di fornire contesto ai modelli linguistici, permettendo loro di accedere a e interagire con una serie di strumenti esterni in modo sicuro e controllato. Grazie a ciò, gli LLM possono eseguire operazioni specifiche, recuperare informazioni o integrare funzionalità aggiuntive, migliorando l'efficacia e la personalizzazione delle risposte generate [2].

Obiettivo di questa tesi è illustrare un progetto volto a dimostrare nella pratica le funzionalità del Model Context Protocol. Il progetto in questione è un sistema di ticket management, che consente agli utenti di interfacciarsi con un chatbot che rende automatica la creazione di ticket delineanti i problemi riscontrarti e l'inoltro degli stessi a degli sviluppatori, che possono visionarli e avviare delle procedure per risolverli.

Large Language Model - Cosa sono e come funzionano

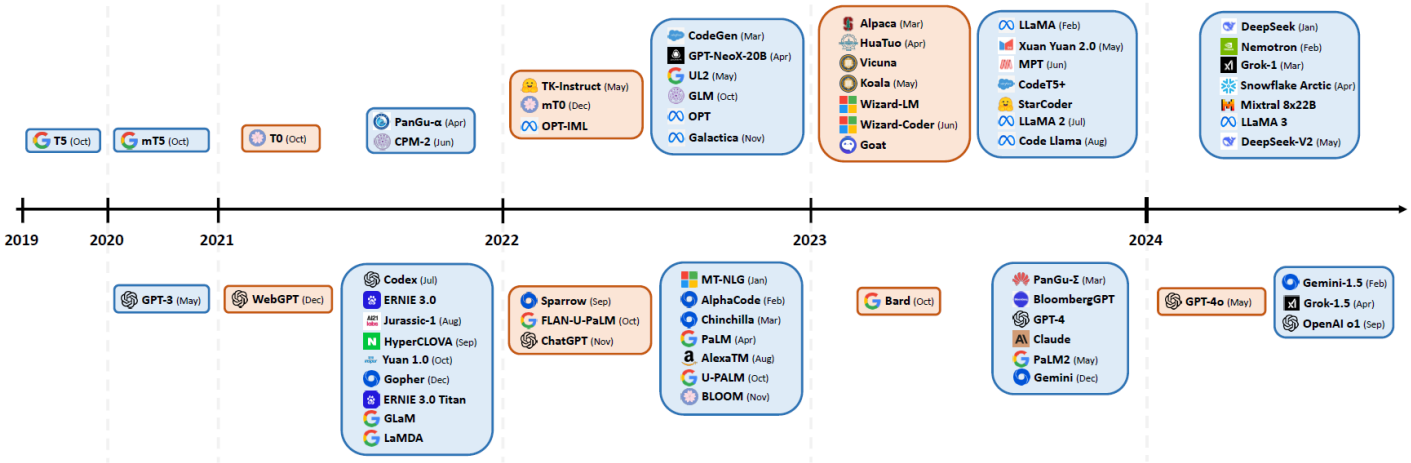

Figura 1: Cronologia dei principali *Large Language Models* (LLM) sviluppati fino al 2024. Sopra la linea ci sono i modelli open-source, mentre sotto ci sono quelli closed-source. [3]

Introduzione

Il linguaggio rappresenta uno strumento fondamentale per gli esseri umani, sia come mezzo di comunicazione e trasmissione di conoscenza, sia come interfaccia per interagire con le macchine. Lo sviluppo di modelli linguistici sempre più potenti è stato quindi al centro della ricerca in *Natural Language Processing* (NLP).

Storicamente, si è passati da approcci statistici (basati su n-grammi e modelli probabilistici), a modelli neurali, con una struttura ispirata a quella del cervello, fino ai modelli pre-addestrati (*Pre-trained Language Models*, PLM). Quest'ultimi, addestrati su grandi corpus testuali, hanno introdotto il concetto di rappresentazioni generali del linguaggio, successivamente adattabili tramite *fine-tuning* a specifici compiti [4][5][6].

Un punto di svolta è stato l'introduzione dei ***Transformer*** [7], che hanno reso possibile scalare i modelli a miliardi di parametri grazie al meccanismo di attenzione, alla parallelizzazione e alla disponibilità di hardware e dataset sempre più estesi. Questo ha portato alla nascita dei **Large Language Models** (LLM).

Funzionamento

Illustriamo ora i concetti fondamentali per comprendere la struttura e il funzionamento dei *Large Language Models* (LLM).

Panoramica

Il testo dell'utente, il cosiddetto *prompt*, viene prima pre-processato e ridotto in *token*, ossia diviso in frammenti più piccoli, i quali vengono poi trasformati in rappresentazioni numeriche che attraversano una rete neurale particolare chiamata Transformer, basata su meccanismi di attenzione, che restituisce un singolo token, che viene poi riconvertito in una parola. Il testo iniziale con l'aggiunta del token generato viene poi dato in input all'LLM per generare un nuovo token e così via. Infine il modello viene addestrato, cioè i suoi parametri vengono strategicamente modificati, per generare risposte sensate e coerenti. Le diverse scelte in ciascuna di queste fasi (tokenizzazione, qualità dei dati, obiettivo di training, architettura) determinano in gran parte le capacità e i limiti del modello.

Tokenizzazione e pre-processing dei dati

La tokenizzazione [8] spezza il testo in unità elementari chiamate token: possono essere singoli caratteri, sottoparole [9], simboli [10] o parole complete. Prima della tokenizzazione si eseguono operazioni di pre-processing: filtraggio della qualità, deduplicazione e rimozione di informazioni sensibili. Un buon pre-processing dei dati riduce inoltre rumore, bias e contenuti non desiderati, migliorando la generalizzazione del modello.

Il Transformer: principio e componenti fondamentali

Il Transformer è l'architettura alla base degli LLM moderni. Si basa su un principio chiamato *self-attention* [11]: nel rappresentare matematicamente i token del prompt,

trasformandoli in oggetti chiamati *embeddings*, si tiene conto non solo della parola isolata ma anche dell'intera frase nella quale si trova, aggiungendo così il contesto.

Ci sono poi diverse tipologie di Transformer:

- ***Encoder-decoder***: due parti separate, una legge l'input (*encoder*) e l'altra genera l'output (*decoder*). Utile per traduzione o riassunti.
- ***Decoder autoregressivo***: genera una parola alla volta guardando solo le parole già prodotte (utilizzato nei modelli GPT).
- ***Mixture-of-Experts (MoE)***: il modello ha molti "esperti" specializzati e per ogni input ne attiva solo alcuni, migliorando così l'efficienza [12][13][14].

Stadi di adattamento: dal pre-training all'uso

Il ciclo di vita di un LLM si articola tipicamente in tre fasi principali:

1. **Pre-training**: apprendimento auto-supervisionato su grandi corpus testuali per acquisire conoscenze linguistiche generali.
2. **Fine-tuning / Alignment**: adattamento a compiti specifici o preferenze umane.
3. **Prompting / Utilizzo**: impiego diretto tramite prompt.

Pre-training degli LLM

Il ***pre-training*** è la fase cruciale che consente ai Large Language Models (LLM) di acquisire conoscenza linguistica generale e capacità di rappresentazione del testo prima di qualsiasi adattamento specifico.

Obiettivi di pre-training

Gli obiettivi di addestramento definiscono cosa il modello apprende:

- ***Full Language Modeling***: al modello viene chiesto di predire i token futuri dati i token precedenti.
- ***Prefix Language Modeling***: un insieme di alcuni token viene scelto casualmente e solo i token rimanenti devono essere previsti dal modello correttamente.
- ***Masked Language Modeling***: i token, o gli span (una sequenza di token), vengono mascherati casualmente e al modello viene chiesto di predire i token mascherati dato il contesto intorno a loro. [15]

Dataset di Addestramento

La qualità e la varietà dei dati di pre-training sono fondamentali. Le principali categorie di dataset includono:

- **Corpus web**: testi raccolti da siti web, ad esempio *Common Crawl*.
- **Enciclopedie e risorse strutturate**, come Wikipedia.
- **Libri e testi lunghi**, ad esempio *BooksCorpus*.
- **Forum e codice**, inclusi dataset come *The Pile*, *Github*, *StackExchange*.

Prima dell'addestramento, i dataset subiscono processi di filtraggio e deduplicazione.

Sfide del Pre-training

L'addestramento degli LLM pone sfide di tipo computazionale e metodologico:

- **Costo computazionale**: i modelli con centinaia di miliardi di parametri richiedono settimane di addestramento su supercomputer con migliaia di GPU.
- **Efficienza dei dati**: la disponibilità di dati di alta qualità è limitata.
- **Rumore e bias**: i dati web contengono rumore, contenuti tossici e bias culturali che vengono appresi dal modello.
- **Controllo della qualità**: è necessario implementare filtri e curare la selezione delle fonti.

Fine-tuning e Adattamento degli LLM

Dopo il pre-training, gli LLM vengono tipicamente sottoposti a una fase di adattamento, che ne affina le capacità su compiti specifici o ne migliora l'allineamento con le intenzioni umane.

Supervised Fine-tuning (SFT)

Consiste nell'addestrare ulteriormente l'LLM su un dataset etichettato per compiti specifici (es. traduzione, classificazione, QA).

- Viene usato per adattare un modello generalista a un dominio ristretto (es. medicina, diritto).
- Può essere effettuato con dataset relativamente piccoli rispetto al pre-training.
- Rischia di causare *catastrophic forgetting*, ossia perdita delle conoscenze generali.

Instruction Tuning

Gli LLM vengono ottimizzati per comprendere e seguire istruzioni in linguaggio naturale.

- Basato su dataset di tipo *prompt-response*, costruiti manualmente o tramite crowdsourcing.
- Introduce la capacità di rispondere in modo coerente a comandi generali senza ulteriori adattamenti.

[16]

Reinforcement Learning from Human Feedback (RLHF)

Il **RLHF** [17] rappresenta una tecnica fondamentale per allineare i modelli alle preferenze umane.

1. Si raccoglie un dataset di risposte generate dal modello valutate da esseri umani.
2. Si addestra un modello di reward che impara a prevedere la valutazione di cui prima.
3. L'LLM viene ulteriormente ottimizzato con tecniche di reinforcement learning.

Questo approccio è stato utilizzato, ad esempio, per ChatGPT.

Domain Adaptation

Gli LLM possono essere specializzati in domini specifici:

- ***Continued Pre-training***: ulteriore addestramento su corpus specializzati.
- ***Fine-tuning* supervisionato su task di dominio**.
- **Approcci ibridi**: combinazioni di pre-training e PEFT (*Parameter-Efficient Fine-Tuning*).

Alignment Tuning

Per garantire sicurezza, utilità e rispetto delle preferenze etiche, si introducono strategie di *alignment*:

- ***Constitutional AI***: uso di principi predefiniti come linee guida per il comportamento del modello.
- ***Self-critique* e revisione**: i modelli generano e valutano autonomamente le proprie risposte. [18][19]

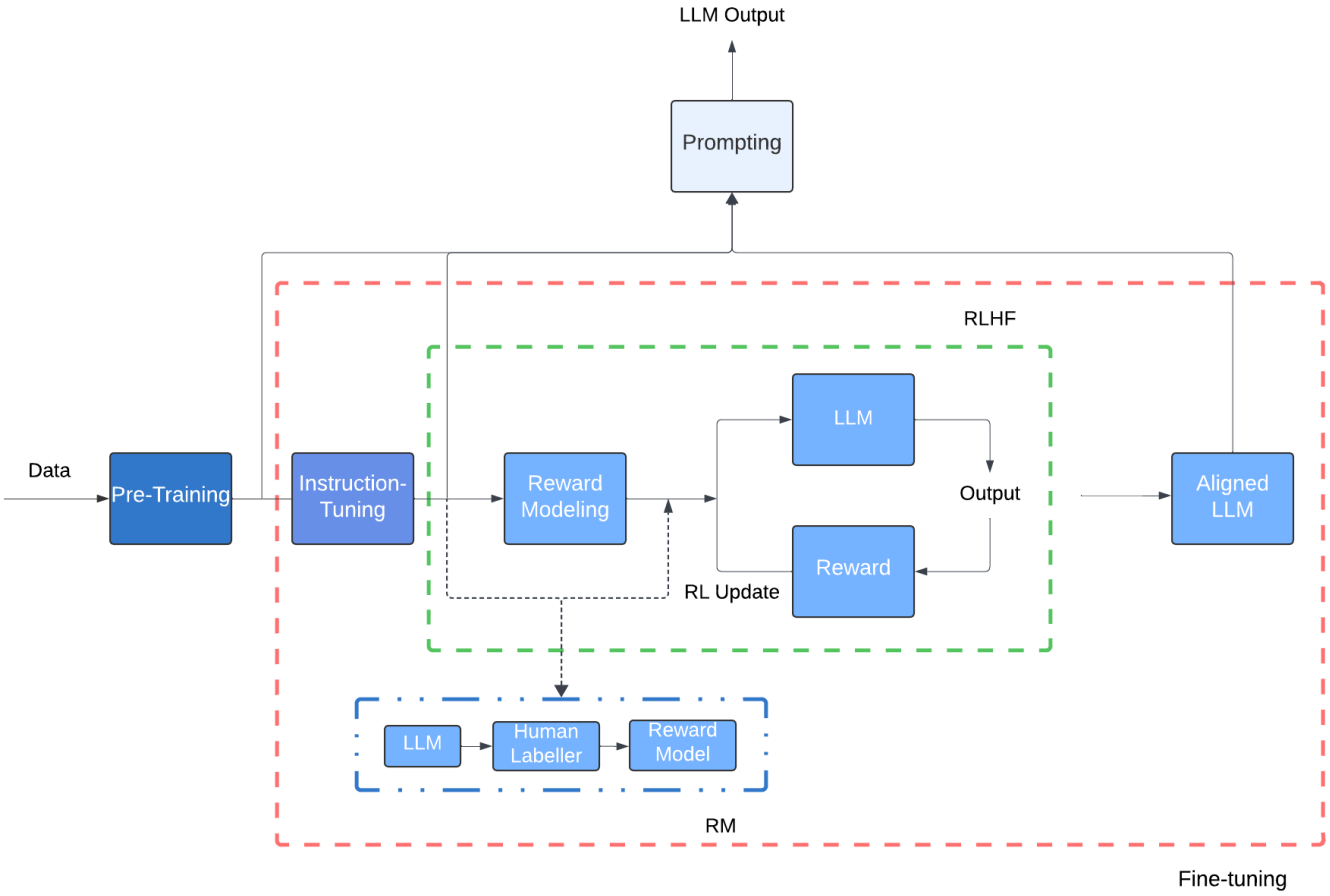
LLM Output
Prompting
RLHF
LLM
Data
Pre-Training
Instruction-Tuning
Reward Modeling
Output
Aligned LLM
Reward
RL Update
LLM
Human Labeller
Reward Model
RM
Fine-tuning

Figura 2: [3]

Tecniche di Prompting e In-Context Learning

Zero-shot Prompting

Nel ***zero-shot prompting*** il modello riceve soltanto una descrizione del compito in linguaggio naturale, senza esempi.

- Dimostra la capacità di generalizzazione appresa durante il pre-training.
- Funziona bene per compiti semplici e largamente rappresentati nei dati di training.
- È la modalità più naturale di interazione con un LLM.

Few-shot Prompting e In-Context Learning (ICL)

Nel **few-shot prompting**, al modello vengono forniti alcuni esempi di input-output all'interno del prompt. Consente al modello di inferire lo schema del compito e adattarsi in tempo reale. Questo comportamento è noto come ***in-context learning*** (ICL) [20].

Prompt Engineering

La formulazione del prompt influenza drasticamente la qualità delle risposte. Sono dunque emerse tecniche sistematiche di **prompt engineering**:

- **Instruction-style prompts**: forniscono istruzioni chiare e contestualizzate.
- **Template-based prompting**: utilizzo di schemi predefiniti per generare input coerenti.
- **Role prompting**: assegnazione di un "ruolo" al modello (es. "Sei un medico con decenni di esperienza…").

Prompting per il Ragionamento

Gli LLM possono essere spinti a mostrare capacità di ragionamento tramite prompt opportuni. Le tecniche principali sono:

- ***Chain-of-Thought* (CoT)**: incoraggia il modello a produrre spiegazioni passo-passo [21].
- ***Self-Consistency***: genera più catene di ragionamento e seleziona la risposta più frequente [22].
- ***Tree-of-Thought* (ToT)**: esplora diversi percorsi di ragionamento, con possibilità di backtracking [23].
- ***Generated Knowledge Prompting***: il modello produce prima conoscenza di supporto, poi la usa per rispondere.

Multi-turn Prompting

Alcuni compiti complessi richiedono interazioni iterative con il modello.

- ***Single-turn prompting***: il modello riceve tutte le informazioni in un unico prompt.
- ***Multi-turn prompting***: l'interazione è suddivisa in più turni, con feedback e aggiustamenti progressivi. Questo approccio è alla base degli **agenti autonomi** basati su LLM.

Tecniche Avanzate di Prompting

Sono state sviluppate ulteriori strategie per aumentare l'affidabilità:

- ***Prompt Chaining***: suddividere un compito complesso in più sotto-compiti concatenati.

- ***Automatic Prompt Generation***: generazione automatica di prompt efficaci tramite ottimizzazione o meta-modelli.
- ***Retrieval-Augmented Prompting***: combinazione con sistemi di recupero documentale per fornire contesto aggiornato e ridurre le allucinazioni.

Applicazioni degli LLM

Gli **Large Language Models** (LLM) hanno trovato applicazioni in un'ampia gamma di domini, dalla generazione di linguaggio naturale alla programmazione, dalla ricerca scientifica alla medicina. Questo capitolo presenta le principali aree di utilizzo e gli scenari emergenti.

Generazione e Comprensione del Linguaggio Naturale

Gli LLM eccellono in compiti tradizionali di NLP:

- **Traduzione automatica**: modelli come GPT, T5 e mBART hanno raggiunto performance vicine ai sistemi dedicati.
- **Riassunto testuale**: generazione di sintesi coerenti e concise da testi lunghi.
- **Parafrasi e riformulazione**: utile per la scrittura assistita e l'elaborazione creativa.
- **Conversazione e chatbot**: applicazioni come *ChatGPT* e Claude forniscono assistenza virtuale multi-dominio.

Programmazione e Ingegneria del Software

Gli LLM specializzati sul codice hanno mostrato grande impatto nello sviluppo software:

- **Code completion**: completamento automatico di funzioni e snippet.
- **Code generation**: traduzione da linguaggio naturale a codice eseguibile.
- **Code explanation e refactoring**: spiegazione e ottimizzazione di codice esistente.
- Strumenti come *Codex*, *Copilot* e *AlphaCode* hanno reso accessibile la programmazione anche a utenti non esperti.

Supporto alla Ricerca e alla Scienza

Gli LLM vengono sempre più utilizzati come strumenti di accelerazione scientifica:

- **Recupero e sintesi della letteratura scientifica**.
- **Generazione di ipotesi**: suggerimenti per esperimenti o linee di ricerca.
- **Analisi dati e interpretazione**: supporto a scienziati in fisica, biologia, chimica.
- **Scoperta di molecole e farmaci**: integrazione con modelli di chimica computazionale.

Medicina e Assistenza Sanitaria

Il settore medico rappresenta un campo sensibile ma promettente:

- **Supporto diagnostico**: analisi di sintomi e cartelle cliniche.
- **Assistenza alla scrittura clinica**: redazione automatica di referti.
- **Educazione medica**: generazione di materiali di studio personalizzati.
- **Chatbot sanitari**: fornire informazioni mediche preliminari ai pazienti.

Sono necessarie forti misure di sicurezza per ridurre rischi di allucinazioni e bias.

Educazione e Apprendimento

Gli LLM possono agire da tutor personalizzati:

- **Creazione di materiali didattici adattivi**.
- **Assistenza individuale agli studenti** con spiegazioni su misura.
- **Valutazione automatizzata** di esercizi e saggi.
- Supporto all'apprendimento delle lingue attraverso dialoghi interattivi.

Business e Produttività

Le applicazioni aziendali sono tra le più immediate:

- **Automazione del customer service** con chatbot multilingue.
- **Generazione di report e analisi** a partire da dati aziendali.
- **Supporto alle decisioni** tramite analisi predittive.
- **Content creation**: marketing, copywriting e produzione di contenuti digitali.

Applicazioni Multimodali

Gli LLM vengono estesi per gestire input multimodali:

- ***Vision-Language Models***, come *CLIP*, *Flamingo*, GPT-4 multimodale.
- **Image captioning e visual question answering**.
- **Audio e speech**: trascrizione, traduzione e sintesi vocale.
- **Robotica**: uso di LLM per interpretare istruzioni linguistiche e guidare azioni fisiche.

Agenti Autonomi e Tool Use

Un campo emergente riguarda l'integrazione degli LLM in agenti autonomi:

- **Tool-augmented LLMs**: modelli che interagiscono con strumenti esterni come motori di ricerca, calcolatrici o database.
- **Agenti multi-step**: in grado di pianificare sequenze di azioni.
- Framework come *LangChain*, *AutoGPT* e *BabyAGI* mostrano le potenzialità degli LLM come agenti generici.

Sfide, Limitazioni ed Aspetti Etici degli LLM

Nonostante i notevoli progressi, gli LLM presentano ancora numerose sfide tecniche, limitazioni pratiche e rischi etici. Questo capitolo esamina i principali problemi legati alla loro adozione e le preoccupazioni sociali connesse al loro utilizzo.

Allucinazioni e Aderenza ai Fatti

Gli LLM spesso generano contenuti plausibili ma falsi, un fenomeno noto come **allucinazioni**. Possono inventare dati, riferimenti bibliografici o fatti storici inesistenti. Ciò riduce l'affidabilità in contesti critici come medicina, diritto o scienza. Tecniche come *retrieval-augmented generation* (RAG) sono state proposte per mitigare questo problema.

Bias e Equità

Gli LLM ereditano i bias presenti nei dati di addestramento.

- Riproducono stereotipi di genere, etnia, religione, orientamento sessuale.
- Possono amplificare discriminazioni già esistenti nella società.

Sono in corso ricerche su dataset bilanciati e metodi di de-biasing.

Sicurezza e Abusi Potenziali

Gli LLM possono essere usati impropriamente per generare contenuti dannosi.

- **Disinformazione**: generazione automatica di fake news e propaganda.
- **Cybersecurity**: creazione di codice malevolo o phishing su larga scala.
- **Deepfakes multimodali**: se integrati con modelli di immagini o audio.

Interpretabilità e Trasparenza

Un limite importante riguarda la natura **black-box** degli LLM.

- È difficile spiegare come vengano prese le decisioni interne al modello.
- La mancanza di interpretabilità riduce la fiducia in applicazioni sensibili.

Sono state proposte tecniche di *model probing*, analisi delle rappresentazioni interne e spiegazioni post-hoc.

Sostenibilità e Impatto Ambientale

Il training di LLM richiede un enorme consumo energetico.

- Addestrare un singolo modello può generare tonnellate di CO_2.
- Questo solleva preoccupazioni ambientali e di sostenibilità.

Approcci di efficientamento (quantizzazione, pruning, distillazione) mirano a ridurre tali costi.

Accessibilità e Disuguaglianze

Gli LLM di punta sono sviluppati principalmente da grandi aziende tecnologiche.

- L'alto costo computazionale limita la ricerca accademica indipendente.
- Si rischia una concentrazione di potere e conoscenza in poche mani.

Iniziative open-source (es. *LLaMA*, *Falcon*, *MPT*) cercano di democratizzare l'accesso.

Allineamento con Valori Umani

Un'altra sfida riguarda l'**alignment** con preferenze etiche e valori umani.

- Gli LLM devono essere utili, onesti e innocui.
- Tecniche come **RLHF** e ***Constitutional AI*** sono state sviluppate per migliorare l'allineamento.
- Rimane complesso garantire coerenza su culture e comunità diverse.

Aspetti Legali e Regolamentazione

Con la diffusione degli LLM emergono questioni legali:

- **Copyright**: rischio di violazioni dovute a testi generati simili a fonti protette.
- **Privacy**: possibilità che vengano riprodotte informazioni sensibili dai dati di training.
- **Regolamentazione**: discussioni in corso a livello internazionale su AI Act e framework etici.

Prospettive Future e Direzioni di Ricerca

Gli LLM rappresentano uno dei campi più dinamici dell'intelligenza artificiale. Dopo i progressi ottenuti negli ultimi anni, la comunità scientifica si interroga su come evolveranno i modelli futuri e quali direzioni di ricerca saranno più rilevanti. Questo capitolo discute le prospettive principali.

Efficienza e Sostenibilità

Una priorità sarà lo sviluppo di modelli più efficienti dal punto di vista computazionale e energetico.

- Tecniche di compressione (quantizzazione, pruning, distillazione) sempre più sofisticate.
- Architetture ibride che combinano Transformer con nuovi meccanismi di attenzione.
- Algoritmi di addestramento ottimizzati per ridurre costi e tempi.

Modelli Multimodali

Gli LLM tenderanno a integrarsi con altre modalità sensoriali:

- **Vision-Language Models**: unione di testo e immagini.
- **Speech e audio**: modelli in grado di comprendere e generare linguaggio parlato.
- **Video e robotica**: comprensione di sequenze visive e comandi linguistici per il controllo di agenti fisici.

L'obiettivo è costruire **modelli universali** in grado di gestire input multimodali.

Memory-Augmented LLMs

Attualmente gli LLM hanno memoria limitata alla finestra contestuale.

- Ricerca su architetture con memoria esterna persistente.
- Meccanismi per richiamare informazioni apprese in sessioni precedenti.
- Possibilità di modelli che accumulano conoscenza nel tempo, riducendo la necessità di retraining massivo.

Autonomia e Agenti Intelligenti

Un filone emergente è l'integrazione degli LLM in agenti autonomi.

- Capacità di pianificazione multi-step tramite prompting avanzato.
- Integrazione con tool esterni: database, motori di ricerca, calcolatrici, API.
- Applicazioni in domini complessi come finanza, ricerca scientifica, sviluppo software.

Interpretabilità e Controllo

La comunità scientifica riconosce la necessità di modelli più trasparenti:

- Strumenti di **explainable AI** per comprendere le decisioni interne.
- Analisi delle rappresentazioni neurali per capire l'emergere delle abilità.
- Tecniche di controllo fine-grained per modulare il comportamento del modello.

Sicurezza e Robustezza

Un'altra direzione riguarda la costruzione di LLM affidabili e sicuri:

- Difese contro attacchi avversari e prompt injection.

- Riduzione delle allucinazioni tramite retrieval e grounding su basi di conoscenza.
- Maggiore robustezza a input rumorosi o manipolati.

Allineamento e Governance

Il tema dell'allineamento con valori umani resterà centrale.

- Evoluzione di tecniche come RLHF, Constitutional AI e feedback multimodale.
- Creazione di framework etici condivisi per la progettazione di modelli sicuri.
- Discussione su regolamentazione internazionale e open science.

Verso l'Intelligenza Artificiale Generale (AGI)

Infine, gli LLM sono considerati possibili passi verso l'AGI.

- Studi sull'emergere di capacità generali di ragionamento e apprendimento.
- Integrazione con sistemi simbolici e neuro-simbolici.
- Possibilità di architetture future che superino i limiti dei Transformer attuali.

Model Context Protocol

Funzionamento

MCP è un protocollo open-source che standardizza il modo in cui le applicazioni forniscono contesto agli LLM. Si può pensare a MCP come a una porta USB-C per le applicazioni IA. Proprio come USB-C fornisce un modo standardizzato per collegare i dispositivi a varie periferiche e accessori, MCP fornisce un modo standardizzato per collegare i modelli IA a diverse fonti di dati e strumenti. MCP consente di creare agenti e flussi di lavoro complessi basati sugli LLM e connette i modelli con il mondo. [24]

Il protocollo MCP include i seguenti progetti:

- Specifiche MCP: una specifica di MCP che delinea i requisiti di implementazione per client e server.
- SDK MCP: *Software Development Kit* (SDK) per diversi linguaggi di programmazione che implementano MCP.
- Strumenti di sviluppo MCP.
- Implementazioni di server MCP di riferimento.

MCP si concentra esclusivamente sul protocollo per lo scambio di contesto, senza stabilire come le applicazioni IA utilizzino gli LLM o gestiscano il contesto fornito. [25]

Partecipanti

MCP segue un'architettura client-server in cui un host MCP, un'applicazione IA come Claude Code o Claude Desktop, stabilisce connessioni ad uno o più server MCP. L'host MCP realizza questo creando un client MCP per ogni server MCP. Ogni client MCP mantiene una connessione uno-a-uno dedicata con il suo server MCP corrispondente. I principali partecipanti all'architettura MCP sono:

- Host MCP: l'applicazione IA che coordina e gestisce uno o più client MCP.
- Client MCP: un componente che mantiene una connessione a un server MCP ed ottiene, da un server MCP, il contesto che l'host MCP può utilizzare.
- Server MCP: un programma che fornisce contesto ai client MCP; possono essere eseguiti sia in locale che in remoto. [25]

Livelli

MCP è costituito da due livelli:

- Livello dati: definisce il protocollo, basato su JSON-RPC, per la comunicazione client-server, inclusa la gestione del ciclo di vita e le primitive principali, come strumenti, risorse, prompt e notifiche.
- Livello trasporto: definisce i meccanismi e i canali di comunicazione che consentono lo scambio di dati tra client e server, inclusi l'instaurazione di connessioni specifiche per il trasporto, il framing dei messaggi e l'autorizzazione.

Concettualmente, il livello dati è il livello interno, mentre il livello trasporto è il livello esterno. [25]

Livello dati

Il livello dati implementa un protocollo di scambio basato su JSON-RPC 2.0 che definisce la struttura e la semantica dei messaggi. Questo livello include:

- Gestione del ciclo di vita: gestisce l'inizializzazione della connessione, la negoziazione delle capacità e la terminazione della connessione tra client e server.
- Funzionalità del server: consente ai server di fornire funzionalità di base, inclusi strumenti per azioni IA, risorse per dati di contesto e richieste per schemi di interazione da e verso il client.
- Funzionalità del client: consente ai server di chiedere al client di campionare dall'LLM, ottenere input dall'utente e registrare messaggi al client.
- Funzionalità di utilità: supporta funzionalità aggiuntive come notifiche per aggiornamenti in tempo reale e monitoraggio dei progressi per operazioni di lunga durata. [25]

Livello di trasporto

Il livello di trasporto gestisce i canali di comunicazione e l'autenticazione tra client e server. Gestisce la creazione della connessione, il framing dei messaggi e la comunicazione sicura tra i partecipanti. MCP supporta due meccanismi di trasporto:

- *Stdio Transport*: utilizza flussi di input/output standard per la comunicazione diretta tra processi locali sulla stessa macchina, garantendo prestazioni ottimali senza sovraccarico di rete.
- *Streamable HTTP transport*: utilizza metodi HTTP POST per i messaggi client-server con eventi inviati dal server opzionali per le funzionalità di streaming. Questo trasporto consente la comunicazione con il server remoto e supporta metodi di autenticazione HTTP standard, inclusi *bearer token*, chiavi API e intestazioni personalizzate.

MCP consiglia di utilizzare OAuth per ottenere i token di autenticazione. Il livello di trasporto astrae i dettagli di comunicazione dal livello di protocollo, consentendo lo stesso formato di messaggio del protocollo JSON-RPC 2.0 su tutti i meccanismi di trasporto. [25]

Protocollo del livello dati

Una parte fondamentale di MCP è la definizione dello schema e della semantica tra client e server MCP. Il livello dati è la parte di MCP che definisce le modalità con cui gli sviluppatori possono condividere il contesto dai server MCP ai client MCP. MCP utilizza JSON-RPC 2.0 come protocollo *Remote Procedure Call* (RPC). Client e server si inviano richieste e rispondono di conseguenza. Le notifiche possono essere utilizzate quando non è richiesta alcuna risposta. [25]

Primitive

Le primitive MCP sono il concetto più importante all'interno di MCP. Definiscono ciò che client e server possono offrirsi reciprocamente. Queste primitive specificano i tipi di informazioni contestuali che possono essere condivise con le applicazioni IA e la gamma di azioni che possono essere eseguite. MCP definisce tre primitive principali che i server possono esporre:

- *Tools*: funzioni eseguibili che le applicazioni IA possono invocare per eseguire azioni (ad es.: operazioni su file, chiamate API, query di database).
- Risorse: fonti di dati che forniscono informazioni contestuali alle applicazioni IA (ad es.: contenuto di file, record di database, risposte API).
- Prompt: schemi riutilizzabili che aiutano a strutturare le interazioni con i modelli linguistici (ad es.: prompt di sistema, prompt *few-shot*).

Ogni tipo di primitiva ha metodi associati per la scoperta (`*/list`), il recupero (`*/get`) e, in alcuni casi, l'esecuzione (`tools/call`). I client MCP utilizzeranno i metodi `*/list` per scoprire le primitive disponibili. Ad esempio, un client può prima elencare tutti gli strumenti disponibili (`tools/list`) e poi eseguirli. Questa progettazione consente di creare elenchi dinamici.
Come esempio concreto, si consideri un server MCP che fornisce contesto su un

database. Può esporre strumenti per interrogare il database, una risorsa che contiene lo schema del database e un prompt che include esempi di interazione con gli strumenti.
MCP definisce anche le primitive che i client possono esporre. Queste primitive consentono agli autori del server MCP di creare interazioni più ricche.

- Campionamento: consente ai server di richiedere il completamento del modello linguistico dall'applicazione IA del client. Questa funzionalità è utile quando gli autori del server desiderano accedere a un modello linguistico, ma vogliono rimanere indipendenti dal modello e non includere un SDK del modello linguistico nel proprio server MCP. Possono utilizzare il metodo `sampling/complete` per richiedere il completamento del modello linguistico dall'applicazione IA del client.

- Elicitazione: consente ai server di richiedere informazioni aggiuntive agli utenti. Questa funzionalità è utile quando gli autori del server desiderano ottenere maggiori informazioni dall'utente o chiedere la conferma di un'azione. Possono utilizzare il metodo `elicitation/request` per richiedere informazioni aggiuntive all'utente.

- Logging: consente ai server di inviare messaggi di log ai client a scopo di debug e monitoraggio. [25]

Notifiche

Il protocollo supporta notifiche in tempo reale per abilitare aggiornamenti dinamici tra server e client. Ad esempio, quando gli strumenti disponibili su un server cambiano, come quando vengono rese disponibili nuove funzionalità o vengono modificati strumenti esistenti, il server può inviare notifiche di aggiornamento per informare i client connessi di tali modifiche. Le notifiche vengono inviate come messaggi di notifica JSON-RPC 2.0 (senza attendere una risposta) e consentono ai server MCP di fornire aggiornamenti in tempo reale ai client connessi. [25]

Ticket Management System

Il *Ticket Management System* è un'applicazione software che ha lo scopo di aiutare gli utenti nella creazione e gestione di ticket, creati per segnalare la presenza di problemi individuati, e nel loro inoltro a degli sviluppatori che possano analizzare e poi risolvere tali problemi. Funzionalità centrale dell'applicazione è il fatto che l'utente si interfaccia con un chatbot, un'istanza della risorsa Azure OpenAI che usa il modello GPT-4.1 nello specifico, per usufruire dei servizi offerti.

Il *Ticket Management System* è stato progettato come una *solution* in C# suddivisa, secondo un approccio modulare, in più progetti indipendenti ma interconnessi. Tale organizzazione riflette i principi delle moderne architetture software a livelli, secondo il principio *Separation of Concerns*, favorendo la manutenibilità e la possibilità di estendere il sistema senza introdurre dipendenze circolari o accoppiamenti eccessivamente rigidi.

La composizione della solution prevede cinque progetti principali, ciascuno con un ruolo ben definito: *TM.Shared*, *TM.Data*, *TM.CQRS*, *TM.Function* e *TM.Client*.

Progetti nella solution

TM.Shared

Il progetto *TM.Shared* definisce tutte le classi degli oggetti che vengono creati internamente nel codice a seguito dell'interazione con l'utente. Tali classi sono:

- `Ticket`: elemento centrale del sistema; rappresenta i ticket creati dagli utenti, con i relativi `Comment`, le `Task` legate, il `TicketStatus` e la `TicketPriority`.
- `Category`: ogni `Ticket` appartiene a una categoria.
- `Comment`: i commenti associati a un determinato `Ticket`.
- `Task`: i task sono le operazioni affidate agli sviluppatori, generati dal sistema a seguito della creazione di un `Ticket`.
- `User`: gli utenti che usano il sistema; sono intesi anche gli sviluppatori che devono risolvere le `Task`.

Oltre alle classi però, il progetto definisce per ognuna dei DTO (*Data Transfer Objects*) [26]. I DTO sono classi compatte che assomigliano alle classi originali ma che contengono meno informazioni, solo quelle ritenute necessarie. L'obiettivo è migliorare le performance legate al trasferimento dati tra i diversi sottosistemi, oltre a nascondere alcune logiche interne delle varie classi.

TM.Data

Il progetto *TM.Data* ha il compito di gestire la persistenza delle informazioni, fornendo l'accesso al database ed archiviando gli oggetti creati. Il database è archiviato all'interno dell'ambiente cloud *Azure* ed utilizza il linguaggio SQL.

TM.CQRS

Il progetto *TM.CQRS* introduce l'implementazione del pattern *Command Query Responsibility Segregation* [27]. Il CQRS è un pattern architetturale che ha lo scopo di isolare le operazioni di lettura (`Queries`) e di scrittura (`Commands`) in un progetto separato, che possono poi essere richiamate da ognuno degli altri progetti per effetturare query sul database.

TM.Function

Il cuore della logica applicativa è rappresentato da *TM.Function*. Essa è un'*Azure Function*, ovvero una soluzione serverless che consente agli sviluppatori di scrivere e eseguire codice sul cloud in maniera più efficace e compatta, senza dover ricorrere alla creazione di un progetto [28]. Prende il ruolo dell'MCP Server, l'applicazione IA che gestisce la logica principale del Model Context Protocol. Verrà discusso più approfonditamente in seguito.

TM.Client

Infine, *TM.Client* costituisce l'interfaccia utente, nonchè l'MCP Client. Esso contiene la sola classe `Program.cs`, occupandosi di avviare l'intera solution ed integrare i vari altri progetti, oltre che inizializzare la comunicazione con il chatbot tramite terminale.

L'MCP in pratica

Nucleo centrale del progetto è la classe `TicketManagementTools` del progetto TM.Function. Il suo obiettivo è quello di definire una serie di *tools* che l'LLM invoca per eseguire le operazioni richieste. Vediamo un esempio:

```
[Function("AggiungiTicket")]
public async Task<TicketDtoCreate?> AggiungiTicketAsync(
    [McpToolTrigger("AggiungiTicket", "Aggiunge un ticket")] ToolInvocationContext context,
    [McpToolProperty("Ticket", "string", DtoJsonSchemas.TicketDtoCreateSchema)] string ticketDto)
{
    var options = new JsonSerializerOptions
    {
        PropertyNameCaseInsensitive = true,
        Converters = { new JsonStringEnumConverter() }
    };

    var dto = JsonSerializer.Deserialize<TicketDtoCreate>(ticketDto, options);
    if (dto == null || !dto.IsValid())
    {
        return null;
    }

    var entity = dto.ToEntity();
    await WriteRepo<Ticket>().AddAsync(entity);
    await WriteRepo<Ticket>().SaveChangesAsync();

    return dto;
}
```

Questo metodo, come si può facilmente dedurre, permette di creare un oggetto `Ticket` (più precisamente un DTO `TicketDtoCreate`). Caratteristica del Model Context Protocol sono le sue decorazioni: `McpToolProperty` e `McpToolTrigger`. Curiosamente, queste decorazioni esplicitano i propri dettagli operativi in linguaggio naturale e non in un linguaggio di programmazione, C# in questo caso. Nonostante ciò, l'LLM è in grado di interpretare alla lettera queste istruzioni e utilizzare correttamente i tool a sua disposizione.

Da notare l'uso di JSON all'interno del metodo. Per comunicare al chatbot la struttura degli oggetti da creare, gli vengono passati degli schemi JSON ai quali l'LLM dev'essere conforme. Nel frammento mostrato, lo schema è fornito tramite `DtoJsonSchemas.TicketDtoCreateSchema` nell'attributo `McpToolProperty`; questo schema definisce la forma del JSON atteso e permette di vincolare l'output dell'LLM alla struttura prevista.

Il parametro `ToolInvocationContext` nell'esempio fornisce, come si vede dalla firma, il contesto dell'invocazione dello tool: in pratica rappresenta il punto di integrazione tra la chiamata esterna (o generata dall'LLM) e l'implementazione della Function stessa. La presenza di tale contesto e delle specifiche in linguaggio naturale nelle decorazioni facilita la correlazione tra le istruzioni fornite al modello e le operazioni concrete eseguite dall'applicazione.

Questi elementi insieme mostrano come il Model Context Protocol consenta di esporre funzioni applicative al modello in modo strutturato: le descrizioni in linguaggio naturale rendono comprensibili i comportamenti dei tool all'LLM, mentre gli schemi JSON e la logica di validazione garantiscono che i dati prodotti rispettino le aspettative del dominio applicativo.

Verifica pratica

Verifichiamo ora se il sistema effettivamente funziona e se l'LLM è in grado di eseguire le operazioni che gli vengono chieste. Per farlo, effettueremo un esempio: gli chiederemo di creare un ticket. Segue una trascrizione dell'interazione avuta con il chatbot.

```
Prompt:  vorrei creare un ticket
Per creare un ticket ho bisogno di alcune informazioni:

1.  Titolo del ticket
2.  Descrizione del problema o richiesta
```

```
3.  Priorità (Low, Medium, High)
4.  Il tuo ID utente (o i tuoi dati se non sei ancora registrato)
5.  Categoria del ticket (selezionabile da elenco categorie
disponibili)

Puoi fornirmi queste informazioni?  Se non conosci la categoria o
il tuo ID utente, posso aiutarti a recuperarli.
Prompt:  il titolo è "Problema nell'autenticazione"
Ottimo, hai scelto il titolo "Problema nell'autenticazione".

Ora ho bisogno delle seguenti informazioni per completare la
creazione del ticket:

1.  Descrizione del problema
2.  Priorità (Low, Medium, High)
3.  Il tuo ID utente (oppure il tuo username/email/nome e cognome
per identificarti)
4.  Categoria del ticket (se non la conosci, posso mostrarti la
lista delle categorie disponibili)

Per favore, forniscimi questi dettagli.
Prompt:  la descrizione è "Ho problemi nell'autenticazione.  Non
riesco ad inserire username e password nel form."
Perfetto, ho annotato la descrizione:

"Ho problemi nell'autenticazione.  Non riesco ad inserire username e
password nel form."

Mi servono ancora alcune informazioni:

1.  Priorità del ticket (Low, Medium, High)
2.  Il tuo ID utente (oppure username/email/nome e cognome, se non
conosci l'ID)
3.  Categoria del ticket (se non la conosci posso elencare le
categorie disponibili)
```

```
Puoi fornirmi questi dettagli?  Vuoi che ti mostri le categorie
disponibili?
Prompt:  priorità alta
Le categorie disponibili sono:

1.  Hardware:  Categoria relativa a hardware e dispositivi fisici
2.  Software Development:  Category for software development related
tasks
3.  Networking:  Categoria relativa alle reti di computer
4.  Software Testing:  Categoria per attività di testing software

Quale categoria vuoi assegnare al ticket?  Inoltre, puoi fornirmi
il tuo ID utente o, se non lo conosci, il tuo username/email/nome e
cognome per identificarti?
Prompt:  vorrei creare una nuova categoria
Per creare una nuova categoria, ho bisogno di:

1.  Nome della categoria
2.  Descrizione della categoria (opzionale, ma consigliata)

Per favore, forniscimi questi dettagli.
Prompt:  "Problemi di autenticazione - Problemi legati
all'autenticazione nel sistema"
La nuova categoria "Problemi di autenticazione" è stata creata con
successo.

Ora, per completare il ticket, puoi fornirmi il tuo ID utente
oppure il tuo username/email/nome e cognome?  In questo modo potrò
associare il ticket al tuo profilo.
Prompt:  l'username è adelprete
Il ticket "Problema nell'autenticazione" con priorità alta è stato
creato nella categoria "Problemi di autenticazione" ed è stato
associato al tuo profilo (username:  adelprete).
```

```
Se hai bisogno di aggiungere commenti, task, o apportare modifiche
al ticket, fammelo sapere!
```

Come si evince dalla conversazione, l'operazione è stata un successo: l'LLM ha correttamente compreso quali sono i parametri dell'oggetto Ticket e ce li ha chiesti in ordine. Ha inoltre reperito l'informazione delle categorie già esistenti nel database. Come prova definitiva, accediamo al database sulla piattaforma Azure e osserviamo il contenuto della tabella *dbo.Tickets* prima e dopo la conversazione di cui sopra.

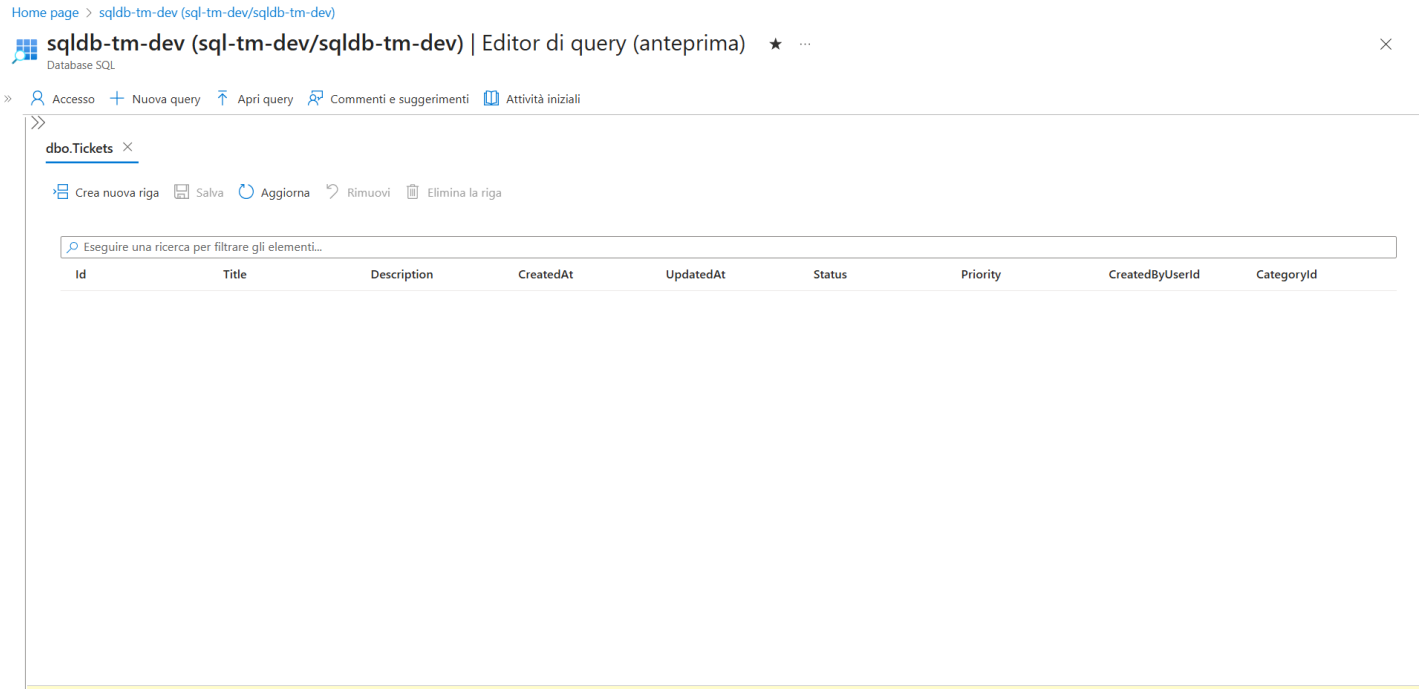

Figura 3: Prima dell'operazione. Non ci sono ancora ticket.

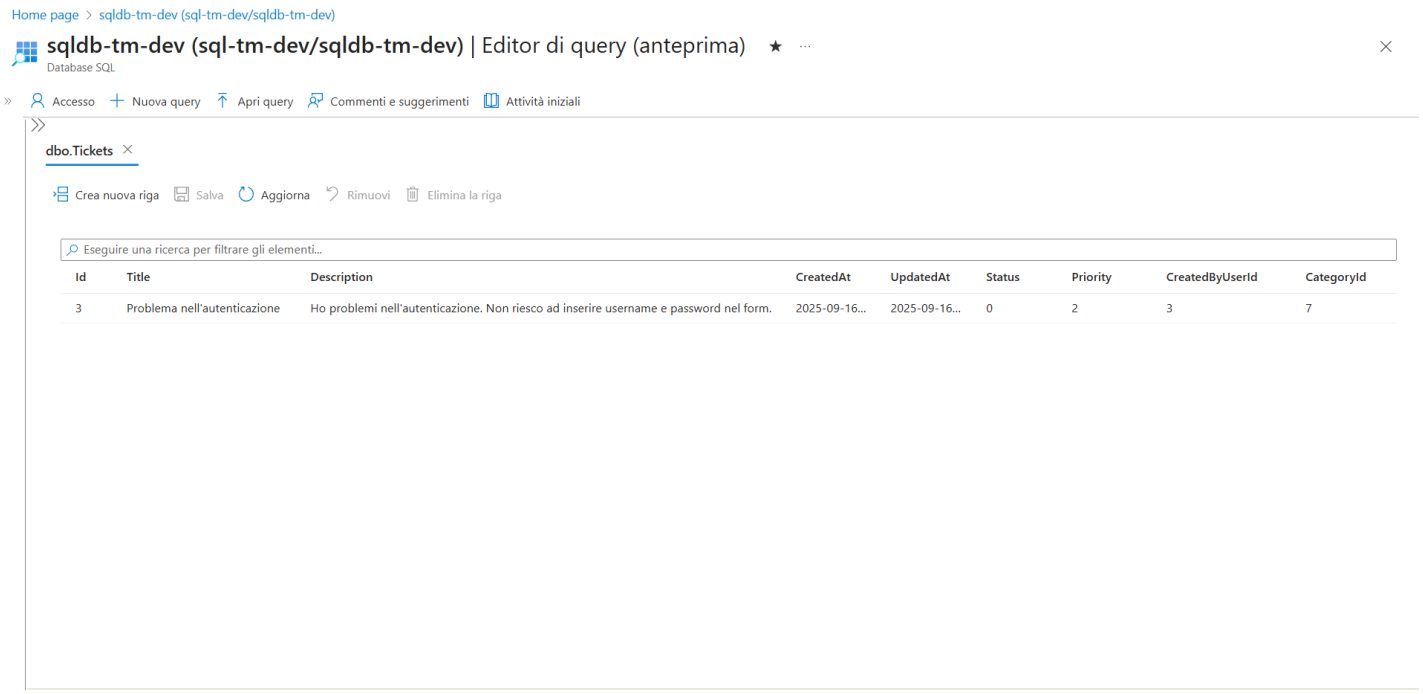

Figura 4: Dopo l'operazione. Il ticket è stato aggiunto.
Nota: le due date e lo status sono stati generati automaticamente.

Conclusioni e sviluppi futuri

Il *Ticket Management System* è pensato per essere posizionato all'interno di un contesto più grande, ossia all'interno di un sistema software per il quale un utente può avere necessità di un supporto tecnico. Il sistema che verrà preso in esame è 3FS©.

3FS© - Panoramica del funzionamento

Il sistema 3FS© (*Fleet and Financial Flexible System*) rappresenta una soluzione software avanzata per la gestione integrata delle flotte aziendali, concepita per rispondere in maniera automatizzata e completa ad un'ampia gamma di esigenze operative, commerciali, contabili, finanziarie e normative.

L'obiettivo primario della piattaforma è quello di ottimizzare l'intero ciclo di vita dei veicoli, dalla fase di acquisizione fino alla dismissione, garantendo al contempo un equilibrio tra minimizzazione dei costi di mantenimento e massimizzazione dei ricavi derivanti dal noleggio. La natura modulare del sistema consente alle imprese di adottare un modello "pay-per-use", sostenendo esclusivamente i costi relativi alle funzionalità effettivamente utilizzate, il che ne amplifica la flessibilità e la scalabilità.

Un aspetto rilevante di 3FS© è la gestione automatizzata degli accordi e della rotazione del parco veicoli. Attraverso strumenti di analisi predittiva, il sistema è in grado di stimare i prezzi di acquisto dei veicoli, tenendo conto di sconti e condizioni contrattuali, e di generare automaticamente la configurazione ottimale della flotta operativa. Questo approccio consente non solo di migliorare il potere negoziale con

i costruttori automobilistici, ma anche di predisporre processi di rotazione basati su criteri di riduzione dei costi o incremento della redditività.

La piattaforma integra inoltre funzionalità avanzate per la distribuzione e messa in servizio dei veicoli, assicurando che le consegne avvengano in coerenza con i target mensili della flotta operativa. Ogni fase della logistica, dalla pre-ispezione fino alla distribuzione presso le stazioni di noleggio finale, è monitorata e tracciata, riducendo i rischi di non conformità. A livello normativo e assicurativo, il sistema automatizza adempimenti quali registrazione delle targhe, pagamento delle tasse di possesso e gestione delle coperture assicurative, rendendo i veicoli disponibili al noleggio in tempi rapidi.

Durante il ciclo di vita dei veicoli, 3FS© gestisce in maniera integrata attività quali manutenzione programmata, rinnovi assicurativi, pagamento delle tasse automobilistiche e risoluzione di danni e sinistri. Una caratteristica distintiva è l'interazione diretta con le autorità pubbliche per la gestione delle multe, semplificando così procedure tradizionalmente onerose. La fase di de-fleeting, infine, è supportata da processi automatizzati di selezione dei veicoli da dismettere, dalla gestione delle aste alla produzione di documentazione fiscale, fino alla sincronizzazione con i sistemi contabili aziendali.

Sul piano finanziario e contabile, la piattaforma consente la registrazione e la riconciliazione in tempo reale di fatture di acquisto e vendita, nonché la gestione dei flussi di cassa e la produzione di reportistica sia statutaria che gestionale. L'integrazione con sistemi contabili esterni amplia ulteriormente le possibilità di controllo e trasparenza, supportando la chiusura mensile e le analisi manageriali. [29]

Il Ticket Management System applicato a 3FS©

Il Ticket Management System è per ora solo un proof-of-concept, ma in futuro si potrà integrare all'interno di 3FS© per fornire supporto tecnico agli utenti. Obiettivo aggiuntivo è quello di fornire al modello una dettagliata conoscenza del funziona-

mento del sistema software, così da poter aiutare nella risoluzione dei problemi più semplici.

L'integrazione con 3FS© prevede più livelli funzionali: raccolta dei ticket da interfacce utente (web, mobile e punti di assistenza), correlazione automatica con moduli del sistema (es. logistica, contabilità, gestione veicoli) e instradamento verso gli operatori o verso risposte automatiche basate su conoscenza contestualizzata. Il proof-of-concept dimostra la fattibilità della correlazione semantica fra descrizioni dei problemi e componenti software, migliorando i tempi di diagnosi e la qualità delle soluzioni proposte.

Vantaggi attesi

L'integrazione del Ticket Management System con 3FS© mira a ottenere diversi benefici misurabili:

- riduzione del tempo medio di risoluzione grazie a suggerimenti contestuali e triage automatico;
- diminuzione del carico operativo umano per attività ripetitive e di basso valore;
- miglioramento della qualità del servizio e della soddisfazione utente tramite risposte coerenti e tempestive;
- raccolta di conoscenza operativa che può alimentare analisi predittive e piani di miglioramento.

Conclusione

Il Model Context Protocol offre un nuovo modo per potenziare le capacità degli LLM, già di per sé strumenti estremamente capaci e potenti. Questa tesi ha illustrato una possibile applicazione di questa tecnologia.

Bibliografia

[1] UBS. Latest House View Daily, 2023. URL: https://www.ubs.com/global/en/wealthmanagement/insights/chief-investment-office/house-view/daily/2023/latest-25052023.html. Citato a pagina 6.

[2] Model Context Protocol. Getting Started - Introduction, 2024. URL: https://modelcontextprotocol.io/docs/getting-started/intro. Citato a pagina 7.

[3] Humza Naveed, Asad Ullah Khan, Shi Qiu, Muhammad Saqib, Saeed Anwar, Muhammad Usman, Naveed Akhtar, Nick Barnes, and Ajmal Mian. arXiv preprint arXiv:2307.06435, 2023. Accessed: 2025-09-02. URL: https://arxiv.org/abs/2307.06435, arXiv:2307.06435. Citato alle pagine 8 and 16.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018. Accessed: 2025-09-02. URL: https://arxiv.org/abs/1810.04805, arXiv:arXiv:1810.04805. Citato a pagina 9.

[5] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 2227–2237. Association for Computational Lingui-

stics, 2018. Accessed: 2025-09-02. URL: https://aclanthology.org/N18-1202. Citato a pagina 9.

[6] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension, 2019. Accessed: 2025-09-02. URL: https://arxiv.org/abs/1910.13461, arXiv:arXiv:1910.13461. Citato a pagina 9.

[7] Andrey Chernyavskiy, Dmitry Ilvovsky, and Preslav Nakov. Transformers: "the end of history" for natural language processing? In *Machine Learning and Knowledge Discovery in Databases. Research Track: European Conference, ECML PKDD 2021, Bilbao, Spain, September 13–17, 2021, Proceedings, Part III*, volume 12977 of *Lecture Notes in Computer Science*, pages 677–693. Springer, 2021. Accessed: 2025-09-02. doi:10.1007/978-3-030-86523-8_41. Citato a pagina 9.

[8] J. J. Webster and C. Kit. Tokenization as the initial phase in NLP. In *Proceedings of the 14th International Conference on Computational Linguistics (COLING), Volume 4*, 1992. Citato a pagina 10.

[9] Taku Kudo. Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 66–75, 2018. URL: https://aclanthology.org/P18-1007. Citato a pagina 10.

[10] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural Machine Translation of Rare Words with Subword Units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, 2016. URL: https://aclanthology.org/P16-1162. Citato a pagina 10.

[11] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. Archived (PDF) from the original on 2024-02-21. Retrieved 2024-01-21. URL: `https://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf`. Citato a pagina 10.

[12] William Fedus, Barret Zoph, and Noam Shazeer. Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity. *Journal of Machine Learning Research*, 23(1):5232–5270, 2022. Citato a pagina 11.

[13] N. Du, Y. Huang, A. M. Dai, S. Tong, D. Lepikhin, Y. Xu, M. Krikun, Y. Zhou, A. W. Yu, O. Firat, et al. GLAM: Efficient Scaling of Language Models with Mixture-of-Experts. In *Proceedings of the International Conference on Machine Learning*, pages 5547–5569. PMLR, 2022. Citato a pagina 11.

[14] X. Ren, P. Zhou, X. Meng, X. Huang, Y. Wang, W. Wang, P. Li, X. Zhang, A. Podolskiy, G. Arshinov, et al. Pangu-P: Towards Trillion Parameter Language Model with Sparse Heterogeneous Computing, 2023. URL: `https://arxiv.org/abs/2303.10845`, `arXiv:arXiv:2303.10845`. Citato a pagina 11.

[15] Tao Wang, Adam Roberts, David Hesslow, Teven Le Scao, Hyung Won Chung, Iz Beltagy, Jérémy Launay, and Colin Raffel. What Language Model Architecture and Pretraining Objective Works Best for Zero-Shot Generalization? In *Proceedings of the International Conference on Machine Learning*, pages 22964–22984. PMLR, 2022. URL: `https://proceedings.mlr.press/v162/wang22a.html`. Citato a pagina 12.

[16] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Saurabh Brahma, et al. Scaling instruction-finetuned language models, 2022. Accessed: 2025-09-02. URL:

https://arxiv.org/abs/2210.11416, arXiv:arXiv:2210.11416. Citato a pagina 14.

[17] Daniel M. Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B. Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. Fine-tuning language models from human preferences. *arXiv preprint*, 2019. URL: https://arxiv.org/abs/1909.08593, arXiv:1909.08593. Citato a pagina 15.

[18] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Sharan Batra, Puneet Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models, 2023. Accessed: 2025-09-02. URL: https://arxiv.org/abs/2307.09288, arXiv:arXiv:2307.09288. Citato a pagina 15.

[19] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pages 27730–27744, 2022. Accessed: 2025-09-02. URL: https://proceedings.neurips.cc/paper_files/paper/2022/hash/b1efde53be364a73914f58805a001731-Abstract-Conference.html. Citato a pagina 15.

[20] Q. Dong, L. Li, D. Dai, C. Zheng, Z. Wu, B. Chang, X. Sun, J. Xu, and Z. Sui. A survey for in-context learning, 2023. URL: https://arxiv.org/abs/2301.00234, arXiv:2301.00234. Citato a pagina 17.

[21] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed H. Chi, Quoc V. Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837, 2022. Citato a pagina 18.

[22] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves

chain of thought reasoning in language models, 2022. URL: https://arxiv.org/abs/2203.11171, arXiv:2203.11171. Citato a pagina 18.

[23] S. Yao, D. Yu, J. Zhao, I. Shafran, T. L. Griffiths, Y. Cao, and K. Narasimhan. Tree of thoughts: Deliberate problem solving with large language models, 2023. Accessed: 2025-09-02. URL: https://arxiv.org/abs/2305.10601, arXiv:2305.10601. Citato a pagina 18.

[24] Model Context Protocol. Getting Started - Introduction, 2024. Accessed: 2025-08-16. URL: https://modelcontextprotocol.io/docs/getting-started/intro. Citato a pagina 29.

[25] Model Context Protocol. Learn - Architecture, 2024. Accessed: 2025-08-16. URL: https://modelcontextprotocol.io/docs/learn/architecture. Citato alle pagine 29, 30, 31, 32, and 33.

[26] Microsoft Docs. Using Web API with Entity Framework (Part 5). Microsoft Docs. Accessed: 2025-09-02. URL: https://learn.microsoft.com/it-it/aspnet/web-api/overview/data/using-web-api-with-entity-framework/part-5. Citato a pagina 35.

[27] Microsoft Docs. CQRS pattern. Microsoft Docs. Accessed: 2025-09-02. URL: https://learn.microsoft.com/it-it/azure/architecture/patterns/cqrs. Citato a pagina 36.

[28] Microsoft Docs. Azure Functions overview. Accessed: 2025-08-16. URL: https://learn.microsoft.com/it-it/azure/azure-functions/functions-overview?pivots=programming-language-csharp. Citato a pagina 36.

[29] ProgeSoftware. 3FS - ProgeSoftware, 2021. Accessed: 2025-09-02. URL: https://3fs.progesoftware.it/. Citato a pagina 44.