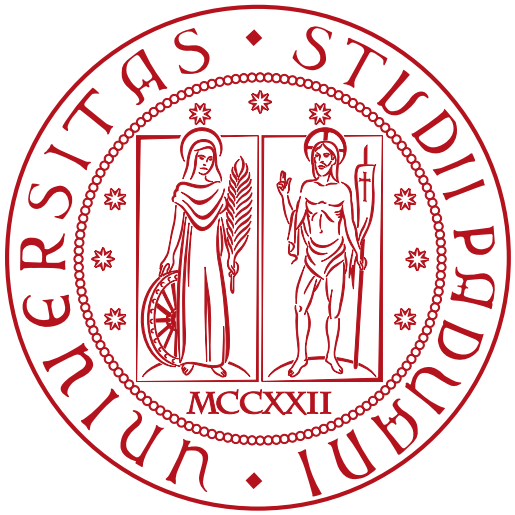
Università degli studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica



Sviluppo di un chatbot basato su Model Context Protocol per il supporto alla pianificazione delle attività aziendali

Tesi di laurea triennale

Relatore
Francesco Ranzato

Laureando
Leonardo Preo
Matricola 2101092

ANNO ACCADEMICO 2025-2026

«Life is a tragedy when seen in close-up, but a comedy in long-shot.»

—— Charlie Chaplin

«There is no light for those who don't know darkness…»

—— Takehiko Inoue

Ringraziamenti

Desidero esprimere la mia profonda gratitudine al Prof. Francesco Ranzato, relatore di questa tesi, per la guida, i preziosi consigli e la completa disponibilità dimostrati durante l'intero percorso di stesura.
Un ringraziamento speciale va ai miei genitori, per avermi permesso di intraprendere questo percorso universitario e per avermi sempre sostenuto con fiducia.
Grazie a mio fratello, per la sua presenza costante e per quel legame profondo e silenzioso, tipico dei fratelli, che ha reso speciale la nostra crescita insieme.
Un pensiero colmo d'affetto è rivolto ad Alessia, la mia dolce metà: grazie per essermi stata accanto, con amore e pazienza, nei momenti più difficili che hanno segnato questi anni. Ringrazio inoltre tutti i miei parenti, che mi sono stati vicini durante tutto il mio percorso di studi.
Dedico questo traguardo ai miei nonni. A Lino, che pur non potendo festeggiare con me questo importante obiettivo, sento vicino in ogni momento. A mia nonna Bianca: sebbene la malattia abbia offuscato i ricordi, rimane per me la figura luminosa di sempre. Un grazie profondo a Gemma e Vittorio, che si sono presi cura di me fin da piccolo e che, ancora oggi, sostengono con affetto me e mio fratello.
Ringrazio i miei amici, che con la loro presenza e la loro allegria hanno reso questi anni di studio più leggeri, aiutandomi ad affrontare ogni sfida con il sorriso.
Infine, desidero ringraziare i colleghi di Omega S.r.l.. e il mio tutor aziendale, Marco Ortali, per l'opportunità di lavorare a questo progetto. Un ringraziamento particolare va al loro supporto, sia tecnico che umano, ricevuto durante le settimane di tirocinio; la loro vicinanza è stata fondamentale per superare le difficoltà personali che hanno caratterizzato questo periodo.

Salzano, 8 Luglio 2026 *Leonardo Preo*

Sommario

Il presente documento descrive il lavoro svolto durante il periodo di stage curricolare, della durata di circa trecentoventi ore, dal laureando Leonardo Preo presso l'azienda Omega S.r.l.. Lo stage è stato condotto sotto la supervisione del tutor aziendale Marco Ortali, mentre il prof. Francesco Ranzato ha ricoperto il ruolo di tutor accademico.

Questa tesi tratta la progettazione e lo sviluppo di **un modulo** ***Chatbot***$_G$ integrato all'interno del gestionale Agilis, volto a supportare il reparto *Delivery* nella pianificazione delle attività operative. L'obiettivo è quello di fornire uno strumento di supporto decisionale basato su intelligenza artificiale, in grado di interagire in linguaggio naturale con gli operatori, suggerendo soluzioni di pianificazione intelligente e segnalando eventuali conflitti o vincoli non rispettati.

Organizzazione del testo

Il primo capitolo presenta l'azienda, introduce il progetto e illustra le motivazioni che mi hanno portato a sceglierlo, definendone gli obiettivi e le peculiarità;

Il secondo capitolo ripercorre lo stato dell'arte su cui poggia il lavoro: l'evoluzione dei *Large Language Models*, il paradigma del *Tool Use*$_G$ e della *function calling*, il *Model Context Protocol* e i *Framework*$_G$ di orchestrazione;

Il terzo capitolo analizza gli utenti e i casi d'uso del sistema e ne deriva i requisiti, distinti in funzionali, non funzionali e vincoli, rimandando all'Appendice A per la specifica completa;

Il quarto capitolo descrive le tecnologie adottate, motivando le ragioni che ne hanno guidato la scelta;

Il quinto capitolo descrive la progettazione del sistema: l'architettura di distribuzione, i componenti principali e le loro interazioni, il modello di interazione uomo-AI e le principali decisioni architetturali;

Il sesto capitolo descrive l'implementazione, illustrando le scelte tecniche più significative e le sfide affrontate;

Il settimo capitolo trae le conclusioni: il consuntivo tra piano di lavoro e lavoro effettivo, il grado di raggiungimento degli obiettivi, i rischi occorsi e i possibili sviluppi futuri.

Convenzioni tipografiche

Durante la stesura del testo ho scelto di adottare le seguenti convenzioni tipografiche:

- Gli acronimi, le abbreviazioni e i termini di uso non comune menzionati vengono definiti nel glossario, situato alla fine del documento (p. 75);
- Per la prima occorrenza dei termini riportati nel glossario viene utilizzata la seguente nomenclatura: *termine*$_G$;
- I termini in lingua straniera non di uso comune o facenti parti del gergo tecnico sono evidenziati con il carattere *corsivo*;
- I nomi di funzioni o variabili appartenenti ad un linguaggio di programmazione vengono scritte con un carattere `monospaziato`;
- Le citazioni ad un libro o ad una risorsa presente nella bibliografia (p. 79) saranno affiancate dal rispettivo numero identificativo, es. [1];
- I blocchi di codice sono rappresentati nel seguente modo

```C#
using System;

namespace EsempioTesi
{
    class Program
    {
        // Metodo principale di esecuzione
        static void Main(string[] args)
        {
            int risultato = CalcolaSomma(5, 7);
            Console.WriteLine($"Il risultato è: {risultato}");
        }

        /// <summary> Calcola la somma di due numeri interi.</summary>
        static int CalcolaSomma(int a, int b)
        {
            return a + b;
        }
    }
}
```

Codice 1: Codice d'esempio.

Indice

Elenco delle Figure

Elenco delle Tabelle

Elenco dei Codici Sorgente

Capitolo 1

Introduzione

In questo capitolo presento l'azienda presso cui ho svolto il tirocinio, introduco il progetto e illustro le motivazioni che mi hanno portato a sceglierlo, definendone gli obiettivi e il contributo originale.

1.1 Il contesto aziendale e l'ambito del progetto

Figura 1: Logo di Omega S.r.l.

Il presente lavoro di tesi nasce all'interno dell'esperienza di tirocinio curricolare svolta presso *Omega Gruppo*[1] [2], realtà informatica specializzata nello sviluppo di soluzioni software gestionali ed *ERP*$_G$ per l'ottimizzazione dei processi aziendali. In un mercato guidato dall'efficienza operativa, la digitalizzazione e l'automazione dei flussi di lavoro costituiscono i pilastri portanti per essere competitivi. In scenari enterprise più ampi, un ERP dialoga spesso anche con il *Frontend*$_G$ applicativo e con sistemi *MES*$_G$ e *WMS*$_G$ per mantenere allineati produzione e magazzino.
Tra le soluzioni *core* dell'azienda figura Agilis, un sistema gestionale strutturato per la pianificazione e il monitoraggio delle risorse aziendali. In particolare, il modulo dedicato alla gestione della *delivery* e della pianificazione delle attività (denominato internamente *XSPMPRIS*) costituisce uno degli strumenti più critici per gli operatori aziendali. Attraverso questo modulo, i responsabili della pianificazione coordinano l'assegnazione dei task, la gestione dei carichi di lavoro e la schedulazione delle commesse, interfacciandosi costantemente con un'architettura complessa e con un patrimonio informativo di grandi dimensioni, storicizzato su *Database*$_G$ relazionali e mediato da un ecosistema di servizi *REST*$_G$ (denominato Symposium).

[1] https://www.omegagruppo.it/

1.2 Definizione del problema e motivazioni

La definizione delle attività all'interno di un contesto aziendale moderno è un compito ad alto carico cognitivo. Gli operatori si trovano spesso a dover consultare simultaneamente molteplici schermate, verificare la disponibilità delle risorse umane, incrociare scadenze stringenti e rispettare vincoli di competenze tecniche. Questo processo, sebbene supportato dalle interfacce grafiche tradizionali dei gestionali, risulta intrinsecamente rigido: ogni operazione richiede una sequenza precisa di interazioni manuali (clic, inserimento dati, navigazione tra menu) e una ottima padronanza del software.
Negli ultimi anni, l'avvento dei *LLMs*$_G$ ha rivoluzionato il paradigma di interazione uomo-macchina, introducendo la possibilità di elaborare e comprendere il linguaggio naturale con un livello di flessibilità in costante crescita. Tuttavia, l'adozione di queste tecnologie in contesti *Enterprise* si scontra spesso contro due barriere fondamentali:

1. **L'isolamento informativo:** i modelli commerciali o *Open-Source*$_G$ nascono privi di conoscenza rispetto ai dati in tempo reale residenti all'interno dei sistemi gestionali privati.
2. **L'affidabilità e la sicurezza dei dati:** permettere a un *Agente*$_G$ basato su intelligenza artificiale di scrivere direttamente o modificare in autonomia record sensibili su un database di produzione introduce rischi inaccettabili in termini di consistenza e conformità software.

La sfida centrale di questo progetto risiede quindi nel superare la rigidità delle interfacce tradizionali senza compromettere la sicurezza del sistema informativo preesistente, offrendo agli utenti uno strumento in grado di tradurre l'intento espresso tramite linguaggio naturale in azioni concrete e verificate.

1.3 Obiettivi del progetto e contributi originali

L'obiettivo principale di questo lavoro è la progettazione e lo sviluppo di un assistente conversazionale intelligente, integrato nativamente nell'interfaccia utente del modulo di pianificazione di Agilis. Per garantire un'integrazione flessibile, standardizzata e disaccoppiata tra l'applicazione *Client*$_G$ e i modelli di intelligenza artificiale, si è scelto di adottare il *Model Context Protocol* o *MCP*$_G$, un protocollo emergente aperto che standardizza il modo in cui i modelli *AI*$_G$ si connettono a sorgenti di dati e strumenti esterni.
Il perimetro del lavoro si inserisce in un ecosistema in cui il gestionale client Agilis, lo strato di servizi *Backend*$_G$ Symposium e il database erano già consolidati e preesistenti. Il contributo originale di questa tesi si concentra sul **livello di integrazione conversazionale**, sull'architettura di orchestrazione e sulla necessaria **espansione dello strato dei servizi**. Nello specifico, l'attività di ricerca e sviluppo ha riguardato:

- La progettazione e l'implementazione della sezione chatbot direttamente all'interno dell'interfaccia utente di Agilis.
- La creazione di nuove *API*$_G$ all'interno di Symposium, progettate specificamente per esporre la logica di business ai nuovi servizi.

- Lo sviluppo di un'architettura client/*Server*$_G$ basata su protocollo MCP, in grado di esporre in sicurezza le funzionalità del gestionale sotto forma di «strumenti» (tool) invocabili dall'LLM.
- L'orchestrazione dei flussi di pensiero del modello e la gestione dello streaming della risposta in tempo reale.
- L'ideazione di un meccanismo transazionale di **anteprima in memoria**: il chatbot non effettua scritture dirette sul database, ma genera una proposta strutturata che l'operatore può valutare visivamente e confermare esplicitamente prima della persistenza definitiva.
- La gestione della sicurezza e del ciclo di vita dei *token* di autenticazione lungo tutta la catena dei processi separati.

Questi sei contributi corrispondono agli obiettivi O01–O07 (obbligatori) e F01 (facoltativo) formalizzati nel Piano di Lavoro, il cui stato di raggiungimento è rendicontato nel capitolo conclusivo (Sezione 7). L'approccio proposto si configura come un sistema di supporto decisionale: l'intelligenza artificiale estende le capacità operative dell'utente, riducendo i tempi di inserimento e ricerca, ma l'ultima parola e il controllo sulla validità dell'operazione rimangono saldamente in mano all'operatore umano.

Capitolo 2

Stato dell'Arte

In questo capitolo ripercorro lo stato dell'arte su cui si fonda il progetto: l'evoluzione dei Large Language Models, il paradigma del tool use e della function calling, il Model Context Protocol e i principali framework di orchestrazione per applicazioni basate su intelligenza artificiale.

2.1 L'evoluzione dei Large Language Models

Gli LLM hanno conosciuto negli ultimi anni un'evoluzione radicale, ridefinendo i confini dell'ingegneria del software e dell'intelligenza artificiale applicata. Basati prevalentemente sull'architettura *Transformer*[2] introdotta da Vaswani et al. nel 2017 [3], questi modelli vengono pre-addestrati su enormi volumi di testo non strutturato per apprendere le relazioni statistiche e semantiche tra i *token*.
Inizialmente concepiti come predittori statistici di testo, i modelli più recenti hanno mostrato capacità emergenti avanzate, tra cui il ragionamento logico-deduttivo, la comprensione del contesto e la generazione di codice sorgente sintatticamente corretto. Sebbene posseggano queste potentissime capacità, l'architettura pura di un LLM presenta un limite intrinseco, l'*isolamento informativo*: il modello risponde basandosi esclusivamente sulla conoscenza cristallizzata al momento del suo *Knowledge Cutoff*$_G$, risultando cieco rispetto ai dati dinamici, in tempo reale o privati. A ciò si aggiunge il fenomeno delle *allucinazioni*, ovvero la generazione di risposte plausibili ma fattualmente errate.

2.2 Il paradigma del Tool Use e della Function Calling

Per superare l'isolamento informativo e ridurre l'incidenza delle *Allucinazioni*$_G$, la ricerca si è orientata verso *Sistemi Aumentati*$_G$, facendo evolvere il ruolo dell'LLM da generatore di testo ad *agente* in grado di interagire con l'ambiente esterno. Questo cambiamento di paradigma è noto come *tool use* o *function calling*.
In questo modello l'LLM non viene più interrogato per fornire direttamente la risposta a un problema che richiede dati esterni; viene invece istruito a riconoscere quando una richiesta necessita di informazioni integrative o di un'azione applicativa. In tal caso il modello interrompe la generazione e produce un output strutturato — tipicamente in formato *JSON*$_G$ — che specifica il nome di una

[2] https://arxiv.org/abs/1706.03762

funzione e i relativi parametri di invocazione. L'applicazione ospite intercetta questo output, esegue la funzione chiamando le *API* o effettuando le query necessarie e restituisce il risultato al modello, che lo rielabora per formulare la risposta finale. Il meccanismo sposta il carico dell'elaborazione del dato dall'intelligenza artificiale al software deterministico, garantendo la precisione del risultato.

2.3 Il Model Context Protocol

Sebbene la *function calling* sia ampiamente supportata dai principali provider di modelli, la sua implementazione ha storicamente sofferto di una forte frammentazione. Prima dell'avvento di MCP, l'integrazione tra LLM e strumenti esterni era affidata a meccanismi proprietari e mutuamente incompatibili: le *OpenAI Functions* definivano i tool tramite uno schema JSON specifico con vincoli di formato rigidi; *Anthropic Tool Use* adottava un proprio schema di descrizione e un diverso protocollo di invocazione; i *Plugin*$_G$ system di ChatGPT imponevano un modello di autenticazione e manifest separato. Ogni piattaforma e ogni framework imponeva quindi schemi di definizione dei tool, formati di serializzazione e modelli di trasporto proprietari. Di conseguenza, integrare lo stesso set di strumenti con modelli differenti ha spesso richiesto la riscrittura di complessi *layer* di adattamento.
Per risolvere il suddetto problema di interoperabilità è stato introdotto il *Model Context Protocol*[3] [1], un protocollo aperto e standardizzato — sviluppato da Anthropic[4] [4] — che formalizza la comunicazione tra le applicazioni guidate dall'AI e le sorgenti di dati o strumenti esterni. Per analogia, il protocollo MCP rappresenta per un'applicazione basata su intelligenza artificiale ciò che lo standard USB-C rappresenta per l'hardware: un livello di connessione universale.
L'architettura si basa su una chiara separazione dei ruoli secondo un pattern *host-client-server*. Un **server MCP** è un processo indipendente che espone in modo standardizzato tre tipi di risorse primitive: i *Prompt*$_G$ (modelli di istruzioni preconfigurati), le *resource* (dati testuali o binari in sola lettura) e i tool (funzioni eseguibili che interrogano o modificano sistemi esterni, descritte tramite JSON Schema) [5]. Un **host** — come un *IDE*$_G$ o un assistente conversazionale — istanzia, per ciascun server a cui si collega, un **client MCP** che ne media la comunicazione: il client interroga il server per scoprirne le capacità, ne inoltra gli schemi al modello e, quando questo richiede l'invocazione di uno strumento, instrada la chiamata al server e ne restituisce la risposta.

[3] https://modelcontextprotocol.io/docs/getting-started/intro
[4] Società statunitense di ricerca e sviluppo nell'intelligenza artificiale.

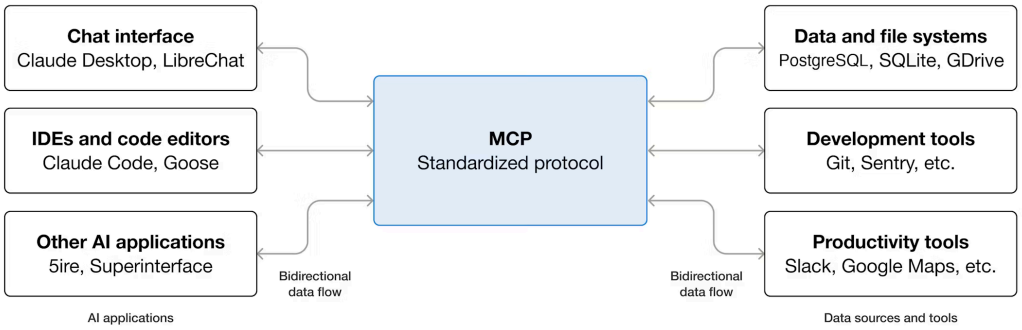


Figura 2: Rappresentazione architetturale di alto livello del Model Context Protocol (MCP).
Source: *https://modelcontextprotocol.io/docs/getting-started/intro* [1]

A livello di trasporto il protocollo utilizza canali standard — i flussi di input/output standard (*stdio*) per processi locali o connessioni persistenti (come i *Server Sent Events* su HTTP) per le architetture distribuite — serializzando i messaggi secondo lo standard JSON-RPC 2.0 [6].

2.4 Framework di orchestrazione e vincoli aziendali

La traduzione pratica del paradigma ad agenti in contesti *enterprise* richiede framework di orchestrazione robusti, capaci di gestire i cicli di esecuzione, i contesti di memoria e i filtri software. Nel panorama Microsoft i due principali riferimenti sono *LangChain*[5] [7] e *Microsoft Semantic Kernel*[6] [8].
Semantic Kernel è un SDK_G *open source* sviluppato da Microsoft per facilitare l'integrazione di modelli AI in applicazioni .NET. Rispetto ad altre librerie si distingue per una forte astrazione concettuale e per un'integrazione nativa con i meccanismi di *dependency injection* e di programmazione asincrona tipici del C#. Il framework astrae i canali di comunicazione con i differenti provider AI, offrendo un'interfaccia unificata per la gestione dello *streaming* dei *token* e per l'esecuzione dei *plugin*; si presta quindi come strato ideale per implementare i filtri di controllo e di sicurezza che intercettano le risposte dell'LLM prima della loro visualizzazione.
LangChain è un'alternativa *open-source* anche per l'ecosistema Microsoft e Python, meno integrata con .NET Framework ma più flessibile. Nel progetto non è stata utilizzata perché Semantic Kernel offre una migliore astrazione per il contesto .NET e una più stretta integrazione con i meccanismi di *dependency injection* nativi di C#. LangChain sarebbe stata la scelta preferita in un

[5] https://www.langchain.com/
[6] https://learn.microsoft.com/it-it/semantic-kernel/overview/

ecosistema *polyglot*, ma l'eterogeneità dei *runtime* (.NET 4.8 *legacy* vs .NET 10 moderno, cfr. Sezione 4) avrebbe richiesto ulteriori *layer* di traduzione.
Lo sviluppo di soluzioni basate su queste tecnologie deve però fare i conti con le infrastrutture aziendali preesistenti. Molti ERP di classe *enterprise* poggiano su versioni classiche del framework .NET — come .NET Framework 4.8 — che presentano rigidi vincoli di compatibilità verso le librerie moderne pensate per .NET Core o .NET 8/10. Questa asimmetria tecnologica impone scelte architetturali mirate, separando nettamente l'interfaccia utente *legacy* dai moderni motori di orchestrazione AI mediante strategie di disaccoppiamento basate su processi locali isolati o servizi web dedicati.

Capitolo 3

Analisi dei requisiti

In questo capitolo analizzo gli utenti e i casi d'uso del sistema e ne derivo i requisiti, distinti per tipologia in funzionali, non funzionali e vincoli; per non appesantire la trattazione, la specifica completa e le user stories sono riportate in Appendice A.

3.1 Analisi degli utenti

Il prodotto è concepito per gli operatori del reparto *Delivery* di un'azienda cliente del Gruppo Omega che adotta il gestionale Agilis: personale incaricato di gestire le consegne ai clienti, organizzare i turni di lavoro e coordinare le attività con le altre divisioni aziendali. L'obiettivo è fornire una soluzione concreta che incrementi l'efficienza e la produttività del reparto, automatizzando il supporto decisionale e liberando tempo da destinare ad attività collaterali altrimenti trascurate. Poiché il chatbot è integrato nel modulo di pianificazione di Agilis — un'area ad accesso ristretto agli operatori *Delivery* — questi rappresentano l'unica tipologia di utenza del sistema, e ogni interazione avviene sempre in seguito all'autenticazione nel gestionale.

3.2 Metodo: user stories e codifica dei requisiti

I requisiti funzionali sono stati raccolti tramite *user stories*, strumento dello sviluppo agile che esprime un requisito dal punto di vista dell'utente nella forma

«come [utente], voglio [obiettivo], in modo da [beneficio]»

ed è corredato di *acceptance criteria* verificabili. Le storie sono organizzate in *epic*, aggregazioni tematiche che facilitano la pianificazione. A ciascun requisito è poi associato un codice del tipo `(F/Q/C)(M/D/O)R`, che ne indica la tipologia - funzionale, qualitativo o di vincolo - e la necessità - *mandatory*, *desirable* od *optional*. La lista integrale delle *user stories*, con i relativi *acceptance criteria*, e le tabelle di tracciamento requisito-fonte sono riportate in Appendice A.

3.3 Macro-requisiti

Le *user stories* raccolte possono essere raggruppate in tre macro-aree funzionali, che definiscono l'architettura generale e fungono da linee guida per i requisiti di

dettaglio.
La prima riguarda la **comprensione del contesto operativo**: l'assistente deve estrarre specifiche entità — quali operatori, commesse, date e tipologie di task — da richieste destrutturate in lingua italiana, eventualmente arricchite da allegati.
La seconda riguarda il **supporto decisionale non distruttivo**: il sistema formula piani di lavoro anche complessi senza alterare in autonomia il database di produzione, lasciando all'utente il controllo su ogni operazione.
La terza riguarda la **trasparenza del flusso logico**: l'operatore deve poter ispezionare il processo di ragionamento del modello e gli strumenti utilizzati, così da poterne verificare e validare le proposte.

3.4 Quadro dei requisiti

L'analisi ha prodotto **trenta requisiti funzionali**, **sette requisiti qualitativi** e **quattro vincoli**.
I requisiti funzionali coprono l'intero ciclo d'uso: la gestione del contesto (allegati e conversazioni come fonte), l'interrogazione informativa sui dati del gestionale, la generazione di proposte di pianificazione e il loro affinamento iterativo, l'anteprima e l'applicazione selettiva delle operazioni, la creazione guidata di appuntamenti, la riorganizzazione delle agende a fronte di imprevisti e la gestione della conversazione.
I requisiti qualitativi fissano le proprietà trasversali: coerenza delle risposte con il contesto, chiarezza delle notifiche, allineamento tra gestionale e database, integrazione visiva con Agilis, tempi di risposta ragionevoli e tracciabilità delle elaborazioni.
I vincoli, infine, recepiscono le condizioni imposte dal contesto aziendale: sviluppo su framework .NET compatibile con Agilis, integrazione con Symposium, divieto di modifiche autonome alla pianificazione e architettura modulare.
La Tabella 1 ne riassume la distribuzione per tipologia e priorità; il tracciamento puntuale di ciascun requisito verso la *user story* o la fonte di origine è in Appendice A.

Tipo	Mandatory	Desirable	Optional	Somma
Functional	17	11	2	30
Qualitative	4	3	0	7
Constraint	4	0	0	4
Totale	**25**	**14**	**2**	**41**

Tabella 1: Riepilogo dei requisiti.

Capitolo 4

Tecnologie utilizzate

In questo capitolo illustro le tecnologie utilizzate per lo sviluppo del progetto, descrivendone brevemente le caratteristiche principali e motivando le ragioni che mi hanno portato a sceglierle.

4.1 La piattaforma .NET

Figura 3: Logo di .NET.

.NET[7] [9] è la piattaforma di sviluppo ed esecuzione creata da Microsoft su cui vengono eseguiti i linguaggi C# e Visual Basic .NET; fornisce un'estesa libreria di classi base e gestisce l'esecuzione sicura del codice tramite *Common Language Runtime* e *garbage collection*. Occorre però distinguere due famiglie:

- **.NET Framework**: storico e disponibile solo su Windows, la cui versione finale è la 4.8;
- **.NET moderno**: multipiattaforma e *open-source*, giunto alla versione 10.

Questa distinzione è centrale per il progetto, poiché le due famiglie non sono interoperabili all'interno dello stesso processo. Il gestionale Agilis è realizzato in Visual Basic .NET su .NET Framework 4.8; i nuovi componenti di intelligenza artificiale (server e client MCP) sono invece sviluppati in C# su .NET 10, condizione necessaria per poter adottare le librerie più recenti dell'ecosistema — come l'SDK di MCP — non disponibili sul Framework *legacy*. L'impossibilità di caricare assembly .NET 10 all'interno di un processo .NET Framework 4.8 ha reso la separazione in processi distinti non una semplice scelta di disaccoppiamento, ma una necessità tecnica (cfr. Sezione 5.2).

[7] https://dotnet.microsoft.com/it-it/

4.2 C#

Figura 4: Logo di C#.

C# è un linguaggio orientato agli oggetti, fortemente tipizzato e multi-paradigma, sviluppato da Microsoft. Grazie alla sua sintassi chiara, al controllo dei tipi a tempo di compilazione e alla gestione automatica della memoria, rappresenta uno standard industriale per le applicazioni *enterprise*. Nel progetto è stato il linguaggio principale per la logica di *back-end*, rivelandosi ideale per la realizzazione del server MCP e per l'orchestrazione conversazionale.

4.3 Visual Basic .NET

Visual Basic .NET è un linguaggio orientato agli oggetti che, al pari di C#, viene compilato per la piattaforma .NET. Nel progetto è stato impiegato per il modulo di interfaccia integrato nel gestionale. La scelta non è stata discrezionale, ma dettata da un vincolo di omogeneità: il modulo di pianificazione del reparto *Delivery* in cui il chatbot si innesta è interamente scritto in Visual Basic .NET, e l'integrazione richiede di operare direttamente sugli oggetti dell'interfaccia esistente — leggere lo stato del pianificatore, applicare in anteprima le proposte e agganciarsi al flusso di salvataggio — attività che impongono di sviluppare nello stesso linguaggio e nello stesso processo del modulo ospitante.

4.4 Microsoft Semantic Kernel

Semantic Kernel è il framework di orchestrazione LLM utilizzato nel client. Fornisce l'astrazione del *Kernel* (host del modello e dei *plugin*), il *function calling* automatico — è il framework a decidere e auto-invocare i tool in base al ragionamento del modello — e una catena di filtri che permette di intercettare ogni invocazione di strumento.
Quest'ultimo meccanismo è stato sfruttato per l'anti-loop, l'osservabilità e il rinnovo del *token*. I tool MCP esposti dal server vengono mappati in funzioni del *kernel* e resi invocabili dal modello.

4.5 SDK del Model Context Protocol

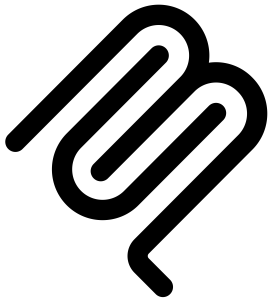


Figura 5: Logo di MCP.

L'integrazione del protocollo MCP, descritto concettualmente nella Sezione 2, è stata realizzata avvalendosi del relativo SDK per .NET [10]. Questa libreria costituisce la dipendenza fondamentale che ha imposto l'uso di .NET 10 per i nuovi componenti del sistema, essendone un requisito tecnico imprescindibile. Dal punto di vista implementativo, l'SDK fornisce tutti i blocchi costitutivi per l'architettura: per il lato server, semplifica l'esposizione delle funzionalità aziendali e gestisce la comunicazione bidirezionale tramite i canali standard di *input/output*; per il lato client, offre meccanismi per interrogare il server, mappare a *runtime* gli strumenti esposti e tradurli in interfacce comprensibili per il modello linguistico.

4.6 *Endpoint*$_G$ LLM e Ollama

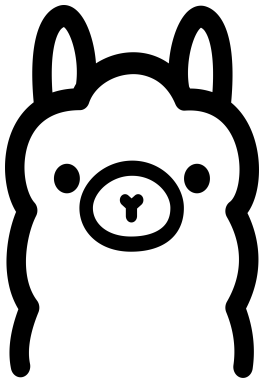


Figura 6: Logo di Ollama.

L'inferenza è delegata a un *endpoint* LLM remoto *OpenAI-compatibile*, ospitato su un server aziendale con runtime di tipo *Ollama*[8] [11]. Ollama è uno strumento *open source* che facilita l'esecuzione, la gestione e il *deployment* di LLM in ambienti locali: a differenza delle soluzioni *Cloud*$_G$, consente di eseguire i modelli

[8] https://ollama.com/

direttamente sull'infrastruttura ospite, garantendo privacy dei dati e assenza di latenza verso API di terze parti, e offre un'interfaccia conforme allo standard delle API OpenAI[9] [12]. Questa compatibilità ha permesso di interrogare il modello tramite librerie client standard, mantenendo il sistema indipendente dallo specifico motore di inferenza. L'esecuzione in locale è resa praticabile dalla *Quantizzazione*$_G$, tecnica che riduce la precisione numerica dei pesi della rete abbassandone i requisiti di memoria e di calcolo.

Figura 7: Logo di Qwen.

Per il motore di inferenza è stato adottato un modello della famiglia ***Qwen***[10] [13], sviluppata da Alibaba Cloud. Qwen è una famiglia di *Large Language Model open-source* con spiccate capacità di ragionamento logico-deduttivo e comprensione multilingue, italiano compreso. La famiglia offre un supporto nativo alla catena di ragionamento esplicita (*thinking chain*): il modello può separare il proprio monologo interno (visibile all'operatore come canale di trasparenza) dalla risposta destinata all'utente, abilitando questo comportamento tramite un semplice *flag* `think` nel *payload* della richiesta HTTP. Qwen è disponibile in diverse taglie; per questo progetto sono state valutate la variante a 9 miliardi di parametri (9B) e quella a 27 miliardi (27B), entrambe in versione quantizzata per ridurre i requisiti hardware. Le dinamiche che hanno guidato la selezione della versione e taglia dello specifico modello sono descritte nella Sezione 5.7.1.

4.7 Visual Studio 2026

Figura 8: Logo di Visual Studio 2026.

[9] Società di ricerca e sviluppo che fornisce API per modelli linguistici generativi.
[10] https://qwen.ai/home

L'intero ciclo di produzione del software è stato orchestrato tramite *Visual Studio 2026*[11] [14], l'ambiente di sviluppo integrato (IDE) impiegato per la stesura, il *debugging* e la compilazione del codice. Scelto per la profonda integrazione con l'ecosistema .NET e il linguaggio C#, offre analisi statica, suggerimenti contestuali e profilazione delle prestazioni, oltre a un'interfaccia visuale per il versionamento integrata nell'ambiente.

4.8 Git

Figura 9: Logo di Git.

Git[12] [15] è un sistema di versionamento distribuito, sviluppato da Linus Torvalds nel 2005, che permette di tracciare le modifiche al codice sorgente e di collaborare in modo efficiente. I suoi elementi principali sono la *working directory*, la *staging area*, i *commit* (istantanee immutabili dello stato del progetto), i *branch* (linee di sviluppo parallele) e i *repository* locale e remoto, riconciliati tramite l'operazione di *merge*. In Omega le funzionalità di Git vengono utilizzate prevalentemente attraverso l'area di versionamento integrata in Visual Studio (Figura 10).

[11] https://visualstudio.microsoft.com/it/vs/
[12] https://git-scm.com/

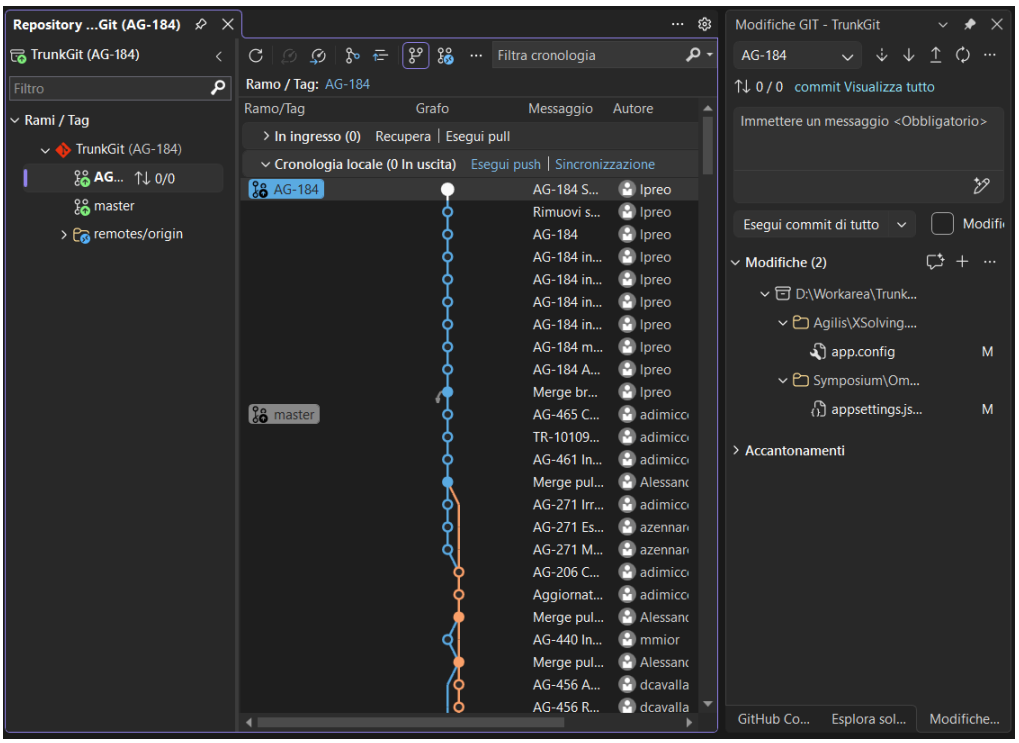


Figura 10: Sezione Visual Studio 2026 dedicata a versionamento.

Capitolo 5

Progettazione

In questo capitolo affronto la progettazione del sistema: descrivo l'architettura di distribuzione sui tre nodi, i componenti principali e le loro interazioni, il modello di interazione uomo-AI ad anteprima e conferma, la gestione della sicurezza e del ciclo di vita del token*, e le principali decisioni architetturali con i relativi compromessi.*

5.1 Architettura di distribuzione: i tre nodi

La soluzione si distribuisce su tre nodi di rete distinti. È il punto da cui discende ogni altra scelta progettuale, poiché determina firewall, sicurezza e *deployment*.

- **Nodo A - Workstation operatore.** Ospita l'interfaccia nativa di Agilis e l'intera catena MCP locale: il client e il server MCP sono processi figli, avviati *on-demand* all'apertura del pannello chatbot. Tutto il carico di orchestrazione locale è confinato qui.
- **Nodo B - Server AI aziendale.** È il servizio centralizzato e remoto che esegue il modello di linguaggio, raggiunto via HTTP. Nessun modello gira sulla workstation: il carico inferenziale è confinato sul server, così da non gravare sulle postazioni degli operatori.
- **Nodo C - Server applicativo Agilis.** Ospita lo strato dei servizi REST di Symposium, il *plugin* MCP e il database del gestionale.

I flussi sono rigidamente direzionali. Il nodo A è il fulcro: apre un flusso HTTP verso il nodo B per inviare i *prompt* e ricevere lo *streaming* dei *token*, e interroga in parallelo il nodo C tramite chiamate REST per recuperare i dati gestionali. I nodi B e C non comunicano mai direttamente tra loro, garantendo che il *back-end* aziendale resti isolato dall'infrastruttura AI.

5.2 I componenti software e l'isolamento in processi

Il sistema si articola in quattro componenti:

1. **OmegaGruppo.McpServer** (.NET 10, nodo A) - espone i dati del gestionale come tool MCP, chiamando le API REST di Symposium.
2. **OmegaGruppo.McpClient** (.NET 10 + Semantic Kernel, nodo A) - è l'host dell'LLM e l'orchestratore conversazionale: avvia il server MCP, espone

una *Named Pipe* verso Agilis e dialoga con l'LLM remoto.
La comunicazione tra Agilis e Omegagruppo.McpClient avviene, appunto, tramite *Named Pipe* locale, riducendo l'*overhead* di rete e mantenendo l'isolamento dei processi. La *pipe* è denominata `OmegaChatbotPipe_{PID}_{GUID:N}` dove `PID` è l'identificativo del processo Agilis e `GUID` un identificativo univoco generato all'apertura del pannello, garantendo che più sessioni simultanee possano coesistere sulla stessa workstation senza conflitti.

3. **Agilis / XSPMPRIS - Chatbot** (Visual Basic .NET, nodo A) - la *UI*$_G$ del pannello, la lettura del contesto dello *scheduler*, l'anteprima e l'applicazione delle proposte, l'acquisizione e il *refresh* del *token*.
4. **Symposium - *Plugin* MCP** (ASP.NET Core, nodo C) - il *back-end* REST `mcp/*` che interroga il database del gestionale.

Dei quattro, i primi tre sono stati progettati ex-novo nell'ambito di questo lavoro; il quarto estende il *back-end* Symposium preesistente.
Sul nodo A i tre processi sono deliberatamente separati per tre ragioni. Anzitutto l'**isolamento di runtime**: Agilis è *legacy* (.NET Framework / VB), mentre client e server MCP richiedono .NET 10 e Semantic Kernel; processi separati evitano alla radice i conflitti di runtime e di dipendenze, soddisfacendo il vincolo di piena compatibilità con il gestionale (`CMR1`). In secondo luogo il **ciclo di vita gestito**: ogni apertura del pannello avvia un client MCP, che a sua volta avvia il server; la chiusura effettua uno *shutdown* coordinato a cascata che evita i processi orfani. Infine la natura **self-contained** del server MCP, pubblicabile come singolo file e indipendente da un runtime .NET preinstallato sulla workstation.

5.3 Il flusso end-to-end

Si consideri la richiesta *«Pianifica l'attività di collaudo per Marco la prossima settimana»*. Il flusso si svolge così:

1. **Raccolta del contesto.** Agilis raccoglie il messaggio e costruisce il contesto (periodo visibile, risorse e appuntamenti nella *viewport*) ed eventuali allegati estratti in testo.
2. **Invio sulla pipe.** Inoltra la richiesta al client MCP sulla *Named Pipe* (A $\rightarrow$ A).
3. **Inoltro all'LLM.** Il client MCP inoltra la richiesta all'LLM remoto (A $\rightarrow$ B) via Semantic Kernel, in *streaming*.
4. **Decisione e auto-invocazione dei tool.** Guidato dal *System Prompt*, il modello decide quali tool invocare; Semantic Kernel li auto-invoca (ad esempio `get_turni_calendario`, obbligatorio prima di proporre, poi `get_tasks`, `get_risorse`, `get_appuntamenti`).
5. **Esecuzione REST.** Il server MCP esegue ogni tool chiamando l'*endpoint* `mcp/*` di Symposium (A $\rightarrow$ C) con il *Bearer JWT*$_G$ Symposium interroga il database e restituisce JSON.

6. **Produzione della risposta e cattura delle proposte.** L'LLM produce in *streaming* il testo visibile e il ragionamento (*think*) su un canale separato, abilitato dal *flag* `think` nel *payload* della richiesta. A differenza degli approcci classici basati sul *parsing* di un JSON generato a fine risposta, il sistema cattura le alternative di pianificazione in modo reattivo: il modello le dichiara invocando un tool dedicato alla proposta, che funge da validatore deterministico.
7. **Eventi verso Agilis.** Il client MCP invia ad Agilis una sequenza di eventi (*delta*, *thinking*, *tool*, *end*); l'evento finale contiene le alternative strutturate.
8. **Selezione e anteprima.** Agilis mostra le alternative come *card*; selezionandone una, l'anteprima viene applicata allo *scheduler* su oggetti in memoria, senza alcuna scrittura sul database.
9. **Decisione finale.** L'operatore conferma (salvataggio standard di Agilis), annulla (*rollback*) oppure affina la proposta.

5.4 Il modello di interazione AI-uomo: anteprima e conferma

È il cuore concettuale del sistema e la risposta progettuale al vincolo per cui il chatbot non deve mai modificare la pianificazione in autonomia (`CMR3`). L'assistente non scrive mai sul database: produce un *changeset*, ovvero un insieme di operazioni di creazione, aggiornamento o cancellazione sugli appuntamenti, che viene applicato **solo in memoria** come anteprima sullo *scheduler*. Gli elementi modificati sono marcati come proposti dal chatbot e collegati a un identificativo di *changeset*; prima di mutarli viene salvato uno *snapshot* (pattern *Memento*$_G$) che consente il *rollback*.
La conferma non è un'azione speciale dell'AI: è il normale flusso di salvataggio di Agilis, innescato dall'operatore esattamente come per una modifica manuale. L'annullamento ripristina lo *snapshot* pre-modifica. Si ottiene così una netta separazione delle responsabilità: l'AI **propone**, l'operatore **dispone**, e la persistenza resta nel codice collaudato del gestionale. Il pannello è governato da una macchina a stati (Idle → AwaitingResponse → Browsing → Selected → Previewing → Refining) che rende espliciti e mutuamente esclusivi gli stati dell'interazione.

5.5 Sicurezza e ciclo di vita del *token*

I dati di autorizzazione nascono nella sessione Agilis dell'operatore già autenticato e fluiscono lungo la catena di processi. Vale una distinzione netta, dettata da un principio di minimizzazione dei segreti: le **credenziali** (la password) restano nel processo Agilis, dove servono solo al login iniziale verso Symposium, e non vengono mai passate ai processi MCP; l'**identità di tenant** (azienda, ditta, nome utente) viene propagata come variabile d'ambiente al client MCP, che la ribadisce al server, il quale la trasforma in *header* fissi delle chiamate REST, usati da Symposium per il filtro *multi-tenant* che confina ogni query alla

ditta dell'operatore; il ***token* JWT** [16] è l'unico segreto che attraversa i confini di processo, e solo come *Bearer*.
La scadenza del JWT durante una conversazione — la sessione usa *token* a tempo — è gestita con una difesa stratificata, articolata su più livelli: un margine di scadenza che rinnova il *token* poco prima della sua effettiva invalidazione senza attendere l'errore 401; una rilettura del provider che, su 401, ritenta con il *token* più recente già disponibile; un *refresh* reattivo che, riconosciuto un apposito segnale di *token* scaduto, ne richiede uno nuovo ad Agilis e ritenta l'operazione; un *refresh* proattivo che aggiorna il *token* tra un turno e l'altro; e un unico punto di verità che custodisce il *bearer* corrente. La meccanica di dettaglio di questi livelli è descritta nella Sezione 6.

5.6 Pattern Architetturali adottati

I pattern architetturali non sono fini a sé stessi [17]: ciascuno risolve un problema specifico imposto dai vincoli o dai requisiti non funzionali.

- ***Adapter*$_G$.** Il `ChatbotPanelViewModel` non dipende mai direttamente dalle classi concrete del modulo `XSPMPRIS`, ma solo da interfacce: `IPlanningContextProvider` per la lettura del contesto dallo *scheduler*, `IChangesetApplier` per l'applicazione e il *rollback* dell'anteprima, `IAttachmentExtractionService` per l'estrazione del testo dagli allegati, `IChatbotOrchestratorClient` per la comunicazione col client MCP. Questo disaccoppiamento isola il pannello dai dettagli interni del gestionale e rende i componenti sostituibili e testabili indipendentemente.
- ***Decorator*$_G$.** Il `DeferredOrchestrator` (il cui codice è in Appendice B) implementa `IChatbotOrchestratorClient` e avvolge l'orchestratore reale: all'apertura del pannello avvia la costruzione dell'orchestratore in un *task* di *background*, restituendo immediatamente il controllo al *thread* di interfaccia. Quando arriva il primo messaggio, il `DeferredOrchestrator` attende (`await`) il completamento della *factory* e inoltra la richiesta all'orchestratore ormai pronto, senza aver mai bloccato la UI.
- ***Chain of Responsibility*$_G$.** I tre filtri di auto-invocazione registrati nel client MCP formano una catena ordinata: `MaxToolRoundsFilter` (anti-loop) è il più esterno, `ToolCaptureFilter` (osservabilità) è intermedio, `SymposiumAuthRefreshFilter` (*refresh token*) è il più interno. Ciascun filtro decide se inoltrare la chiamata al successivo, modificare il contesto o terminare la catena.
- ***Macchina a stati*$_G$.** Tre automi a stati finiti governano punti distinti del sistema: il `PanelMode` (cfr. Sezione 6.6) disciplina gli stati dell'interazione utente-chatbot (`Idle`, `AwaitingResponse`, `Browsing`, `Selected`, `Previewing`, `Refining`); lo *stripper* dello *stream SSE*$_G$ gestisce i marcatori `thinking` eventualmente spezzati tra *chunk*; il ciclo di vita degli allegati (`InEstrazione`, `Pronto`, `Errore`, `NonSupportato`) isola il *payload* del modello dai fallimenti di estrazione.

Il catalogo completo dei pattern, con la relativa motivazione, è riportato in Appendice A.

5.7 Decisioni architetturali e *trade-off*

Le scelte di fondo del progetto possono essere lette come una sequenza di compromessi consapevoli. La separazione in tre processi sul nodo A isola il runtime .NET 10 dal processo *legacy*, a fronte dell'impossibilità di mantenere modello e orchestratore in-process. La collocazione dell'LLM su un server remoto centralizza GPU e modello e mantiene leggera la workstation, scartando l'opzione del modello locale per il costo hardware su ogni macchina. L'accesso ai dati esclusivamente via REST mantiene i dati dietro le porte già presidiate, evitando di duplicare logica e di scavalcare la sicurezza con un accesso diretto al database. Il principio per cui l'AI non scrive mai sul database, applicando i *changeset* come anteprima in memoria, scarta la scrittura diretta come rischio inaccettabile su un gestionale di produzione. Lo *streaming* a canali separati privilegia una UX_G reattiva rispetto a un'unica risposta a fine elaborazione. Il *refresh* del *token* a più livelli garantisce robustezza alla scadenza del JWT senza interrompere la conversazione. La costruzione *lazy* dell'orchestratore evita il *deadlock* all'apertura del pannello. Infine, il contenimento del contesto alla sola *viewport* evita che il modello ragioni su migliaia di record fuori vista, contenendo costo di *token* e rumore informativo. La tabella sinottica di queste decisioni, con le alternative scartate, è riportata in Appendice A.

5.7.1 Scelta del modello e *trade-off* inferenziali

L'elaborazione semantica e decisionale del sistema è affidata a un modello della famiglia Qwen, selezionato per il suo supporto nativo a una catena di ragionamento esplicito (*thinking*).
Durante la fase di sviluppo, l'adozione di questa architettura ha richiesto una valutazione comparativa tra la variante **Qwen 3.5 a 9 miliardi di parametri (9B)** e **Qwen 3.6 a 27 miliardi di parametri (27B)**. I test hanno evidenziato un *trade-off* netto: nelle prove svolte solo la variante a 27B ha mostrato un'aderenza pressoché completa ai dati recuperati tramite i tool (cfr. requisito `QMR1`), sebbene anche il modello più grande fosse vulnerabile alla generazione manuale di JSON strutturato — problema comune a entrambe le taglie prima dell'introduzione dei tool compositi, come dettagliato nella Sezione 6.9. Tuttavia, l'esecuzione del modello a 27B introduce requisiti hardware e latenze di risposta attualmente incompatibili con l'infrastruttura di test e con i target di reattività del sistema.
Si è pertanto deciso di mantenere la variante 9B come base per lo sviluppo e l'implementazione del prototipo. Poiché il modello 9B, durante elaborazioni prolungate, tende fisiologicamente a sviluppare catene logiche che si autoalimentano sfociando in allucinazioni, si è reso necessario un profondo intervento architetturale sul software circostante per mitigarne i limiti.

5.7.2 Mitigazione delle allucinazioni tramite delega deterministica

La criticità inferenziale del modello 9B è stata affrontata applicando un principio di progettazione rigoroso: la riduzione sistematica del carico operazionale demandato al modello. Ogni complessità deterministica è stata spostata dall'LLM al software circostante.

- **Transizione da JSON a *Tool-Calling* per le proposte.** Inizialmente la creazione delle proposte era demandata interamente al modello, a cui veniva richiesto di scrivere manualmente un blocco JSON testuale nella chat, successivamente parsato dall'interfaccia grafica. Questo approccio risultava fragile. È stato quindi introdotto un *plugin* specifico che permette di registrare la proposta come se fosse un tool a tutti gli effetti, tramite l'inserimento parametrizzato. Il tool associato verifica programmaticamente l'assenza di conflitti in agenda, scartando a monte le proposte invalide. Questo allevia il carico sull'LLM, rendendo il flusso più fluido in quanto il modello non deve più preoccuparsi della formattazione manuale del JSON.
- **Tool ad alta granularità.** Sono stati sviluppati tool complessi (es. calcolo delle disponibilità) che restituiscono risultati finiti, evitando che il modello debba comporre mentalmente più strumenti elementari e dedurne le intersezioni temporali.
- **Gestione e contenimento del contesto.** Il *prompt* è stato arricchito con tecniche *few-shot* (esempi di chiamate), limitando contestualmente il rumore informativo (es. imponendo un tetto massimo di 20 appuntamenti visibili per finestra temporale).

L'insieme di queste strategie ha permesso di abbattere drasticamente il tasso di occorrenza delle allucinazioni, rendendo il prototipo basato su 9B operativo e affidabile per la maggior parte delle casistiche. Tuttavia, per garantire la totale sicurezza decisionale richiesta da un ambiente di produzione, l'adozione del modello a 27B rimane un requisito strutturale imprescindibile.

Capitolo 6

Implementazione

In questo capitolo descrivo come sono state realizzate le scelte architetturali illustrate nel capitolo precedente, organizzando l'esposizione per problema affrontato e riportando i frammenti di codice più significativi; i listati estesi sono raccolti in appendice.

6.1 Le API REST di Symposium per la pianificazione

Il primo strato realizzato è quello dei dati: senza un canale di accesso al patrimonio informativo del gestionale, il modello sarebbe cieco rispetto allo stato reale di risorse, task e agende. L'esposizione avviene tramite un *plugin* REST aggiunto a Symposium, sviluppato ex-novo come contributo di questo lavoro.
Il *plugin* è realizzato come modulo *pluggable*: i *controller* risiedono nel *namespace* `OmegaGruppo.Plugins.MCP.Controllers` e vengono registrati nel contenitore di *dependency injection* di Symposium senza alcuna modifica al codice del *backend* preesistente. Tutti gli *endpoint* condividono il prefisso di rotta `mcp/<Risorsa>` e restano logicamente separati dalle API storiche del gestionale. La superficie esposta copre l'intero dominio della pianificazione — risorse e competenze, task di commessa, calendario lavorativo dei turni, appuntamenti — affiancata da *endpoint* di area gestionale (documenti, ordini, offerte, scadenze, anagrafica, articoli) e da pochi *endpoint* di scrittura (CreaDocumento, CreaOfferta). L'elenco completo è riportato in Appendice B.
È importante notare che il *plugin* è deliberatamente più ampio di quanto il *chatbot* utilizzi: al modello vengono esposti unicamente otto tool di sola lettura legati alla pianificazione (cfr. Sezione 6.4), affiancati da un ***plugin* nativo locale** per la validazione delle proposte. Gli *endpoint* di scrittura restano fuori dal perimetro dell'assistente, coerentemente con il principio per cui l'AI non persiste mai direttamente sul database.
Il secondo aspetto critico è la sicurezza *multi-tenant* e l'isolamento dei dati, che concretizza quanto anticipato nella Sezione 5.5. L'identità di *tenant* non è mai un parametro di *query* controllabile dal chiamante, ma è derivata esclusivamente dal *token*.
La validazione delle richieste entranti è affidata al *middleware* `SympAuthorizeMiddleware`, che intercetta tutte le comunicazioni prima che qualsiasi *controller* venga istanziato. Il *middleware* opera secondo i seguenti passaggi:

1. **Estrazione**: recupera il *Bearer Token*$_G$ dall'header `Authorization` della richiesta HTTP.
2. **Validazione**: verifica la firma crittografica del JWT contro la chiave pubblica di Symposium.
3. **Verifica**: controlla la validità temporale del *token* (claim `exp`).
4. **Estrazione Claim**: ricava i parametri di contesto (`Azienda` e `Ditta`) direttamente dal *payload* del JWT.
5. **Iniezione**: rende disponibile tale contesto ai *controller* tramite il servizio `ICallContextDataService`.

Questa architettura garantisce che il confinamento *multi-tenant* sia applicato *by design* a livello di *middleware*, riducendo drasticamente il rischio di bypass accidentali. A complemento di questa protezione, il confinamento è ulteriormente blindato a livello di esecuzione: ciascun *controller*, prima di interrogare il database, recupera il codice ditta dal contesto e lo applica come filtro obbligatorio in ogni *query*:

```C#
var codditt = _callContextDataService.GetRequestBase().Ditta;
// ...ogni query LINQ vincola i risultati alla ditta dell'operatore:
var risultati = _db.Risorse.Where(og => og.codditt == codditt && /* altri filtri */);
```

6.2 Il server MCP: bootstrap e tool

Il primo problema implementativo è esporre le risorse del gestionale come strumenti invocabili, garantendo che il *token* corretto prevalga sempre. Il server MCP è costruito con `Host.CreateApplicationBuilder` e l'ordine delle fonti di configurazione è deliberato: le variabili d'ambiente prevalgono su `appsettings.json`, perché il *token* aggiornato arriva dall'host come variabile d'ambiente e non deve essere sovrascritto da un eventuale *token* di sviluppo — scaduto — presente nel file di configurazione. In avvio vengono registrati il *provider* del *token* come *singleton* e i tool che espongono le risorse del gestionale; il trasporto è `stdio`.

6.2.1 Bootstrap dettagliato del server MCP

Il server MCP avvia una sequenza di inizializzazione rigorosa:

1. **Configurazione (0-50ms).**
   - Lettura di `appsettings.json` (*endpoint* Ollama, URI Symposium, *tenant* di *default* per lo sviluppo).
   - Sovrascrittura della configurazione tramite variabili d'ambiente (priorità massima).
   - Validazione dei parametri con *fallback* su valori predefiniti in caso di assenza.

2. **Registrazione dei provider (50-100ms).**
   - Iniezione di `SymposiumTokenProvider` nel contenitore di *Dependency Injection* con ciclo di vita *Singleton*.
   - Configurazione del client HTTP e delle logiche di intercettazione e *retry* (es. gestione reattiva dei codici `401 Unauthorized`).
   - Chiusura e validazione del contenitore IoC (*Inversion of Control*).
3. **Registrazione dei Tool (100-200ms).**
   - Registrazione esplicita delle classi dei tool (es. `AppuntamentiTool`, `RisorseTool`).
   - Analisi dei metadati tramite l'attributo `[McpServerTool]`.
   - Generazione della mappa di instradamento (*routing*) e validazione dei parametri contro gli schemi JSON richiesti dal protocollo.
4. **Setup del trasporto (200-250ms).**
   - Apertura dei canali di I/O standard (`stdio`) per la comunicazione bidirezionale con l'host.
   - Avvio del gestore di messaggi (*message pump*) conforme allo standard JSON-RPC 2.0.
   - Registrazione dei gestori per la terminazione sicura del processo (*graceful shutdown*).

L'intero *bootstrap* si completa in meno di 300 millisecondi, garantendo un'apertura quasi istantanea del pannello *chatbot* all'interno del gestionale senza impattare sulla reattività dell'interfaccia utente.

6.2.2 Architettura e tassonomia dei Tool

Ogni tool è una classe con un metodo annotato `[McpServerTool]` e `[Description(...)]` che riceve via *dependency injection* il *resolver* del client REST. Il server espone due tipologie di strumenti. La maggior parte sono tool **pass-through**, che mappano direttamente una risorsa del dominio (appuntamenti, task, risorse, competenze, calendario lavorativo): costruiscono i parametri opzionali, chiamano l'*endpoint* `mcp/<Risorsa>` e ne formattano la risposta. Gli altri sono tool **compositi**, che non restituiscono una risorsa grezza ma un risultato derivato, calcolato in modo deterministico aggregando più fonti — è il caso degli spazi liberi di una risorsa e della ricerca della risorsa disponibile, dettagliati nella Sezione 6.4. Fa eccezione, infine, il tool di *control-plane* per l'aggiornamento del *bearer*, riservato all'host ed escluso dai tool visibili al modello. La tabella completa dei tool e degli *endpoint* corrispondenti è in Appendice B.
Per la visione di un esempio di definizione di tool MCP vedere Codice 8.

6.3 Il client MCP: orchestrazione e catena di filtri

Il client MCP è l'host dell'LLM e il coordinatore dell'intera conversazione. In avvio crea il *Kernel* di Semantic Kernel puntato all'*endpoint* remoto, avvia il server MCP ereditando le variabili d'ambiente di autenticazione, mappa i tool

MCP in funzioni del *Kernel* — escludendo il tool di *control-plane* — e registra i filtri di auto-invocazione.
L'ordine di registrazione è significativo, perché il primo filtro registrato è il più **esterno** della catena (eseguito per primo in ingresso). Si registrano, nell'ordine:

1. `MaxToolRoundsFilter`: un filtro **anti-loop** che termina l'esecuzione al superamento di un numero massimo di round di tool, vedi Codice 9;
2. `ToolCaptureFilter`: un filtro di **osservabilità** che accoda nome, parametri e risultato di ogni invocazione in una coda concorrente (`ConcurrentQueue`) a beneficio dell'interfaccia UI, vedi Codice 10;
3. `SymposiumAuthRefreshFilter`: un filtro che — riconosciuto il segnale di *token* scaduto — ne richiede il rinnovo e ritenta l'operazione. La meccanica di quest'ultimo è dettagliata nella Sezione 6.8.

6.4 Implementazione delle strategie anti-allucinazione e dei *Plugin*

Le tre strategie di delega deterministica introdotte in Sezione 5.7.2 si traducono in altrettanti elementi concreti nell'ambiente Semantic Kernel.

- **Validazione esterna delle proposte.** Il fulcro è il tool `pianifica_e_proponi`, che riceve dal modello un *array* tipizzato di operazioni — crea, modifica, elimina — e ne verifica programmaticamente la consistenza. In presenza di sovrapposizioni tra le operazioni o con l'agenda esistente non registra nulla e restituisce al modello l'elenco degli errori insieme alla lista completa da correggere e ripassare. Solo a validazione superata la proposta viene depositata nel `ProposteCapturePlugin`, un contenitore passivo che l'orchestratore legge a fine turno per emetterla come *card* verso l'interfaccia. La validazione resta così responsabilità del tool, mentre il *plugin* di cattura si limita a trasportare una proposta già verificata.
  Il `ProposteCapturePlugin` è volutamente minimale: espone una lista mutabile di alternative e un metodo `Imposta` che la **sostituisce**, non la estende. Questa scelta fa sì che ogni turno produca esattamente una proposta, mai più d'una:

```C#
public sealed class ProposteCapturePlugin
{
    public List<AlternativaPayload>? Ultime { get; private set; }
    public void Reset() => Ultime = null;
    public void Imposta(List<AlternativaPayload> alternative) =>
        Ultime = alternative ?? new List<AlternativaPayload>();
```

```
7 }
```

Codice 2: Il `ProposteCapturePlugin`: contenitore passivo delle proposte. `Imposta` rimpiazza la lista, non appende.

A valle della validazione, `PianificaPlugin` costruisce una singola `AlternativaPayload` e la deposita:

```
1 var alt = new AlternativaPayload                                    C#
2 {
3     Id = "A",
4     Etichetta = "Pianificazione",
5     Criteri = new List<string> { "operazioni indicate",
      "nessuna sovrapposizione" },
6     Operazioni = payloadOps
7 };
8 proposte.Imposta(new List<AlternativaPayload> { alt });
```

Codice 3: Costruzione della singola proposta in `PianificaPlugin`. L'identificativo `"A"` e la chiamata a `Imposta` con lista di un elemento confermano che viene generata una sola alternativa per turno.

- **Tool ad alta granularità.** I tool `trova_risorsa_disponibile` e `get_spazi_liberi` non sono semplici letture, ma calcoli compositi: invece di esporre al modello gli strumenti elementari (calendario, appuntamenti) lasciandogli dedurre mentalmente le intersezioni temporali — operazione su cui un modello a 9B allucina facilmente — restituiscono fasce orarie libere già calcolate. La logica risiede nel componente `DisponibilitaCore`, interno al server MCP, che recupera dalle API di Symposium il calendario dei turni e gli impegni della risorsa nell'intervallo richiesto e ne calcola la differenza insiemistica. Il metodo `FinestreLavorative` traduce le regole del calendario aziendale in slot lavorativi concreti per ogni giorno del periodo (escludendo giorni non lavorativi e pause); il metodo `Libere` sottrae da queste finestre gli impegni esistenti, con una scansione lineare a cursore, come si vede in Codice 11. Il risultato è un elenco ordinato e privo di sovrapposizioni, restituito al modello come dato finito. Spostando questo calcolo nel software deterministico si elimina alla radice una delle principali fonti di errore aritmetico-temporale del modello, in linea con la strategia di delega descritta nella Sezione 5.7.2.
- **Contenimento dell'esposizione e del contesto.** Dei numerosi *endpoint* disponibili in Symposium sono stati resi invocabili unicamente gli otto tool *core* legati alla pianificazione (cfr. Tabella 13), riducendo la superficie decisionale del modello.

6.4.1 *Plugin* nativi: Knowledge e WebFetch

Oltre ai tool MCP remoti, due *plugin* Semantic Kernel nativi arricchiscono il contesto senza dipendere da Symposium:

- `KnowledgePlugin`. Carica dinamicamente i documenti in formato Markdown presenti nella cartella `Knowledge/` all'avvio del sistema. Questa base di conoscenza, facilmente aggiornabile senza ricompilazione del codice, espone al modello procedure operative e FAQ di Agilis, permettendogli di consultare istruzioni tecniche specifiche durante il ragionamento.
- `WebFetchPlugin`. Fornisce un'interfaccia per il recupero di contenuti testuali da URL remoti. È configurato con un User-Agent standard e un timeout di 30 secondi, ed è progettato per supportare operazioni di ricerca informativa (sfruttando motori come DuckDuckGo Lite) o la consultazione di documentazione web, mantenendo la pulizia del contenuto tramite la rimozione dei *tag* HTML e la normalizzazione del testo.

6.5 L'integrazione lato Agilis: contesto e allegati

Affinché il modello ragioni su dati pertinenti, il lato Agilis deve tradurre lo stato vivo del pianificatore e gli eventuali documenti forniti dall'operatore in un contesto testuale conciso. Questa responsabilità è affidata a due componenti distinti, entrambi agganciati al pannello tramite interfacce (vedi Codice 6), in applicazione del pattern *Adapter* (cfr. Sezione 5.6).

- **Costruzione del contesto.** Il componente `XspmprisContextAdapter` legge dal *ViewModel* dello *scheduler* la finestra temporale effettivamente renderizzata, le risorse visibili e gli appuntamenti che vi ricadono, applicando gli stessi filtri attivi nell'interfaccia (periodo, risorsa selezionata). Il punto delicato è il contenimento del rumore informativo, anticipato nella Sezione 5.7.2: un contesto troppo ampio degrada la qualità delle risposte e gonfia il costo in *token*. Per questo il numero di appuntamenti trasmessi è limitato a un tetto rigido; in caso di superamento, l'eccesso viene troncato in ordine cronologico e la circostanza viene segnalata esplicitamente al modello tramite un *flag* di contesto, così che possa invitare l'operatore a restringere il periodo anziché ragionare su dati parziali in modo silenzioso:

```vb
Const MaxAppuntamentiContesto As Integer = 20
If ctx.AppuntamentiVisibili.Count > MaxAppuntamentiContesto Then
    Dim totale = ctx.AppuntamentiVisibili.Count
    ctx.AppuntamentiVisibili = ctx.AppuntamentiVisibili _
        .OrderBy(Function(a) a.DataInizio) _
        .Take(MaxAppuntamentiContesto).ToList()
    ctx.FiltriAttivi("appuntamentiTroncati") =
```

```
8           $"mostrati {MaxAppuntamentiContesto} di {totale}:
            restringi il periodo o le risorse"
9  End If
```

Codice 4: Tetto al numero di appuntamenti immessi nel contesto, con segnalazione del troncamento al modello.

- **Estrazione degli allegati.** Per soddisfare i requisiti di arricchimento del contesto tramite file, un servizio dedicato (`AttachmentExtractionService`) normalizza documenti eterogenei in testo. Il servizio funge da dispatcher verso estrattori specializzati, selezionati per estensione: i fogli Excel vengono convertiti in tabelle *Markdown* tramite `DevExpress.Spreadsheet` (con un tetto di righe per foglio); i PDF vengono estratti con `DevExpress.Pdf`; le email sono gestite nel formato `.msg` tramite la libreria *MsgReader* e nel formato `.eml` tramite *MimeKit*, producendo un'intestazione strutturata (mittente, destinatari, data, oggetto) seguita dal corpo ripulito dalle citazioni. Per contenere l'occupazione di contesto a circa 50K *token* massimo per turno (empiricamente sufficiente per 5 allegati + contesto visuale + storia conversazionale), ogni allegato è troncato a circa 16K caratteri (circa 4K *token* per allegato). Il numero massimo di allegati è limitato a 5 per turno.
  Una limitazione nota riguarda i PDF privi di livello testuale (documenti acquisiti come immagine): non essendo previsto un livello di riconoscimento ottico dei caratteri, l'estrazione restituisce un esito vuoto e l'allegato viene marcato in errore con un messaggio esplicativo. Ogni allegato attraversa infatti un proprio ciclo di vita (`InEstrazione`, `Pronto`, `Errore`, `NonSupportato`) e solo gli allegati nello stato `Pronto` concorrono alla costruzione del *payload* inviato al modello.
  Per le email, una limitazione importante riguarda gli allegati **incorporati**: l'estrattore elenca i nomi dei file allegati all'email ma non ne estrae il contenuto testuale. Come mostrato nel frammento seguente, il codice per `.msg` (via `MsgReader`) e per `.eml` (via `MimeKit`) itera sugli `Attachments` raccogliendone i nomi, ma non apre né decodifica i file: vedi Codice 12.

6.6 Il modulo chatbot in Agilis: *changeset*, anteprima e macchina a stati

Il lato Agilis del sistema (Visual Basic .NET, nodo A) traduce le proposte in un'anteprima visiva sullo scheduler, senza alcuna scrittura sul database. Il componente centrale è `XspmprisChangesetApplier`, che applica un *changeset* — un insieme di operazioni di creazione, modifica ed eliminazione — agli oggetti già presenti in memoria nel *ViewModel* della timeline.
Ogni elemento toccato viene marcato come proposto dal chatbot (`IsProposedByChatbot`) e collegato all'identificativo del *changeset*, così da essere distinguibile dal programma e ripristinabile in blocco. Il punto delicato

è il *rollback*: la classe `TimelineItem` adotta un *dirty tracking* automatico per cui ogni assegnazione sporca il *flag* `HasChanges`. Per non perdere lo stato originale si applica il pattern **Memento**, salvando uno snapshot pre-modifica (`PreChatbotSnapshot`) prima di mutare l'oggetto; il valore «pulito» di `HasChanges` viene catturato prima della clonazione e riassegnato per ultimo. Vedi Codice 13.
La conferma non è un'azione speciale dell'AI: `ConfermaAnteprima` si limita a scartare lo *snapshot* di *rollback* e a ripulire i *marker*, lasciando `HasChanges = True` affinché la modifica venga persistita dal normale flusso di salvataggio di Agilis. L'annullamento (`AnnullaAnteprima`) percorre la strada inversa: ripristina gli `Update` dal `PreChatbotSnapshot`, rimuove le creazioni (riconoscibili dal `Progr` temporaneo negativo) e reintegra le eliminazioni dalla collezione `DeletedItems`.
L'intera interazione è governata da una macchina a stati esplicita (`PanelMode`) che rende mutualmente esclusivi gli stati del pannello e ne disciplina le transizioni: `Idle`, `AwaitingResponse`, `Browsing`, `Selected`, `Previewing`, `Refining`.

6.7 Gestione e normalizzazione dello streaming

Lo streaming dei *token* dall'*endpoint* remoto pone due problemi: separare il testo destinato all'utente dalla catena di ragionamento (*thinking*), e farlo su un flusso SSE in cui i marcatori possono arrivare spezzati a cavallo di due *chunk*.
La separazione è realizzata da un `DelegatingHandler` HTTP (`OllamaThinkingHandler`) interposto sul client: in fase di richiesta inietta il *flag* `think` per abilitare il *reasoning* del modello; in fase di risposta avvolge lo stream in un `ReasoningToThinkStream` che normalizza l'*output* marcando il ragionamento con *tag* `<think>...</think>`.

C#

```
protected override async Task<HttpResponseMessage>
SendAsync(
    HttpRequestMessage request,
    CancellationToken ct)
{
    if (InjectThink && request.Content is not null &&
    request.Method == HttpMethod.Post)
    {
        var body = await
        request.Content.ReadAsStringAsync(ct);
        if (JsonNode.Parse(body) is JsonObject obj)
        {
            obj["think"] = true;       // abilita il
            reasoning
```

```
                request.Content = new
11              StringContent(obj.ToJsonString(),
                Encoding.UTF8, "application/json");
12          }
13      }
14      var response = await base.SendAsync(request, ct);
15      if (InjectThink && response.Content is not null)
16      {
17          var original = await
            response.Content.ReadAsStreamAsync(ct);
18          response.Content = new StreamContent(new
            ReasoningToThinkStream(original, Debug));
19      }
20      return response;
21  }
```

Codice 5: *Handler* che abilita il *thinking* e normalizza lo *stream* SSE.

A valle, l'orchestratore consuma il flusso normalizzato con una macchina a stati (lo *stripper*) che instrada i frammenti sui canali corretti — testo visibile e ragionamento — gestendo i marcatori `<think>` eventualmente spezzati tra due *chunk* e ripulendo l'*output* da blocchi residui. Da qui nascono gli eventi `delta` e `thinking` inviati ad Agilis. Questa scelta di disaccoppiamento garantisce inoltre l'indipendenza dallo specifico motore: con un *proxy* che espone già i *tool-call* SSE nativi, l'iniezione è disabilitata e il flusso passa invariato.

6.8 Ciclo di vita del *token* in dettaglio

La difesa stratificata anticipata della Sezione 5.5 si concretizza su livelli collocati in punti diversi della catena di processi, secondo il principio di minimizzazione dei segreti: la password non lascia mai Agilis, e solo il JWT attraversa i confini di processo, come *Bearer*.

1. **Margine di scadenza con *fallback*(lato Agilis).** Il connettore verso Symposium rinnova il *token* **prima** della sua invalidazione, senza attendere l'errore 401, sfruttando il *refresh token*. Se anche il *refresh token* risulta scaduto — caso tipico dopo lunghe inattività — il connettore non propaga l'errore: azzera la sessione e ricade su un *login* completo con le credenziali ancora custodite nel processo Agilis, ripristinando in modo trasparente una sessione valida. Vedi Codice 14.
2. **Ritentativo col *token* corrente (lato server MCP).** Prima di arrendersi, il client REST del server è configurato con `WithActionOnUnauthorized`: su 401 rilegge dal provider il *token* più recente disponibile, riconfigura il *bearer* e ritenta la chiamata una volta. Solo se anche questo tentativo fallisce, il

server traduce il 401 in un *Sentinel Value*$_G$ testuale, `__AUTH_EXPIRED`, poiché il canale verso il client trasporta solo stringhe e non può veicolare eccezioni tipizzate. Vedi Codice 15.

3. **Refresh reattivo (lato client).** Un filtro di auto-invocazione (`SymposiumAuthRefreshFilter`) riconosce la *sentinel* nel risultato di un tool, richiede un nuovo *token* ad Agilis attraverso la *pipe*, lo applica al server tramite il tool di *control-plane* `SetSymposiumToken` e ri-invoca l'operazione una sola volta. Un semaforo serializza i 401 paralleli in un unico *refresh*. Vedi Codice 16.
4. ***Refresh* proattivo tra turni e** 5. **unico punto di verità.** Tra un turno e l'altro Agilis può inviare al client un *token* aggiornato, propagato al server con lo stesso tool di *control-plane*; il `SymposiumTokenProvider`, **singleton** *thread-safe* nel server MCP, custodisce l'unica copia mutabile del *bearer* corrente.

La combinazione dei cinque livelli rende la conversazione resiliente alla scadenza del JWT senza mai interromperla.

6.9 Testing e Performance dei Modelli

Al fine di validare il sistema, sono state testate sul campo tutte le funzionalità implementate: la creazione di proposte di pianificazione, l'estrapolazione di dati da file allegati, le domande relative al contesto che l'utente sta visualizzando nell'interfaccia e le domande che richiedono informazioni non immediatamente visibili. Per ciascuna di queste quattro funzionalità sono stati identificati 5 scenari realistici, per un totale di 20 situazioni di prova distinte. Ogni scenario è stato eseguito 10 volte per ogni modello preso in considerazione, per un totale di 200 test complessivi sul modello **Qwen3.5-9B** e 200 test sul modello **Qwen3.6-27B**.
Durante questa fase si è scelto di non misurare i tempi di risposta (ad esempio i millisecondi di latenza), poiché la velocità di esecuzione dipendeva in modo determinante dalla potenza dell'hardware locale che ospitava i modelli. L'analisi si è quindi concentrata sull'affidabilità e sulla correttezza dell'output.

6.9.1 Definizione di errore

Un'esecuzione è stata classificata come *«errata»* al verificarsi di almeno una delle seguenti quattro condizioni:

- **Allucinazioni.** Il modello produce valori o informazioni che appaiono realistici ma non corrispondono ai dati reali presenti nel sistema.
- **Errori di formato nella chiamata dei tool.** Il modello durante la chiamata ai tool esposti non rispetta il formato ed i nomi dei parametri che il tool MCP si aspetta, con la conseguente impossibilità di eseguire correttamente l'operazione. In seguito ad un certo numero di errori di questo tipo, l'LLM procede allucinando e restituendo all'utente valori inventati.

- **Errori nei valori dei parametri passati ai tool.** Stesso problema del punto precedente, ma con parametri correttamente nominati e formattati: il modello passa valori errati, ad esempio date fuori range o codici risorsa inesistenti.
- **Errore nella generazione del JSON di proposta.** Questo solo precedentemente all'introduzione del *plugin* `pianifica_e_proponi`, che ha eliminato la necessità di generare manualmente il JSON di proposta.

Di fatto tutti e quattro questi tipi di errore sono riconducibili a un unico fenomeno: l'allucinazione del modello, che può manifestarsi in modi diversi ma che ha come conseguenza finale la produzione di un output non valido.

6.9.2 Risultati dei test

Nei test iniziali — prima dell'introduzione dei tool compositi e dell'ottimizzazione del *System Prompt* descritte nella Sezione 6.4 — durante operazioni complesse che richiedevano l'invocazione di più tool in sequenza, il modello da 9B tendeva a generare allucinazioni nel 75% dei casi, commettendo errori sia nel formato sia nei valori inseriti nelle richieste ai tool o nella creazione del blocco JSON per le proposte. Anche il modello più avanzato da 27B — nonostante il più elevato numero di parametri, che però lo rendevano estremamente più lento a causa dei limiti fisici dell'hardware ospitante — mostrava un tasso di errore significativo, del 10%, presentando le stesse tipologie di errori del modello da 9B.
Come descritto in precedenza, gran parte del problema derivava dall'assenza iniziale di tool compositi e dal fatto che il modello dovesse generare manualmente un JSON per proporre la pianificazione. Per alleggerire il carico sull'LLM, sono stati introdotti i tool compositi e il *plugin* per registrare le proposte tramite l'inserimento diretto dei parametri.
Tuttavia, nonostante l'introduzione di questo strumento, il modello da 9B mostrava una persistente tendenza a ricadere nel comportamento precedente, continuando in modo persistente a generare codice JSON testuale all'interno della risposta. Per risolvere questa anomalia si è intervenuti radicalmente sul *System Prompt*: è stato introdotto l'obbligo esplicito e stringente di utilizzare unicamente il *plugin* dedicato per le proposte, spiegandone meticolosamente il funzionamento e vietando la generazione di formati non richiesti.
Dopo queste ottimizzazioni — tool compositi, `pianifica_e_proponi` per la validazione esterna delle proposte, e *System Prompt* rafforzato — si sono ripetuti i test eseguiti in precedenza: l'accuratezza del modello da 27B ha raggiunto il 100%, abbassando il tasso di errore allo 0%, mentre l'accuratezza del modello da 9B è salita al 90%, riducendo di molto il tasso d'errore e portandolo al 10%. Sebbene quest'ultimo fatto rappresenti un traguardo tecnico notevole per un modello quantizzato di piccole dimensioni, un tasso di affidabilità del 90% non è ancora ritenuto sufficiente per la messa in produzione, poiché il perimetro applicativo in cui si inserisce questo gestionale esige una precisione assoluta sulle assegnazioni aziendali.

Capitolo 7

Conclusioni

In questo capitolo traggo le conclusioni del lavoro: confronto il piano di lavoro con quanto effettivamente svolto, verifico il grado di raggiungimento degli obiettivi e di copertura dei requisiti, ripercorro i rischi occorsi con le relative mitigazioni e propongo possibili sviluppi futuri.

7.1 Consuntivo finale

Al fine di garantire una maggiore trasparenza e un tracciamento più puntuale delle attività effettivamente svolte, in fase di consuntivazione (Tabella 3) si è scelto di disaggregare due delle macro-fasi originariamente accorpate nel Piano di Lavoro (Tabella 2). Nello specifico, la mappatura tra i due prospetti è la seguente:

- La voce preventiva **Formazione iniziale e studio del sistema** rimane invariata nel consuntivo.
- La voce preventiva **Analisi requisiti e progettazione** è stata esplosa in due fasi distinte: **Analisi dei requisiti** e **Progettazione del sistema**.
- La voce preventiva **Sviluppo del chatbot e integrazione con il pianificatore** rimane invariata nel consuntivo.
- La voce preventiva **Test funzionali e miglioramento del sistema** è stata esplosa in due fasi distinte: **Test funzionali** e **Miglioramento e ottimizzazione del sistema**.
- La voce preventiva **Documentazione tecnica e revisione finale** rimane invariata nel consuntivo.

Il monte ore complessivo di 304 ore previsto per lo stage è stato rigorosamente rispettato; tuttavia, la distribuzione effettiva dell'impegno ha subito variazioni fisiologiche rispetto a quanto preventivato nel Piano di Lavoro iniziale.
Come si evince dal confronto tra la Tabella 2 e la Tabella 3, lo scostamento principale ha riguardato le fasi di sviluppo, test e ottimizzazione del sistema, che hanno richiesto un *effort* congiunto superiore a quanto inizialmente stimato (212 ore effettive contro le 190 preventivate). Questa dilatazione dei tempi è direttamente imputabile alle sfide architetturali e inferenziali emerse in corso d'opera con il modello a 9 miliardi di parametri.
Come analizzato nel sesto capitolo, la marcata tendenza del modello ad allucinare formati e valori durante la generazione di output JSON ha costretto ad

abbandonare tale approccio in favore di una strategia basata su *tool-calling* reattivo. Questa transizione ha comportato lo sviluppo imprevisto di tool compositi e di un *plugin* di registrazione dedicato. Di conseguenza, la successiva fase di test è risultata più onerosa, concentrandosi in larga misura sulla profonda e iterativa ottimizzazione del *System Prompt*, operazione rivelatasi indispensabile per portare l'accuratezza delle risposte al 90%.
Di contro, una fase di formazione iniziale più snella e una stesura della documentazione tecnica più rapida del previsto hanno permesso di compensare l'assorbimento orario dello sviluppo, consentendo di chiudere il progetto esattamente nel tetto delle 304 ore stabilite.

Fase Preventivata	**Ore**
Formazione iniziale e studio del sistema	30
Analisi requisiti e progettazione	44
Sviluppo del *chatbot* e integrazione con il pianificatore	140
Test funzionali e miglioramento del sistema	50
Documentazione tecnica e revisione finale	40
Totale	**304**

Tabella 2: Monte ore preventivato nel Piano di Lavoro.

Fase Effettiva	**Ore**
Formazione iniziale e studio del sistema	24
Analisi dei requisiti	16
Progettazione del sistema	28
Sviluppo del *chatbot* e integrazione con il pianificatore	150
Test funzionali	22
Miglioramento e ottimizzazione del sistema	40

Fase Effettiva	Ore
Documentazione tecnica e revisione finale	24
Totale	**304**

Tabella 3: Consuntivo orario effettivo a fine progetto.

7.2 Raggiungimento degli obiettivi

Con riferimento agli obiettivi prefissati in fase di avvio e formalizzati nel Piano di Lavoro, è possibile confermare il pieno soddisfacimento di tutti i traguardi obbligatori e il conseguimento parziale dell'obiettivo facoltativo. Occorre precisare che tali traguardi definiscono le finalità formative e macro-architetturali del progetto nel suo complesso, distinguendosi pertanto dai requisiti software di dettaglio analizzati nella Sezione 3.
La Tabella 4 riassume lo stato di completamento per ciascun obiettivo identificato.

Codice	Obiettivo	Esito
O01	Comprendere parte dell'architettura del gestionale Agilis	Soddisfatto
O02	Progettare e sviluppare un server MCP funzionante	Soddisfatto
O03	Implementare la comunicazione con il sistema LLM	Soddisfatto
O04	Realizzare funzionalità di accesso ai dati del gestionale	Soddisfatto
O05	Documentare il sistema sviluppato	Soddisfatto
O06	Implementare logiche di analisi dei dati e suggerimento di azioni	SSoddisfattoì

Codice	Obiettivo	Esito
O07	Strutturare il codice secondo buone pratiche di sviluppo	Soddisfatto
F01	Estendere il sistema a nuovi casi d'uso applicativi	Soddisfatto parzialmente

Tabella 4: Riepilogo del raggiungimento degli obiettivi di progetto.

Sul piano implementativo, gli obiettivi legati alla comprensione dell'infrastruttura (**O01**) e all'accesso ai dati (**O04**) si sono tradotti nella proficua integrazione con il modulo `XSPMPRIS` e nello sviluppo del nuovo strato di API REST all'interno di Symposium. Parallelamente, l'esigenza di stabilire un canale di comunicazione sicuro e isolato con il modello linguistico (**O03**) è stata soddisfatta attraverso la realizzazione del server Model Context Protocol (**O02**) e l'adozione del framework Microsoft Semantic Kernel come motore di orchestrazione.
Il requisito relativo all'implementazione di logiche di supporto decisionale (**O06**) è stato assolto delegando le criticità deterministiche al software circostante: l'impiego di tool ad alta granularità e il meccanismo transazionale di anteprima in memoria prevengono infatti alla radice l'insorgenza di conflitti e allucinazioni in fase di pianificazione. Inoltre, l'intero ciclo di vita del software è stato guidato da rigorosi principi architetturali (**O07**), impiegando design pattern consolidati (quali *Adapter*, *Memento*, *Decorator* e *Chain of Responsibility*) per disaccoppiare nativamente i nuovi moduli dalla base di codice *legacy*; il percorso analitico e progettuale è stato infine documentato in dettaglio all'interno di un elaborato lasciato a disposizione dell'azienda per un possibile futuro sviluppo. (**O05**).
L'unico traguardo considerato raggiunto in maniera «Parziale» è l'estensione del sistema a nuovi casi d'uso (**F01**). Come anticipato nel sesto capitolo, l'infrastruttura di back-end necessaria per estendere l'operatività del chatbot ad altre aree del gestionale (ad esempio ordini, fatturazione o anagrafica clienti) è stata interamente predisposta, pubblicando i relativi *endpoint* in Symposium. Tuttavia, i limiti cognitivi e la sensibilità al rumore informativo del modello quantizzato a 9 miliardi di parametri hanno sconsigliato l'inclusione massiva di tali strumenti all'interno del *System Prompt* nell'attuale iterazione del prototipo. La piena applicazione di questo traguardo è pertanto rimandata a un futuro potenziamento dell'infrastruttura hardware, che renderà sostenibile l'esecuzione in produzione del più robusto e flessibile modello da 27B.

7.3 Requisiti soddisfatti

La Tabella 5 riassume il grado di copertura dei requisiti al termine del progetto. Tutti i **25 requisiti mandatory** — 17 funzionali, 4 qualitativi e 4 di vincolo

— sono stati integralmente soddisfatti, a conferma che il nucleo irrinunciabile del sistema è stato realizzato in ogni sua parte.

Sul fronte funzionale, il bilancio dei requisiti **desirable** è parziale (7 su 11). Sono soddisfatti l'anteprima non distruttiva nel pianificatore con *rollback* via pattern *Memento* (`XspmprisChangesetApplier`, `FDR4`), la visualizzazione dell'elenco delle operazioni previste (`FDR5`), l'accettazione o il rifiuto selettivo delle singole operazioni prima dell'applicazione (`FDR6`), l'annullamento dell'anteprima con ripristino dello stato precedente (`FDR7`), e i due requisiti di trasparenza: il canale di *thinking* che espone il ragionamento in tempo reale (`FDR9`) e il `ToolCaptureFilter` che registra nome, parametri ed esito di ogni strumento invocato (`FDR10`). È inoltre soddisfatta l'interruzione dell'elaborazione in corso (`FDR11`), realizzata tramite cancellazione cooperativa end-to-end: `ChatbotPanelViewModel.Annulla()` invia un messaggio `kind="cancel"` sulla *pipe*, il `PipeServer` cancella il `CancellationTokenSource` del turno corrente, e `StreamTurnAsync` propaga l'`OperationCanceledException` producendo l'evento «Turno annullato».

Quattro requisiti **desirable** non sono stati soddisfatti nell'attuale iterazione del prototipo. `FDR1` (estrazione dei contenuti dagli allegati incorporati nelle email) non è implementato: il `MsgAttachmentExtractor` elenca i nomi degli allegati ma non ne estrae il contenuto testuale, limitandosi a includere il corpo dell'email e i nomi dei file allegati nel contesto del modello. `FDR2` (generazione di più alternative di pianificazione) e `FDR3` (confronto e selezione tra alternative) non sono soddisfatti perché il `PianificaPlugin` produce esattamente una proposta per turno: il metodo `ProposteCapturePlugin.Imposta()` rimpiazza la lista anziché accodare, e il costruttore della risposta in `PianificaPlugin` crea una singola `AlternativaPayload`. Senza alternative multiple, anche la selezione (`FDR3`) è degenere: l'operatore può solo accettare o rifiutare l'unica proposta generata, al massimo scegliendo quali operazioni applicare. `FDR8` (simulazione degli effetti di modifiche ipotetiche alla pianificazione) non è stato implementato. La ragione è architetturale, non omissiva: la strategia anti-allucinazione (Sezione 5.7.2) affida ogni operazione di pianificazione al tool deterministico `pianifica_e_proponi`, che produce **sempre** un *changeset* reale, convalidato contro i conflitti in agenda e pronto per l'anteprima. Non esiste un tool separato per scenari ipotetici, né il *plugin* di proposta supporta una modalità «sola analisi». Questa scelta è coerente con il principio per cui l'LLM non deve mai ragionare su dati che non siano stati verificati dal software deterministico: una simulazione senza effetti sarebbe un ragionamento puramente statistico del modello, esposto al rischio di allucinazioni che l'architettura mira a prevenire.

I due requisiti funzionali **optional** — `FOR1` e `FOR2`, relativi all'apertura dell'agenda di una risorsa tramite comando in chat e alla relativa disambiguazione — non sono stati implementati. L'infrastruttura di base (invocazione di programmi Agilis via *named pipe*) è predisposta, ma il caso d'uso specifico di apertura dell'agenda non è stato attivato nell'attuale iterazione del prototipo.

Sul fronte qualitativo, tutti e quattro i requisiti **mandatory** e due dei tre **desirable** sono soddisfatti. L'unico requisito qualitativo non coperto è `QDR2`, che prescrive la distinguibilità visiva degli appuntamenti proposti dal *chatbot* rispetto a quelli già presenti nel pianificatore. La proprietà `IsProposedByChatbot` esiste come flag dati sulla classe `TimelineItem` ed è correttamente impostata dal `XspmprisChangesetApplier` durante l'anteprima, ma non è agganciata ad alcun converter XAML o stile grafico: il `AppointmentItemBorderConverter`, unico converter che agisce sull'aspetto degli appuntamenti nello *scheduler*, verifica esclusivamente la proprietà `DataVincolata`. L'effetto visivo dell'anteprima è limitato alla comparsa e scomparsa degli elementi creati o eliminati; gli appuntamenti modificati non presentano alcuna differenziazione cromatica o grafica rispetto a quelli non toccati dalla proposta.

Tutti e 4 i **constraint** sono pienamente rispettati: lo sviluppo in .NET (`CMR1`) è attestato dai progetti .NET 10 di client e server MCP e dal progetto Visual Basic .NET del modulo `XSPMPRIS.Chatbot`; l'integrazione con Symposium (`CMR2`) è realizzata dal *plugin* REST `OmegaGruppo.Plugins.MCP`; il divieto di modifiche autonome alla pianificazione (`CMR3`) è garantito dal modello ad anteprima e conferma, in cui l'AI propone un *changeset* applicato solo in memoria e l'operatore dispone il salvataggio; l'architettura modulare (`CMR4`) è conseguita tramite l'isolamento in tre processi separati, l'uso di interfacce per disaccoppiare il pannello *chatbot* dal *ViewModel* dello *scheduler*, e la catena di filtri (`MaxToolRoundsFilter`, `ToolCaptureFilter`, `SymposiumAuthRefreshFilter`) per le responsabilità trasversali.

Complessivamente, il sistema copre **34 requisiti su 41** individuati in fase di analisi, con una copertura completa dei mandatory e un tasso di soddisfacimento dei desirable pari al 64%.

Tipo	Mandatory	Desirable	Optional	Somma
Functional	17/17	7/11	0/2	24/30
Qualitative	4/4	2/3	0/0	6/7
Constraint	4/4	0/0	0/0	4/4
Totale	**25/25**	**9/14**	**0/2**	**34/41**

Tabella 5: Riepilogo dei requisiti soddisfatti.

7.4 Prodotto sviluppato

In Figura 11 e in Figura 12 si può osservare l'interfaccia grafica sviluppata, la sua integrazione nella sezione dedicata alla gestione delle attività aziendali ed un esempio di richiesta di pianificazione in tutte le sue fasi.

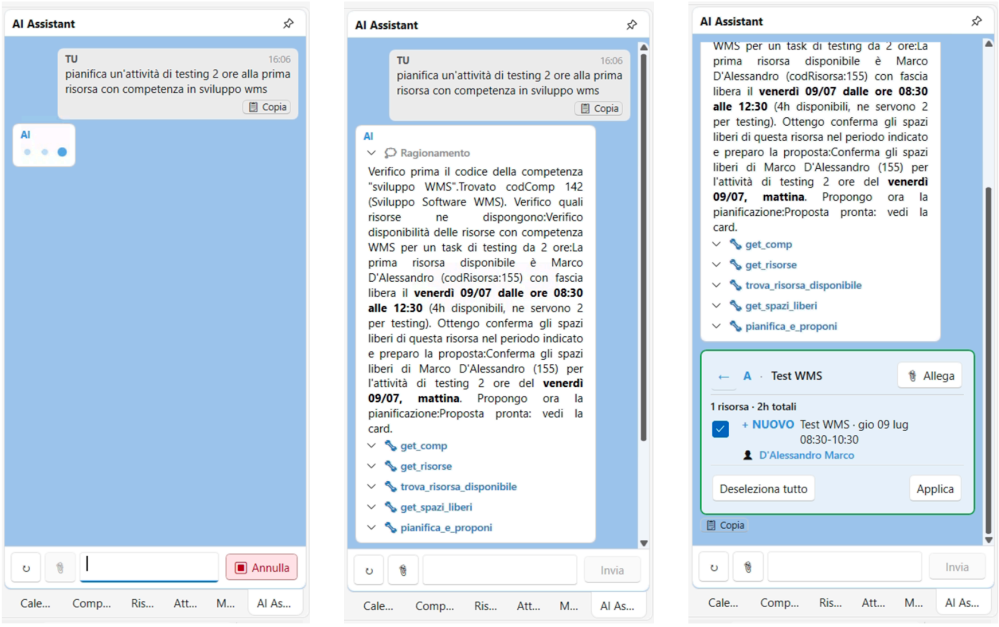


Figura 11: Interazione utente-chatbot durante l'invio di una richiesta di pianificazione.

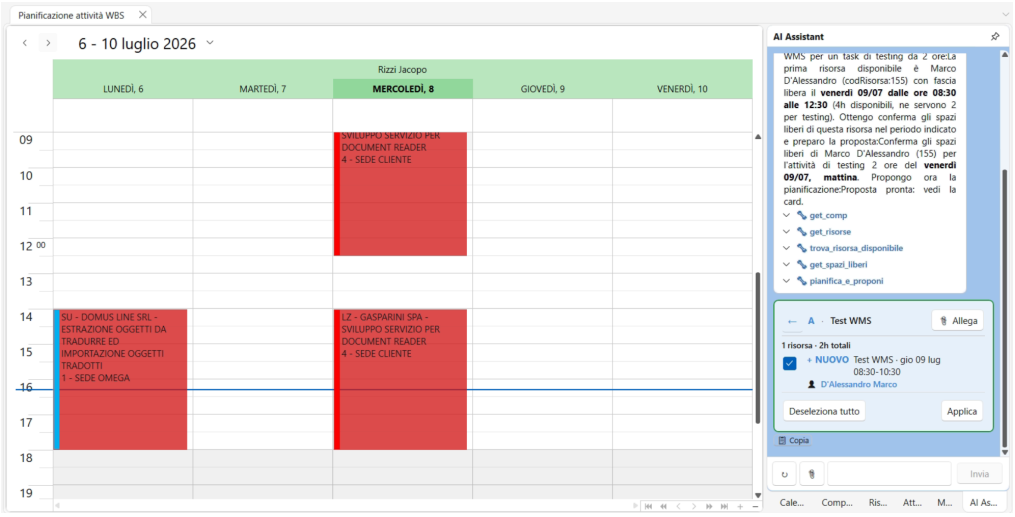


Figura 12: Interfaccia integrata: chatbot a destra e calendario attività ed appuntamenti a sinistra.

7.5 Rischi occorsi e mitigati

I rischi emersi durante lo stage sono riportati in Tabella 6.

Descrizione	Mitigazione
R1 – Complessità architettura *legacy* di Agilis e difficoltà di orientamento nel dominio applicativo.	Confronti periodici con i colleghi per approfondire i flussi logici e il patrimonio informativo aziendale.
R2 – Difficoltà di apprendimento nell'implementazione delle nuove tecnologie (MCP, Semantic Kernel, .NET 10).	Supporto del tutor, ricerca documentale autonoma e utilizzo mirato dell'IA per l'analisi dei punti critici.
R3 – Imprevisti medici e difficoltà logistiche negli spostamenti verso la sede aziendale.	Rimodulazione delle presenze in accordo con l'azienda e attivazione dello smart working parziale per rispettare le scadenze.

Tabella 6: Rischi occorsi con la loro mitigazione.

7.6 Sviluppi futuri e messa in produzione

Il lavoro svolto ha dimostrato l'efficacia di un'architettura ad agenti integrata nel gestionale Agilis. Grazie all'ottimizzazione estrema dei tool e del contesto, il sistema riesce a operare con un modello quantizzato a 9 miliardi di parametri mantenendo un tasso di allucinazioni estremamente contenuto. Tuttavia, permane un margine fisiologico di errore intrinseco alla taglia del modello. Il principale sviluppo futuro in vista di un rilascio in produzione consiste nell'adeguamento dell'infrastruttura hardware (Nodo B), al fine di ospitare definitivamente il modello Qwen 3.6 a 27 miliardi di parametri, il quale, a seguito delle ottimizzazioni, ha offerto un'accuratezza totale del 100% nelle prove svolte. Un ulteriore sviluppo previsto è l'applicazione di questo assistente anche in altre pagine del gestionale Agilis, come ad esempio quelle relative agli ordini o ai rapporti con i clienti. A tal fine, su precisa indicazione del tutor aziendale in previsione di espansioni future, sono già stati sviluppati all'interno di Symposium numerosi *endpoint* dedicati, che attualmente giacciono inutilizzati nell'architettura.
La limitazione attuale all'implementazione di queste nuove funzionalità è data dal fatto che il modello da 9B non è in grado di destreggiarsi agilmente con un numero così elevato di strumenti. Come emerso durante i test, affinché il modello interpreti correttamente i tool, necessita di istruzioni precise e dettagliate

all'interno del *System Prompt*. L'aggiunta di nuovi strumenti renderebbe la dimensione del prompt troppo grande, impedendo a un modello di piccola taglia di muoversi e rispondere in maniera rapida e corretta. L'adozione del modello da 27B risulta pertanto la condizione tecnica necessaria non solo per garantire l'affidabilità, ma anche per permettere l'espansione operativa del sistema.

7.7 Valutazione personale

L'attività di stage si è rivelata estremamente formativa per la mia crescita personale e professionale, consentendomi di acquisire competenze in ambiti che il percorso accademico aveva solo parzialmente toccato, come l'ingegneria dei sistemi a linguaggio naturale, l'orchestrazione di agenti AI e l'integrazione di protocolli emergenti in contesti enterprise consolidati.
L'approfondimento del Model Context Protocol e del framework Semantic Kernel ha rappresentato una sfida stimolante: ho dovuto confrontarmi con tecnologie di frontiera in un ecosistema .NET caratterizzato da una forte eterogeneità di runtime (.NET Framework 4.8 *legacy* e .NET 10 moderno), imparando a gestire il disaccoppiamento architetturale attraverso processi isolati e canali di comunicazione inter-processo. La necessità di progettare un sistema robusto nonostante i vincoli imposti da un gestionale maturo, con il suo patrimonio informativo storicizzato e le sue logiche di business consolidate, mi ha insegnato l'importanza di un approccio pragmatico che bilanci innovazione e stabilità operativa.
Particolarmente significativa è stata la progettazione delle strategie anti-allucinazione: spostare la complessità deterministica dall'LLM al software circostante, attraverso tool ad alta granularità e validatori esterni, ha trasformato il mio modo di intendere il ruolo dell'intelligenza artificiale in un sistema produttivo. Non più un oracolo da interrogare, ma un agente specializzato le cui decisioni sono guidate e verificate da strati software deterministici, in un modello di interazione in cui l'AI propone e l'operatore dispone. Questa visione ha richiesto un'attenzione meticolosa alla progettazione delle interfacce e alla gestione degli stati, affinché la trasparenza del ragionamento e la tracciabilità degli strumenti invocati fossero sempre accessibili all'utente finale.
La collaborazione con il team aziendale e il tutor Marco Ortali è stata parte integrante del percorso: il confronto continuo sulle scelte architetturali - dalla separazione dei processi alla gestione del ciclo di vita del *token*, dalla limitazione del contesto alla viewport alla validazione esterna delle proposte - mi ha permesso di comprendere come le decisioni tecniche siano sempre il frutto di compromessi tra esigenze funzionali, vincoli infrastrutturali e requisiti di sicurezza. L'azienda mi ha concesso ampia autonomia organizzativa e libertà di sperimentazione, pur fornendo un supporto costante e puntuale, elemento che ha reso l'esperienza particolarmente arricchente.
Il progetto rappresenta per me un ponte tra la teoria accademica e la pratica professionale: ho potuto applicare nozioni di progettazione software, pattern architetturali e gestione della sicurezza a un caso d'uso concreto, con ripercussioni operative in un ambiente di produzione reale. L'approccio alla mitigazione delle

allucinazioni tramite delega deterministica e il modello di anteprima-conferma per la pianificazione sono esempi di come si possano coniugare le potenzialità dell'AI con l'affidabilità richiesta da un gestionale aziendale.
Quanto appreso — dal funzionamento interno dei modelli linguistici alle tecniche di orchestrazione, dalla gestione dei *token* di autenticazione alla normalizzazione dello streaming — costituisce una solida base per il mio futuro professionale, in un settore, l'*AI engineering*, in rapidissima evoluzione e di grande interesse per possibili sviluppi accademici e lavorativi. Questa esperienza ha affinato il mio metodo di studio e la mia capacità di affrontare progetti complessi, rafforzando in me la consapevolezza che la vera innovazione sta spesso nel saper integrare tecnologie nuove in contesti consolidati, con un'attenzione costante alla sicurezza, all'affidabilità e all'esperienza dell'utente finale.

Appendice A

Requisiti completi e decisioni di progetto

A.1 Lista delle user stories

Epic 1. Gestione del contesto

US1 Allegare un file di specifica task

Descrizione: Come operatore del reparto *Delivery*, voglio poter allegare un file contenente le informazioni dei task da pianificare, in modo da fornire al chatbot il contesto necessario per elaborare proposte di pianificazione.

Acceptance criteria:
1. L'operatore può selezionare un file tramite la funzionalità di aggiunta allegato.
2. In alternativa, l'operatore può trascinare il file nell'area dedicata alla chat.
3. Il file viene aggiunto correttamente al contesto a disposizione del modello.

US2 Allegare una conversazione Outlook

Descrizione: Come operatore del reparto *Delivery*, voglio poter allegare una conversazione email proveniente da Outlook, in modo da includere nel contesto del chatbot le informazioni contenute nella conversazione e nei relativi allegati.

Acceptance criteria:
1. L'operatore può trascinare la conversazione Outlook nell'area dedicata alla chat.
2. La conversazione e gli eventuali allegati vengono aggiunti correttamente al contesto del modello.

Epic 2. Pianificazione dei task

US3 Interrogare le informazioni a schermo

Descrizione: Come operatore del reparto *Delivery*, voglio poter porre domande al chatbot riguardo alle informazioni attualmente visualizzate nel pianificatore,

in modo da ottenere chiarimenti o analisi senza dover navigare manualmente tra i dati.

Acceptance criteria:
1. L'operatore può inserire in chat una domanda relativa ai task o agli appuntamenti visibili a schermo.
2. Il chatbot risponde in modo coerente con le informazioni visualizzate.

US4 Specificare la priorità dei task

Descrizione: Come operatore del reparto *Delivery*, voglio poter indicare esplicitamente il livello di priorità dei task da pianificare, in modo da orientare il chatbot verso proposte che rispettino l'ordine di importanza da me definito.

Acceptance criteria:
1. L'operatore può specificare le priorità dei task tramite testo nel messaggio oppure tramite file allegato.
2. Le informazioni di priorità vengono incluse nel contesto del modello e considerate nella generazione delle proposte.

US5 Indicare la strategia di pianificazione

Descrizione: Come operatore del reparto *Delivery*, voglio poter specificare la strategia da adottare nella pianificazione dei task, in modo da ottenere proposte allineate alle esigenze operative del momento.

Acceptance criteria:
1. L'operatore può descrivere nel prompt la strategia di pianificazione desiderata.
2. La strategia viene inclusa nel contesto del modello e applicata nella generazione delle proposte.

US6 Richiedere la pianificazione dei task

Descrizione: Come operatore del reparto *Delivery*, voglio poter richiedere al chatbot una proposta di pianificazione per i task forniti, in modo da ricevere suggerimenti di allocazione temporale senza doverli elaborare manualmente.

Acceptance criteria:
1. L'operatore può fornire i task tramite file allegato o descrizione testuale nel messaggio.
2. L'operatore può opzionalmente specificare priorità e strategia di pianificazione.
3. Il chatbot genera e visualizza in chat una o più proposte di pianificazione.

US7 Affinare una proposta di pianificazione

Descrizione: Come operatore del reparto *Delivery*, voglio poter richiedere al chatbot di affinare una proposta già generata fornendo indicazioni aggiuntive, in modo da ottenere una soluzione più aderente alle mie esigenze specifiche.

Acceptance criteria:
1. L'operatore può inserire nel prompt indicazioni correttive o preferenze aggiuntive rispetto a una proposta precedente.
2. Il chatbot genera una nuova proposta tenendo conto delle indicazioni fornite.

US8 Approvare e inserire la pianificazione proposta

Descrizione: Come operatore del reparto *Delivery*, voglio poter approvare una proposta di pianificazione generata dal chatbot e inserirla direttamente nel gestionale, in modo da evitare l'inserimento manuale degli appuntamenti.

Acceptance criteria:
1. L'operatore può selezionare la funzionalità di inserimento appuntamenti a partire da una proposta approvata.
2. Gli appuntamenti vengono creati correttamente nel gestionale e nel database.

US9 Confrontare le alternative proposte

Descrizione: Come operatore del reparto *Delivery*, voglio poter confrontare le diverse alternative di pianificazione generate dal chatbot, in modo da scegliere quella più adatta alle mie esigenze.

Acceptance criteria:
1. Il chatbot presenta le alternative come elementi distinti e selezionabili, ciascuno con i relativi punti di forza e compromessi.
2. L'operatore può selezionare un'alternativa per visualizzarne nel dettaglio le operazioni previste.

US10 Visualizzare l'anteprima di una proposta nel pianificatore

Descrizione: Come operatore del reparto *Delivery*, voglio poter visualizzare l'anteprima di una proposta direttamente nel pianificatore prima di confermarla, in modo da valutarne l'effetto sull'agenda.

Acceptance criteria:
1. Selezionando una proposta, le operazioni vengono mostrate in anteprima nel pianificatore **senza** essere salvate nel gestionale.
2. Gli appuntamenti coinvolti sono distinguibili visivamente come proposti dal chatbot.

US11 Accettare o escludere singole operazioni di una proposta

Descrizione: Come operatore del reparto *Delivery*, voglio poter accettare o escludere le singole operazioni che compongono una proposta, in modo da applicare solo le modifiche che ritengo corrette.

Acceptance criteria:
1. L'operatore può visualizzare l'elenco delle operazioni della proposta (creazioni, spostamenti, eliminazioni).
2. L'operatore può accettare o rifiutare ciascuna operazione individualmente prima dell'applicazione.

US12 Annullare l'anteprima di una proposta

Descrizione: Come operatore del reparto *Delivery*, voglio poter annullare l'anteprima di una proposta applicata al pianificatore, in modo da ripristinare lo stato precedente senza conseguenze.

Acceptance criteria:
1. L'operatore può annullare un'anteprima attiva.
2. Il pianificatore ripristina lo stato esistente prima dell'applicazione dell'anteprima.

Epic 3. Creazione guidata degli appuntamenti

US13 Creare un appuntamento in modo guidato

Descrizione: Come operatore del reparto *Delivery*, voglio poter richiedere al chatbot la creazione di un nuovo appuntamento fornendo le informazioni necessarie in linguaggio naturale, in modo da velocizzare l'inserimento senza dover compilare manualmente i campi del gestionale.

Acceptance criteria:
1. L'operatore può descrivere in chat l'appuntamento da creare, specificando le informazioni rilevanti.
2. Il sistema inserisce l'appuntamento nel gestionale e nel database.
3. In caso di conflitto (cliente già impegnato o risorsa non disponibile), il sistema notifica l'operatore.
4. L'operatore può scegliere se procedere comunque, modificare le informazioni o annullare l'operazione.

Epic 4. Riorganizzazione delle agende

US14 Richiedere la riorganizzazione delle agende

Descrizione: Come operatore del reparto *Delivery*, voglio poter segnalare al chatbot il verificarsi di un evento imprevisto e richiedere una proposta

di riorganizzazione degli appuntamenti, in modo da adattare rapidamente la pianificazione alla nuova situazione.

Acceptance criteria:
1. L'operatore può descrivere in chat l'evento imprevisto (risorsa indisponibile, attività urgente, modifica della durata di un task).
2. Il chatbot genera e visualizza in chat una o più proposte di riorganizzazione degli appuntamenti.

US15 Simulare scenari di riorganizzazione

Descrizione: Come operatore del reparto *Delivery*, voglio poter richiedere al chatbot di valutare l'effetto di modifiche alla pianificazione, in modo di farmi un'idea delle conseguenze prima di applicarle.

Acceptance criteria:
1. L'operatore può descrivere in chat una o più modifiche ipotetiche alla pianificazione.
2. Il chatbot illustra l'effetto delle modifiche sulla pianificazione corrente e segnala le eventuali criticità che rileva (ad esempio sovrapposizioni o vincoli temporali a rischio).

US16 Approvare ed eseguire la riorganizzazione

Descrizione: Come operatore del reparto *Delivery*, voglio poter approvare una proposta di riorganizzazione delle agende e applicarla al gestionale, in modo da rendere effettive le modifiche senza doverle inserire manualmente.

Acceptance criteria:
1. L'operatore può selezionare la funzionalità di modifica delle agende a partire da una proposta di riorganizzazione approvata.
2. Gli appuntamenti vengono aggiornati correttamente nel gestionale e nel database.

US17 Ricevere notifica di conflitto non risolvibile

Descrizione: Come operatore del reparto *Delivery*, voglio essere notificato dal chatbot quando viene rilevato un conflitto che non può essere risolto automaticamente, in modo da poter intervenire manualmente con le informazioni necessarie.

Acceptance criteria:
1. Quando il chatbot rileva un conflitto non risolvibile, visualizza in chat un avviso esplicito.
2. L'avviso descrive la natura del conflitto in modo comprensibile, permettendo all'operatore di prendere una decisione informata.

Epic 5. Informazioni e supporto generale

US18 Richiedere informazioni generiche

Descrizione: Come operatore del reparto *Delivery*, voglio poter richiedere al chatbot informazioni su risorse, task, appuntamenti e competenze, in modo da accedere rapidamente ai dati di cui ho bisogno senza dover navigare manualmente nel gestionale.

Acceptance criteria:
1. L'operatore può inserire in chat una domanda relativa a risorse, task, appuntamenti o competenze.
2. Il chatbot recupera e visualizza in chat le informazioni richieste in modo accurato e tempestivo.

US19 Aprire l'agenda di una risorsa

Descrizione: Come operatore del reparto *Delivery*, voglio poter chiedere al chatbot di aprire l'agenda di una specifica risorsa, in modo da visualizzarne direttamente la pianificazione nel gestionale senza doverla cercare manualmente.

Acceptance criteria:
1. L'operatore può richiedere in chat l'apertura dell'agenda indicando la risorsa di interesse.
2. Il sistema apre nel gestionale la sezione corrispondente all'agenda della risorsa indicata.
3. Qualora la risorsa non sia identificabile univocamente, il chatbot richiede una disambiguazione prima di procedere.

Epic 6. Interazione e trasparenza

US20 Comprendere il ragionamento e le fonti del chatbot

Descrizione: Come operatore del reparto *Delivery*, voglio poter visualizzare il ragionamento seguito dal chatbot e gli strumenti che ha consultato, in modo da comprendere e fidarmi delle proposte generate.

Acceptance criteria:
1. Il chatbot mostra, in un canale dedicato, il proprio ragionamento durante l'elaborazione.
2. Il chatbot indica quali strumenti ha invocato e con quali parametri, rendendo tracciabile la provenienza dei dati.

US21 Interrompere l'elaborazione in corso

Descrizione: Come operatore del reparto *Delivery*, voglio poter interrompere un'elaborazione del chatbot già avviata, in modo da non dover attendere una risposta non più necessaria.

Acceptance criteria:
1. Durante l'elaborazione l'operatore può richiedere l'annullamento del turno.
2. Il sistema interrompe la generazione e torna disponibile per una nuova richiesta.

US22 Avviare una nuova conversazione

Descrizione: Come operatore del reparto *Delivery*, voglio poter azzerare la conversazione corrente, in modo da iniziare una nuova richiesta senza l'influenza del contesto precedente.

Acceptance criteria:
1. L'operatore può avviare una nuova conversazione.
2. Il contesto conversazionale precedente viene azzerato.

US23 Porre domande di *follow-up* mantenendo il contesto

Descrizione: Come operatore del reparto *Delivery*, voglio poter porre domande di *follow-up* mantenendo il contesto della conversazione, in modo da affinare richieste o proposte senza ripetere le informazioni già fornite.

Acceptance criteria:
1. Il chatbot mantiene memoria dei messaggi e delle proposte precedenti nella stessa conversazione.
2. Le risposte successive tengono conto del contesto già stabilito.

A.2 Tracciamento dei requisiti

Ad ogni requisito è associato un codice costruito in base alle sue caratteristiche:

(F/Q/C)(M/D/O)R

F (*Functional*): definisce una funzione di un sistema o dei suoi componenti;
Q (*Qualitative*): rappresentano come il sistema deve essere per soddisfare i requisiti dello stakeholder;
C (*Constraint*): rappresentano dei vincoli o dei limiti che il sistema deve rispettare;

M (*Mandatory*): irrinunciabili per qualcuno degli stakeholder;
D (*Desirable*): non strettamente necessari ma a valore aggiunto riconoscibile;
O (*Optional*): relativamente utili oppure contrattabili anche in fasi avanzate del progetto;

R (*Requirement*): requisito

In Tabella 7, Tabella 8 e Tabella 9 sono riassunti i requisiti e il loro tracciamento con gli use case delineati in fase di analisi.
La colonna «Fonti» indica da quale artefatto il requisito è stato derivato:

- US (User Story): requisiti funzionali estratti dalle user story raccolte durante l'analisi
- Piano di lavoro: requisiti non funzionali e di qualità definiti nel documento di project management
- Riunione col tutor: vincoli tecnici e architetturali emersi dalle riunioni di supervisione

Un requisito può avere più fonti, indicando che più stakeholder convergevano sulla stessa esigenza.

Codice	Descrizione	Fonti
FMR1	Il sistema deve permettere all'operatore di allegare file contenenti informazioni sui task da pianificare, sia tramite selezione che trascinamento nell'area chat.	US1
FDR1	Il sistema deve permettere all'operatore di allegare conversazioni Outlook, includendo nel contesto anche gli eventuali allegati.	US2
FMR2	Il sistema deve permettere all'operatore di interrogare il chatbot sulle informazioni attualmente visualizzate nel pianificatore.	US3
FMR3	Il sistema deve permettere all'operatore di specificare il livello di priorità dei task, tramite testo o file allegato.	US4
FMR4	Il sistema deve permettere all'operatore di indicare la strategia di pianificazione da adottare.	US5
FMR5	Il sistema deve generare e visualizzare in chat una o più proposte di pianificazione a partire dai task e dai parametri forniti dall'operatore.	US6
FMR6	Il sistema deve permettere all'operatore di affinare una proposta di pianificazione già generata, fornendo indicazioni aggiuntive in linguaggio naturale.	US7
FMR7	Il sistema deve permettere all'operatore di approvare una proposta di pianificazione e inserire gli appuntamenti risultanti nel gestionale e nel database.	US8

Codice	Descrizione	Fonti
FDR2	Il sistema deve presentare le proposte alternative come elementi distinti e selezionabili, ciascuno corredato dei relativi punti di forza e compromessi.	US9
FDR3	Il sistema deve permettere all'operatore di selezionare una proposta alternativa per visualizzarne nel dettaglio le operazioni previste.	US9
FDR4	Il sistema deve mostrare in anteprima nel pianificatore le operazioni di una proposta selezionata, senza salvarle nel gestionale.	US10
FDR5	Il sistema deve permettere all'operatore di visualizzare l'elenco delle operazioni che compongono una proposta (creazioni, spostamenti, eliminazioni).	US11
FDR6	Il sistema deve permettere all'operatore di accettare o escludere individualmente le singole operazioni di una proposta prima della loro applicazione.	US11
FDR7	Il sistema deve permettere all'operatore di annullare un'anteprima attiva, ripristinando lo stato del pianificatore precedente alla sua applicazione.	US12
FMR8	Il sistema deve guidare l'operatore nella creazione di un nuovo appuntamento a partire da una descrizione in linguaggio naturale, inserendolo nel gestionale e nel database.	US13
FMR9	Il sistema deve notificare l'operatore in caso di conflitto rilevato durante la creazione di un appuntamento.	US13
FMR10	Il sistema deve consentire all'operatore di scegliere come procedere a fronte di un conflitto in fase di creazione: proseguire, modificare le informazioni o annullare l'operazione.	US13

Codice	Descrizione	Fonti
FMR11	Il sistema deve generare proposte di riorganizzazione delle agende in seguito alla segnalazione di un evento imprevisto da parte dell'operatore.	US14
FDR8	Il sistema deve illustrare l'effetto di modifiche ipotetiche alla pianificazione, segnalando le eventuali criticità rilevate (sovrapposizioni o vincoli temporali a rischio).	US15
FMR12	Il sistema deve permettere all'operatore di approvare una proposta di riorganizzazione e applicare le modifiche al gestionale e al database.	US16
FMR13	Il sistema deve segnalare all'operatore, in linguaggio chiaro e comprensibile, le criticità o i conflitti che non è in grado di risolvere nel rispetto di tutti i vincoli, esponendone la natura per consentire un intervento manuale informato.	US17
FMR14	Il sistema deve rispondere a richieste di informazioni generiche riguardanti risorse, task, appuntamenti e competenze.	US18
FOR1	Il sistema deve permettere all'operatore di richiedere l'apertura dell'agenda di una specifica risorsa, aprendo nel gestionale la sezione corrispondente.	US19
FOR2	Il sistema deve richiedere una disambiguazione quando la risorsa indicata per l'apertura dell'agenda non è identificabile univocamente.	US19
FDR9	Il sistema deve mostrare, in un canale dedicato, il ragionamento seguito durante l'elaborazione.	US20
FDR10	Il sistema deve indicare gli strumenti invocati e i relativi parametri, rendendo tracciabile la provenienza dei dati.	US20
FDR11	Il sistema deve permettere all'operatore di interrompere un'elaborazione in corso, tornando disponibile per una nuova richiesta.	US21

Codice	Descrizione	Fonti
FMR15	Il sistema deve permettere all'operatore di avviare una nuova conversazione, azzerando il contesto conversazionale precedente.	US22
FMR16	Il sistema deve mantenere il contesto della conversazione corrente e tenerne conto nelle risposte successive.	US23
FMR17	Il sistema deve mettere a disposizione del chatbot una base di conoscenza interna (istruzioni operative e FAQ sull'utilizzo del gestionale) consultabile durante l'elaborazione, affinché possa interpretare correttamente i dati ed eseguire le operazioni richieste.	Piano di lavoro.

Tabella 7: Tracciamento dei requisiti funzionali.

Codice	Descrizione	Fonti
QMR1	Le risposte del chatbot devono essere coerenti con il contesto fornito dall'operatore e con le informazioni presenti nel gestionale.	Piano di lavoro.
QMR2	Il sistema deve notificare l'operatore in modo chiaro e comprensibile in caso di conflitti o situazioni anomale, senza ricorrere a linguaggio tecnico.	Piano di lavoro.
QMR3	Le operazioni approvate dall'operatore devono essere riportate in modo consistente sia nel gestionale sia nel database.	US8
QMR4	L'interfaccia del chatbot deve essere integrata nel gestionale Agilis in modo visivamente coerente con lo stile grafico esistente.	Piano di lavoro.
QDR1	Il sistema deve rispondere alle richieste dell'operatore in tempi ragionevoli, senza introdurre latenze significative nel flusso di lavoro.	Piano di lavoro.

Codice	Descrizione	Fonti
QDR2	Gli appuntamenti mostrati in anteprima devono essere visivamente distinguibili da quelli già presenti nel pianificatore.	US10
QDR3	Il sistema deve garantire la tracciabilità delle proprie elaborazioni, rendendo evidente all'operatore l'origine delle informazioni utilizzate.	US20

Tabella 8: Tracciamento dei requisiti di qualità.

Codice	Descrizione	Fonti
CMR1	Il sistema deve essere sviluppato utilizzando il framework .NET, garantendo la piena compatibilità con il gestionale Agilis.	Riunione col tutor.
CMR2	Il sistema deve integrarsi con Symposium per l'esposizione delle API e la consultazione dei dati nel database.	Riunione col tutor.
CMR3	Il chatbot non deve mai eseguire modifiche alla pianificazione in modo autonomo: ogni operazione deve essere esplicitamente approvata dall'operatore.	Piano di Lavoro.
CMR4	Il sistema deve essere sviluppato con un'architettura modulare, facilmente estendibile e manutenibile.	Riunione col tutor.

Tabella 9: Tracciamento dei requisiti di vincolo.

A.3 Catalogo dei pattern architetturali

Pattern	Dove	Problema risolto
Adapter	Adattatori di contesto/ changeset, orchestratore	Disaccoppia il pannello chatbot da XSPMPRIS; il *ViewModel* dipende solo da interfacce.
Facade	Modulo chatbot	Nasconde l'assemblaggio di adapter, orchestratore, *ViewModel* e *view* dietro factory statiche.
Decorator	`DeferredOrchestrator`	Costruisce l'orchestratore reale in *background*; evita il *deadlock* sync-su-async sul thread UI.
Proxy remoto / Bridge	Orchestratore lato VB	Proxy verso l'orchestratore C# che vive in un altro processo; DTO speculari sul *wire*.
Chain of Responsibility	Filtri di auto-invocazione	Intercetta ogni invocazione di tool per anti-loop, osservabilità e *refresh* del *token*.
Producer / Consumer	Filtro di cattura verso coda concorrente	Sincronizza la cattura dei tool con lo *streaming* del testo.
State Machine	*Stripper* dello *streaming*	Ripulisce e smista lo *stream* di *chunk*, gestendo i marcatori spezzati a cavallo di due *chunk*.
Sentinel value	`__AUTH_EXPIRED`	Segnala il *token* scaduto attraverso un confine che trasporta solo stringhe.
Memento	*Snapshot* pre-modifica	Consente il *rollback* dell'anteprima allo stato precedente.

Pattern	Dove	Problema risolto
Singleton mutabile thread-safe	Provider del *token*	Unico punto di verità del *bearer* corrente nel server MCP, aggiornabile a runtime.

Tabella 10: Catalogo dei pattern architetturali.

A.4 Decisioni architetturali e trade-off

Decisione	Motivazione	Alternativa scartata
Tre processi separati sul nodo A	Isola il runtime .NET 10 dal processo *legacy*; server MCP *self-contained*.	Modello e orchestratore *in-process*: impossibile per i runtime divergenti.
LLM remoto sul server aziendale	Centralizza GPU e modello; workstation leggera.	Modello locale su ogni postazione: costo hardware elevato.
Accesso ai dati via REST (MCP)	I dati restano dietro le porte già presidiate da Symposium.	Accesso diretto al database: duplica logica e scavalca la sicurezza.
L'AI non scrive mai sul database	*Changeset* applicato come anteprima; persistenza demandata ad Agilis.	Scrittura diretta da parte dell'AI: rischio inaccettabile in produzione.
Streaming a canali separati	UX reattiva; ragionamento su canale dedicato e blocco JSON nascosto.	Risposta unica a fine elaborazione: attesa lunga e output non pulito.
Refresh del *token* a più livelli	Robustezza alla scadenza del JWT senza interrompere la conversazione.	Singolo controllo all'avvio: il *token* scadrebbe a metà sessione.

Decisione	**Motivazione**	**Alternativa scartata**
Costruzione *lazy* dell' orchestratore	Apertura del pannello immediata e UI sempre reattiva.	Costruzione sincrona: *deadlock* e blocco dell'interfaccia.
Contesto limitato alla *viewport*	Evita di far ragionare il modello su record fuori vista.	Invio dell'intero dataset: costo di *token* e rumore informativo.
Validazione delle proposte esterna al modello	Un tool deterministico verifica le sovrapposizioni e scarta a monte le proposte invalide; il modello resta libero dalla verifica di consistenza.	Estrazione da un blocco JSON libero generato dal modello: fragile col modello leggero, frequenti incoerenze.
Tool deterministici ad alta granularità	Incapsulano sequenze complesse (es. calcolo disponibilità) e restituiscono risultati finiti, riducendo i passi di ragionamento.	Tool elementari composti dal modello: ogni composizione è un passo in più e un'occasione di allucinazione.

Tabella 11: Decisioni architetturali e relativi compromessi.

Appendice B

Listati di codice

B.1 *Endpoint* REST

La colonna *Esposto* indica se l'*endpoint* è reso invocabile dal modello (✓) o se è disponibile nel *plugin* ma fuori dal perimetro del chatbot (✘).

Metodo	Rotta	Esposto	Descrizione
GET	`mcp/Appuntamenti`	✓	Appuntamenti del pianificatore (filtri per risorsa, date, stato, conto, lead, commessa, task).
GET	`mcp/Tasks`	✓	Task di commessa (competenza, livello, cliente, scadenze, stato evasione).
GET	`mcp/Risorse`	✓	Risorse interne con competenze associate.
GET	`mcp/Competenze`	✓	Competenze censite, filtrabili per area.
GET	`mcp/Competenze/aree`	✓	Macro-aree di competenza.
GET	`mcp/TurniCalendario`	✓	Calendario lavorativo aziendale (regole, intervalli orari, validità).

Metodo	Rotta	Esposto	Descrizione
GET	`mcp/Documenti`	✘	Testate documenti di magazzino.
GET	`mcp/Documenti/righe`	✘	Righe dei documenti.
GET	`mcp/Documenti/analisi/per-cliente`	✘	Fatturato per cliente.
GET	`mcp/Documenti/analisi/per-articolo`	✘	Vendite per articolo.
GET	`mcp/Documenti/analisi/per-agente`	✘	Fatturato per agente.
GET	`mcp/Ordini`	✘	Testate ordini.
GET	`mcp/Ordini/righe`	✘	Righe ordini (con quantità evasa).
GET	`mcp/Ordini/analisi/per-cliente`	✘	Analisi ordini per cliente.
GET	`mcp/Ordini/analisi/per-articolo`	✘	Analisi ordini per articolo.
GET	`mcp/Offerte`	✘	Testate offerte/ preventivi.
GET	`mcp/Offerte/righe`	✘	Righe offerte.
GET	`mcp/Offerte/analisi/per-agente`	✘	Valore offerte per agente.
GET	`mcp/Scadenze`	✘	Partite aperte/saldate.

Metodo	Rotta	Esposto	Descrizione
GET	`mcp/Scadenze/aperto/per-cliente`	✘	Esposizione aperta per cliente.
GET	`mcp/Scadenze/prossime`	✘	Scadenze entro N giorni.
GET	`mcp/Anagrafica`	✘	Clienti e fornitori (tipo C/F).
GET	`mcp/Leads`	✘	Lead/prospect.
GET	`mcp/Articolo`	✘	Articoli di magazzino.
GET	`mcp/ProgrammiMenu`	✘	Voci di menu del gestionale.
POST	`mcp/CreaDocumento`	✘	Creazione documento di magazzino.
POST	`mcp/CreaOfferta`	✘	Creazione offerta.
POST	`mcp/CreaOfferta/modifica`	✘	Modifica/revisione offerta.

Tabella 12: Lista degli *endpoint* esposti da Symposium per il server MCP.

B.2 Mappa degli strumenti esposti al modello

Di seguito la mappa completa degli strumenti a disposizione del modello linguistico, suddivisi in base alla loro natura architetturale: strumenti remoti esposti tramite protocollo MCP e *plugin nativi* iniettati localmente nel Semantic Kernel.

Tool	Tipo	Fonte
Strumenti MCP		
`get_appuntamenti`	*pass-through*	`mcp/Appuntamenti`

Tool	Tipo	Fonte
`get_tasks`	*pass-through*	`mcp/Tasks`
`get_risorse`	*pass-through*	`mcp/Risorse`
`get_comp`	*pass-through*	`mcp/Competenze`
`get_ar_co`	*pass-through*	`mcp/Competenze/aree`
`get_turni_calendario`	*pass-through*	`mcp/TurniCalendario`
`get_spazi_liberi`	composito	`DisponibilitaCore` (TurniCalendario + Appuntamenti)
`trova_risorsa_disponibile`	composito	`DisponibilitaCore` (TurniCalendario + Risorse + Appuntamenti)
`SetSymposiumToken`	*control-plane*	interno, non esposto al modello
***Plugin* Semantic Kernel**		
`pianifica_e_proponi`	validazione	`PianificaPlugin`

Tabella 13: Mappa tool MCP esposti –> fonti dati.

B.3 Estratti di codice

```vb
' Confine verso l'orchestratore (client MCP). Implementato da McpClientOrchestrator
' e decorato da DeferredOrchestrator per la costruzione asincrona.
```

```
Public Interface IChatbotOrchestratorClient
    Function InitializeAsync() As Task
    Function SendTurnAsync(...) As Task(Of OrchestratorResponse)
    Function RefineAsync(...) As Task(Of OrchestratorResponse)
    Sub OnTokenRefreshed(nuovoToken As String)
    Sub ResetConversation()
    Sub RichiediAnnullamentoTurno()
    Sub Shutdown()
End Interface

' Estrazione del contesto dal _ViewModel_ di XSPMPRIS. Implementato da XspmprisContextAdapter.
Public Interface IPlanningContextProvider
    Function BuildContext() As ChatContext
    Function ResolveAppuntamento(progr As Integer) As AppuntamentoSnapshot
    Function ResolveRisorsa(cod As Integer) As String
    Event ContextChanged As EventHandler
End Interface

' Applicazione/rollback del changeset sullo scheduler. Implementato da XspmprisChangesetApplier.
Public Interface IChangesetApplier
    Function Valida(changeset As ChatbotChangeset) As ValidationResult
    Function ApplicaAnteprima(changeset As ChatbotChangeset) As ApplyResult
    Sub AnnullaAnteprima(changesetId As Guid)
    Sub ConfermaAnteprima(changesetId As Guid)
End Interface

' Estrazione testo dagli allegati. Servizio dispatcher + estrattori specializzati.
Public Interface IAttachmentExtractionService
    ReadOnly Property MaxCaratteriPerAllegato As Integer
    ReadOnly Property MaxAllegatiPerTurno As Integer
```

```
33      Function ClassificaTipo(percorsoLocale As String) As
        AttachmentTipo
34      Function EstraiAsync(target As ChatAttachment) As
        Task
35  End Interface
36
```

Codice 6: Le interfacce che disaccoppiano il pannello chatbot dal modulo `XSPMPRIS`.

```
1   Public Sub New(factory As Func(Of Task(Of                    Visual Basic
    IChatbotOrchestratorClient)))
2     ' La factory effettua una chiamata HTTP a Symposium
    (login): eseguirla nel
3     ' thread UI bloccherebbe l'apertura del pannello. La si
    avvia in background.
4     _innerTask = Task.Run(factory)
5
6     ' Osserva un eventuale fallimento della factory per
    evitare che
7     ' un'eccezione non osservata possa terminare il
    processo.
8     _innerTask.ContinueWith(
9         Sub(t) Debug.WriteLine("[DeferredOrchestrator]
          Factory FAULTED: " &
          t.Exception?.GetBaseException()?.Message),
10        CancellationToken.None,
11        TaskContinuationOptions.OnlyOnFaulted,
12        TaskScheduler.Default)
13  End Sub
14
15  Public Async Function SendTurnAsync(...) As Task(Of
    OrchestratorResponse)
16      Dim inner = Await _innerTask            ' attende la
        costruzione, senza bloccare la UI
17      Return Await inner.SendTurnAsync(...)
18  End Function
```

```
1   [McpServerTool]                                              C#
```

```
2   [Description("Cerca appuntamenti del pianificatore.
    Restituisce titolo, date inizio/fine, luogo, stato,
    commessa, taskid, risorse assegnate, note interne, note
    cliente. Filtri: codRisorsa (og_progr), dataDa, dataA,
    codStato, conto, codLead, commessa, taskId. Possibilità
    di paginazione attraverso skip e take.")]
3   public async Task<string>
    GetAppuntamenti(IRestClientResolver rcr,
4       [Description("Codice appuntamento (ap_progr) per
        filtrare per appuntamento specifico [Opzionale]")]
        int? codApp = null,
5       [Description("Codice risorsa ORGANIG (og_progr) per
        filtrare appuntamenti di una risorsa specifica
        [Opzionale]")] int? codRisorsa = null,
6       [Description("Data/ora inizio da (yyyy-MM-dd o yyyy-
        MM-ddTHH:mm) [Opzionale]")] DateTime? dataDa = null,
7       [Description("Data/ora inizio a (yyyy-MM-dd o yyyy-
        MM-ddTHH:mm) [Opzionale]")] DateTime? dataA = null,
8       [Description("Codice stato appuntamento
        [Opzionale]")] int? codStato = null,
9       [Description("Codice conto cliente collegato
        [Opzionale]")] int? conto = null,
10      [Description("Codice lead collegato [Opzionale]")]
        int? codLead = null,
11      [Description("Codice commessa collegata
        [Opzionale]")] int? commessa = null,
12      [Description("Codice task collegato [Opzionale]")]
        int? taskId = null,
13      [Description("Numero di record da saltare")] int skip
        = 0,
14      [Description("Numero massimo di record")] int take =
        100)
15      {
16          // ...
17      }
```

Codice 8: Esempio di tool MCP che mappa l'*endpoint* `mcp/Appuntamenti` del *plugin* REST di Symposium.

```
1   sealed class MaxToolRoundsFilter(int maxRounds) :
    IAutoFunctionInvocationFilter
```
C#

```
{
    public async Task OnAutoFunctionInvocationAsync(
        AutoFunctionInvocationContext context,
        Func<AutoFunctionInvocationContext, Task> next)
    {
        if (context.RequestSequenceIndex >= maxRounds)
        {
            context.Terminate = true;
            return;
        }
        await next(context);
    }
}
```

Codice 9: Il filtro anti-loop: termina l'esecuzione al superamento del numero massimo di round di tool.

```
public sealed record ToolCallRecord(string Nome,
string ParametriJson, string? Risultato);

public sealed class ToolCaptureFilter :
IAutoFunctionInvocationFilter
{
    private readonly ConcurrentQueue<ToolCallRecord>
    _queue = new();

    public bool TryDequeue(out ToolCallRecord record) =>
    _queue.TryDequeue(out record!);

    public async Task OnAutoFunctionInvocationAsync(
        AutoFunctionInvocationContext context,
        Func<AutoFunctionInvocationContext, Task> next)
    {
        var parametri = SerializeArgs(context.Arguments);
        await next(context);
        var risultato = context.Result?.GetValue<object?
        >()?.ToString();
```

```
            _queue.Enqueue(new
            ToolCallRecord(context.Function.Name, parametri,
            risultato));
        }
    }
```

Codice 10: Il filtro di osservabilità: intercetta ogni invocazione di tool e ne accoda nome, parametri ed esito in una `ConcurrentQueue` a disposizione dell'interfaccia.

```
// Sottrae gli impegni dalle finestre lavorative,
restituendo le fasce libere.
internal static List<Slot> Libere(List<Slot> finestre,
List<Slot> impegni)
{
    var libere = new List<Slot>();
    foreach (var f in finestre)
    {
        var cursore = f.Inizio;
        var interni = impegni
            .Where(i => i.Fine > f.Inizio && i.Inizio <
            f.Fine)
            .OrderBy(i => i.Inizio);
        foreach (var imp in interni)
        {
            if (imp.Inizio > cursore)
                libere.Add(new Slot { Inizio = cursore,
                Fine = imp.Inizio });
            if (imp.Fine > cursore)
                cursore = imp.Fine;   // salta oltre
                l'impegno
        }
        if (cursore < f.Fine)
            libere.Add(new Slot { Inizio = cursore, Fine
            = f.Fine });
    }
    return libere.Where(s => s.Fine > s.Inizio).OrderBy(s
    => s.Inizio).ToList();
}
```

Codice 11: Calcolo deterministico delle fasce libere per sottrazione degli impegni dalle finestre lavorative.

```vb
' .msg via MsgReader: solo nomi degli allegati
If msg.Attachments IsNot Nothing Then
    For Each allegato In msg.Attachments
        Dim att = TryCast(allegato, MsgReader.Outlook.Storage.Attachment)
        If att IsNot Nothing Then
            nomiAllegati.Add(If(att.FileName, "(allegato)"))
        End If
    Next
End If

' .eml via MimeKit: stessa limitazione
For Each allegato In msg.Attachments
    Dim part = TryCast(allegato, MimeKit.MimePart)
    If part IsNot Nothing AndAlso Not String.IsNullOrEmpty(part.FileName) Then
        nomiAllegati.Add(part.FileName)
    End If
Next
```

Codice 12: Estrazione email: gli allegati sono elencati per nome, senza estrazione del contenuto.

```vb
Case OperazioneTipo.Update
    Dim target As TimelineItem = Nothing
    If itemsPerProgr.TryGetValue(op.ProgrTarget.Value, target) Then
        ' Memento: snapshot pre-modifica per il rollback (una sola volta).
        If target.PreChatbotSnapshot Is Nothing Then
            Dim hasChangesPreModifica = target.HasChanges
            Dim snap = target.Clone()
            snap.PreChatbotSnapshot = Nothing
```

```
9               snap.HasChanges = hasChangesPreModifica  '
                valore pulito, riassegnato per ultimo
10              target.PreChatbotSnapshot = snap
11          End If
12          ApplicaModificaSuItem(target, op.DatiNuovi)
13          target.IsProposedByChatbot = True
14          target.ChatbotChangesetId = changeset.Id
15          target.HasChanges = True
16      End If
```

Codice 13: Applicazione di un'operazione di modifica con *snapshot Memento*.

```
1   public async Task AuthenticateIfNecessary()                    C#
2   {
3       // Token ancora valido (margine di 1 minuto): nessuna
        azione.
4       if (_loginResponse != null &&
        _loginResponse.AccessToken.TokenExpiration >
        DateTime.UtcNow.AddMinutes(1))
5           return;
6
7       // Sessione esistente: prova il refresh. Se il
        refresh token è scaduto
8       // o rifiutato, azzera la sessione e ricade nel login
        completo.
9       if (_loginResponse != null)
10      {
11          try
12          {
13              var refresh = await /* POST Login/
                RefreshToken */;
14              if (refresh.IsSuccessful){ /* aggiorna token
                e bearer */ return; }
15          }
16          catch { /* errore di rete --> fallback */ }
17      }
18
19      // Login completo: primo accesso oppure refresh token
        scaduto.
```

```
20      var login = await /* POST Login/Authenticate con le
        credenziali di sessione */;
21      // aggiorna _loginResponse e configura il bearer
22  }
```

Codice 14: Margine di scadenza lato Agilis.

```
1  public static string
   FormatResponse(RestResponseWithData<byte[]> resp)
2  {
3      if (resp.IsSuccessful)      return
       Encoding.UTF8.GetString(resp.Data!);
4      if ((int)resp.StatusCode == 401) return
       AuthExpiredSentinel;   // "__AUTH_EXPIRED"
5      return $"Nessun dato trovato (errore:
       {resp.StatusCode})";
6  }
```

Codice 15: Traduzione del 401 in un *sentinel* lato server MCP.

```
1   public async Task OnAutoFunctionInvocationAsync(
2       AutoFunctionInvocationContext context, Func<…> next)
3   {
4       await next(context);
5       if (!IsAuthExpired(context.Result)) return;
6
7       await _refreshGate.WaitAsync();
8       // un solo refresh per ondata di 401
9       try {
10          var newToken = await
            needTokenAsync(CancellationToken.None);
11          // Agilis via pipe
12          if (string.IsNullOrWhiteSpace(newToken)) return;
13          if (!string.Equals(_lastAppliedToken, newToken,
            StringComparison.Ordinal)) {
14              await mcpServerClient.CallToolAsync(
15                  "SetSymposiumToken",
16                  new Dictionary<string, object?>
                    { ["token"] = newToken });
```

```
17              _lastAppliedToken = newToken;
18          }
19      } finally { _refreshGate.Release(); }
20
        context.Result = await
21      context.Function.InvokeAsync(context.Kernel,
        context.Arguments); // retry
22  }
```

Codice 16: *Refresh* reattivo del *token* su *sentinel* `__AUTH_EXPIRED`.

Codice 7: `DeferredOrchestrator`. *Decorator* che costruisce l'orchestratore reale in *background*, evitando il blocco del *thread* di interfaccia all'apertura del pannello.

Glossario

Adapter: *Pattern* strutturale che converte l'interfaccia di una classe in un'altra attesa dai client, consentendo a classi altrimenti incompatibili di collaborare. 20.

Agente: Un sistema basato su intelligenza artificiale capace di analizzare una richiesta, ragionare per pianificare una sequenza di operazioni e interagire attivamente con l'ambiente esterno tramite l'uso di strumenti. A differenza di un modello standard che si limita a generare testo, un agente prende decisioni e compie azioni (come chiamate API) per raggiungere l'obiettivo prefissato. 2.

***AI* – Artificial Intelligence**: Disciplina dell'informatica che studia i fondamenti teorici, le metodologie e le tecniche per la progettazione di sistemi in grado di simulare le capacità cognitive umane, come l'apprendimento, il ragionamento e la risoluzione di problemi. 2.

Allucinazioni: Fenomeno in cui un'intelligenza artificiale genera informazioni false, inesatte o del tutto inventate, presentandole però con un tono sicuro, logico e convincente come se fossero reali e verificate. 5.

***API* – Application Programming Interface**: Insieme di regole e protocolli che permettono a diverse applicazioni software di comunicare tra loro, consentendo l'accesso a funzionalità o dati specifici senza dover condividere l'intero codice sorgente. 2.

Backend: Il «dietro le quinte» di un'applicazione web. Gestisce la logica di funzionamento, l'elaborazione dei dati e la comunicazione con il database e altri servizi, rimanendo di fatto invisibile all'utente finale. 2.

Bearer Token: *Token* di autenticazione HTTP trasportato nell'*header Authorization* con il prefisso `Bearer` , utilizzato per identificare l'utente nelle richieste REST. 24.

Chain of Responsibility: *Pattern* comportamentale in cui una richiesta attraversa una catena di gestori, ciascuno dei quali può processarla, modificarla o inoltrarla al successivo. 20.

Chatbot: Software progettato per simulare una conversazione con un essere umano. Utilizza l'intelligenza artificiale per comprendere le richieste dell'utente espresse in linguaggio naturale e fornire risposte o compiere azioni pertinenti. vii

Client: Dispositivo o programma (come un browser web o l'interfaccia utente di un'app) che richiede servizi, dati o risorse a un altro computer (chiamato server) all'interno di una rete. 2.

***Cloud* – Cloud Computing**: L'erogazione di servizi informatici (come server, archiviazione dati o *software*) tramite Internet. Nel progetto, si contrappone all'esecuzione «in locale», dove invece i programmi (come l'LLM) girano

direttamente sui computer fisici dell'azienda, garantendo maggiore privacy. 13.

Database: Archivio informatico strutturato (spesso chiamato «base di dati») in cui le informazioni vengono memorizzate e organizzate in modo da poter essere facilmente ricercate, aggiornate e gestite in sicurezza dal *software*. 1.

Decorator: *Pattern* strutturale che aggiunge responsabilità a un oggetto in modo dinamico, avvolgendolo in un componente con la stessa interfaccia. 20.

Endpoint: L'indirizzo web specifico o il punto di contatto esatto attraverso cui due *software* comunicano tra loro (ad esempio, un indirizzo a cui il chatbot invia una richiesta per ottenere la lista degli appuntamenti dal gestionale). ix, 13.

***ERP* – Enterprise Resource Planning**: *Software* di gestione aziendale che integra e coordina i principali processi operativi di un'organizzazione (come vendite, acquisti, gestione magazzino, contabilità) in un unico sistema centralizzato, garantendo la coerenza dei dati in tempo reale. 1.

Framework: Una struttura *software* di base, simile a un'impalcatura precostruita, che fornisce agli sviluppatori un insieme di strumenti, librerie e regole per costruire applicazioni complesse più velocemente e in modo standardizzato, senza dover partire da zero. vii

Frontend: La parte visibile di un *software* o di un sito web, con cui l'utente può interagire direttamente tramite schermi, pulsanti e testi. È strettamente legata al concetto di Interfaccia Utente (UI). 1.

***IDE* – Integrated Development Environment**: Ambiente di sviluppo integrato che fornisce agli sviluppatori un insieme completo di strumenti per scrivere, testare e debuggare il codice, spesso includendo un editor di testo, un compilatore, un debugger e funzionalità di gestione del progetto. 6.

***JSON* – JavaScript Object Notation**: Formato di testo leggero e semplice da leggere, utilizzato per scambiare dati tra sistemi informatici. È lo standard principale per la comunicazione tra le moderne applicazioni web. 5.

***JWT* – JSON Web Token**: Standard di sicurezza utilizzato per trasmettere in modo affidabile informazioni tra due parti. Nel progetto è impiegato per verificare l'identità dell'utente al momento dell'accesso e mantenere la sua sessione sicura nel tempo. 18.

Knowledge Cutoff: Indica la data esatta in cui si interrompe la raccolta dei dati utilizzati per l'addestramento di un modello di intelligenza artificiale (come un LLM). Il modello non possiede alcuna conoscenza intrinseca di eventi, scoperte o informazioni emerse successivamente a tale data. 5.

***LLM* – Large Language Model**: Modello di intelligenza artificiale avanzato, addestrato su enormi quantità di dati testuali, capace di comprendere, elaborare e generare linguaggio naturale con un livello di fluidità e coerenza molto simile a quello umano. 2.

***Macchina a stati* – State machine**: Modello comportamentale che rappresenta il comportamento di un sistema come un insieme di stati finiti e delle transizioni tra essi, attivate da eventi o condizioni. 20.

***MCP* – Model Context Protocol**: Protocollo standardizzato progettato per facilitare e mettere in sicurezza la comunicazione tra i modelli di intelli-

genza artificiale (come gli LLM) e fonti di dati o strumenti esterni (Tool), permettendo all'assistente di recuperare informazioni di contesto specifiche e in tempo reale. 2.

Memento: *Pattern* comportamentale che cattura e esternalizza lo stato interno di un oggetto senza violare l'incapsulamento, permettendone il ripristino successivo. 19.

***MES* – Manufacturing Execution System**: Sistema informatizzato utilizzato nel settore manifatturiero per gestire, monitorare e ottimizzare l'esecuzione delle attività in fabbrica, collegando i sistemi gestionali (come l'ERP) con le macchine operative in produzione. 1.

Open-Source: Modello di sviluppo in cui il codice sorgente di un *software* è reso pubblico, permettendo a chiunque di studiarlo, modificarlo, migliorarlo e distribuirlo liberamente. 2.

Plugin: Un piccolo programma aggiuntivo (spesso chiamato anche estensione o modulo) che si «aggancia» a un software principale per aggiungergli nuove funzionalità specifiche, senza la necessità di modificare il codice del programma di base. 6.

Prompt: Il testo, la domanda o il comando che un utente inserisce per dare un'istruzione a un sistema di intelligenza artificiale. Determina il contesto e l'obiettivo della risposta che il modello andrà a generare. 6.

Quantizzazione: Tecnica di compressione che riduce la precisione numerica dei pesi di una rete neurale (es: da `float32` a `int8`), riducendo memoria e latenza a costo di leggera perdita di accuratezza. 14.

***REST* – Representational State Transfer**: Stile architetturale per la progettazione di servizi web, che utilizza metodi HTTP standard (come GET, POST, PUT, DELETE) per operare su risorse rappresentate in formati come JSON o XML, favorendo l'interoperabilità e la scalabilità. 1.

***SDK* – Software Development Kit**: Pacchetto di strumenti di sviluppo che fornisce a programmatori e sviluppatori librerie, interfacce (API), documentazione e strumenti di debug necessari per creare applicazioni su una specifica piattaforma, semplificando e accelerando il processo di scrittura del codice. 7.

Sentinel Value: Valore speciale usato come segnale o marcatore per comunicare una condizione particolare al programma (es: `__AUTH_EXPIRED` per indicare che il *token* di accesso è scaduto). 32.

Server: Computer o sistema informatico potente che fornisce dati, servizi o risorse ad altri computer (i client) attraverso una rete. Nel contesto web, elabora le richieste degli utenti e ospita il cuore delle applicazioni. 3.

Sistemi Aumentati: Architetture di intelligenza artificiale in cui un modello linguistico non opera in isolamento, ma viene potenziato integrandolo con l'accesso a basi di dati, strumenti (tool) o interfacce di programmazione (API) esterne. Questo approccio permette all'LLM di superare i limiti della propria memoria interna e di recuperare informazioni aggiornate o private. 5.

***SSE* – Server Sent Events**: Protocollo di comunicazione *one-way* dal server al client basato su HTTP, utilizzato per lo *streaming* di dati in tempo reale. 20.

Tool Use: Pratica di utilizzare strumenti esterni per estendere le capacità di un modello di intelligenza artificiale, permettendogli di accedere a dati, eseguire operazioni o interagire con sistemi esterni in modo sicuro ed efficiente. vii

***UI* – User Interface**: Interfaccia Utente, ovvero l'ambiente visivo e interattivo composto da schermate, pulsanti, testi e menu attraverso il quale una persona comunica e interagisce in modo intuitivo con un software. 18.

***UX* – User Experience**: Esperienza Utente, ovvero l'insieme di percezioni, emozioni e risposte che una persona prova quando interagisce con un software. A differenza della UI (che si occupa dell'aspetto visivo), la UX si concentra su quanto il sistema sia facile, intuitivo, utile e soddisfacente da usare nel lavoro quotidiano. 21.

***WMS* – Warehouse Management System**: Software gestionale dedicato al supporto e all'ottimizzazione delle operazioni quotidiane di magazzino, che coordina attività come la ricezione delle merci, lo stoccaggio, il prelievo (*picking*) e la spedizione. 1.

Bibliografia

[1] Anthropic, «Model Context Protocol — Getting Started». 2025. [Online]. Disponibile su: https://modelcontextprotocol.io/docs/getting-started/intro

[2] Omega S.r.l., «Omega Gruppo». 2025. [Online]. Disponibile su: https://www.omegagruppo.it/

[3] A. Vaswani *et al.*, «Attention Is All You Need», 2017. [Online]. Disponibile su: https://arxiv.org/abs/1706.03762

[4] Anthropic, «Anthropic». 2025. [Online]. Disponibile su: https://www.anthropic.com/

[5] JSON Schema Community, «JSON Schema». 2025. [Online]. Disponibile su: https://json-schema.org/

[6] JSON-RPC Working Group, «JSON-RPC 2.0 Specification». 2010. [Online]. Disponibile su: https://www.jsonrpc.org/specification

[7] LangChain Inc., «LangChain». 2025. [Online]. Disponibile su: https://www.langchain.com/

[8] Microsoft, «Semantic Kernel — Overview». 2025. [Online]. Disponibile su: https://learn.microsoft.com/it-it/semantic-kernel/overview/

[9] Microsoft, «Microsoft .NET». 2025. [Online]. Disponibile su: https://dotnet.microsoft.com/it-it/

[10] Anthropic e Microsoft, «Model Context Protocol — C# SDK». 2025. [Online]. Disponibile su: https://github.com/modelcontextprotocol/csharp-sdk

[11] Ollama Inc., «Ollama». 2025. [Online]. Disponibile su: https://ollama.com/

[12] OpenAI, «OpenAI». 2025. [Online]. Disponibile su: https://openai.com/

[13] Alibaba Cloud, «Qwen — Alibaba Cloud». 2025. [Online]. Disponibile su: https://qwen.ai/home

[14] Microsoft, «Visual Studio». 2026. [Online]. Disponibile su: https://visualstudio.microsoft.com/it/vs/

[15] L. Torvalds, «Git — distributed version control system». 2005. [Online]. Disponibile su: https://git-scm.com/

[16] M. B. Jones, J. Bradley, e N. Sakimura, «JSON Web Token (JWT) — RFC 7519». 2015. [Online]. Disponibile su: https://datatracker.ietf.org/doc/html/rfc7519

[17] E. Gamma, R. Helm, R. Johnson, e J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.