Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Informatica

Agenti generativi e pipeline d'avanguardia: Automazione del ciclo di sviluppo software

Relatore:
Chiar.mo Prof.
Fabio VITALI

Presentata da:
Gabriele FOGU

Correlatore Aziendale:
Chiar.mo
Marco PINELLI

Anno Accademico 2024-2025

Indice

Introduzione

Questa tesi affronta uno dei problemi più concreti e ricorrenti nello sviluppo software moderno: la difficoltà di trasferire in modo rapido ed efficace conoscenze tecniche, standard architetturali e convenzioni interne a nuovi sviluppatori, e la conseguente lentezza nell'avvio di un progetto o nella costruzione di prototipi funzionali. In un contesto aziendale basato su soluzioni full stack e su un *template* condiviso, come quello adottato da **Laif S.r.l.**, questa difficoltà si manifesta nella necessità di dedicare tempo significativo all'allineamento iniziale e alla produzione di porzioni di codice ripetitive o standardizzate.

L'uso del *template* svolge un ruolo chiave nel ridurre il lavoro di implementazione del cosiddetto *boilerplate*, ossia quelle componenti strutturali che ricorrono in tutti i progetti e che definiscono l'architettura di base del software: dalla configurazione del back-end e delle sue rotte CRUD, all'organizzazione delle directory, alla gestione della navigazione e dei flussi applicativi. Accanto al template, il *design system* aziendale `laif-ds` fornisce un insieme coerente di componenti front-end, stili e pattern d'interazione, permettendo di mantenere uniformità visiva e comportamentale tra progetti differenti e riducendo ulteriormente il carico di implementazione ripetitiva.

Sebbene questa soluzione riduca l'overhead tecnico, rimane aperto il problema dell'onboarding: comprendere come orientarsi rapidamente all'interno della codebase, come rispettare gli standard, come costruire nuove feature aderendo allo stile architetturale e visuale dell'azienda.

Questa tesi sostiene che tale problema può essere affrontato in modo efficace tramite l'integrazione di un **agente di Intelligenza Artificiale** direttamente nell'IDE, alimentato da una **knowledge base interna** strutturata secondo il *template* aziendale, le *rules* versionate e la documentazione viva esposta tramite **MCP**. L'ipotesi centrale è che, fornendo a un LLM un contesto ricco, organizzato e vincolato, l'agente sia in grado di generare codice di qualità più che accettabile e coerente, riducendo in modo significativo sia il *time-to-productivity* dei nuovi sviluppatori sia i tempi necessari alla produzione di mockup o *proof of concept* per attività *presales*.

Nel corso della ricerca è emerso come questa pipeline agentica consenta allo sviluppatore - anche partendo da zero - di costruire in maniera assistita applicazioni complete, passando rapidamente da richieste semplici alla generazione multi–file di componenti coerenti con lo stack aziendale. L'efficacia osservata non risiede nella sostituzione del lavoro umano, bensì nel supporto continuo fornito dall'agente attraverso un ciclo *plan/act/review* e nell'uso integrato di:

a) regole aziendali versionate e strutturate in Markdown,

b) accesso a specifiche e documentazione tramite **MCP**,

c) conoscenza del *design system* (`laif-ds`) e del *template* full-stack.

Questa integrazione, essendo interna al flusso di sviluppo e basata sulla codebase reale, permette di superare i limiti degli assistenti generici: scarsa consapevolezza degli standard, *context switching* tra ambienti esterni, perdita di coerenza e rischio di *documentation drift*. La pipeline analizzata dimostra invece di poter produrre codice coerente, ripetibile e allineato agli standard aziendali.
Un risultato particolarmente significativo è che la modalità sperimentata in questa ricerca si è rivelata talmente efficace, comoda e produttiva da essere adottata stabilmente nel processo di sviluppo aziendale, diventando parte integrante della pipeline interna.

Panoramica della soluzione

Per comprendere la natura e la portata dell'approccio proposto, è utile esaminare la pipeline agentica da una prospettiva "ad alta quota", osservando come i suoi componenti cooperino nel trasformare un LLM generico in uno strumento operativo profondamente contestualizzato.
L'intero impianto ruota attorno a un agente integrato nell'IDE, capace di interagire con la codebase e con le risorse informative aziendali. La sua efficacia dipende da tre pilastri: **contesto**, **vincoli** e **accesso alla conoscenza**. Il contesto è garantito dal *template* full-stack, che definisce la struttura portante dei progetti - dal modello dati all'organizzazione dei componenti front-end - e funge da riferimento per ogni nuova generazione. I vincoli sono incorporati nelle *rules* versionate, che descrivono convenzioni, standard architetturali, pattern ricorrenti e pratiche di sviluppo da rispettare. L'accesso alla conoscenza, infine, è reso possibile dal **Model Context Protocol (MCP)**, che collega l'agente alle fonti informative interne, come documentazione tecnica, schemi e dati dei database o specifiche funzionali.
La pipeline opera attraverso un ciclo iterativo *plan–act–review*: pianificazione dell'intervento sulla base del contesto, generazione o modifica del codice all'interno dell'IDE, e revisione immediata per verificare la coerenza con gli standard. Questo meccanismo permette all'agente di affrontare attività complesse, come refactoring multi–file, implementazione di nuove feature e costruzione di prototipi funzionali, mantenendo un allineamento continuo con la visione architetturale aziendale.

La ricerca prende forma a partire dal *claim* introdotto, sviluppandosi attraverso l'analisi dei componenti che costituiscono la pipeline agentica e delle modalità con cui essi cooperano all'interno dell'ambiente di sviluppo. Il percorso prosegue con la costruzione di un caso applicativo concreto, in cui la pipeline viene impiegata per realizzare, a partire da zero, il porting completo di un'applicazione reale. Questo scenario permette di osservare l'agente nel suo ciclo operativo, di valutarne il contributo alla generazione multi–file e di verificare la coerenza del codice prodotto rispetto agli standard aziendali. La valutazione empirica conclusiva consente di misurare l'impatto della pipeline sui principali aspetti considerati: la riduzione dei tempi di onboarding, la coerenza

stilistica e architetturale del codice generato, l'accelerazione nella produzione di prototipi e la reale integrazione del metodo nel flusso di lavoro quotidiano. Ne emerge un quadro unitario che collega il problema iniziale alla soluzione proposta e alla sua efficacia operativa.

Claim e obiettivo della ricerca

In termini aziendali, l'adozione di standard condivisi (template back-end/front-end, convenzioni di naming, *design system*, documentazione *standardizzata e condivisa*) riduce i costi di manutenzione e facilita il passaggio di consegne, ma introduce una **barriera d'ingresso** per i nuovi sviluppatori.
I processi tradizionali di onboarding (wiki, affiancamento, *shadowing*[MK24]) sono efficaci ma **costosi** e faticano a rimanere aggiornati. Parallelamente, il reparto commerciale necessita di **prototipi rapidi** e **realistici/plausibili** per validare casi d'uso e presentare demo a potenziali clienti. Un agente AI che "*vive*" nell'IDE, addestrato sulle **regole interne** e connesso a **fonti vive** (*repositories*, basi di dati, documentazione) può diventare un tassello fondamentale nel processo, incrementando la produttività, riducendo i tempi e le iterazioni, tutto ciò senza sacrificare qualità e conformità alla struttura di sviluppo reale.

Tesi (claim).

Un agente AI integrato nell'IDE, alimentato da una knowledge base interna derivata da template, regole e asset progettuali, strutturata in maniera rigida e puntuale, riduce i tempi e i costi di onboarding e prototipazione, aumentando la coerenza del codice rispetto agli standard aziendali.
In particolare, si sostiene che:

- L'agente **accorcia** il percorso che porta un neo-assunto *junior* a diventare un developer in grado di produrre valore in maniera autonoma grazie a **spiegazioni contestuali**, **esempi guidati** e **generazione assistita aderente alle *rules***.

- La generazione di **mockup** (modello dati d'esempio, pagina con UI interagibile e design coerente) è **più rapida** rispetto a pratiche manuali o ad agenti "generici" non allineati al template e prevede anche il riutilizzo "fisico" del mockup nell'eventuale successivo sviluppo dell'applicazione.
- La **qualità** (lint, test, aderenza a naming/struttura, uso del *design system*) **migliora** quando le risposte dell'agente sono vincolate da regole e *playbook* versionati;
- L'**integrazione** con fonti *vive* (MCP verso Notion e DB) mitiga il *documentation drift* e anch'essa **migliora la qualità del codice generato**, riducendo l'ambiguità dei requisiti.

Obiettivo generale.

Progettare, integrare e valutare un **agente IDE-centrico** - basato su regole e con accesso controllato a documentazione e dati - capace di fornire:

a) **Supporto all'onboarding**: spiegazioni contestuali dei file, ricostruzione di pattern ricorrenti, micro-task guidati su codebase reali;

b) **Generazione assistita**: creazione di *unità funzionali* end-to-end (modello dati, migrazione, controller/service, pagina UI) che si integrino con il template e che segua gli standard di sviluppo;

c) **Prototipazione rapida**: produzione di mockup coerenti con `laif-ds` per casi d'uso *presales*, con dati fittizi e fake API ma che rispecchino in termini di UI/UX quello che sarà il prodotto finito;

d) **Enforcement degli standard**: verifica automatica di naming, struttura del codice, gerarchia dei file, impiego dei permessi, prassi di sicurezza, uso dei componenti UI proprietari, aderenza alle convenzioni di programmazione e coerenza del codice generato rispetto alle altre codebase proprietarie.

Capitolo 1

Contesto Scientifico e Tecnologico

La valutazione avviene in uno **stack specifico** (FastAPI/SQLAlchemy/Alembic; Next.js/React; `laif-ds` su *shadcn/ui* [sha]; PostgreSQL) e in un **contesto aziendale** con template e regole già consolidate. L'agente non sostituisce lo sviluppatore, ma agisce come *assistente strutturato* capace di pianificare, proporre *diff* e invocare strumenti secondo politiche controllate.
Si assume la disponibilità di **rules file** aggiornati, accesso MCP a Notion (documentazione/specifiche) e a DB (schema/metadati), e l'uso di un IDE con capacità agentiche (es. Windsurf/Cascade o estensioni VS Code compatibili), fermo restando che la metodologia è *portabile* su strumenti equivalenti.

L'attività di sviluppo software in contesti aziendali caratterizzati da un'elevata standardizzazione di processi e strumenti presenta una duplice sfida.
Da un lato, la definizione di *best practices*, l'adozione di template condivisi e l'impiego di design system proprietari consentono di ridurre i costi di manutenzione e garantire uniformità qualitativa; dall'altro, tali convenzioni generano inevitabilmente una barriera all'ingresso per i nuovi sviluppatori, che devono acquisire rapidamente competenze su tecnologie, metodologie e prassi operative spesso articolate.

Il problema principale può essere quindi formalizzato come segue:

> *Come ridurre i tempi e i costi legati all'inserimento e alla formazione di nuovi sviluppatori in un ambiente di sviluppo standardizzato, garantendo al contempo il rispetto delle convenzioni aziendali e la produttività del team?*

A questa criticità si affianca **un'ulteriore esigenza**: supportare un reparto vendite di dimensioni estremamente ridotte (una sola persona) nella preparazione di prototipi e dimostrazioni da presentare ai potenziali clienti.
La possibilità di generare rapidamente applicazioni *mockup*, anche non pienamente funzionali ma sufficienti a illustrare le potenzialità del prodotto, rappresenterebbe infatti un vantaggio competitivo rilevante.

La questione si complica ulteriormente per aziende di piccole dimensioni o di recente costituzione come la nostra, nelle quali le risorse destinate alla formazione interna risultano limitate, e la necessità di mantenere elevata la produttività rende oneroso sottrarre forza lavoro alle attività di sviluppo. In tale contesto, ridurre il tempo di rampa dei nuovi sviluppatori e garantire l'aderenza agli standard diventa essenziale per la sostenibilità dei progetti.

1.1 Approcci popolari

Negli ultimi anni si è consolidato un insieme di *pattern* ricorrenti nell'impiego di strumenti digitali a supporto dello sviluppo software, sia in ambito accademico che industriale. Da un lato, troviamo pratiche ormai classiche come l'*onboarding* strutturato dei nuovi sviluppatori e la prototipazione rapida di funzionalità o interi prodotti; dall'altro, si è affermata una nuova generazione di strumenti basati su modelli di linguaggio di grandi dimensioni (LLM), integrati sotto forma di assistenti di codice, chatbot e agenti "semi–autonomi".
In particolare, la letteratura e i report industriali descrivono un uso sempre più diffuso di strumenti AI nel ciclo di sviluppo, ma spesso con un grado di integrazione limitato rispetto a processi, standard e architetture specifiche dell'organizzazione [Git23] [Sic24].
In questa sezione vengono quindi discussi, da un lato, gli approcci tradizionali a

onboarding e prototipazione e, dall'altro, le modalità con cui le aziende stanno iniziando a sfruttare assistenti e agenti AI, evidenziandone punti di forza e limiti rispetto al contesto oggetto di questa tesi.

1.1.1 Onboarding e Prototipazione

La letteratura recente mostra come un onboarding strutturato migliori sensibilmente la produttività dei nuovi assunti, incrementi l'engagement e riduca il turnover [Phe24]. Altri lavori riportano incrementi significativi del tempo-alla-produttività per organizzazioni con processi strutturati [Mil22].
Anche la prototipazione è considerata una leva strategica per stimolare agilità e innovazione [Cam+17][Whi24], specialmente in ambito digitale: consente di visualizzare rapidamente un'idea, verificarne la fattibilità e raccogliere feedback prima di investimenti onerosi [Cou23]. L'integrazione di tecniche AI automatizza via via parti del processo, con effetti sui tempi e sulla qualità [BG23]. **L'*onboarding* tradizionale** combina wiki interne, affiancamento e formazione, che risulta essere una soluzione efficace ma costosa. Una linea consolidata è l'uso di *retrieval-augmented generation* (RAG) [Gao+23][Lew+20], per ancorare la generazione alla documentazione interna (anche tramite *MCP* [Hou+25]).

1.1.2 Uso attuale delle AI nello sviluppo software

Negli ultimi due anni, l'adozione di strumenti di AI generativa da parte degli sviluppatori è passata da fenomeno emergente a componente quasi ubiqua del flusso di lavoro quotidiano. Una survey sponsorizzata da GitHub nel 2023, riportata nel report *Octoverse*, indica che il 92% degli sviluppatori utilizza già strumenti di *AI coding* nel lavoro o nel tempo libero [Git23]. Anche analisi successive confermano che l'uso di assistenti di codice e chatbot è ormai percepito come "nuova normalità" nello sviluppo software [Sha23; Dev25].
Il report *State of Developer Ecosystem 2024* di JetBrains, basato su oltre 23 000 sviluppatori, evidenzia come circa quattro aziende su cinque permettano o incoraggino l'uso di strumenti AI di terze parti nei workflow di sviluppo, mentre una quota

crescente di sviluppatori integra funzionalità AI direttamente nei prodotti [Sic24]. In parallelo, analisi di tipo gestionale attribuiscono all'introduzione sistematica di strumenti AI incrementi significativi di produttività (nell'ordine del 30–35%) e riduzioni del *time-to-market* [Dig24].
Nonostante ciò, il modo in cui questi strumenti vengono utilizzati è spesso **puntuale e non strutturato**: nella maggior parte dei casi si tratta di:

- completamento del codice (*code completion*) e generazione di snippet isolati;
- scrittura o "bozza" di test unitari;
- refactoring locale su uno o pochi file;
- chiarimenti su errori di compilazione, API o librerie.

Sia evidenze industriali sia survey qualitative mostrano che gli LLM sono percepiti come molto utili per compiti localizzati, ma raramente integrati in modo profondo con architetture, standard e processi dell'organizzazione [Sic24][Fos25]. Questo è esattamente il *gap* che la soluzione proposta in questa tesi cerca di colmare: passare da un uso occasionale e ad hoc dell'AI a un agente configurato sul contesto aziendale.

1.1.3 Agenti AI via web e strumenti "agentici"

Accanto ai classici chatbot e agli assistenti integrati nell'IDE, si sono diffusi negli ultimi anni strumenti "agentici" accessibili via web (ad esempio ambienti remoti che applicano automaticamente *diff*, esplorano repository e propongono modifiche strutturate al codice). Nella pratica industriale questi agenti vengono spesso utilizzati per:

- esplorare rapidamente una nuova codebase;
- generare prototipi o *mockup* front-end;
- preparare demo o *proof-of-concept* in tempi molto brevi.

Casi di studio riportano ad esempio riduzioni del tempo di prototipazione da alcuni giorni a poche ore (o meno) quando team di sviluppo rendono obbligatorio l'uso di tool

come Cursor o GitHub Copilot per alcune attività esplorative [Ins25a]. Altre survey su team di ingegneria indicano che oltre il 60–70% delle organizzazioni utilizza più di uno strumento AI nel ciclo di sviluppo (IDE assistant, chatbot general-purpose, agenti web dedicati) [Ins25b].
Questi approcci, tuttavia, presentano limitazioni strutturali rispetto a un'integrazione profonda nel contesto aziendale:

a) **Workspace parziale**: l'agente opera spesso su un workspace remoto limitato a sottoinsiemi del repository caricati di volta in volta;

b) **Contesto effimero**: il contesto (file aperti, conversazioni, istruzioni) tende a perdersi tra una sessione e l'altra, rendendo difficile costruire una "memoria" stabile del progetto;

c) **Standard debolmente modellati**: le convenzioni (architettura, naming, policy) non sono tipicamente codificate come vincoli rigidi, ma affidate a prompt generici o linee guida testuali.

Per attività di brainstorming, *spike* esplorativi o prototipazione rapida queste soluzioni sono spesso sufficienti e portano benefici tangibili. Per un'adozione in progetti strutturati, con forte standardizzazione interna, permangono però dubbi su coerenza, sicurezza e mantenibilità del codice generato.

1.1.4 Limiti delle soluzioni popolari

Gli agenti AI "generici" risultano spesso efficaci su compiti isolati, ma presentano limiti pratici quando si richiede aderenza a standard interni e integrazione profonda nel flusso di sviluppo.
In particolare:

a) Non incorporano le convenzioni aziendali (naming, struttura di repository, politiche di sicurezza), generando codice eterogeneo;

b) Aumentano il rischio di *data leakage* perché inducono a copiare nel prompt porzioni sensibili di codice o specifiche;

c) Impongono *context switching* dall'IDE a strumenti esterni, con perdita di continuità operativa;

d) Si appoggiano a wiki e note spesso soggette a *documentation drift* [Gau23], cioè disallineamento progressivo tra documentazione e codice reale e le fonti di informazioni impiegate sono spesso "oscure" all'utente, il che porta più facilmente ad allucinazioni del modello.

Per mitigare tali criticità, risultano più efficaci approcci che: (a) vincolano la generazione a **regole versionate** (file Markdown) e a **playbook** operativi; (b/c) integrano l'agente **direttamente nell'IDE** [Mic], così da lavorare su codebase e toolchain reali (lint, test, migrazioni) senza cambiare contesto; (d) sfruttano **MCP** per l'accesso in sola lettura a fonti "vive" (Notion per documentazione/specifiche, database per schemi e metadati), riducendo ambiguità e drift.

1.1.5 Context switching, hallucinations e documentation drift.

Studi sul lavoro frammentato mostrano che, in media, dopo un'interruzione servono circa 23 minuti per ritornare pienamente concentrati sul compito originario [MGK08], dato spesso citato anche in letteratura divulgativa sul tema della produttività. Nel caso di sviluppatori che alternano continuamente IDE, browser e interfacce web per interagire con l'AI, questo costo cognitivo si traduce in perdita di efficienza e maggiore probabilità di errore.
Sul fronte della qualità delle risposte, survey sistematiche sulle *hallucinations* nei modelli generativi [Ji+23] e linee guida industriali sulla sicurezza dei sistemi generativi [Ser24] sottolineano come l'assenza di una base di conoscenza strutturata e di vincoli espliciti aumenti il rischio di contenuti plausibili ma non corretti. In parallelo, lavori recenti sul fenomeno del *documentation drift* evidenziano che, in assenza di meccanismi automatizzati di allineamento, la documentazione tende rapidamente a divergere dal codice reale, con impatti negativi su manutenzione e onboarding [Gos22] [Moh+25].
Questi elementi motivano l'adozione di approcci che riducono il copia/incolla manuale, ancorano la generazione a fonti "vive" (repository, database, KB) e mantengono le regole aziendali sotto controllo di versione.

1.2 Soluzione proposta

Obiettivo: progettare un **agente AI** integrato nell'IDE che sfrutti:

- documentazione interna (Notion via MCP);
- specifiche e task (Notion via MCP);
- **rules** di generazione (`.md` nel filesystem);
- accesso in lettura al database (PostgreSQL via MCP) per schema e metadati;
- conoscenza del design system `laif-ds`;
- repository completa del progetto (architettura, convenzioni, componenti).

Scopi: assistenza ai nuovi sviluppatori, enforcement degli standard, generazione rapida di mockup a supporto del reparto vendite.
La progettazione e la valutazione di questa architettura agentica saranno descritte nei capitoli successivi, a partire dalla comparazione fra IDE e strumenti assistivi adottati nel contesto aziendale.

Capitolo 2

Descrizione ad alto livello della soluzione

In questo capitolo descriviamo ad alto livello l'architettura della soluzione proposta, partendo dal funzionamento generale degli agenti di generazione di codice, passando per la valutazione comparativa degli strumenti più diffusi e arrivando alla motivazione della scelta finale. L'obiettivo è mostrare come sia possibile passare da un uso occasionale di LLM "general purpose" a un agente configurato sul contesto aziendale, integrato nell'IDE e capace di rispettare regole, template e design system esistenti.
Anticipiamo fin da subito che, sebbene strumenti ormai molto diffusi come **Cursor** e **GitHub Copilot** forniscano già risposte efficaci alle esigenze considerate in questa ricerca, nel contesto specifico di Laif la combinazione fra **Windsurf** e **Lovable** si è rivelata la soluzione complessivamente più adatta. Essa consente infatti di generare soluzioni complete a partire da zero - sia quando l'obiettivo è produrre il mockup di un'applicazione, sia quando si tratta di completare lo sviluppo di un progetto non eccessivamente complesso - mantenendo al contempo un buon controllo sul codice prodotto.
Il ruolo dello sviluppatore resta comunque centrale: è necessario definire e mantenere aggiornate le regole aziendali, curare l'accesso alla documentazione tramite MCP, strutturare prompt in *pseudo-markup* coerenti con i template esistenti e, soprattutto, revisionare e rifinire il codice generato. L'agente non sostituisce quindi il

programmatore umano, ma ne estende la capacità operativa all'interno di un processo più strutturato (*human-in-the-loop* [Wu+22]).

2.1 Come funzionano gli agenti di generazione di codice

Gli agenti di generazione di codice moderni non sono semplici sistemi di completamento automatico, ma architetture composte che combinano modelli linguistici di grandi dimensioni (LLM), protocolli di estensione e meccanismi di revisione iterativa. Rispetto ai primi tool di *autocomplete*, la differenza principale è che l'agente ragiona in termini di obiettivi, piani e azioni, avvicinandosi ai modelli deliberativi della letteratura sugli *intelligent agents* [WJ95].
In termini astratti, il comportamento di un agente di codice può essere descritto come un **ciclo deliberativo** che, nel caso specifico dello sviluppo software, assume tipicamente la forma:

a) **Raccolta del contesto** L'agente costruisce una rappresentazione del problema a partire da diversi input: il prompt dello sviluppatore, i file aperti nell'IDE, la struttura della codebase, eventuali asset di documentazione (ad esempio accessibili via RAG o MCP) e, in alcuni casi, lo stato dei test o del sistema di build. In questa fase il modello deve selezionare e comprimere in un contesto limitato le informazioni più rilevanti [Liu+24] [Yan+25], gestendo il compromesso tra ampiezza (quante parti della codebase considerare) e profondità (quanti dettagli includere).

b) **Pianificazione** Sulla base del contesto, l'agente elabora un piano di azione, che può essere esplicito (lista di passi, *task list*, pseudo–codice) o implicito nelle sue istruzioni interne. Tecniche come la *Chain-of-Thought*, il *Plan-and-Solve* o approcci in stile ReAct [Wei+22] [Wan+23] [Yao+23] mirano proprio a rendere più stabile questa fase di ragionamento, separando il "che cosa fare" dal "come modificare concretamente il codice". In ambito sviluppo, ciò si traduce per

esempio nel decidere quali file creare o modificare, quali endpoint aggiungere, quali test aggiornare, prima ancora di produrre il codice effettivo.

c) **Invocazione di strumenti** Una volta definito il piano, l'agente invoca strumenti esterni per agire sull'ambiente: lettura e scrittura di file, esecuzione di test, interrogazione del database, chiamate a servizi HTTP, ricerca nella documentazione o nel sistema di ticketing. L'uso di protocolli come MCP [Hou+25] e di meccanismi di *tool calling* [Sch+23] consente di modellare queste capacità come funzioni tipizzate, riducendo il rischio di errori (ad esempio, generare percorsi o query non valide) e rendendo più osservabile il comportamento dell'agente.

d) **Revisione e output** Infine, l'agente valuta l'esito delle azioni: confronta il codice modificato con il piano iniziale, interpreta l'output dei test o dei comandi eseguiti, individua eventuali errori e, se necessario, aggiorna il piano e ripete il ciclo. Solo quando il risultato è ritenuto soddisfacente, propone al programmatore un set di *diff* o di file completi da applicare alla codebase. In questo passaggio la presenza del programmatore umano è cruciale per approvare, modificare o rifiutare i cambiamenti, mantenendo il controllo finale sulla qualità del software [Wu+22].

Nella pratica, gli strumenti oggi disponibili nel panorama industriale implementano questo ciclo con gradi diversi di visibilità e configurabilità. Alcuni prodotti si presentano più come "autocomplete potenziati", limitandosi a generare snippet locali all'editor; altri offrono un vero e proprio agente in grado di esplorare la codebase, eseguire comandi e proporre modifiche multi–file. La soluzione proposta in questa tesi appartiene a questa seconda categoria, ma introduce vincoli aggiuntivi legati all'allineamento con standard aziendali, template di progetto e design system condivisi.

2.2 Dalla valutazione comparativa alla scelta di Windsurf e Lovable

Prima di progettare l'architettura definitiva è stato necessario confrontare le principali soluzioni disponibili per lo sviluppo assistito da AI, con particolare attenzione a tre dimensioni:

- **Profondità dell'integrazione con l'IDE**: capacità di lavorare sull'intera codebase, eseguire comandi, lanciare test, gestire *diff* e interagire con il sistema di versionamento;
- **Allineamento con standard e template esistenti**: possibilità di imporre regole di naming, struttura del progetto, uso di pattern e design system proprietari;
- **Supporto a onboarding e prototipazione**: facilità nel creare rapidamente applicazioni dimostrative coerenti con lo stack aziendale, riutilizzabili per l'onboarding dei nuovi sviluppatori e per le attività del reparto vendite.

Strumenti come **Cursor** e **GitHub Copilot** si collocano in una posizione molto forte lungo la prima dimensione: sono pienamente integrati nell'IDE, offrono assistenza contestuale nello sviluppo quotidiano e consentono in alcuni casi di orchestrare task complessi (ad esempio esecuzione di test o refactoring multi–file). Tuttavia, nel contesto specifico di Laif presentano due limiti principali:

a) **Configurabilità limitata rispetto a template e regole aziendali** È possibile guidare il comportamento dell'assistente tramite prompt generali o linee guida, ma non esiste (allo stato dei fatti durante lo svolgimento di questa tesi) un meccanismo nativo per caricare e versionare in modo strutturato un insieme di *rules* Markdown legate a uno specifico repository, differenziate per stack o progetto.

b) **Supporto alla prototipazione end-to-end meno spinto** Sebbene l'assistente sia in grado di generare file e componenti, la responsabilità di orchestrare l'intera

applicazione (struttura del progetto, routing, configurazione dello stack) ricade in larga parte sullo sviluppatore, che deve guidare manualmente ogni passo della creazione.

Per rispondere meglio alle esigenze emerse in precedenza - onboarding guidato, rispetto dei template aziendali, generazione di mockup completi per il reparto vendite - si è quindi scelto di adottare una combinazione di strumenti complementari:

- **Windsurf** come IDE agentico principale, grazie alla sua capacità di orchestrare un agente su un workspace locale, integrare protocolli come MCP e lavorare in modo trasparente su file, comandi e test;
- **Lovable** [Lov] come generatore di prototipi end-to-end, in grado di creare rapidamente applicazioni complete (front-end e back-end) a partire da specifiche testuali, da utilizzare come base di lavoro o come mockup da presentare ai clienti.

La combinazione di questi strumenti permette di coprire in modo sinergico due esigenze: da un lato la generazione rapida di applicazioni coerenti con uno stack moderno (Next.js/React + FastAPI/SQLAlchemy); dall'altro l'integrazione profonda dell'agente nel workflow quotidiano di sviluppo, con accesso controllato alle risorse aziendali.

2.3 Architettura ad alto livello: Windsurf, Lovable, MCP e rules

L'architettura proposta si basa sull'idea di utilizzare **Lovable** per generare rapidamente applicazioni o porzioni di applicazione coerenti con lo stack aziendale, per poi importarle in un workspace **Windsurf** dove un agente configurato opera su tre pilastri principali:

a) **Rules file versionati nel repository** Per ciascun progetto o famiglia di progetti vengono definiti uno o più file Markdown contenenti le regole di sviluppo (*rules file*): convenzioni di naming, struttura delle directory, pattern

architetturali (ad esempio l'uso del template "standard" basato su `RoleBasedCRUDService`), linee guida per l'uso del design system `laif-ds`, vincoli su validazione, sicurezza e gestione degli errori. Questi file sono versionati insieme al codice e possono essere referenziati esplicitamente nei prompt, in modo che l'agente li utilizzi come vincoli operativi.

b) **Accesso MCP alle risorse aziendali** Come descritto nei capitoli precedenti, sono stati configurati server MCP per:

- **PostgreSQL**, che espone strumenti per interrogare la base dati (esecuzione di query SQL, introspezione del catalogo: schemi, tabelle, colonne, tipi, vincoli), permettendo all'agente di ragionare sui modelli reali e non su una descrizione approssimativa;
- **Notion**, che consente di ricercare e recuperare documentazione aziendale relativa a prodotti, progetti, specifiche funzionali e tecniche, mantenendo struttura gerarchica e metadati (titoli, stati, tag, relazioni).

In questo modo l'agente lavora su fonti "vive" e aggiornate, riducendo il *documentation drift* e la necessità di copiare manualmente lunghi estratti nella finestra di prompt.

c) **Integrazione con lo stack e il design system aziendale** Lo stack tecnologico di riferimento comprende FastAPI/SQLAlchemy/Alembic sul backend, Next.js/React sul frontend, il design system `laif-ds` basato su *shadcn/ui* e PostgreSQL come database principale. Le rules codificano in modo esplicito:

- come creare nuovi endpoint seguendo il template standard;
- come definire modelli e schemi coerenti con i modelli SQLAlchemy esistenti;
- come utilizzare i componenti `laif-ds` per produrre interfacce utente coerenti con il resto della piattaforma;
- come organizzare i file e i moduli in linea con le convenzioni interne (naming, cartelle, pattern).

L'agente in Windsurf, operando su questo contesto, è in grado di proporre modifiche che rispettano lo stile aziendale, riducendo la necessità di correzioni manuali a posteriori.

Il flusso tipico può essere così sintetizzato:

- per una nuova funzionalità o un nuovo prodotto dimostrativo, si utilizza **Lovable** per generare un primo mockup applicativo, che fornisce la struttura generale (routing, pagine principali, componenti base);
- il codice generato viene importato nel repository aziendale e aperto in **Windsurf** per essere normalizzato rispetto ai template interni;
- all'interno di Windsurf, l'agente - guidato da rules, MCP e documentazione - si occupa di completare o rifinire l'implementazione: integrazione con il backend, adeguamento ai modelli dati reali, allineamento al design system e alla struttura degli applicativi aziendali e aggiunta della logica di business;
- lo sviluppatore umano revisiona i *diff*, esegue i test, valuta la coerenza con i requisiti e, se necessario, aggiorna le rules sulla base delle lezioni apprese, in un ciclo iterativo di miglioramento continuo.

Questa architettura permette di conciliare due obiettivi: da un lato, accelerare in modo significativo il time-to-prototype e supportare un reparto vendite di dimensioni ridotte nella preparazione di demo e mockup; dall'altro, mantenere un elevato livello di controllo sugli standard di sviluppo, riducendo la barriera all'ingresso per i nuovi sviluppatori e rendendo più ripetibile il processo di generazione assistita. I capitoli successivi mostreranno in dettaglio come questa soluzione sia stata applicata a casi reali e quali risultati siano stati osservati in termini di qualità del codice, velocità di implementazione e facilità di onboarding.

Capitolo 3

Implementazione della soluzione

In questo capitolo si entra nel dettaglio operativo dell'architettura proposta, descrivendo passo per passo le fasi che hanno caratterizzato la descrizione della pipeline agentica utilizzata in Laif. Se nel capitolo precedente è stata presentata una visione ad alto livello del ruolo degli agenti e degli strumenti (Windsurf, Lovable, MCP), qui l'attenzione si sposta sulla loro implementazione concreta: come viene raccolto il contesto, in che modo vengono costruiti i prompt, quali chiamate ai modelli OpenAI vengono effettuate e quali meccanismi di controllo vengono applicati sui risultati. Vengono quindi analizzate le strategie di gestione del contesto (inclusione di documentazione, schema del database, regole versionate), le tecniche di prompting utilizzate e le fasi di generazione e validazione del codice.

3.1 Raccolta del contesto

La comprensione del contesto è il punto di partenza per ogni agente. Essa si articola in due componenti principali: l'analisi *statica* e quella *dinamica*.
Nella fase statica, l'agente legge la struttura del progetto e i file sorgente disponibili. In un progetto software, questo significa estrarre l'albero delle directory, le dipendenze dichiarate (ad esempio in `package.json` o `requirements.txt`) e, quando possibile, analizzare la sintassi tramite *Abstract Syntax Tree* (AST).

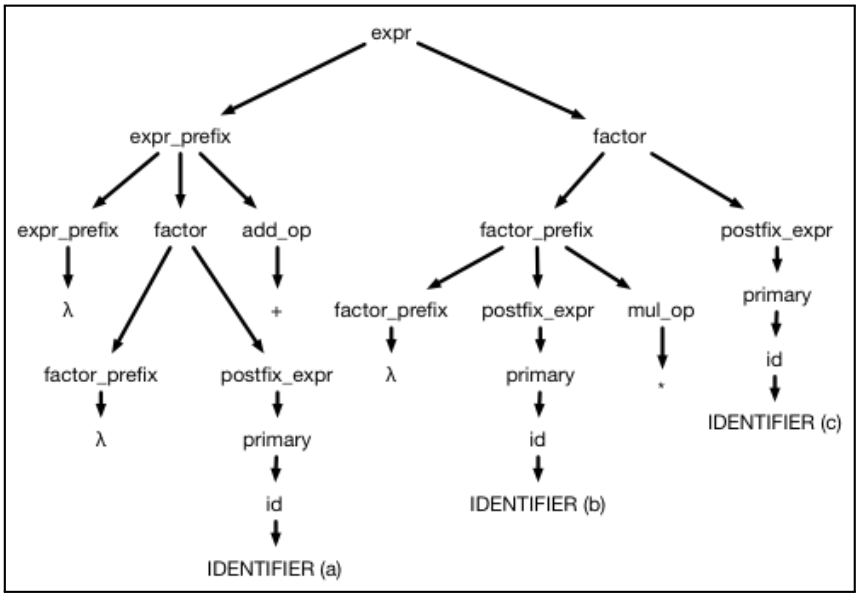

Figura 3.1: AST

Rappresentazione ad albero della struttura sintattica di un programma

L'AST permette di ricostruire le relazioni semantiche tra classi, funzioni e variabili e costituisce una rappresentazione intermedia utile per ragionamenti di tipo semantico (ad esempio, individuare dove una funzione è definita e dove è utilizzata).
Nella fase dinamica, l'agente osserva l'esecuzione del codice e i feedback del sistema. Questo include la raccolta di messaggi d'errore, stack trace, output dei test unitari o funzionali, e persino comandi eseguiti in un terminale controllato (es. `pytest`, `npm run build`, `docker compose`, ecc...). Tali segnali sono fondamentali perché forniscono informazioni non direttamente ricavabili dalla sola analisi sintattica.

Per gestire progetti di grandi dimensioni e, dunque, leggere i file disponibili, l'agente non si limita ad aggregare tutto il codice sorgente nella **finestra contestuale** del modello. Un LLM infatti, ha una finestra di contesto limitata e può leggere e ragionare solo su un certo numero di token. D'altra parte, un progetto può contenere anche centinaia di milioni di righe di codice organizzate in centinaia, se non migliaia, di file diversi.
Gli LLM più all'avanguardia hanno a disposizione finestre di contesto molto estese (fino

a 10M token per Llama 4 Scout; 2,097,152 token per Gemini 1.5 Pro), che permettono di fornire all'LLM parti ampie del progetto o corpora multi-documento; tuttavia, *non* è sufficiente "incollare tutto" nel prompt. Studi sistematici mostrano che l'aggiunta di contenuto irrilevante o la collocazione sfavorevole dell'informazione chiave causano **degradazione della qualità**: il fenomeno *lost-in-the-middle* documenta come l'accuratezza cali quando le informazioni rilevanti si trovano in posizioni centrali di un contesto molto lungo [Liu+24], mentre benchmark controllati evidenziano che **contesto irrilevante** distrae il ragionamento e riduce la precisione, specialmente con più distrattori [Yan+25]. In scenari RAG, l'aumento dei passaggi recuperati oltre un certo punto può introdurre "*hard negatives*" che peggiorano l'output, richiedendo il riordino del retrieval o del fine-tuning mirato alla robustezza [Jin+25].
Quando un agente o un sistema RAG recupera documenti per arricchire il contesto del modello, non tutti i documenti sono ugualmente utili; li dividiamo in:

- **Positivi**: frammenti che contengono la risposta o informazioni rilevanti.
- **Negativi facili** (easy negatives): documenti evidentemente irrilevanti (es. un articolo sportivo quando si cerca documentazione Python).
- **Negativi difficili** (hard negatives): documenti che sono molto simili superficialmente alla query (quindi hanno embedding vicini), ma non contengono la risposta corretta o contengono informazioni fuorvianti o contraddittorie.

Tabella 3.1: Confronto sintetico di modelli per finestra di contesto [LLM].

Modello	**Contesto (token)**
Llama 4 Scout	10,000,000
Gemini 1.5 Pro	2,097,152
Gemini 1.5 Flash	1,048,576
Gemini 1.5 Flash 8B	1,048,576
Gemini 2.0 Flash	1,048,576
Gemini 2.0 Flash-Lite	1,048,576
Gemini 2.5 Flash	1,048,576
Gemini 2.5 Flash-Lite	1,048,576
Gemini 2.5 Pro	1,048,576
Gemini 2.5 Pro Preview 06-05	1,048,576
GPT-4.1	1,047,576
GPT-4.1 mini	1,047,576
GPT-4.1 nano	1,047,576
Llama 4 Maverick	1,000,000
GPT-5	400,000
GPT-5 mini	400,000
GPT-5 nano	400,000

Il numero rappresenta la massima finestra di contesto gestibile (in token) per ciascun modello per singola richiesta.

Se anche fosse possibile fornire tutto il codice sorgente ad un ipotetico LLM con finestra di contesto illimitata, il modello riceverebbe molto rumore irrilevante che farebbe calare drasticamente la qualità delle risposte.
Viene dunque costruito un **contesto organizzato**:

$$C = \left(C_{\text{static}},\ C_{\text{dynamic}},\ C_{\text{retrieved}}\right)$$

dove:

- C_{static} rappresenta la conoscenza ottenuta dall'analisi dei file e dell'AST;
- C_{dynamic} raccoglie log, errori e output runtime;

- $C_{\text{retrieved}}$ è l'insieme di documenti recuperati da basi esterne (wiki, database, API) mediante tecniche di *retrieval-augmented generation* (RAG) [Lew+20].

Il recupero avviene tramite **modelli di embedding** (es. `text-embedding-3-large` di OpenAI), che mappano documenti e query in uno spazio vettoriale $\mathbb{R}^d$ (dove d è la dimensione del vettore di embedding). L'agente calcola quindi la similarità (tipicamente coseno) per selezionare i frammenti più rilevanti per consentire all'agente di includere nel prompt solamente i frammenti più pertinenti selezionati dal *retriever*, spingendo il contesto effettivo ben oltre la capacità di attenzione del modello, senza sovraccaricarlo con tutto il codice sorgente.

3.2 Pianificazione esplicita (*plan/act*)

Una volta acquisito il contesto, l'agente passa alla fase di pianificazione. Diversamente dagli approcci basati esclusivamente su predizione sequenziale, gli agenti moderni adottano un ciclo deliberativo, producendo, come già accennato, un vero e proprio piano testuale multi-step, che può includere azioni come "creare un nuovo file", "scrivere una classe `Customer`", "eseguire i test di integrazione" o "leggere la documentazione dell'API". Questo approccio, teorizzato nel paradigma **ReAct** [Yao+23], prevede la combinazione di ragionamento testuale (*reasoning*) e azioni concrete (*acting*).

Molti agenti implementano un ciclo deliberativo basato su **chain-of-thought** nascosta [Wei+22] e su **Planning con Large Language Models** [Wan+23]. Un tipico algoritmo adottato (ispirato a ReAct) è:

a) il modello genera un piano multi-step (`Plan`);

b) l'utente o un `Policy Manager` approva il piano;

c) il modello esegue azioni atomiche (`Act`), come creare un file o lanciare i test;

d) il ciclo continua fino al raggiungimento dell'obiettivo.

OpenAI Codex e le versioni specializzate per GitHub Copilot sfruttano varianti di **reinforcement learning from human feedback (RLHF)** per ottimizzare la pianificazione e l'esecuzione, riducendo errori e allucinazioni.

Il valore aggiunto di questo schema è duplice: da un lato garantisce trasparenza nei confronti dell'utente, che ha la possibilità di esaminare e modificare il piano proposto; dall'altro consente di correggere eventuali errori durante l'esecuzione, seguendo un ciclo iterativo che ricorda da vicino quello adottato da uno sviluppatore umano.

Formalmente, ad ogni iterazione t si può descrivere il processo come

$$(a_t, r_t) \sim \pi_\theta(a \mid C, h_{t-1}),$$

dove a_t rappresenta l'azione scelta, r_t la risposta dell'ambiente (ad esempio l'esito di un test), C il contesto corrente e h_{t-1} la storia delle interazioni precedenti.
Il simbolo "$\sim$" indica che la coppia (a_t, r_t) è campionata da una **policy stocastica** π_θ: non si tratta dunque di una relazione deterministica, ma di un processo probabilistico in cui, dato lo stato definito dal contesto e dalla storia, la policy specifica una distribuzione sulle possibili azioni da cui viene selezionata a_t. L'ambiente restituisce quindi una ricompensa o un segnale di feedback r_t in funzione dell'azione compiuta.

Nel contesto di OpenAI, come anche di altri LLM paragonabili, questa fase di scelta viene ulteriormente potenziata da tecniche di addestramento basate su ***reinforcement learning from human feedback (RLHF)*** [Chr+17] e sul più recente ***reinforcement learning from AI feedback (RLAIF)*** [Bai+22]. In entrambi i casi, la policy π_θ non viene ottimizzata solo tramite dati supervisionati, ma anche attraverso segnali di preferenza esterni: nel caso di RLHF derivano da annotatori umani, mentre in RLAIF sono generati automaticamente da altri modelli o da metriche predefinite, come la correttezza di compilazione o il successo nei test.

3.3 Tooling e protocolli

Un elemento che distingue un agente realmente efficace è la sua capacità di interfacciarsi con strumenti esterni. A questo scopo sono stati sviluppati protocolli come il **Model Context Protocol** (MCP) [Hou+25], che forniscono un'interfaccia standard per collegare l'agente a database, sistemi di knowledge management o API aziendali. MCP consente di trattare tali risorse come vere e proprie "estensioni di contesto", richiamabili on demand dal modello.

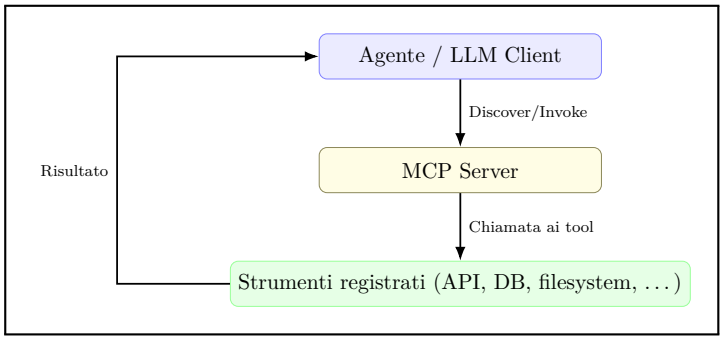

Figura 3.2: Schema di flusso di funzionamento degli **MCP**: l'agente comunica con il server MCP per scoprire e invocare strumenti, che a loro volta accedono a risorse esterne e ritornano i risultati.

OpenAI ha introdotto inoltre la **Function Calling API** [Opea], che trasforma le predizioni testuali in invocazioni strutturate di funzioni. In questo schema, l'LLM non produce soltanto testo libero ma genera output in formato JSON, immediatamente traducibile in chiamate a procedure lato client (ad esempio `create_file()` o `query_database()`). Ciò garantisce maggiore robustezza e riduce l'ambiguità semantica tipica dell'output naturale.

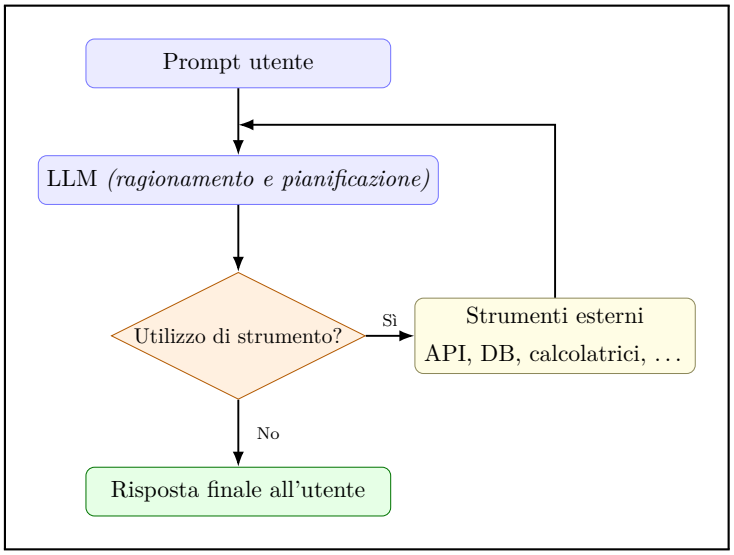

Figura 3.3: **Function Calling API**: il modello non esegue codice; emette JSON strutturato che il client traduce in chiamate di funzione controllate.

Più in generale, l'integrazione di strumenti è al centro di paradigmi come **Toolformer** [Sch+23], dove i modelli vengono addestrati a decidere autonomamente *quando* e *quali* strumenti invocare. Tuttavia, questo introduce anche sfide di **sicurezza**: l'esecuzione di codice o l'accesso a dati sensibili richiede **sandboxing**, **autorizzazioni esplicite** e **tracciamento delle azioni** per mantenere affidabilità e controllo umano.

3.3.1 Revisione e sicurezza

Ogni modifica proposta dall'agente deve essere tracciabile e sicura. A tal fine, i sistemi più maturi adottano strategie di **diff-based editing**, dove l'output consiste in patch incrementali applicabili ai file, piuttosto che in interi blocchi di codice.
Questo non solo facilita la revisione manuale, ma riduce il rischio di sovrascritture accidentali.

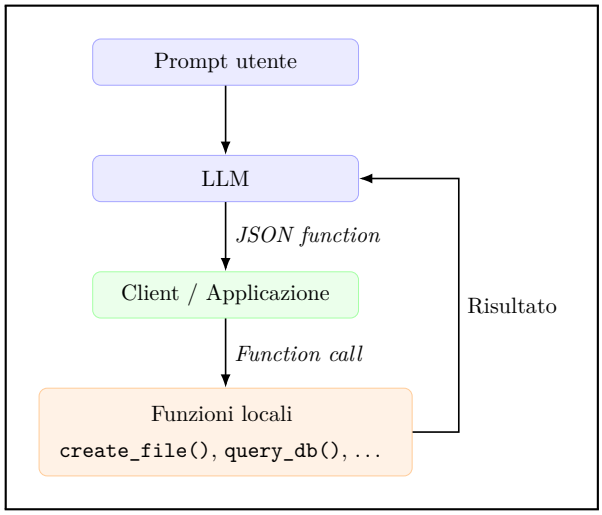

Figura 3.4: Paradigma **Toolformer**: il modello decide se usare uno strumento; in caso affermativo invoca il tool, ne integra il risultato e prosegue lungo il flusso principale.

Figura 3.5: Diff-Based Editing su **Windsurf**

Dal punto di vista architetturale, si stanno affermando soluzioni **client-first** o **BYOK** (*bring your own key*), in cui il codice e le credenziali rimangono sulla macchina locale e non vengono trasmessi a server remoti senza esplicita autorizzazione dall'utente, Questo tipo di approccio riduce la superficie di attacco e i rischi di *data leakage*. Un esempio concreto è l'estensione **Cline** per VS Code [Mar25a], che adotta un'architettura in cui l'LLM remoto riceve solo prompt sintetizzati e mai l'intero contenuto sensibile del progetto, lasciando l'esecuzione dei comandi e l'accesso ai file al client locale.

A livello di sicurezza, OpenAI integra sistemi di **content moderation** e **policy**

enforcement [Opeb], progettati per prevenire la fuoriuscita di dati personali, segreti aziendali o informazioni protette. Questi meccanismi si basano su filtri addestrati per intercettare input/output rischiosi e bloccare l'esecuzione di operazioni non conformi. Tecniche complementari includono la **red-teaming automation** [Per+22], in cui squadre di ricercatori (o agenti automatici) generano scenari malevoli per testare la robustezza dei filtri e la **Constitutional AI** [Bai+22], un insieme di principi normativi e linee guida predefinite (es. "non produrre contenuti dannosi", "fornire risposte utili ed oneste"), che il modello utilizza come criteri di revisione, riducendo la necessità di supervisione umana.

Più in generale, la letteratura recente in sicurezza degli LLM sottolinea la necessità di combinare diversi livelli di difesa: isolamento del contesto (sandboxing), controlli sulle chiamate a strumenti esterni, e auditing delle interazioni [She+23; Kas22]. Questo porta verso un modello di sicurezza stratificato, in cui i protocolli di interfacciamento (es. MCP), i *moderation systems* e le architetture client-first concorrono insieme a garantire affidabilità e *compliance* normativa.

3.3.2 Sintesi

In definitiva, un agente di generazione di codice può essere visto come un **sistema ibrido**, dove un LLM funge da policy centrale che orchestra strumenti, contesto e revisione. L'evoluzione recente mostra una tendenza verso agenti sempre più autonomi e integrabili, capaci non solo di generare codice ma di **pianificare**, **agire**, **verificare** e **interagire** in un ciclo simile a quello di uno sviluppatore umano, con il vantaggio di una velocità e di una capacità di scalare informazioni che superano ampiamente quelle delle pratiche tradizionali.

3.4 Impostazione dello studio

La ricerca è stata organizzata come **lavoro di gruppo a 5**, coordinata da un responsabile ed è stata suddivisa in 4 fasi:

a) **Fase 1 – Ricerca dell'IDE (o del plugin VSCode)**: La prima fase è stata una fase di sperimentazione pratica atta a determinare quale fosse l'IDE "agentico" o il plugin per VSCode da impiegare come standard aziendale.

b) **Fase 2 – Definizione delle regole e dei server MCP**: La seconda fase è stata la fase di definizione di regole in formato **markdow** che coprissero tutte le specifiche da sottoporre all'agente in fase di generazione delle risposte in modo da ottenere codice che seguisse gli standard d'azienda e da poter fornire un *rules-set* ad ogni sviluppatore in Laif. Oltre a ciò in questa fase è stato definito l'insieme dei server MCP da far utilizzare all'agente.

c) **Fase 3 – Test di validazione**: La terza fase è stata quella forse più importante ed interessante. La fase di test e valutazione dell'efficacia del lavoro svolto: è stato eseguito il *porting* di un'applicazione reale per un cliente reale tentando di impiegare meno risorse umane possibile. In questa fase sono stati raccolti dati sia quantitativi che qualitativi finalizzati alla valutazione definitiva della soluzione trovata.

d) **Fase 4 – Raccolta e analisi dei risultati**: L'ultima fase è stata quella di raccolta, elaborazione e studio dei dati raccolti in **fase 3** con l'obiettivo di determinare la validità dello strumento (o pipeline di strumenti) definiti ed eventualemente renderlo uno standard aziendale.

3.4.1 Ricerca dell'IDE (o del plugin VSCode)

Il gruppo di ricerca è formato da sei individui, di cui uno (**M.P.**) è il responsabile del lavoro; Il lavoro è stato organizzato per testare quattro soluzioni possibili selezionate fra le più popolari ed efficaci, mantenendo, quando possibile, il workflow su VS Code (per evitare cambi di IDE):

- **Cline** (estensione VS Code), assegnato a **il sottoscritto, G.F.**, è un agent open-source per VS Code con *Plan/Act*, uso trasparente dei token, BYOK e forte integrazione MCP; gira *client-side* (codice e segreti non passano su server proprietari) [Cli; Mar25a; Doc25].
- **Roo Code** (fork potenziato di Cline), assegnato a **C.V.**, nasce come fork potenziato di Cline; aggiunge *modi* configurabili (code-review, test), esegue comandi e integra MCP; una review tecnica indipendente riporta buoni risultati pratici ma un consumo di token rilevante (BYOK) e una sessione per finestra [Qub25; Cod25; Mar25b].
- **Windsurf** (IDE di Codeium), assegnato a **M.P. e C.P.**, è un IDE con agente (*Cascade*) che orchestra passaggi multipli, raccoglie contesto automaticamente e propone piani eseguibili; diverse analisi gli attribuiscono un'ottima *codebase awareness* e una UI curata, con pricing competitivo [Cod] [Bui25][Zap][Dat25].
- **Cursor** (IDE basato su un fork di VS Code), assegnato a **S.B.**, è un IDE "AI-first" basato su un fork di VS Code, con assistente integrato e supporto a più modelli (OpenAI/Anthropic ecc.), orientato a refactor multi-file, debug e implementazioni guidate [Lab25] [Ver25]. Confronti indipendenti lo descrivono come molto rapido e con grande base utenti, ma non sempre il migliore nel ragionamento profondo su codebase estese rispetto a Windsurf [Zap] [AI25] [Dat25].

Il lavoro di ricerca dell'IDE è stato diviso in due fasi della durata complessiva di **15 giorni lavorativi**; per i primi 5 giorni ci è stato chiesto di iniziare ad impiegare gli agenti (in modalità **Pro** tramite una *Api key* ***OpenAI*** fornita dall'azienda) per lo svolgimento dei *task* quotidiani relativi al nostro ruolo in azienda, il tutto a partire da due file di regole e descrizione dei flussi e delle tecnologie di lavoro in formato markdown condivisi fra tutti da fornire all'agente:

- Il file **laif-rules.md** contenente una descrizione dettagliata della struttura dello stack tecnologico di Laif (quindi descrizione del template, delle librerie impiegate, dei flussi di lavoro e dell'organizzazione generale di un qualunque progetto dell'azienda).

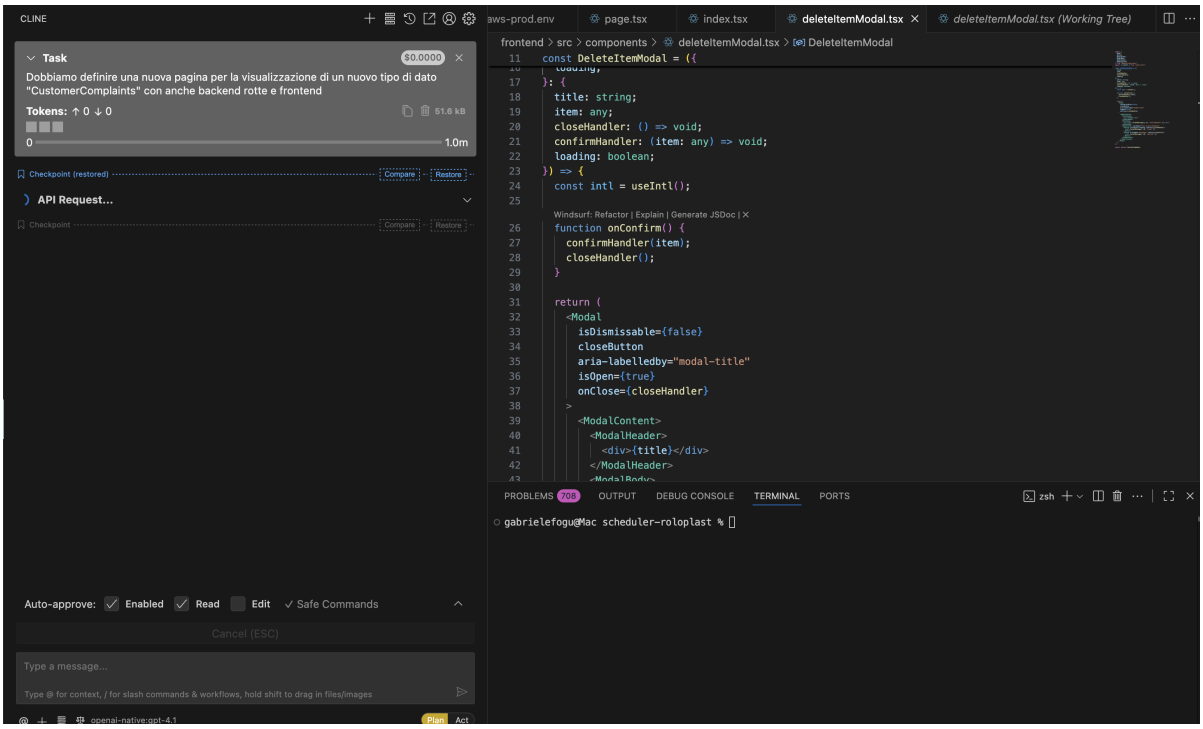

Figura 3.6: Interfaccia della finestra integrata di Cline all'interno di VSCode durante una richiesta in fase di planning.

- Il file **global.md** Contenente indicazioni su come fornire le risposte e su come organizzare il flusso di generazione del codice, dunque:

  a) Partire dalla definizione di un **data-model** se non presente e del relativo **schema**.

  b) La definizione delle rotte CRUD e dei service necessari a soddisfare i requisiti definiti nel prompt.

  c) L'invocazione di uno script proprietario per la generazione di un *data-access layer* TypeScript completo e tipizzato, composto da **types.gen.ts** per tutte le definizioni di tipo, **client.gen.ts** che emette un **client Axios** configurabile, **sdk.gen.ts** che genera una funzione **low-level 1:1** con ogni endpoint più altri file **wrapper** e di **configurazione**.

  d) In fine l'implementazione del frontend e il *binding* fra interfaccia e dati.

Questa fase si è conclusa con una riunione di condivisione dei risultati ottenuti fra tutti i componenti del team di ricerca.
Dal mio canto è stato evidente da subito che Cline fosse un plugin già di per se molto efficace nella gestione della parte backend in special modo. L'agente è stato in grado di definire data-model, *schemas*, rotte backend e soprattutto **servizi anche molto articolati** in python senza troppi problemi.

Nello specifico, Cline fornisce due modalità per l'agente che lo rendono particolarmente efficace e di facile utilizzo:

- La modalità **pianificazione** nella quale l'agente definisce la scaletta delle azioni che intende svolgere per soddisfare le richieste descritte dal prompt; in questa fase è possibile interagire tramite la chat integrata a VSCode per correggere eventuali errori o scelte poco chiare che l'agente intende seguire.

- La modalità **"*execute*"** nella quale l'agente passa all'effettiva modifica della *codebase* per l'implementazione pratica delle richieste.

Anche per quanto riguarda il frontend il lavoro mi era stato agevolato notevolmente, tuttavia è apparso evidente che l'agente non fosse stato in grado di interpretare correttamente le mie richieste in termini di estetica, di usabilità e di organizzazione del layout.
Se da una parte il mio intervento è stato essenziale al perfezionamento del codice in modo che fosse anche solo eseguibile senza errori, d'altra parte è stato da subito evidente che il potenziale dello strumento fosse di elevato interesse per l'azienda in quanto l'agente era stato in grado di fornire anche a frontend un ottimo punto di partenza alla mia fase di sviluppo "umano" se paragonato al "foglio bianco" con cui si ha a che fare nello sviluppo tradizionale. Va considerato inoltre che si tratta di quella che potremmo identificare come la fase **1.1** della ricerca e che non ci si aspettava risultati neanche vicini all'ottimalità.

Inoltre, è importante ricordare che in questa fase **il design system di Laif non era ancora stato rilasciato** e il frontend delle nostre codebase era costituito da componenti ed interfacce poco standardizzate, per chiarirci: non era insolito trovare una stessa soluzione implementata utilizzando componenti e librerie differenti. Una volta condivisa la mia esperienza, ascoltata quella degli altri e presa nota dei punti di forza e di debolezza di ogni soluzione testata, non potendo affermare con certezza quale agente fosse il migliore relativamente all'utilizzo che ne volevamo fare, si è passati alla fase successiva del lavoro, nella quale ci è stato chiesto di continuare ad utilizzare gli agenti che ci erano stati assegnati, questa volta però con l'obiettivo di renderli il più raffinati possibile tramite la stesura di regole markdown più specifiche ed elaborate e tramite l'impiego di server MCP a nostra discrezione.

3.4.2 Definizione delle regole e dei server MCP

Fase della durata di 10 giorni che si è rivelata particolarmente dinamica e complessa, poiché non era possibile definire a priori un insieme di regole che potessero considerarsi ottimali in modo definitivo. Per questo motivo, una volta scelto l'agente da utilizzare in modo arbitrario, ho deciso di adottare un approccio iterativo articolato in più step:

a) In primo luogo, individuavo un *task* reale del progetto su cui stavo lavorando e

fornivo all'agente un prompt il più possibile dettagliato e privo di ambiguità, chiedendogli di proporre una soluzione completa.

b) Poiché l'output iniziale risultava spesso distante dal risultato atteso, la fase successiva consisteva nel definire o affinare un insieme di **regole in formato Markdown** pensate per ridurre l'ambiguità del prompt e orientare meglio la generazione del codice.

c) In base alla qualità dell'output prodotto:

   - se l'output era **inutilizzabile**, scartavo il codice e riproponevo la richiesta in una **nuova istanza di chat**, così da evitare che il contesto della conversazione precedente influenzasse le risposte successive;
   - se l'output risultava **accettabile**, mantenevo la conversazione e proseguivo lo sviluppo a partire da quel risultato, eventualmente estendendolo o correggendolo.

   In entrambi i casi continuavo la conversazione fino al completamento del task o all'avvio di uno nuovo.

d) Infine, quando mi trovavo ad affrontare un task simile a uno già gestito in precedenza, procedevo a **raffinare ulteriormente la regola corrispondente**, incorporando le osservazioni e le correzioni emerse dalle iterazioni precedenti.

Parallelamente alla definizione progressiva delle regole, ho configurato una serie di **server MCP** (Model Context Protocol) per fornire all'agente accesso diretto e strutturato alle risorse aziendali. Nello specifico, ho individuato due integrazioni fondamentali:

- **PostgreSQL**, che tramite una singola *connection string* espone all'agente un insieme di strumenti per l'esecuzione di query SQL e per l'introspezione del catalogo (schemi, tabelle, colonne, vincoli, tipi – inclusi gli `ENUM`), offrendo così un accesso **diretto** e **tipizzato** alla base dati;

- **Notion**, che attraverso un *token* con ambito ristretto all'area di interesse, permette all'agente di ricercare e recuperare la documentazione aziendale (pagine, database, blocchi), preservandone la struttura gerarchica e le proprietà (titoli, campi, relazioni, tag), fornendo un accesso **organizzato** alla *knowledge base* documentale.

Queste due connessioni rappresentano i primi esempi di **integrazione diretta tra agente e risorse aziendali**, consentendo un retrieval di informazioni più affidabile e contestualizzato rispetto all'uso di fonti esterne o di prompt puramente testuali.
L'agente, grazie a tali connessioni, è in grado di combinare la generazione di codice con la conoscenza strutturata, migliorando sensibilmente la qualità e la coerenza delle risposte.
Una volta decorso il periodo di 10 giorni che avevamo disposto per questa fase dello studio, abbiamo affrontato una riunione di condivisione delle informazioni e discussione dei risultati ottenuti da ognuno dei partecipanti, valutati tramite parametri soggettivi di qualità dello strumento e del codice in output.
A questo punto, sotto indicazione del *Project Manager*, **C.V.** ha ricevuto il compito di sintetizzare dai file di regole prodotti da ognuno di noi un insieme ristretto di regole comuni da impiegare nell'ultimo passaggio di questa fase: predisporre una base condivisa per testare l'efficacia degli strumenti su un **benchmark unificato**.

3.4.3 Definizione del benchmark

Per garantire omogeneità nei test, il gruppo ha concordato di misurare le performance di ciascun agente nella generazione di una **pagina completa end-to-end**, comprendente:

- la definizione del **data model** con campi principali e relazioni;
- la generazione delle **rotte API** (CRUD) e del relativo **service** backend;
- la creazione del **frontend** conforme al design system aziendale;
- una **pagina di visualizzazione dati** sotto forma di tabella, con funzionalità di creazione, aggiornamento ed eliminazione dei record.

L'obiettivo era verificare se ciascun agente fosse in grado di produrre una funzionalità completa, coerente con gli standard aziendali, riducendo al minimo gli interventi manuali e mantenendo un equilibrio tra qualità del codice, coerenza architetturale e produttività complessiva.

3.4.4 Risultati emersi

Nei nostri scenari di test, costituiti da repository `TypeScript/React` per il frontend e `FastAPI/SQLAlchemy` con migrazioni per il backend, **Windsurf** ha offerto il miglior equilibrio tra:

- *codebase awareness* (capacità di ricostruire pattern di progetto e rispettare i vincoli architetturali);
- qualità dei piani multi-file (refactor e implementazioni additive con minima perdita di contesto);
- integrazione "senza attrito" con terminale, task, lint e test;
- costo prevedibile su base *seat*/trial rispetto agli altri flussi sperimentati.

Cursor si è dimostrato estremamente fluido e veloce per iterazioni individuali e incrementi di piccola scala, mentre **Cline** e **Roo Code** hanno convinto quando l'obiettivo era restare *dentro VS Code*, mantenendo pieno controllo sui modelli (BYOK) e un'integrazione diretta con MCP.
I risultati ottenuti sono risultati coerenti con le analisi pubbliche disponibili in letteratura e nei test comparativi esterni [AI25; Dat25; Zap], confermando che **Windsurf** risulta leggermente superiore nel *deep codebase work*, mentre **Cursor** eccelle in rapidità e diffusione tra la community.
In sintesi, il benchmark condiviso ha permesso di evidenziare punti di forza e limiti concreti di ogni soluzione, fornendo al gruppo le evidenze necessarie per convergere su **Windsurf** come strumento di riferimento per la fase successiva di validazione su progetto reale.

3.5 Test di validazione

Questa terza fase della ricerca ha rappresentato il punto di svolta del lavoro, in quanto si è trattato del primo test di **validazione su progetto reale** partendo da zero. L'obiettivo principale era verificare la reale efficacia combinata delle componenti selezionate - l'IDE **Windsurf**, i **ruleset** aziendali e le integrazioni **MCP** - all'interno di un flusso di sviluppo effettivo, misurando quanto la *pipeline* agentica potesse ridurre l'intervento umano nel porting e nella ricostruzione di un'applicazione esistente.

In questo stesso periodo (tra la seconda e la terza fase dello studio) era emerso un nuovo strumento: **Lovable** [Lov], una piattaforma web-based di *AI App Building* sviluppata con l'obiettivo di consentire la creazione di interfacce applicative complete partendo da prompt in linguaggio naturale. Lovable adotta un approccio simile a quello dei moderni ambienti di generazione controllata (es. Replit AI o Builder.io), ma si distingue per alcune caratteristiche tecniche rilevanti nel contesto di questo studio:

- integra nativamente un **motore LLM multi-step** capace di generare struttura, componenti UI e logica di interazione in un unico ciclo;
- supporta la modalità **"Remix"**, che consente di *forkare* o duplicare un progetto esistente e di intervenire su di esso con nuovi prompt incrementali, favorendo l'iterazione rapida e il riutilizzo di codice;
- offre un **builder visuale interattivo** per modifiche dirette all'interfaccia, con esportazione immediata in `React/Next.js`, facilitando il passaggio da mockup a prototipo reale;
- produce codice front-end leggibile, modulare e aderente ai principi di componentizzazione tipici dei design system moderni (nel nostro caso, analoghi a `laif-ds`).

Si tratta di una piattaforma particolarmente adatta alla fase di *rapid prototyping*, e per questo motivo è stata integrata sperimentalmente nel processo di validazione: non come sostituto dell'agente IDE, ma come strumento complementare per accelerare la

definizione dell'interfaccia e fornire un primo "scheletro" di navigazione e layout coerente con lo stile aziendale.

Il **test di validazione** è stato assegnato a me e consisteva nell'eseguire il **porting di un'applicazione reale sviluppata da un'altra azienda verso il nostro ecosistema**, adattando la *codebase* al template e agli standard interni di *Laif S.r.l.* L'obiettivo operativo era duplice:

a) verificare la capacità dell'agente (tramite Windsurf e ruleset) di adattare codice preesistente ad un nuovo contesto architetturale e stilistico;

b) misurare quanto fosse possibile ridurre il carico manuale di riscrittura, delegando all'agente e ai suoi strumenti di pianificazione l'implementazione e la rifinitura dei moduli.

Per avviare questa fase, ho iniziato definendo un **set di prompt** all'interno di **Lovable**, partendo da un *remix* di una **demo predefinita** preparata dal *Project Manager* come baseline di riferimento. La demo rappresentava un'interfaccia utente con struttura e logica analoghe a quelle delle nostre applicazioni interne - quindi sidebar di navigazione, header con breadcrumb, tabella dati e viste CRUD - ed era stata arricchita da alcune **pagine aggiuntive** che simulavano funzioni tipiche dei nostri progetti, fungendo da *mockup* di un prodotto reale dell'azienda.

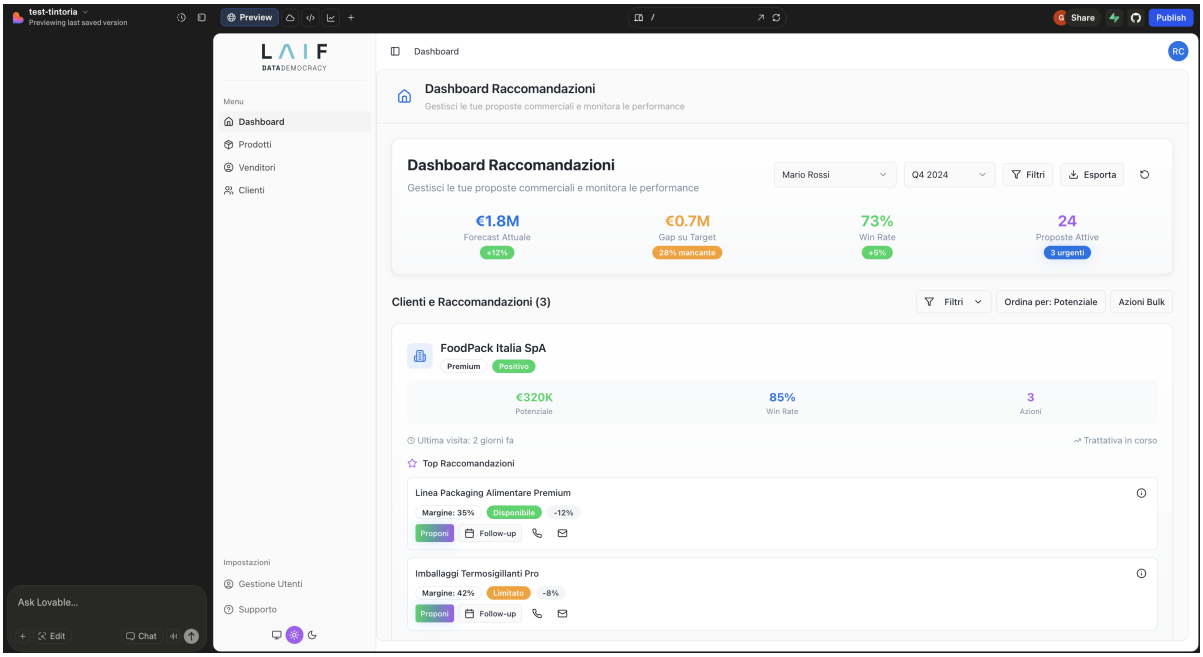

Figura 3.7: Interfaccia di *Lovable*: a sinistra la chat LLM per i prompt; a destra il progetto **laif-demo** usato come base dei **Remix**.

Attraverso il comando **Remix**, Lovable ha generato una nuova istanza del progetto (una sorta di branch parallelo) su cui ho potuto intervenire liberamente, combinando input visivo e testuale. A partire da questo punto, ho iniziato ad interagire con il sistema tramite prompt mirati, che specificavano:

- la struttura e l'organizzazione generale delle pagine (ad esempio: "crea una sezione di gestione utenti con tabella e form di creazione");
- il comportamento dei componenti dinamici (dialog di editing, validazioni, messaggistica);
- la coerenza stilistica con il design system aziendale (palette, tipografia, spaziatura coerenti con `laif-ds`);
- eventuali integrazioni simulate con backend (mock API o dati fittizi).

Il risultato di questa prima fase in Lovable è stato un **prototipo navigabile e coerente**, utile come base visiva e semantica per la fase successiva: il porting effettivo

in Windsurf. In questa seconda parte del test, ho importato l'output generato da Lovable nel mio workspace di **Windsurf**, dove ho ripreso il flusso di sviluppo utilizzando:

- il **ruleset aziendale** (in formato Markdown) per vincolare la generazione del codice ai nostri standard architetturali;
- i **server MCP** già configurati (PostgreSQL e Notion), così da fornire all'agente contesto reale sulla struttura dati e sulla documentazione interna;
- i comandi nativi dell'agente *Cascade* per orchestrare la generazione multi-file (schema → controller → service → frontend).

Durante questa fase, Windsurf ha dimostrato un'elevata capacità di **ricostruire la logica applicativa esistente**, riconoscendo pattern e nomi delle entità provenienti dalla codebase originale e traducendoli in strutture aderenti al nostro template FastAPI/SQLAlchemy sul lato backend e React/Next.js sul lato frontend.
Il sistema MCP ha rivestito un ruolo determinante nel processo di adattamento automatico delle entità e, più in generale, nella comprensione strutturale del dominio dati dell'applicativo da migrare. Grazie all'accesso in lettura **al database PostgreSQL dell'azienda di provenienza**, l'agente WindSurf ha potuto **ispezionare direttamente i datamodel dell'applicazione sorgente**, analizzando in modo sistematico schemi, tabelle, vincoli, relazioni e tipi di dato.
Questa capacità di esplorazione semantica ha consentito di generare una **migrazione iniziale automatica** in grado di portare il database interno del progetto - originariamente nello stato "template" - a una configurazione completamente **allineata 1:1 con il database dell'applicazione originaria**. Tale processo non si è limitato alla mera riproduzione delle strutture, ma ha incluso anche la definizione coerente dei vincoli di integrità, delle chiavi esterne e delle relazioni tra le entità, assicurando una corrispondenza logica e funzionale con il sistema di partenza. Una volta ottenuta questa corrispondenza strutturale, è risultato immediato procedere al **trasferimento dei dati reali** tramite lo script interno `transfer_data.py`, concepito per copiare in modo controllato i contenuti da un database all'altro. In questo modo, l'accesso MCP ha rappresentato non solo un ponte conoscitivo tra due domini dati, ma anche il punto

di partenza per un processo di allineamento e popolamento completamente automatizzato, che ha permesso di integrare senza soluzione di continuità l'applicativo migrato all'interno della nostra infrastruttura.

Il test, in sintesi, ha rappresentato un **caso realistico di porting aziendale** in cui Lovable e Windsurf sono stati impiegati in sinergia: il primo per accelerare la fase di prototipazione e di definizione dell'interfaccia, il secondo per assicurare l'aderenza del codice generato agli standard aziendali e la completa integrazione con il backend. La combinazione dei due strumenti ha permesso di verificare in pratica quanto il ciclo *plan/act/review* potesse essere applicato anche a scenari complessi, nei quali l'obiettivo non è generare da zero, ma **adattare e integrare codice preesistente** con il minimo sforzo manuale e massima coerenza stilistica e funzionale.

3.6 Descrizione dell'applicativo da migrare

L'applicativo oggetto della migrazione, di seguito denominato semplicemente **Tintoria**, è una piattaforma gestionale focalizzata sulla *schedulazione operativa* dei flussi di lavorazione in tintoria industriale. Originariamente realizzato e mantenuto da un altro fornitore, il progetto è stato successivamente **trasferito al nostro team** a seguito dell'impossibilità, da parte del precedente manutentore, di proseguirne il supporto evolutivo. Tale passaggio ha reso necessario un intervento di **migrazione strutturale e tecnologica**, al fine di assicurarne la continuità operativa, l'allineamento agli standard infrastrutturali interni e l'integrazione con la nostra toolchain.

3.6.1 Descrizione funzionale

Lo scopo dell'applicativo è **ottimizzare la pianificazione giornaliera delle lavorazioni** per massimizzare il numero di ordini completati, rispettando vincoli di capacità e sequenziamento delle macchine/impianti. Ogni *ordine* è composto da uno o più *item*, ciascuno associato a una sequenza di *fasi/operazioni* (routing) su specifiche risorse produttive. Il sistema supporta differenti **criteri di priorità**, configurabili per la generazione dei piani:

- *Priorità al cliente* (es. clienti strategici);
- *Priorità alla data di consegna* (minimizzazione del ritardo).

Gli **attori** principali sono:

- **Pianificazione**: definisce priorità, orizzonte temporale, vincoli e scenari; avvia lo scheduler e valida il piano;
- **Responsabile di produzione**: supervisiona saturazione risorse, colli di bottiglia e ribilanciamenti;
- **Operatori di linea**: consultano le code di lavorazione e eseguono avanzamenti;
- **Direzione**: monitora KPI di puntualità, throughput e saturazione.

I **processi principali** sono:

a) **Acquisizione e normalizzazione degli ordini**: import/registrazione di ordini e item;

b) **Definizione vincoli**: capacità per macchina/centro di lavoro, turni e calendari, tempi di set-up, compatibilità per materiale/trattamento, lotti min/max;

c) **Schedulazione**: generazione del piano tramite algoritmo ottimizzante (obiettivi tipici: massimizzare ordini completati, minimizzare tardività, ridurre tempi di attesa/setup), secondo il criterio di priorità selezionato;

d) **Esecuzione e avanzamento**: pubblicazione delle sequenze su ogni macchina, registrazione start/stop e stati operazione; gestione eccezioni (fermi, rilavorazioni);

e) **Monitoraggio e ricalcolo**: analisi KPI (*quality*, *saturation*, *OTIF*, *WOTIF*) ed eventuale riplanificazione;

f) **Ciclo giornaliero**: consolidamento serale e preparazione del piano del giorno successivo.

Le **entità dati** principali includono:

- *Ordine*, *Item*, *Operazione/Fase*, *Routing*;
- *Macchina/Centro di lavoro*, *Calendario/Turno*, *Matrice di setup/compatibilità*;
- *Lotto di lavorazione* (aggregazione di item omogenei), *Piano/Schedule*, *Run di schedulazione* (traccia esiti e parametri).

Black-box scheduling ed ETL

Per motivi organizzativi e di vincoli temporali, lo **schedulatore** e il **processo ETL** sono stati trattati come ***black-box***. Entrambi risultavano già funzionanti e validati in esercizio presso l'azienda di provenienza; di conseguenza, una loro revisione interna avrebbe comportato un'attività di ricerca e ottimizzazione che esulava dagli obiettivi della migrazione.
Durante il porting, il lavoro si è quindi concentrato sull'adattamento del contesto applicativo, sull'integrazione con l'infrastruttura esistente e sulla conservazione della piena compatibilità con i moduli di schedulazione preesistenti, garantendo la continuità operativa e la riproducibilità dei risultati.

3.6.2 Descrizione dell'interfaccia

L'interfaccia utente offre viste operative specifiche:

- **Gantt o Tabella di produzione**: sequenze per macchina con evidenza di colli di bottiglia e slack;
- **Visualizzazione degli ordini non *schedulati* e delle motivazioni** (macchine non disponibili, articoli senza parametri necessari, ecc...);
- **Visualizzazione e modifica dei parametri**: ogni schedulazione viene lanciata con parametri specifici che ne definiscono le priorità ed in generale **la funzione da minimizzare**;
- **Interfaccia per il lancio di una nuova schedulazione**;
- **Indicatori KPI**: (*quality*, *saturation*, *OTIF*, *WOTIF*)

Per completezza, si riportano di seguito alcune schermate significative dell'interfaccia utente, che illustrano le principali funzionalità operative del sistema.

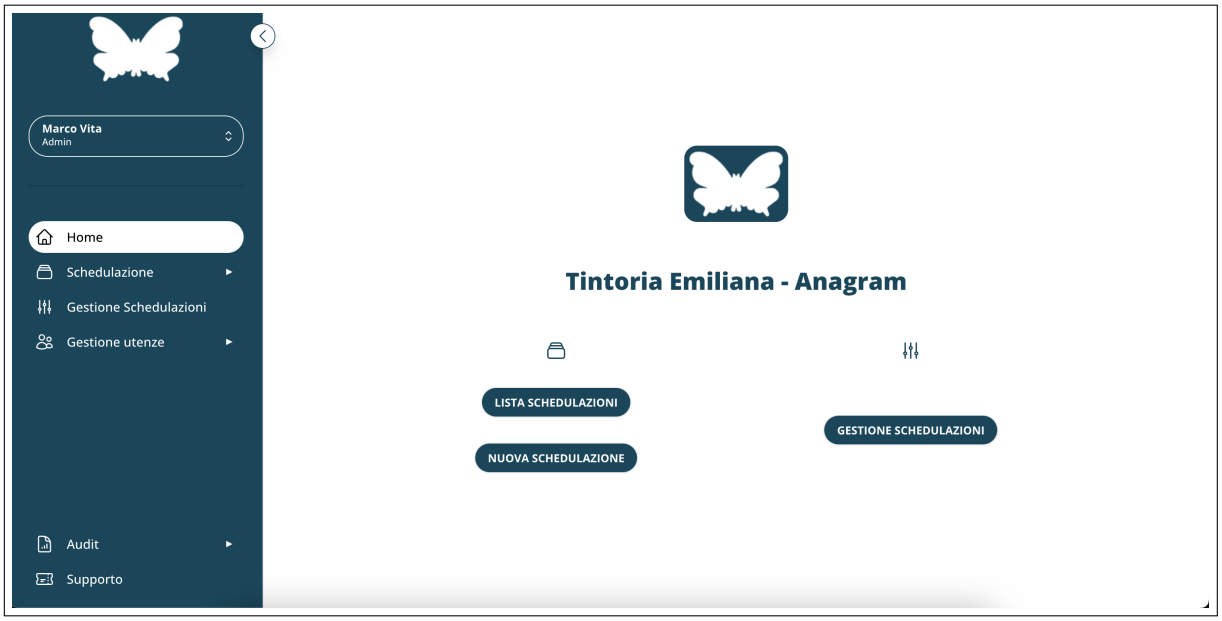

Figura 3.8: *Home* dell'applicativo

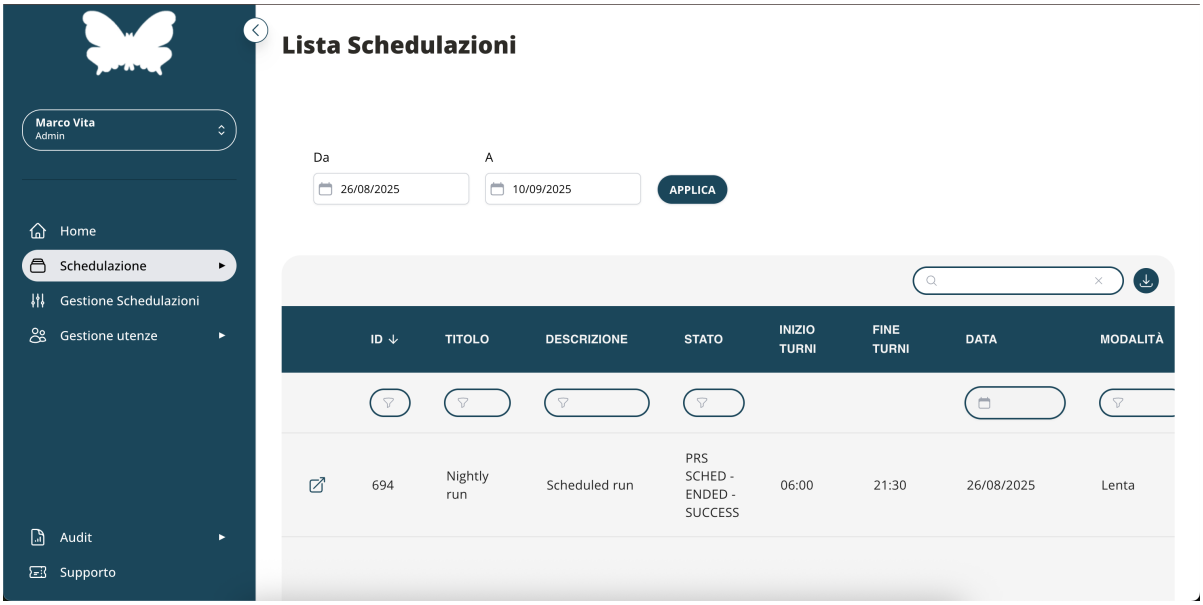

Figura 3.9: *Lista delle schedulazioni* con vista tabellare delle schedulazioni (nell'immagine l'unica schedulazione visibile per questioni tecniche è quella identificata dall'id **694**)

Cliccando con il mouse su di un elemento della tabella, si viene reindirizzati alla pagina di dettaglio, nella quale è possibile visualizzare le specifiche operazioni della schedulazione in modalità gantt o tabellare e gli **ordini non schedulati**:

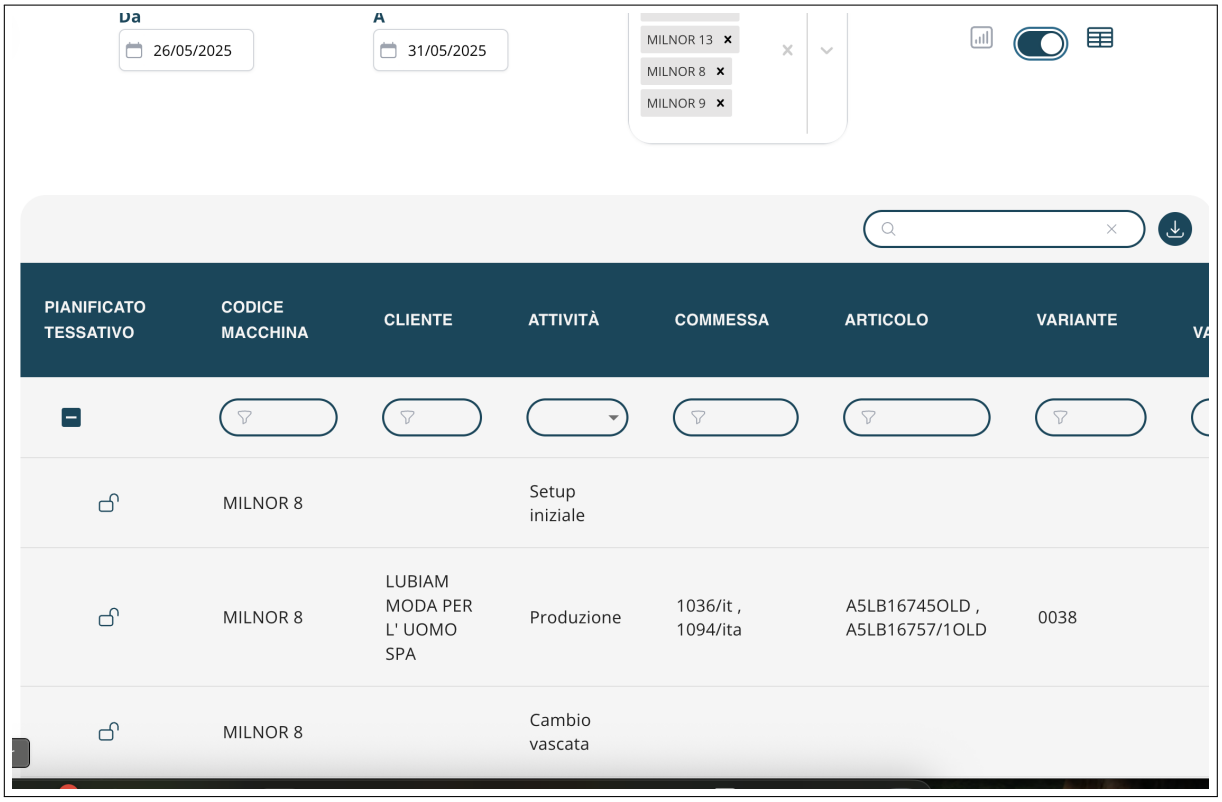

Figura 3.10: La vista tabellare è disponibile a prescindere dai filtri inseriti, mentre quella con grafico Gantt diventa disponibile tramite bottone *Switch* solo dopo aver selezionato almeno un centro di lavorazione nella *Select* in alto a sinistra della pagina.

Figura 3.11: Il Gantt permette la visualizzazione delle singole *task* della schedulazione, portando il mouse in *hover* su di un blocco del Gantt ne si vedono i dettagli e cliccandoci sopra si apre un modale che fornisce le informazioni in modo dettagliato. Notiamo poi un pulsante "MODIFICA" che abilita la modalità di *edit* dei valori dei blocchi tramite il modale di cui sopra. Inoltre, è possiblie tramite un input *Range* modificare l'intervallo di tempo del quale visualizzare i task.

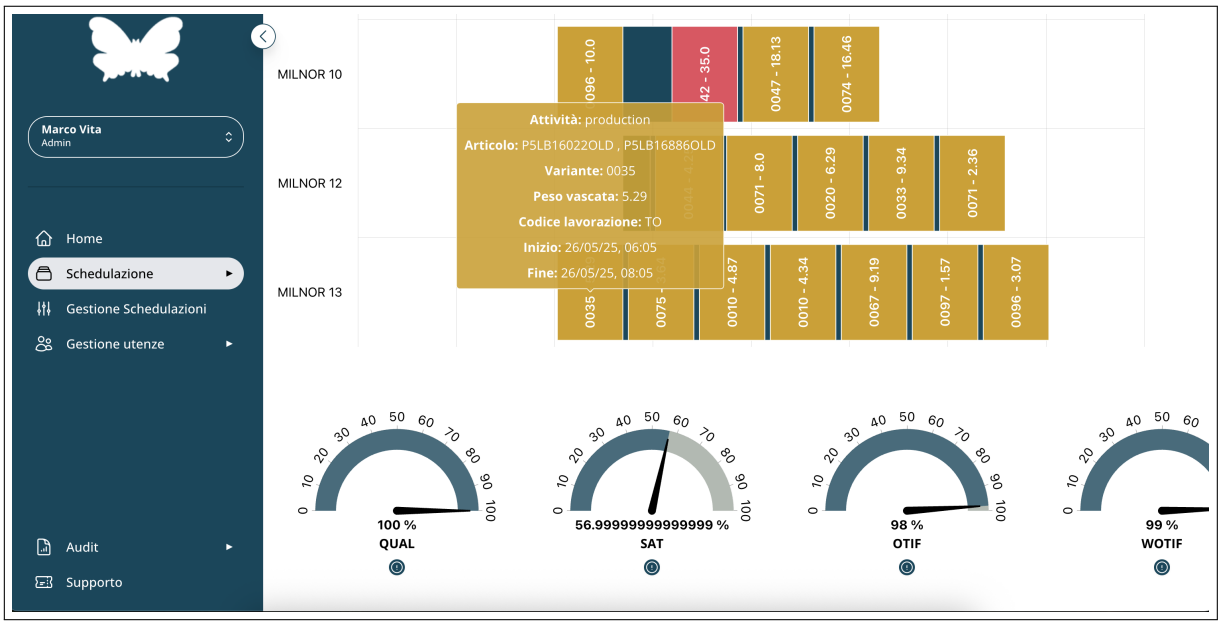

Figura 3.12: Oltre al Gantt, in questa visualizzazione sono visibili anche i valori dei *KPI*.

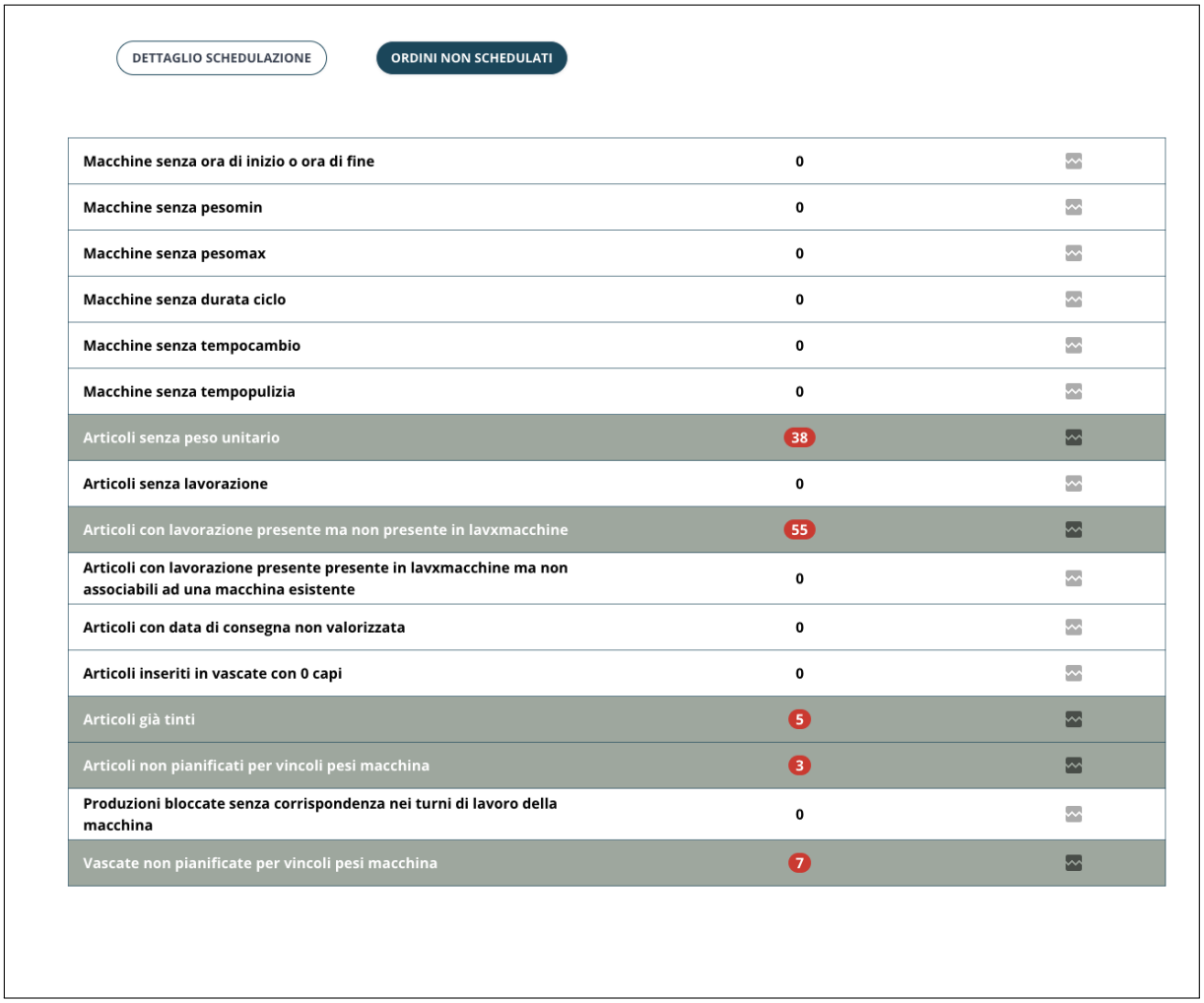

Figura 3.13: Gli ordini non schedulati organizzati per "problematica", in alto possiamo notare i bottoni che permettono di passare da questa vista a quella del dettaglio.

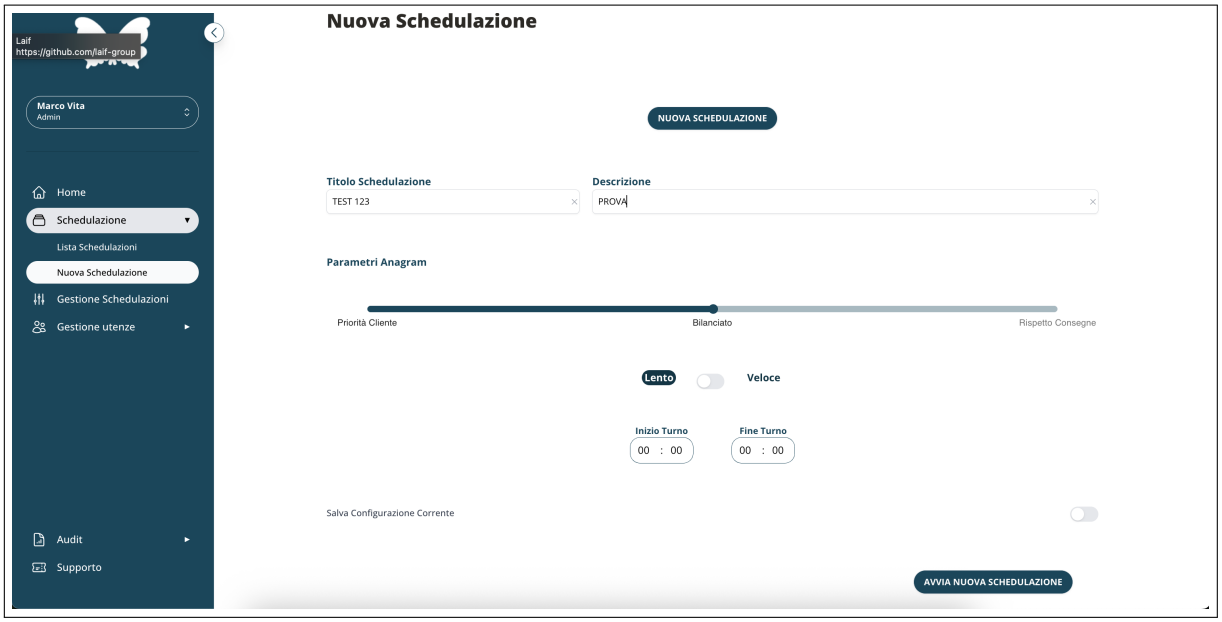

Figura 3.14: Pagina per il lancio di una nuova schedulazione, dove è possibile impostare titolo e descrizione oltre che ai parametri dell'ottimizzatore.

Per brevità non viene riportata la pagina della "Gestione Schedulazioni" che è possibile notare nella navigazione laterale, in quanto identica a quella del lancio di una nuova schedulazione a meno dei campi "titolo" e "descrizione" e della possibilità di lanciare la schedulazione, in quanto è una pagina dedicata alla definizione della configurazione *default*.

Come possiamo osservare, le pagine sono sostanzialmente 6, di cui solamente 4 esposte dalla barra di navigazione laterale:

1. **Home** del progetto, vedremo che è stato deciso di rimuovere questa pagina in quanto ritenuta superflua;

2. **Lista schedulazioni**: Pagina adibita alla visualizzazione tabellare dello storico delle schedulazioni, una volta selezionato un elemento dalla tabella si accede alla **Pagina dettaglio della schedulazione**, costituita da due pagine non esposte nella *navbar*:

2.1. **Pagina dettaglio di base**: fornisce la vista **gantt o tabellare** della schedulazione;

2.2. **Pagina ordini non schedulati**: fornisce la vista tabellare del numero di ordini non schedulati raggruppati per la causa della non-schedulazione;

3. **Nuova schedulazione**: pagina dedicata alla configurazione e al lancio manuale di una nuova schedulazione;

4. **Gestione schedulazioni**: pagina di impostazione della configurazione *default* delle schedulazioni;

3.6.3 Descrizione tecnica e architettura

Tintoria eredita l'impostazione tecnologica di un template generalizzato per applicazioni web e pipeline ETL, con esecuzione locale e distribuita.

- **Frontend**: SPA React servita via Nginx; viste Gantt/queue e pannelli KPI;
- **Backend**: FastAPI (Uvicorn) con CLI Typer (`run.py`) per ruoli e migrazioni Alembic;
- **Database**: PostgreSQL (schema versionato); entità per ordini, item, routing, risorse, calendari, schedule e run;
- **Schedulazione**: job Celery per generazione piani; code SQS per orchestrazione; persistenza dei risultati e dei parametri run;
- **Automazioni**: run giornaliero (notturno) per consolidamento e preparazione piano successivo; ricalcolo on-demand;
- **Documentazione e test**: Sphinx; pytest/httpx per validazione API e logiche di pianificazione;
- **Infrastruttura**: Docker Compose per ambienti locali; deploy su AWS tramite strumenti di provisioning (es. aCli/CDK).

Sicurezza e operatività

L'applicativo adotta pratiche standard di sicurezza e controllo operativo. I container backend vengono eseguiti con utenti non-root e le credenziali sono gestite tramite servizi di secret management in cloud. Le pipeline CI/CD garantiscono tracciabilità dei rilasci e coerenza tra ambienti. Le migrazioni di schema sono gestite con Alembic, mentre i run di schedulazione sono pienamente *auditabili* e riproducibili, con storicizzazione dei parametri e degli esiti di ogni esecuzione.

3.7 Fase "Lovable" - Remix demo, prompt e specifiche eseguibili

In questa fase **partendo dal progetto `laif-demo`** (mockup minimale con layout, architettura, componenti `laif-ds` già configurati), sfruttiamo la funzione **Remix** di *Lovable* per generare automaticamente una nuova codebase che replichi *struttura* e *funzionalità* dell'applicazione target, mantenendo stile e pattern Laif.
Questa fase è caratterizzata da 3 passaggi chiave:

- **Remix dal `laif-demo`**: Lovable clona la base e genera una **nuova repository GitHub** modificata in tempo reale per rispecchiare lo stato dell'applicazione fornita in output da Lovable.
- **Allineamento UI** dagli **screen forniti**: gli screenshot dell'applicazione da imitare fungono da specifica visuale (layout, densità, gerarchie, interazioni) e conseguenti *commit* sulla *repository*.
- **Cascata di prompt (iterazioni)**: applichiamo manualmente una *catena di prompt* di raffinamento (migliorie, correzioni, aggiunte) fino a soddisfare i criteri di accettazione.

Partendo dalla configurazione **Lovable** definita dal responsabile della ricerca (M.P.) - che manterremo privata per strategia aziendale - è stato eseguito un remix della **demo Laif**, che va immaginato, come detto, come un *fork* di una *repository GitHub*. Dunque è iniziato il processo iterativo fornendo il primo prompt al modello LLM di Lovable.

3.7.1 Prima iterazione

Prompt:

```
Dobbiamo modificare pagine e contenuti, utilizza laif-ds
per tutto ove possibile, le pagine disponibili saranno:
- Schedulazioni
-- Lista schedulazioni
-- Nuova schedulazione
- Gestione Schedulazioni

[page] Lista Schedulazioni:
    Input Date dat_start, label "Da";
    affianco Input Date dat_end, label "A";
    affianco Button "Applica" per filtrare il contenuto in quel range.
    Sotto, Tabella con search input,
    filtri (proprietà della tabella laif-ds) e Button di download.
    Le colonne sono:
    [ID, Titolo (con hyperlink a "Dettaglio schedulazione"), descrizione,
    Stato (enum mockup), Inizio turni (hh:mm), fine turni (hh:mm),
    data (date), Modalità (Enum Veloce/Lento), Saturazione, OTIF, Utente]

[page] Dettaglio Schedulazione:
[tabs] Tab con:

- Dettagli
- Ordini non schedulati

[tab] Dettagli:
    AppSelect "Centro di lavorazione", con opzioni
    [Tutti, Campionario, Campionario cesto unico, lavorazioni speciali,
    olandese, rotativa a pressione,
    rotativa a scomparti, rotativa cesto unico];
```

affianco popover "Visualizza Parametri" con:
input slider con valori "priorità cliente", "bilanciato",
"priorità consegne"; sotto "inizio turno: hh:mm" e "fine turno hh:mm";
sotto "modalità: mode" (es. enum Veloce/Lento);
affianco AsyncSelect "Macchine" multi con dei badge
per tutta la lunghezza
della pagina con wrap per ogni valore selezionato che mostrano il nome
della macchina e hanno una x per essere rimossi dalla selezione;
sotto "Da" input Date, "A" input Date;
affianco (se e solo se AppSelect "Centro di lavorazione"
non ha selezionato il valore "Tutti") switch button "Gantt"/"Tabella"
(Gantt default se disponibile, se nulla è selezionato nel select
"Seleziona un centro di lavorazione",
se selezionato tutti solo tabella visibile);
sotto (sse selezionato valore in AppSelect "Centro di lavorazione")
tabella come la tabella di prima
(filtri per ogni colonna, search, download button), con colonne:
[Pianificato Tassativo (T/F con icona di lucchetto
chiuso/aperto associata),
Codice macchina, cliente (str), attività (enum: Setup Iniziale
/Produzione/Candeggio/Cambio vascata), commessa (str cod),
Articolo (str[]),
variante (str cod), peso vascata (double), Lavorazioni vascata
(enum mockup "lavorazioni"),
scadenza (date), quantità capi, numero vascata, inizio (datetime),
fine (Datetime), lista lavorazioni (enum lavorazioni[]),
scadenza originale (date)].

Sotto (sse gantt disponibile) {
Button "modifica" (se clicco su modifica compaiono button
"conferma" e "annulla";

```
    durante la modifica posso cliccare su un blocco e ottengo
    il modale in screenshot #3;
    ho sempre un hover sui blocchi come quello in screenshot #4 e
    quando non è in modifica
    ho un modale di solo recap #5);
    sotto gantt con righe macchine e colonne datetime
    (con zoom range draggable)
    da primo datetime schedulazione a ultimo,
    i blocchi gant sono le attività schedulate.

    Sotto a Gantt (altrimenti nulla) 4 indicatori CircularProgress
    0-100 con (label, IconInfoHover):
    [(QUAL, "Percentuale di qualità della produzione"),
    (SAT, "Percentuale di saturazione delle ore lavorative"),
    (OTIF, "Percentuale di ordini in orario"),
    (WOTIF, "Percentuale di ordini in orario pesata
    per priorità degli ordini")].
    }

[tab] Ordini non schedulati: vedi screenshot con componente a righe.

[page] Nuova schedulazione: Input nome, Input descrizione;
    sotto label "parametri anagram";
    sotto {
    input slider come prima, switch button "veloce/lento" e
    input hh:mm
    Inizio/fine Turno;
    }
@config sotto checkbox (default=F) "Salva configurazione corrente";
affianco Button "avvia nuova schedulazione".
```

```
[page] Configurazione schedulazioni:
    come @config; sotto Button "Imposta come default".
```

Il prompt è stato poi arricchito dagli stessi *screenshot* descritti nel capitolo 3.6.2, oltre a immagini raffiguranti elementi più specifici dell'applicazione, nello specifico:

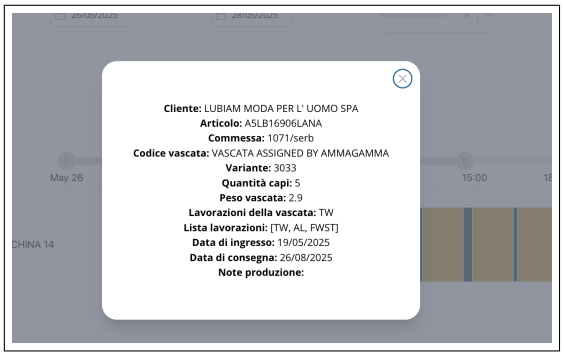

Figura 3.15: Modale di visualizzazione dei dettagli di un *Gantt Block*.

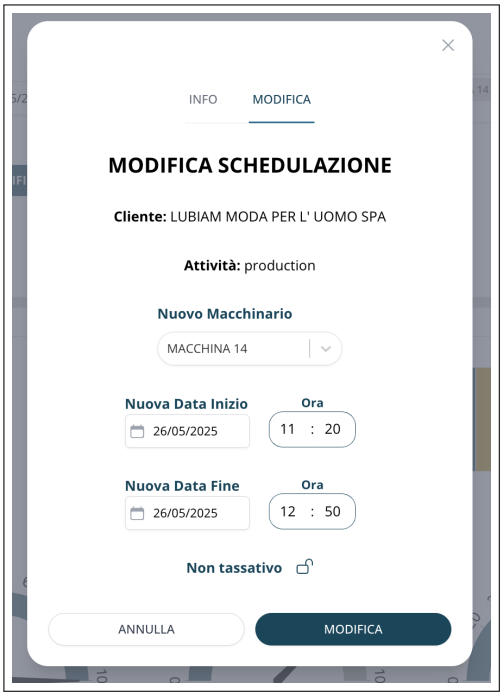

Figura 3.16: Lo stesso modale della figura 3.15 in modalità *modifica*.

Per agevolare l'interpretazione del prompt da parte del modello LLM, ho definito uno **pseudo-markup testuale** concepito come linguaggio intermedio tra la descrizione naturale e la struttura formale di un layout. Tale sintassi adotta una notazione a blocchi ispirata alla gerarchia delle interfacce React, con marcatori delimitati tra parentesi quadre che indicano la natura e la profondità semantica di ciascun elemento. Gli elementi principali dello pseudo-markup sono:

- `[page]` - identifica una pagina o vista logica dell'applicativo (es. `[page] Lista Schedulazioni`);

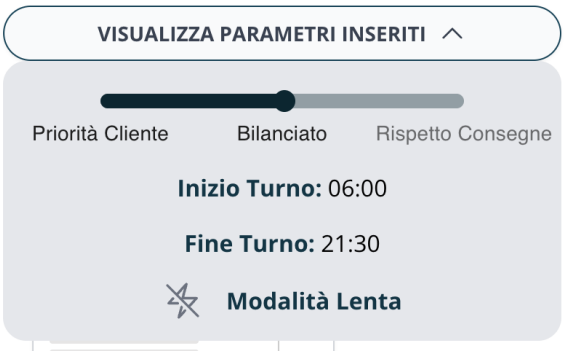

Figura 3.17: Dropdown del tasto *"VISUALIZZA PARAMETRI INSERITI"*

- `[tab]` - definisce una sezione interna a una pagina, normalmente corrispondente a un componente con tabulazione o contenuto dinamico;
- `[component]` o blocchi impliciti - descrivono i singoli elementi UI, come `Input`, `Button`, `AppSelect`, `AsyncSelect`, `Switch` o `Gantt`;
- le *indentazioni* (rafforzate dall'uso di *"-"*) e le strutture annidate rappresentano la gerarchia visiva e funzionale tra i componenti.

Questa forma testuale, pur non essendo un linguaggio formale, è risultata efficace nel rendere il prompt **autoesplicativo e non ambiguo**, in quanto univa la leggibilità del linguaggio naturale a una rappresentazione strutturata dell'interfaccia. In pratica, lo pseudo-markup ha permesso di comunicare al modello non solo il contenuto, ma anche la relazione gerarchica tra gli elementi, il loro comportamento e il contesto d'uso, in questo modo, la combinazione tra il prompt pseudo-strutturato e gli screenshot di supporto ha permesso di fornire al sistema un contesto visivo e sintattico preciso, analogo a una *specification by example*[Adz11].

Tabella 3.2: Metriche di esecuzione del primo prompt

Parametro	Valore	Unità
File modificati	7	–
Tempo di scrittura del prompt	∼22	min
Tempo di *reasoning*	16	s
Tempo di elaborazione modello	3.57	min
Tempo totale	**∼26.13**	**min**

Primo output generato

Il primo output generato dal modello a seguito del prompt di schedulazione ha mostrato risultati parzialmente coerenti con le specifiche fornite. In questa fase iniziale, l'interfaccia prodotta da Lovable ha riprodotto correttamente la struttura generale delle pagine e la navigazione specifica di *Tintoria*, ma ha evidenziato diverse semplificazioni e imprecisioni nella logica dei componenti, come illustrato nelle figure seguenti.

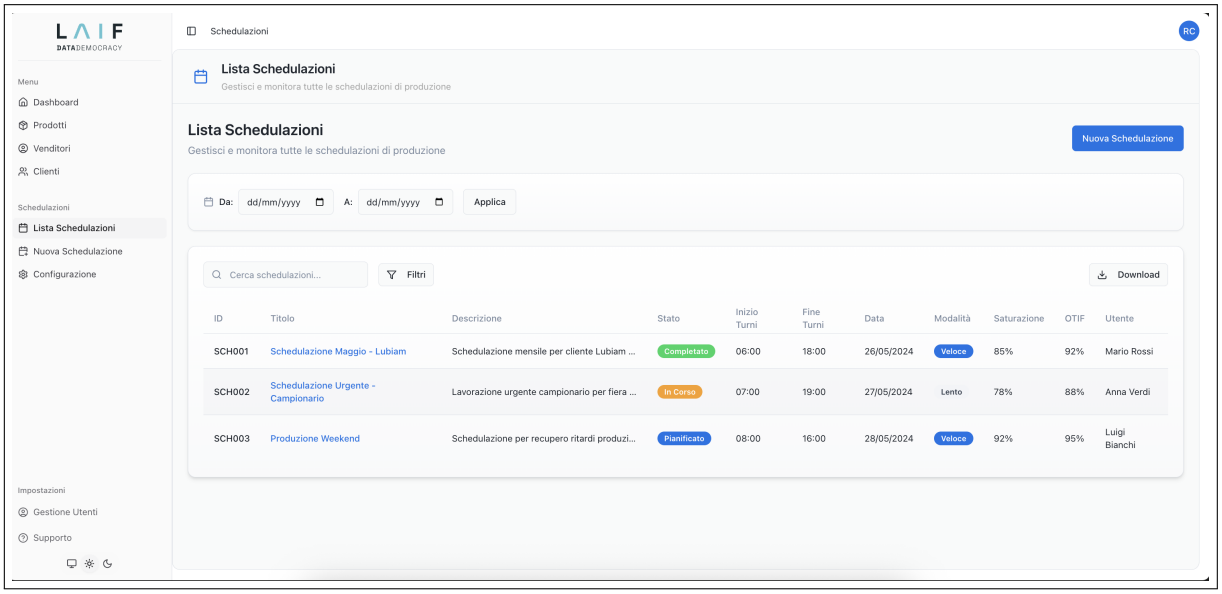

Figura 3.18: Pagina principale con pulsante *Filtri* e collegamento corretto a "Nuova schedulazione".

La navigazione mockup precedente è stata mantenuta erroneamente: si notano infatti due sezioni di navigazione, *Menu* e *Schedulazioni*, quando ci si sarebbe aspettati unicamente la seconda come sezione principale. La pagina "Lista schedulazioni" è stata tuttavia generata in modo corretto anche nei dettagli.

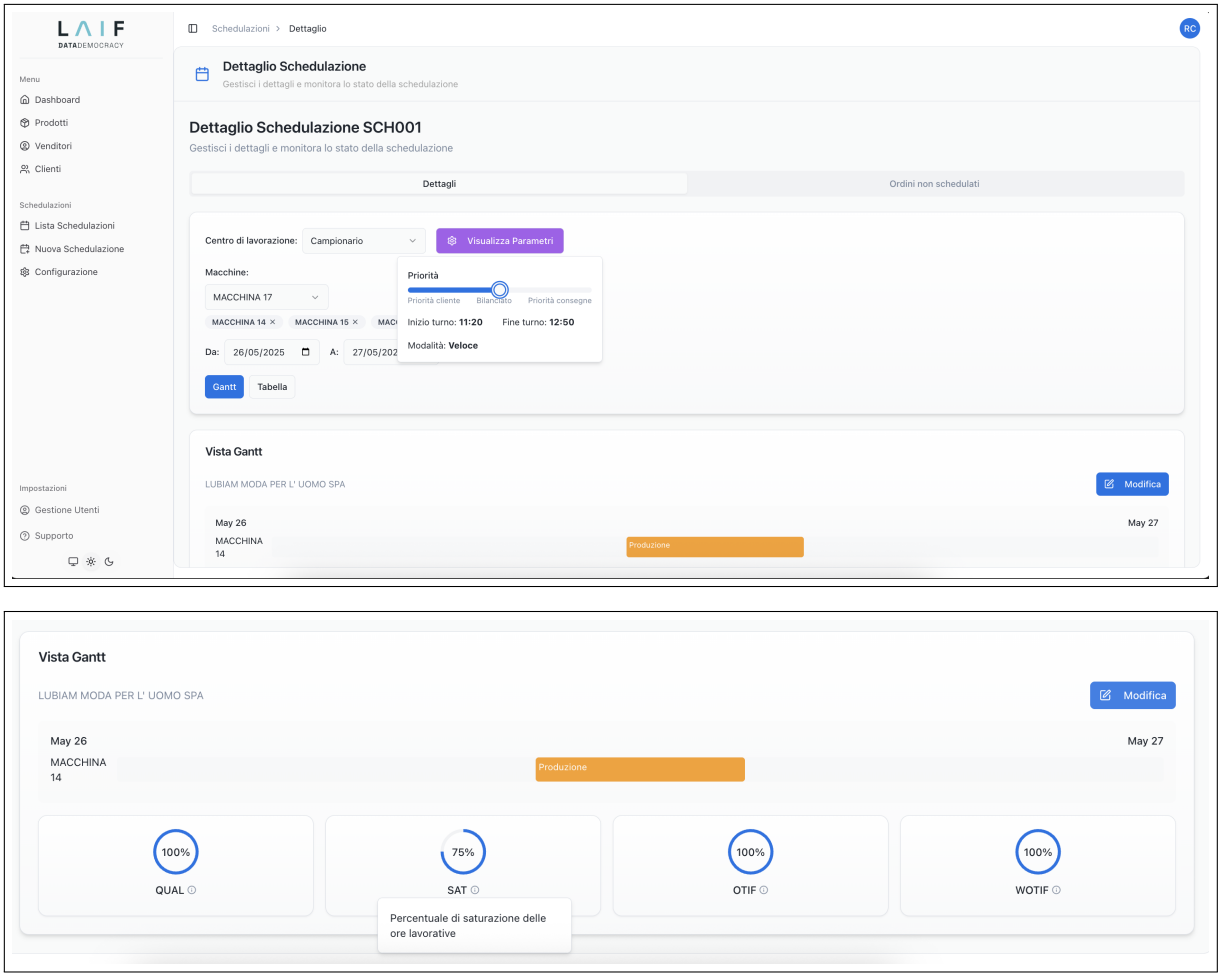

Figura 3.19: Vista Gantt generata dal modello: nella parte superiore (in alto) la sezione dei controlli e filtri, in basso la rappresentazione grafica delle lavorazioni. Il layout risulta approssimativo, ma la logica di visualizzazione e interazione è correttamente interpretata.

Nel complesso si tratta di un output di buona qualità: pur con alcune approssimazioni grafiche, il modello ha compreso la logica dei filtri e la relazione tra i componenti. Il grafico Gantt in sé risulta tuttavia incompleto e privo della modalità di modifica.

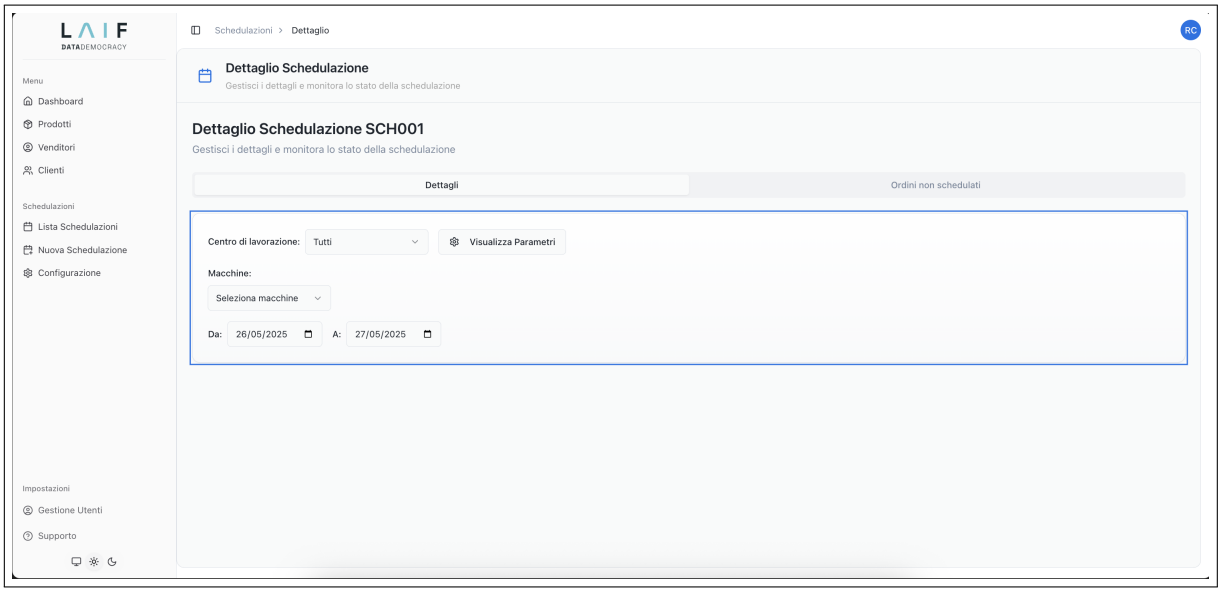

Figura 3.20: Pagina di dettaglio: vista tabellare assente.

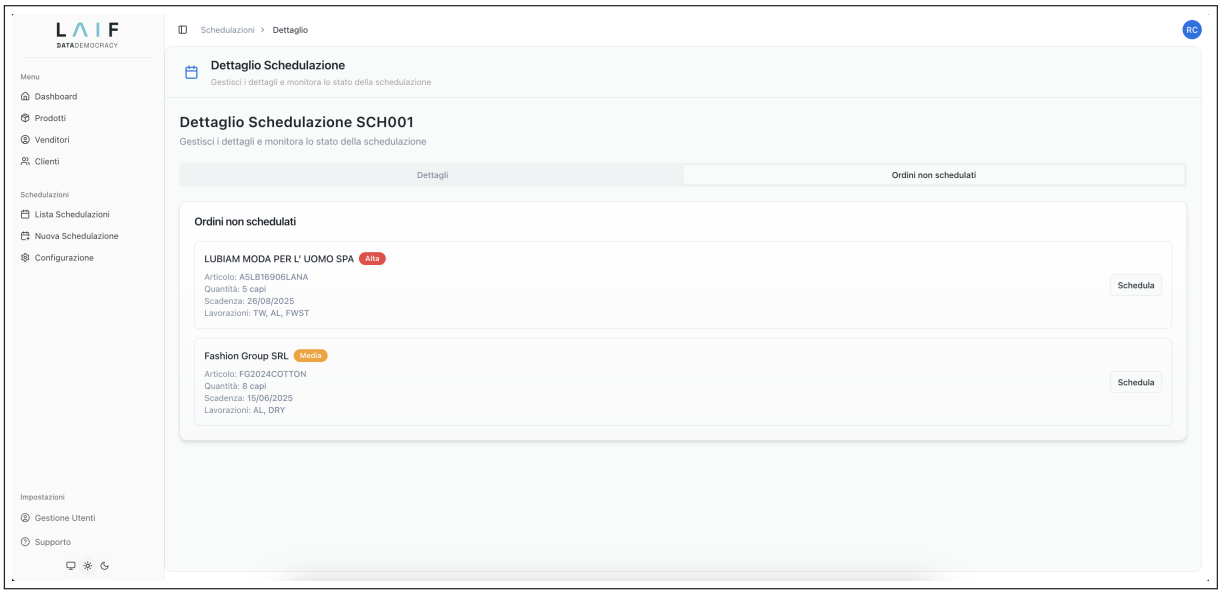

Figura 3.21: Tab “Ordini non schedulati”: contenuto inventato completamente, non corrispondente alle specifiche del prompt.

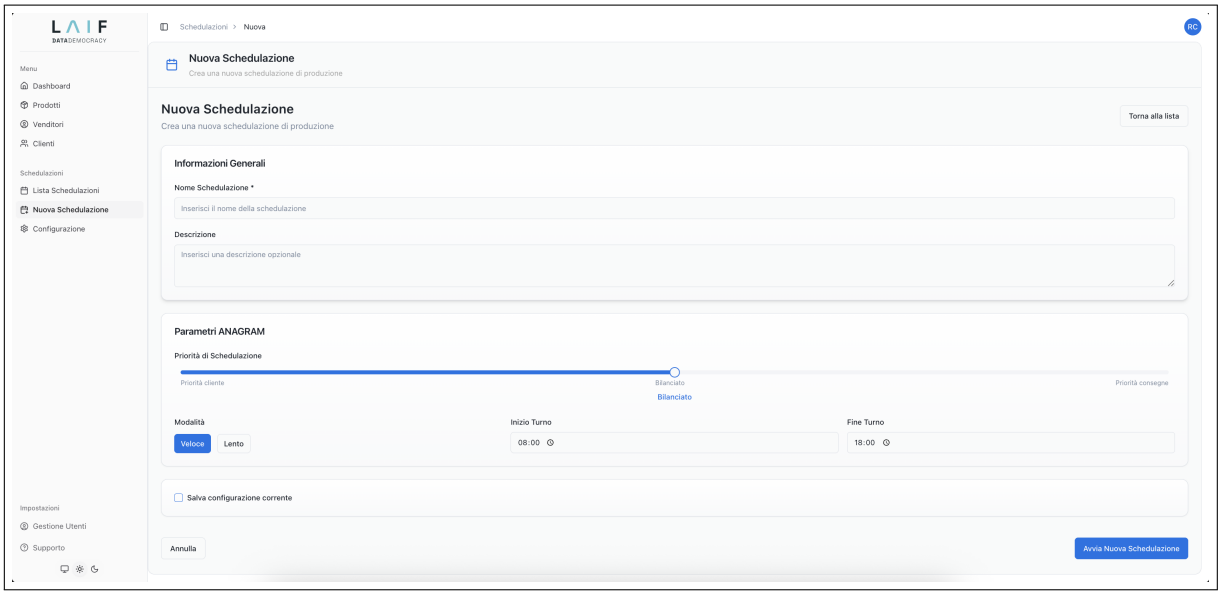

Figura 3.22: Sezioni “Configurazione” e “Parametri avanzati”: pressoché corrette, con l’unico errore dello *slider* espresso su scala 0–100 invece che su tre valori discreti.

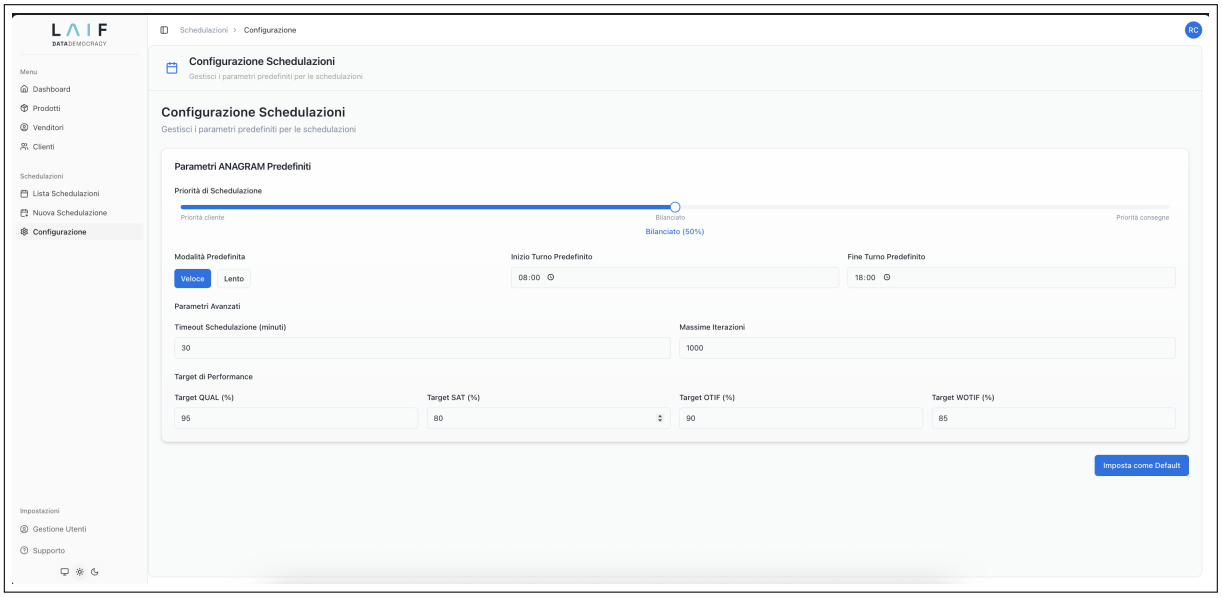

Figura 3.23: Pagina di configurazione: corretta nella parte superiore, ma con sezione “Parametri avanzati” aggiunta spontaneamente dal modello e non richiesta dal prompt.

Nel complesso, il primo output può essere considerato una **versione solo parzialmente conforme** alle specifiche richieste. Pur avendo individuato correttamente la struttura generale delle pagine, il modello ha mostrato una comprensione incompleta della logica applicativa, introducendo elementi arbitrari (come la sezione "Parametri avanzati") e omettendo funzionalità fondamentali quali i filtri tabellari e la modalità di modifica del Gantt. L'interfaccia risultante, sebbene coerente nella forma, si discosta in più punti dalle direttive del prompt, evidenziando la necessità di un intervento correttivo più mirato.

A partire da queste criticità è stato elaborato un **secondo prompt**, concepito per ottenere un output più aderente e controllato. In questa seconda iterazione, le istruzioni sono state rese più vincolanti e dettagliate, con l'obiettivo di limitare la libertà interpretativa del modello e di correggere gli errori emersi nella fase precedente.

3.7.2 Seconda iterazione

Per ragioni di sintesi, verranno presentate meno immagini, concentrandosi sui punti salienti e sulle differenze più rilevanti rispetto al primo output.

Prompt:

```
modifiche da apportare:
- globali:
-- PER TUTTE LE PAGINE (mantieni la sezione di titolo,
sottotitolo e icona solo nella navbar superiore;
eliminala dal contenuto delle pagine)
-- aumenta il numero di dati mockup
-- elimina le voci di navigazione precedentemente esistenti
    (Menu[Dashboard, Prodotti, Venditori, Clienti])
-- le tabelle devono usare le proprietà del laif-ds e devono
    essere paginate (con più dati mockup sarà possibile testarlo)
-- gli slider di priorità possono assumere solo i valori
    agli estremi e al centro (-1, 0, 1)
```

```
- [page] Configurazione (rename in Configurazioni):
-- elimina la sezione Parametri avanzati
- [page] Lista Schedulazioni:
-- Il button nuova schedulazione impostalo come customNavComponent
-- la searchbar e i filtri sono impostabili tramite proprietà della
    table laif-ds
- [page] Dettaglio schedulazione:
-- Quando seleziono "Tutti" la tabella deve essere visibile,
    la label deve essere "Tutti (tabella per sola esportazione")
-- Modifica deve aggiungere un tab al modale dei task del gantt,
    ti invio sia il modale con selezionato il tab in questione
    (screenshot #1) che il gantt (#2) in quanto il tuo è approssimativo:
    manca lo zoom, gli orari, i blocchi sono piccoli, on hover deve
    mostrare i dettagli principali dell'attività (task) (#3)
-- Alla tabella mancano i filtri, la searchbar e il button download
-- La tab degli ordini non schedulati è errata, ti mando l'immagine
    #4 come reference
```

Anche per questo prompt sono state fornite delle immagini di riferimento.

Tabella 3.3: Metriche di esecuzione del secondo prompt

Parametro	Valore	Unità
File modificati	8	–
Tempo di scrittura del prompt	∼12	min
Tempo di *reasoning*	17	s
Tempo di elaborazione modello	6.08	min
Tempo totale	**∼18.25**	**min**

Secondo output generato

Il secondo output, prodotto a seguito del nuovo prompt correttivo, ha mostrato un sensibile miglioramento nella coerenza complessiva e nell'aderenza alle specifiche

funzionali.

Le **modifiche globali** sono state quasi tutte applicate correttamente: la rimozione delle sezioni di titolo e sottotitolo dalle pagine è avvenuta come richiesto, le voci di navigazione superflue sono state eliminate, la pagina degli ordini schedulati è finalmente corretta e priva di allucinazioni e il layout generale risulta più ordinato e leggibile. Il problema principale riguarda l'implementazione delle tabelle: il modello non ha potuto applicare le proprietà aggiornate del `laif-ds` a cui si faceva riferimento nel prompt, probabilmente a causa dell'assenza di una versione recente della libreria nel contesto di generazione. Oltre a questo, la pagina del Gantt presenta numerose problematiche che verranno affrontate nel prompt successivo.

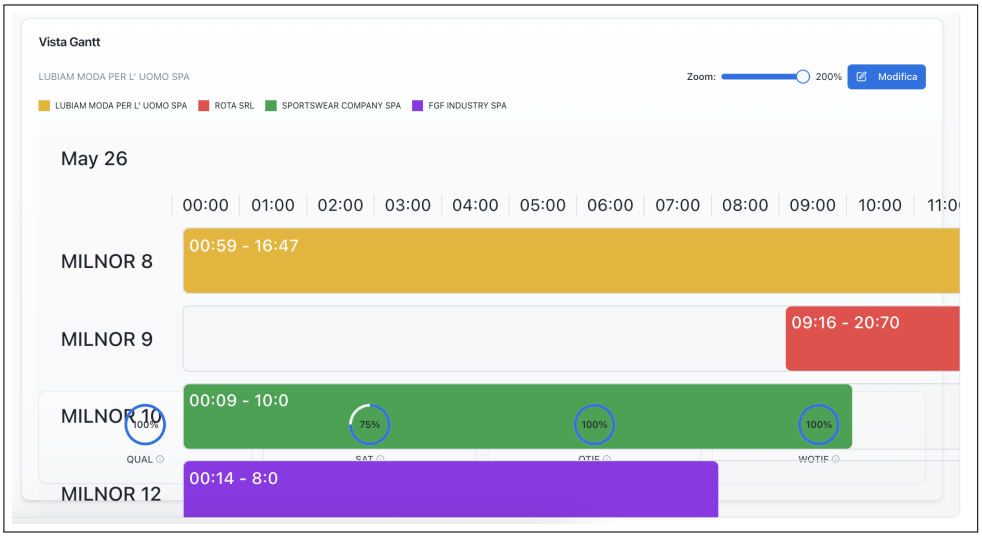

Figura 3.24: La pagina dedicata alla visualizzazione Gantt dei task risulta ancora errata: i filtri sono errati e non funzionanti, mentre gli indicatori dei *KPIs* sono sovrapposti al grafico

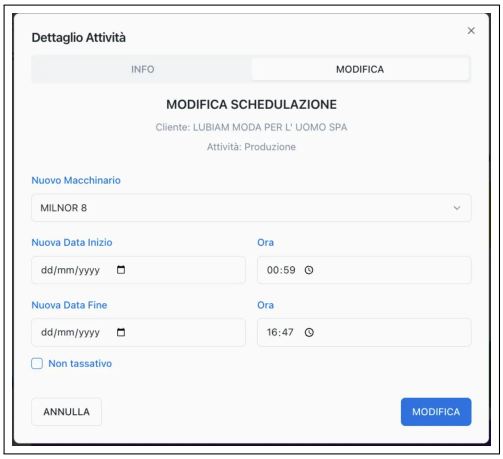

Figura 3.25: La tab di modifica all'interno del modale è corretta, l'unica inesattezza è che deve essere visibile esclusivamente dopo averla attivata tramite l'omonimo pulsante sopra il grafico.

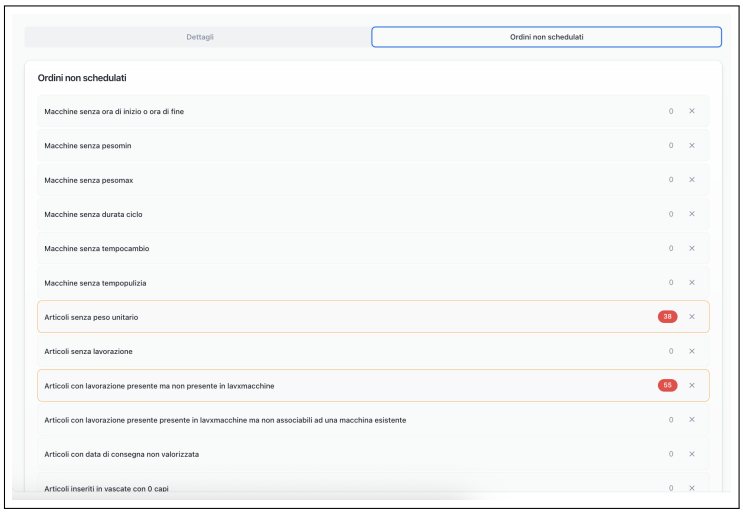

Figura 3.26: Sezione degli ordini schedulati corretta.

In conclusione, il secondo output rappresenta un passo avanti rispetto alla prima iterazione, sia in termini estetici e di usabilità, sia di aderenza logica alle specifiche. Tuttavia, la qualità finale è risultata condizionata da un fattore esterno non noto al momento dei test: la discrepanza tra lo **stato della documentazione del design system** (fornita al modello come riferimento) e la **versione effettivamente in uso**

nella generazione di codice del `laif-ds`. Quest'ultima, infatti, risultava già più avanzata e includeva componenti tabellari e parametri aggiornati non ancora descritti nella documentazione ufficiale. Questa incongruenza ha inevitabilmente influenzato la generazione dei componenti, in particolare delle tabelle e dei filtri, determinando errori **non imputabili alla logica di interpretazione del modello**.

3.7.3 Terza iterazione

Convinto che il problema dipendesse da una formulazione ancora troppo vaga delle istruzioni, nella terza iterazione, oltre a fornire le istruzioni su come migliorare la pagina del Gantt, ho fornito indicazioni per una ristrutturazione delle tabelle.

Prompt:

```
Modifiche da apportare:
- qui c'è la documentazione del laif-ds:
    https://laif-group.github.io/ds/
    ristruttura le tabelle usando la documentazione come riferimento.
- in Dettaglio schedulazione
-- Visualizza parametri:  lo slider deve essere read-only
-- Filtri: Niente deve essere selezionabile finché
    centro di lavorazione non è stato selezionato
-- lo zoom del gantt deve essere uno slider con due button draggable,
    lungo quanto il gantt, se entrambi i button sono ai
    rispettivi estremi viene visualizzato tutto il range delle attività,
    spostandoli si modifica datetime di inizio e fine
    visualizzazione dei task: se ad esempio i task vanno dal 26 maggio
    al 4 giugno, con tutto il range si vedono tutti i giorni,
    più lo restringo e più dettaglio vedo sulla barra del gantt
    (da che si vedono solo i giorni, vedo anche la divisione in ore)
-- la modifica attiva il secondo tab "modifica" del modale
    delle attività, se non siamo in stato di modifica il tab non è
    nemmeno visibile
```

```
-- Ordini schedulati, le righe non hanno un button x per essere rimosse
- Configurazioni
-- rimuovi il "(value)" da sotto allo slider,
- Nuova configurazione
-- non deve essere possibile preme avvia finché mancano nome e descrizione
```

Tabella 3.4: Metriche di esecuzione del terzo prompt

Parametro	Valore	Unità
File modificati	4	–
Tempo di scrittura del prompt	∼6	min
Tempo di *reasoning*	9	s
Tempo di elaborazione modello	2.57	min
Tempo totale	**∼9.06**	**min**

Terzo output generato

Dato l'output pressochè perfetto della pagina del Gantt e data anche l'assenza di risultati tangibili nella modifica delle tabelle causata dall'incongruenza discussa nella sottosezione 3.7.2, questa è stata l'ultima iterazione eseguita tramite Lovable.

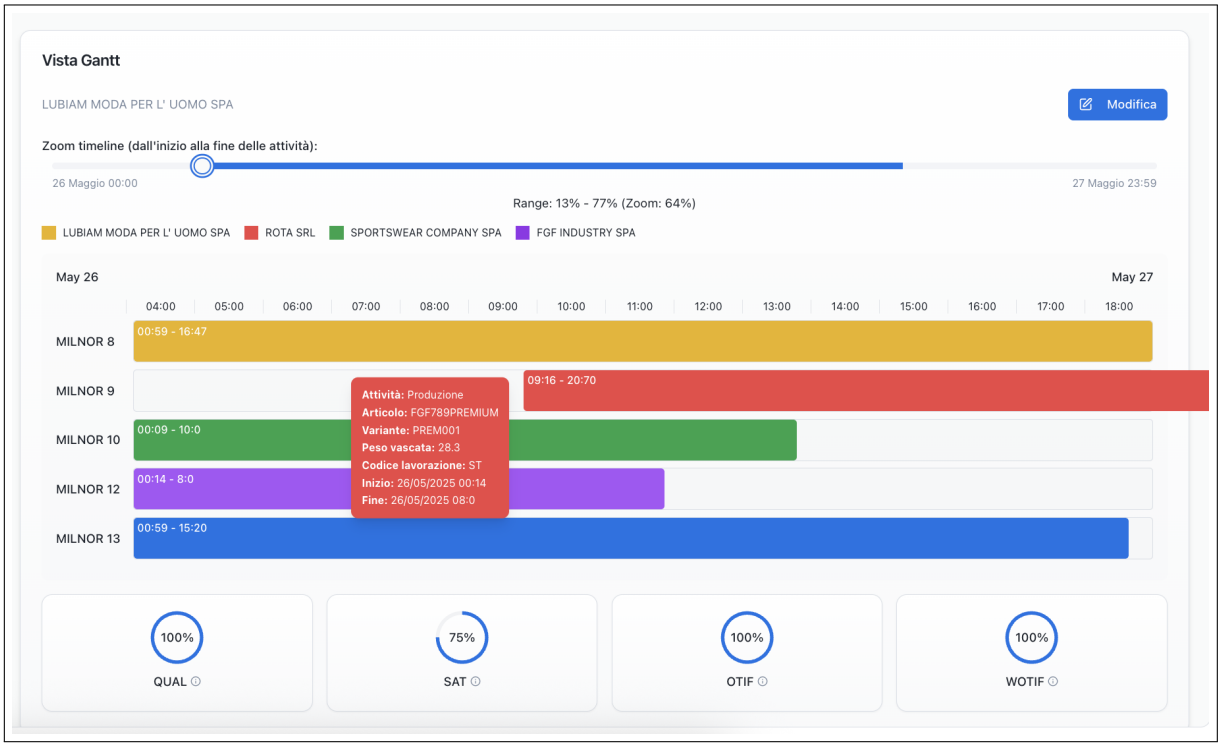

Figura 3.27: Visualizzazione Gantt pressoché corretta ad eccezione del secondo pulsante di *drag* assente e dei task che vanno in *overflow* oltre il blocco del grafico.

Il Gantt risulta coerente e ben strutturato nella resa grafica: il meccanismo di zoom mediante *slider* è effettivamente funzionante e permette di modificare l'intervallo temporale visualizzato, riproducendo il comportamento previsto dal prompt. Permangono tuttavia alcune imperfezioni marginali che ho considerato non significative per gli obiettivi del test. In questa fase, l'intervento tramite ulteriori prompt sarebbe risultato più oneroso e meno efficace rispetto a una correzione manuale diretta nel codice sorgente.

Tabella 3.5: Tabella delle tempistiche complessive

Iterazione	**Totale parziale (min)**
Iterazione 1	**∼26.13**
Iterazione 2	**∼18.25**
Iterazione 3	**∼9.06**
Totale	**∼53.44**

A questo punto, il processo di generazione può considerarsi sostanzialmente completato. Il risultato ottenuto è stabile e coerente con le specifiche, anche se rimane un certo margine di miglioramento, in particolare sul fronte della standardizzazione dei prompt tramite la definizione di un linguaggio di istruzioni più uniforme. In ogni caso, il modello ha raggiunto un livello di qualità tale da consentire il passaggio alla fase operativa successiva: l'esportazione della *codebase* generata e la sua integrazione all'interno della repository ottenuta dal *fork* del template aziendale, così da consolidare il porting all'interno dell'architettura del progetto principale.

3.8 Integrazione e merge con Windsurf

Conclusa la fase di generazione in *Lovable*, il passo successivo - e conclusivo - è consistito nell'**unire il codice prodotto da Lovable** con l'architettura del **template aziendale**, così da ottenere un'unica codebase coerente con gli standard Laif. Per farlo, ho utilizzato **Windsurf** come agente di integrazione e refactoring, affidandogli il compito di adattare la struttura del progetto generato ai vincoli architetturali e stilistici imposti dai *ruleset* interni.

3.8.1 Processo di integrazione

Le due *repository* - quella generata da *Lovable* e quella del progetto Laif - sono state collocate nella stessa directory, da cui è stato avviato l'agente. Questa scelta ha permesso al modello di operare su un contesto unificato, comprendente sia il codice applicativo completo delle interfacce prodotto da Lovable, sia la repository aziendale dedicata al progetto, da cui desumere struttura, organizzazione e convenzioni del template (frontend e backend). In questo modo è risultato possibile facilitare il trasferimento della logica e dei componenti provenienti da Lovable all'interno delle sezioni `app/` del progetto aziendale, mantenendo intatta la separazione con la parte `template/`, condivisa da tutte le istanze.

Per guidare l'agente nella comprensione del progetto e delle sue dipendenze, ho fornito un file descrittivo, `PROJECT_DESCRIPTION.md`, impiegato come *manifesto tecnico* della repository. Il documento riportava la struttura del progetto, la suddivisione dei moduli, le principali librerie frontend e backend, e le istruzioni di script, build e deploy. In questo modo Windsurf poteva pianificare le modifiche rispettando i vincoli di organizzazione e compatibilità dell'ecosistema Laif.

Dopo aver predisposto il contesto, ho avviato una nuova sessione di Windsurf e fornito un prompt sintetico, concepito per chiedere all'agente di migrare i file e riallineare import, dipendenze, configurazioni e scelte strutturali, mantenendo piena aderenza al *rules-set* aziendale.

Il processo è stato condotto come un ciclo di iterazioni convalidato manualmente: Windsurf ha analizzato entrambe le repository, identificato le corrispondenze e proposto piani di merging multi–file. A ogni iterazione ho verificato e approvato i passaggi principali, intervenendo manualmente dove necessario.

Nel complesso, l'intero flusso di merge e refactoring - comprendente analisi, iterazioni con l'agente e interventi manuali - si è completato in poco più **dodici ore di lavoro effettivo**. Il risultato finale è stato un progetto perfettamente integrato, con il codice di Lovable incorporato nel framework aziendale e pienamente compatibile con la struttura monorepo (FastAPI + Next.js).

È importante sottolineare, tuttavia, che il percorso che ha portato alla versione finale e funzionante non è stato affatto lineare. Alcune *rules* si sono rivelate troppo ambigue per essere interpretate correttamente dal modello, e non sono mancati episodi di allucinazioni, errori logici, incongruenze di tipizzazione nei fra frontend e backend e internamente ai componenti, oltre a innumerevoli difetti di coerenza strutturale. Ogni iterazione del prompt ha richiesto una serie di interventi correttivi spesso *time–consuming*, necessari per riportare il progetto in uno stato consistente.
Di seguito riporto, a titolo esemplificativo, alcuni dei prompt iniziali utilizzati durante la fase di integrazione.

3.8.2 Prompt forniti a Windsurf

Prompt 1 (elaborazione: ∼13.43 min).

```
Ti ho fornito due repository: @tintoria-lovable
(output completo di Lovable) e @tintoria-sched
(repo Laif con architettura template).
Trasferisci tutte le funzionalità implementate in tintoria-lovable
dentro l'app Laif, collocando i file nelle sezioni
@frontend/app e @backend/app rispettando integralmente
le rules presenti nella cartella @rules/.
Presta attenzione a navigation, datamodel, schema e CRUD,
```

```
adeguando tutto agli standard del template.
Segui @PROJECT_DESCRIPTION.md per panoramica del progetto.
```

Prompt 2 (elaborazione: ∼8.12 min).

```
Ho generato tutti i tipi TypeScript per il frontend sulla base
dei nuovi schema e delle chiamate CRUD che hai creato nel backend,
seguendo la stessa organizzazione delle altre app del template Laif.
Integra i tipi nei componenti migrati da tintoria-lovable,
allinea gli import e correggi gli errori.
```

Prompt 3 (elaborazione: ∼9.02 min).

```
Procedi con le correzioni:
- La navigation va fatta seguendo quella del template,
    non all'interno dei singoli componenti ma in @navigation.tsx
- Utilizza le chiamate CRUD generate presenti in @clien.gen.ts
    e verifica la coerenza tra datamodel, schema e servizi.
Continua a proporre fix fino a ottenere una build pulita lato frontend
e backend mantenendo la piena conformità al template.
```

Come anticipato in precedenza, a seguito di ciascun output generato da Windsurf si sono resi necessari numerosi interventi correttivi: alcuni dovuti ad ambiguità presenti nelle *rules*, altri a errori di tipizzazione, incoerenze tra datamodel e schema, o semplicemente a scelte strutturali non allineate al template aziendale. Le iterazioni riportate sopra rappresentano quindi solo una parte del lavoro svolto: ogni prompt ha innescato una catena di aggiustamenti manuali e validazioni successive, indispensabili per riportare il progetto in uno stato consistente e conforme agli standard Laif. Nonostante ciò, il processo ha evidenziato la capacità dell'agente di fornire una base solida su cui intervenire, permettendo di concentrare l'effort umano sulle decisioni architetturali più complesse e non sulla riscrittura del codice.

In sintesi, questa fase ha rappresentato la chiusura del test di validazione, dimostrando la possibilità di un flusso unificato e replicabile *Lovable* → *Windsurf* → *Template Laif*, in cui la generazione, l'adattamento e l'integrazione del codice possono avvenire in modo coordinato, pur richiedendo un intervento umano non solo di supervisione

architetturale e gestione dei conflitti, ma anche di programmazione tradizionale nei casi in cui l'agente non era in grado di completare correttamente i task richiesti.

Capitolo 4

Valutazione del sistema

La valutazione del sistema presenta alcune peculiarità: trattandosi di un processo interattivo, non deterministico e fortemente dipendente dal contesto, non è possibile costruire un vero e proprio benchmark numerico. Le prestazioni dell'agente variano infatti in funzione della qualità dei *rules*, del contenuto della knowledge base, della natura del prompt e **dell'intervento umano** nelle fasi di refactoring.
Per questa ragione, le poche metriche quantitative disponibili (tempi di elaborazione e numero di iterazioni necessarie) vanno lette come indicazioni empiriche e non come misure assolute. La valutazione che segue adotta quindi un approccio misto, in cui alcuni dati numerici servono da supporto a un'analisi principalmente qualitativa, centrata su coerenza strutturale, qualità del codice, robustezza del processo e riduzione del carico di lavoro effettivo.
Accanto a queste osservazioni è stata introdotta una piccola componente di valutazione soggettiva, tramite questionari ispirati alla *System Usability Scale* (SUS) [Bro96], per raccogliere in modo più sistematico la percezione di usabilità e utilità degli strumenti da parte del gruppo di ricerca.

4.1 Metodologia di valutazione

La strategia di valutazione combina tre fonti principali di evidenza:

- **Analisi qualitativa del codice e del processo**, lungo l'intero flusso di lavoro: dalla generazione iniziale in *Lovable* all'integrazione finale in *Windsurf*;
- **Osservazioni quantitative leggere**, quali tempi indicativi e numero di iterazioni agentiche richieste per completare un task;
- **Valutazione soggettiva tramite questionario**, rivolta ai membri del gruppo di ricerca.

Il questionario è stato progettato prendendo a riferimento la *System Usability Scale*, una scala standardizzata a 10 domande chiuse con risposte su scala Likert a 5 punti, ampiamente utilizzata per misurare l'usabilità percepita di sistemi interattivi. La struttura è stata adattata al dominio degli strumenti agentici per lo sviluppo software e riducendo il numero di domande, conservando però l'idea di un giudizio sintetico e confrontabile.
Il questionario è stato somministrato ai sei membri del gruppo di ricerca (incluso l'autore), tutti con esperienza nello sviluppo software e, in misura diversa, con esposizione pregressa a strumenti AI per il codice. Per ciascuno dei due strumenti considerati (*Windsurf* e *Lovable*) sono stati previsti due momenti di compilazione:

- un **questionario iniziale**, al termine della prima fase di ricerca e sperimentazione;
- un **questionario finale**, al termine dell'ultima fase descritta nei capitoli precedenti.

La numerosità ridotta del campione non consente inferenze statistiche forti; i risultati vanno quindi letti come un complemento qualitativo all'analisi tecnica, utile per capire come gli strumenti vengano percepiti dopo un uso realistico.

4.2 Efficacia e analisi qualitativa

La valutazione qualitativa considera il comportamento del sistema lungo l'intero flusso di lavoro, dalla generazione iniziale in *Lovable* all'integrazione finale in *Windsurf*. Dato che le attività svolte non sono pienamente ripetibili in modo deterministico, l'attenzione è posta su:

- coerenza strutturale del codice prodotto;
- aderenza al template aziendale;
- robustezza delle modifiche multi–file;
- reale riduzione del carico operativo rispetto a un approccio interamente manuale.

4.2.1 Contributo di Lovable

La fase di generazione in *Lovable* ha mostrato un'elevata efficacia nella ricostruzione dell'interfaccia utente. A partire dagli screenshot e dallo pseudo–markup, il modello ha prodotto:

- una struttura navigabile completa e coerente;
- pagine e componenti UI vicini allo stile desiderato;
- un Gantt implementato in React (task non banale) via via più accurato attraverso le iterazioni;
- uno scheletro di progetto abbastanza robusto da poter essere riutilizzato.

Nonostante errori, approssimazioni e alcune allucinazioni, Lovable si è dimostrato **molto efficace** nella fase di **rapid prototyping**, riducendo drasticamente il tempo necessario per ottenere una base visiva e funzionale da perfezionare nelle fasi successive. In termini pratici, la qualità e l'ampiezza del risultato ottenuto in poche ore corrispondono a quello che, con uno sviluppo tradizionale, avrebbe verosimilmente richiesto **almeno 2/3 di giorni di lavoro** (≈20 ore–uomo [UNI16]).

4.2.2 Contributo di Windsurf

L'integrazione tramite *Windsurf* ha rappresentato il momento di consolidamento dell'intero progetto. L'agente è riuscito a:

- applicare in larga parte gli standard architetturali del template Laif;
- generare e allineare datamodel, schema e CRUD;
- propagare modifiche multi–file in modo coerente;
- integrare le componenti Lovable in un contesto monorepo FastAPI+Next.js.

Questa fase ha però richiesto un contributo umano significativo: correzione di errori logici, rifinitura dei tipi, aggiustamento dei componenti, chiarimento delle *rules* e, in diversi casi, interventi di programmazione tradizionale. In altre parole, Windsurf si è comportato da **forte acceleratore** per molte attività ripetitive e di impianto, ma non è risultato affidabile in autonomia sull'intero flusso.
Questo quadro è coerente con quanto riportato in letteratura: gli LLM eccellono nel riconoscere pattern locali e nel generare blocchi di codice plausibili, ma incontrano difficoltà nel mantenere una **coerenza architetturale globale** quando le modifiche coinvolgono molti file e dipendenze [Bor23][HL25]. Studi su modelli specializzati nel codice (come Codex e derivati [Che+21]) mostrano limiti analoghi nel rispetto di vincoli impliciti e nella gestione di progetti reali con logiche complesse [YTÖ22].
Un ulteriore elemento emerso in questo studio è la forte dipendenza dalla qualità del *prompt* e delle *rules*: se un vincolo non è espresso in modo chiaro, il modello tende a "colmare i vuoti" con soluzioni plausibili ma non sempre corrette. Questo comportamento, spesso descritto come **completamento superficiale di pattern**, è una delle ragioni per cui la supervisione umana resta necessaria [Zho+24].
Nel complesso, i risultati confermano il paradigma **human-in-the-loop**: l'agente è più efficace quando lo sviluppatore mantiene il ruolo di supervisore critico e orchestratore del processo, sfruttando l'AI per accelerare boilerplate, CRUD, componenti UI e refactoring locali, ma riservando le decisioni architetturali e le verifiche di coerenza globale al giudizio umano

4.2.3 Qualità complessiva del risultato

L'integrazione tra Lovable e Windsurf ha prodotto un sistema in larga parte coerente con il template aziendale, ma non privo di imperfezioni. Pur avendo generato una base applicativa solida e riutilizzabile, alcune componenti hanno richiesto:

- riorganizzazione di file e cartelle;
- correzione di dipendenze non gestite correttamente;
- riallineamento tra datamodel e schema;
- riscrittura parziale di logiche applicative.

In più punti sono stati necessari più cicli di revisione per raggiungere una qualità considerata accettabile. Il risultato finale rappresenta quindi un punto di partenza **valido ma non rifinito**: sufficiente per impostare le fasi successive di sviluppo, ma non ancora pronto per un uso produttivo senza un ulteriore passaggio di consolidamento umano.

4.3 Risultati del questionario

I risultati del questionario offrono una conferma, dal punto di vista soggettivo, del quadro emerso dall'analisi qualitativa. In generale, le variazioni tra questionario iniziale e finale sono **moderate** per Lovable e più marcate per Windsurf, in particolare su facilità d'uso e utilità percepita. È importante sottolineare che questi miglioramenti non derivano soltanto da una maggiore familiarità degli sviluppatori con gli strumenti, ma soprattutto dal lavoro di **configurazione mirata** svolto sulla pipeline: definizione di rules dedicate, integrazione MCP verso Notion e PostgreSQL, raffinamento progressivo dei prompt e numerosi cicli di test iterativi.
La Tabella seguente riporta i punteggi medi (scala 1–5) per le principali dimensioni considerate, distinti per strumento e per momento di compilazione (pre/post).

	Lovable		Windsurf	
Dimensione	Pre	Post	Pre	Post
Facilità d'uso	3.6	4.0	3.0	4.2
Fiducia nello strumento	3.4	3.6	2.8	3.6
Controllo percepito	3.0	3.2	3.2	3.6
Utilità complessiva	3.8	4.2	3.0	4.2

Tabella 4.1: Punteggi medi del questionario per Lovable e Windsurf (5 partecipanti, scala 1–5).

In sintesi:

- **Lovable** parte da una percezione già positiva, soprattutto per facilità d'uso e utilità complessiva, e mostra un miglioramento contenuto ma coerente su tutte le voci. Questo è in linea con il suo ruolo: uno strumento **molto efficace per la prototipazione rapida** che beneficia solo in parte della personalizzazione, poiché lavora comunque in un contesto meno vincolato agli standard aziendali.
- **Windsurf** parte da valori iniziali più bassi, ma dopo la fase di **configurazione ad hoc** (rules versionate, integrazione MCP, prompt strutturati, numerosi cicli di prova e correzione) registra un aumento più deciso, fino a raggiungere punteggi comparabili o leggermente superiori a quelli di Lovable su alcune dimensioni, in particolare su facilità d'uso percepita e utilità complessiva nel flusso reale di sviluppo.

La Tabella seguente riassume le variazioni medie ($\Delta = \text{post} - \text{pre}$) per ciascuna dimensione.

Dimensione	Δ Lovable	Δ Windsurf
Facilità d'uso	+0.4	+1.2
Fiducia nello strumento	+0.2	+0.8
Controllo percepito	+0.2	+0.4
Utilità complessiva	+0.4	+1.2

Tabella 4.2: Variazione media dei punteggi tra questionario iniziale e finale.

Queste variazioni vanno lette in chiave **configurativa** più che "formativa": non si tratta solo di sviluppatori che imparano gradualmente a usare un nuovo IDE, ma di uno strumento che diventa via via più utile perché viene **modellato sul contesto aziendale**. Nel caso di Windsurf, l'aumento dei punteggi è strettamente legato a:

- la presenza di rules specifiche per il template Laif e per lo stack tecnologico adottato;
- l'accesso strutturato a schema e metadati via MCP verso PostgreSQL;
- la possibilità di interrogare la documentazione reale tramite MCP Notion e accesso diretto al design system proprietario;
- il raffinamento iterativo di prompt e flussi di lavoro sulla base di numerosi test pratici.

Per alcuni partecipanti almeno una dimensione è rimasta pressoché invariata tra pre e post, mentre nelle altre si è osservato un miglioramento più o meno marcato. Anche questo è coerente con il ridimensionamento delle aspettative iniziali: l'idea di un agente "magico" lascia spazio a una visione più realistica, in cui il valore non è dato dallo strumento in sé, ma dall'integrazione stretta tra **pipeline configurata ad hoc** e supervisione umana.
Nel complesso, il questionario restituisce una valutazione **cautamente positiva**: Lovable e Windsurf sono percepiti come strumenti che migliorano il flusso di lavoro e riducono il carico operativo, ma non eliminano i problemi di complessità architetturale,

standardizzazione e qualità del codice. Il loro contributo emerge soprattutto quando fanno parte di una pipeline definita e mantenuta su misura per l'azienda e per l'architettura che impiega e inserita in un modello *human-in-the-loop* in cui lo sviluppatore mantiene il controllo sulle decisioni chiave. Nel capitolo conclusivo queste evidenze verranno rilette in chiave più ampia, discutendo il ruolo strategico di una pipeline Lovable → Windsurf configurata per il contesto Laif nei processi di sviluppo aziendali.
Per completezza, di seguito vengono riportate le tabelle complessiva delle valutazioni per Lovable e Windsurf.

	Lovable (pre)				Lovable (post)			
Partecipante	FdU	Fid	Ctrl	Ut	FdU	Fid	Ctrl	Ut
G.F.	4	3	3	4	4	4	3	5
M.P.	4	3	3	4	4	4	3	4
C.V.	4	4	3	4	4	4	4	4
C.P.	3	4	3	4	4	3	3	4
S.B.	3	3	3	3	4	3	3	4
Media	3.6	3.4	3.0	3.8	4.0	3.6	3.2	4.2

Tabella 4.3: Valutazioni individuali per Lovable .

	Windsurf (pre)				Windsurf (post)			
Partecipante	FdU	Fid	Ctrl	Ut	FdU	Fid	Ctrl	Ut
G.F.	3	3	3	3	4	4	3	4
M.P.	3	3	3	3	4	4	4	5
C.V.	3	3	4	3	5	4	4	5
C.P.	4	3	3	4	5	3	4	4
S.B.	2	2	3	2	3	3	3	3
Media	3.0	2.8	3.2	3.0	4.2	3.6	3.6	4.2

Tabella 4.4: Valutazioni individuali per Windsurf.

Conclusioni

In questo lavoro non si è inteso valutare la bontà di uno strumento di generazione in sé, quanto piuttosto verificare **l'efficacia complessiva** di una pipeline agentica configurata sul contesto aziendale Laif nel soddisfare il *claim* introdotto nel Capitolo 1. In quell'occasione era stato formulato il presupposto teorico secondo cui un agente integrato nell'IDE, guidato da *rules* versionate, alimentato dal template aziendale e supportato da fonti vive tramite MCP, potesse ridurre tempi e costi delle prime fasi di sviluppo, migliorando al contempo la coerenza del codice e mitigando fenomeni di *documentation drift*. L'intero percorso di ricerca - dall'analisi preliminare alla sperimentazione su progetto reale - è stato strutturato per verificare se tali ipotesi trovassero riscontro pratico.

I risultati ottenuti mostrano che **tutte le ipotesi del claim risultano soddisfatte**. Nel Capitolo 1 è stato affrontato il problema di un onboarding tradizionalmente lento e costoso, in cui la comprensione dell'architettura e delle convenzioni interne rappresenta un ostacolo significativo per i nuovi sviluppatori. La pipeline proposta interviene direttamente su questo limite, mettendo l'agente in condizione di generare fin da subito codice aderente agli standard del team e riducendo così il carico cognitivo iniziale.

Nel Capitolo 2 è stata approfondita la struttura tecnica dell'approccio, evidenziando la sinergia fra Lovable, Windsurf, MCP e il sistema di *rules*. La sperimentazione ha confermato che tale combinazione permette all'agente di operare in modo coerente con lo stack Laif, producendo file e interfacce che rispettano pattern ricorrenti, strutture dati aziendali e linee guida architetturali. Ne emerge che l'efficacia della pipeline non

dipende tanto dall'autonomia generativa dello strumento, quanto dalla qualità del contesto, delle istruzioni e dei vincoli che lo guidano.

Il Capitolo 3 ha mostrato come questo impianto teorico prenda forma nella pratica, mettendo in luce il ruolo del ciclo *plan–act–review* e la capacità dell'agente di operare su progetti multi–file, interpretare modifiche trasversali e riportare codice eterogeneo a un impianto coerente. La validazione descritta nel Capitolo 4, condotta mediante il porting di un'applicazione reale, ha confermato in modo netto l'impatto della pipeline: la riduzione dell'effort umano nelle fasi ripetitive, l'accelerazione nella generazione del primo impianto applicativo e il miglioramento della consistenza interna del codice prodotto costituiscono evidenze robuste a supporto del claim.

Nel complesso, la pipeline Lovable → Windsurf, orientata da *rules* e integrata con MCP, **risolve in modo efficace le problematiche delineate nei capitoli introduttivi** e rappresenta una soluzione praticabile per ridurre tempi, costi e complessità nelle prime fasi di sviluppo software. L'approccio non sostituisce lo sviluppatore, ma ne amplifica l'efficacia, trasformando l'agente in uno strumento operativo che aumenta la qualità del lavoro e accelera i processi senza compromettere il controllo progettuale.

A conferma della sua efficacia, la modalità introdotta durante questa ricerca si è rivelata talmente comoda, funzionale e produttiva da essere adottata stabilmente nel flusso di lavoro aziendale, diventando parte integrante della pipeline di sviluppo. Questa transizione dallo stadio sperimentale all'uso quotidiano costituisce, in ultima analisi, la validazione più evidente della sua utilità.

AI come fattore competitivo

Nel panorama industriale contemporaneo, **l'adozione di strumenti di generazione automatica non può più essere considerata opzionale**. Le aziende che riusciranno a integrare efficacemente gli agenti nei processi di sviluppo godranno di un vantaggio competitivo significativo, mentre ignorare tali tecnologie comporterà un ritardo crescente rispetto ai concorrenti. Pur non sostituendo lo sviluppatore, l'AI attuale agisce come un moltiplicatore di produttività: **non utilizzarla equivarrebbe**, nei prossimi anni, **a rinunciare a una leva strategica fondamentale**.

Valore della pipeline Lovable → Windsurf

Il contributo più evidente emerso dallo studio è la capacità della pipeline, **quando configurata sulle regole e sull'architettura Laif**, di anticipare rapidamente la fase di prototipazione. *Lovable* consente di ottenere, in poche ore, un'interfaccia navigabile, credibile e ricca di componenti complessi, mentre *Windsurf* permette di integrare tale struttura nell'architettura aziendale, riducendo drasticamente la produzione manuale di boilerplate, CRUD e logiche ripetitive.
Questo produce un valore immediato in due direzioni:

- **prototipazione rapida per clienti e stakeholder**, utile per validare funzionalità e ottenere feedback immediati;
- **riduzione del cost-to-build nelle prime fasi di sviluppo**, dove tradizionalmente si concentra il costo umano maggiore [YTÖ22].

La pipeline **così configurata** consente di ridurre in modo significativo il lavoro necessario alla definizione del "primo impianto" dell'applicazione, pur richiedendo rifinitura umana nelle fasi successive, in attinenza con il paradigma *human-in-the-loop* [Wu+22].

L'impatto ambientale: un costo sistemico

Accanto ai benefici operativi, non può essere ignorato l'impatto ambientale legato all'adozione massiva di LLM. Gli studi più recenti mostrano come la fase di addestramento rappresenti solo una parte del problema: anche l'uso quotidiano di modelli di grandi dimensioni comporta consumi energetici rilevanti [SGM19].
L'aumento della domanda di generazione - se esteso all'intero settore industriale - potrebbe portare a un incremento significativo delle emissioni, aggravando un trend già noto per i data center [Jon18].
Inoltre, l'evoluzione verso modelli sempre più grandi rischia di aumentare drasticamente i costi ambientali per ogni token generato [Sch+20], a meno di progressi sostanziali in efficienza hardware e modellistica.
Si tratta di un costo sistemico che l'industria non può ignorare. La letteratura recente sottolinea infatti come la sostenibilità dell'AI dipenda non solo dall'efficienza del training, ma anche dall'ottimizzazione dei modelli in fase di inferenza [Sch+20].
In futuro sarà quindi cruciale introdurre:

- pratiche di prompting più efficienti e consapevoli del costo energetico [Hen+20];
- modelli *smaller-but-smarter*, ottenuti tramite distillazione o ottimizzazioni architetturali per contesti specifici [Pat+21];
- strategie di caching, distillazione e pruning, già riconosciute come strumenti chiave per ridurre drasticamente il costo di esecuzione [Jou+21];
- metriche ambientali integrate nei processi decisionali aziendali e nella documentazione tecnica dei modelli [SGM19].

Sviluppi futuri

Il lavoro svolto apre a numerose direzioni di miglioramento, molti dei quali legati alla necessità di rendere la pipeline più matura, ripetibile e collettivamente adottabile all'interno del team.

Estensione del processo al team

Un primo sviluppo riguarda l'adozione sistematica della pipeline da parte dell'intero gruppo di sviluppo. Ciò richiede linee guida condivise per prompting, naming e gestione del contesto, così da ridurre la variabilità individuale e garantire risultati più stabili. Parallelamente, anche tramite i feedback dei componenti del team, le *rules* dovranno essere rese più granulari e meno ambigue, in modo da fornire all'agente vincoli più chiari.

Potenziamento della knowledge base MCP e test automatici

Un'altra direzione fondamentale consiste nel rafforzamento della KB MCP, arricchendola con documentazione strutturata, esempi, *anti–pattern* [Bro+98] e casi d'uso reali. Una knowledge base più completa aumenta la capacità del modello di rispettare convenzioni architetturali e riduce la necessità di correzioni manuali. In questo contesto, strumenti per la valutazione automatica della qualità del codice generato permetterebbero di identificare rapidamente errori e incoerenze.

Integrazione nei workflow aziendali

Infine, sarà cruciale standardizzare i workflow di prototipazione e integrazione, collegando la pipeline a processi CI/CD che possano orchestrare generazione, validazione e deploy. In questa prospettiva, l'agente potrà evolvere da strumento sperimentale a componente stabile dell'infrastruttura di sviluppo aziendale.

Considerazioni finali

Il ruolo dell'AI nello sviluppo software non è quello di sostituire l'ingegnere, ma di *amplificarne* le capacità: ridurre il carico operativo, aumentare la superficie creativa e abbreviare la distanza tra idea, prototipo e implementazione. I risultati ottenuti mostrano che una pipeline intelligente, **configurata ad hoc sul contesto aziendale e ben orchestrata**, può trasformare il modo in cui i team affrontano porting, refactoring e design di nuove applicazioni.

La direzione è ormai chiara: i sistemi agentici diventeranno parte integrante dei processi di sviluppo, e la loro adozione non dipenderà più dalla disponibilità tecnologica, ma dalla capacità delle aziende di integrarli in modo consapevole e sostenibile.
La sfida che si prospetta non riguarda più l'opportunità del loro utilizzo, ma la capacità di indirizzarne l'impiego in modo rigoroso, controllato e sostenibile.

Bibliografia

[Adz11] G. Adzic. *Specification by Example: How Successful Teams Deliver the Right Software*. Manning, 2011. ISBN: 9781638351368. URL: https://books.google.it/books?id=fDszEAAAQBAJ.

[Bai+22] Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, Carol Chen, Catherine Olsson, Christopher Olah, Danny Hernandez, Dawn Drain, Deep Ganguli, Dustin Li, Eli Tran-Johnson, Ethan Perez, Jamie Kerr, Jared Mueller, Jeffrey Ladish, Joshua Landau, Kamal Ndousse, Kamile Lukosuite, Liane Lovitt, Michael Sellitto, Nelson Elhage, Nicholas Schiefer, Noemi Mercado, Nova DasSarma, Robert Lasenby, Robin Larson, Sam Ringer, Scott Johnston, Shauna Kravec, Sheer El Showk, Stanislav Fort, Tamera Lanham, Timothy Telleen-Lawton, Tom Conerly, Tom Henighan, Tristan Hume, Samuel R. Bowman, Zac Hatfield-Dodds, Ben Mann, Dario Amodei, Nicholas Joseph, Sam McCandlish, Tom Brown e Jared Kaplan. «Constitutional AI: Harmlessness from AI Feedback». In: (2022). DOI: 2212.08073v1. URL: https://arxiv.org/abs/2212.08073v1.

[BG23] Stephan Böhm e Stefan Graser. «AI-based Mobile App Prototyping Status Quo, Perspectives and Preliminary Insights from Experimental Case Studies». In: nov. 2023.

[Bor23] Ali Borji. «A Categorical Archive of ChatGPT Failures». In: (feb. 2023). DOI: 10.21203/rs.3.rs-2895792/v1.

[Bro+98] William H. Brown, Raphael C. Malveau, Hays W. "Skip" McCormick e Thomas J. Mowbray. *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis*. 1st. USA: John Wiley & Sons, Inc., 1998. ISBN: 0471197130.

[Bro96] John Brooke. «SUS: A "Quick and Dirty" Usability Scale». In: *Usability Evaluation in Industry*. A cura di Patrick W. Jordan, Bruce Thomas, Ian L. McClelland e Bernard A. Weerdmeester. Taylor & Francis, 1996.

[Cam+17] Bradley Camburn, Vimal Viswanathan, Julie Linsey, David Anderson, Daniel Jensen, Kevin Otto e Kristin Wood. «Design prototyping methods: State of the art in strategies, techniques, and guidelines». In: *Design Science* 3 (ago. 2017). DOI: 10.1017/dsj.2017.10.

[Che+21] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde, Jared Kaplan, Harri Edwards, Yura Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray e Wojciech Zaremba. «Evaluating Large Language Models Trained on Code». In: (lug. 2021). DOI: 10.48550/arXiv.2107.03374.

[Chr+17] Paul F. Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg e Dario Amodei. «Deep reinforcement learning from human preferences». In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 4302–4310. ISBN: 9781510860964.

[Gao+23] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Qianyu Guo, Meng Wang e Haofen Wang. «Retrieval-Augmented Generation for Large Language Models: A Survey». In: *ArXiv* abs/2312.10997 (2023). URL: https://api.semanticscholar.org/CorpusID:266359151.

[Hen+20] Peter Henderson, Jie Hu, Joshua Romoff, Emma Brunskill, Dan Jurafsky e Joelle Pineau. «Towards the Systematic Reporting of the Energy and

Carbon Footprints of Machine Learning». In: *ArXiv* abs/2002.05651 (2020). URL: https://api.semanticscholar.org/CorpusID:211096620.

[HL25] Nam Huynh e Beiyu Lin. «Large Language Models for Code Generation: A Comprehensive Survey of Challenges, Techniques, Evaluation, and Applications». In: *ArXiv* abs/2503.01245 (2025). URL: https://api.semanticscholar.org/CorpusID:276742040.

[Hou+25] Xinyi Hou, Yanjie Zhao, Shenao Wang e Haoyu Wang. «Model Context Protocol (MCP): Landscape, Security Threats, and Future Research Directions». In: *ArXiv* abs/2503.23278 (2025). URL: https://api.semanticscholar.org/CorpusID:277452486.

[Ji+23] Zihao Ji, Nayeon Lee, Michael A. Hedderich, Danqi Chen e Percy Liang. «Survey of Hallucination in Natural Language Generation». In: *ACM Computing Surveys* 55 (2023). DOI: 10.1145/3571730.

[Jin+25] Bowen Jin, Jinsung Yoon, Jiawei Han e Sercan O Arik. «Long-Context LLMs Meet RAG: Overcoming Challenges for Long Inputs in RAG». In: *The Thirteenth International Conference on Learning Representations*. 2025. URL: https://openreview.net/forum?id=oU3tpaR8fm.

[Jon18] Nicola Jones. «How to Stop Data Centres from Gobbling Up Electricity». In: *Nature* 561.7722 (2018).

[Jou+21] Norman P. Jouppi, Doe Hyun Yoon, Matthew Ashcraft, Mark Gottscho, Thomas B. Jablin, George Kurian, James Laudon, Sheng Li, Peter Ma, Xiaoyu Ma, Thomas Norrie, Nishant Patil, Sushma Prasad, Cliff Young, Zongwei Zhou e David Patterson. «Ten Lessons From Three Generations Shaped Google's TPUv4i : Industrial Product». In: *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. 2021, pp. 1–14. DOI: 10.1109/ISCA52012.2021.00010.

[Kas22] Atoosa Kasirzadeh. «Taxonomy of Risks posed by Language Models». In: ACM: ACM, 2022, pp. 214–229. URL: https://philsci-archive.pitt.edu/21523/.

[Lew+20] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel e Douwe Kiela. «Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks». In: *Advances in Neural Information Processing Systems*. A cura di H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan e H. Lin. Vol. 33. Curran Associates, Inc., 2020, pp. 9459–9474. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf.

[Liu+24] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni e Percy Liang. «Lost in the Middle: How Language Models Use Long Contexts». In: *Transactions of the Association for Computational Linguistics* 12 (feb. 2024), pp. 157–173. ISSN: 2307-387X. DOI: 10.1162/tacl_a_00638. eprint: https://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00638/2336043/tacl_a_00638.pdf. URL: https://doi.org/10.1162/tacl_a_00638.

[MGK08] Gloria Mark, Daniela Gudith e Ulrich Klocke. «The cost of interrupted work: more speed and stress». In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '08. Florence, Italy: Association for Computing Machinery, 2008, pp. 107–110. ISBN: 9781605580111. DOI: 10.1145/1357054.1357072. URL: https://doi.org/10.1145/1357054.1357072.

[MK24] Ieva Margevica-Grinberga e Aija Kaleja. «Job Shadowing as a Method in Further Education». In: *Society. Integration. Education. Proceedings of the International Scientific Conference*. Vol. 2. 2024. DOI: 10.17770/sie2024vol2.7916.

[Moh+25] Abdelrahman Mohamed, Mariam Jan, Rahma Badran, Sama Mohamed, Yousra Amr e Nada Shorim. «A Review on Detecting and Managing Documentation Drift in Software Development». In: *2025 International*

Mobile, Intelligent, and Ubiquitous Computing Conference (MIUCC). 2025, pp. 546–552. DOI: 10.1109/MIUCC66482.2025.11196773.

[Pat+21] David A. Patterson, Joseph Gonzalez, Quoc V. Le, Chen Liang, Lluís-Miquel Munguía, Daniel Rothchild, David R. So, Maud Texier e Jeff Dean. «Carbon Emissions and Large Neural Network Training». In: *ArXiv* abs/2104.10350 (2021). URL: https://api.semanticscholar.org/CorpusID:233324338.

[Per+22] Ethan Perez, Saffron Huang, Francis Song, Trevor Cai, Roman Ring, John Aslanides, Amelia Glaese, Nat McAleese e Geoffrey Irving. «Red Teaming Language Models with Language Models». In: *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. A cura di Yoav Goldberg, Zornitsa Kozareva e Yue Zhang. Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, dic. 2022, pp. 3419–3448. DOI: 10.18653/v1/2022.emnlp-main.225. URL: https://aclanthology.org/2022.emnlp-main.225/.

[Sch+20] Roy Schwartz, Jesse Dodge, Noah A. Smith e Oren Etzioni. «Green AI». In: *Commun. ACM* 63.12 (nov. 2020), pp. 54–63. ISSN: 0001-0782. DOI: 10.1145/3381831. URL: https://doi.org/10.1145/3381831.

[Sch+23] Timo Schick, Jane Dwivedi-Yu, Roberto Dessí, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda e Thomas Scialom. «Toolformer: language models can teach themselves to use tools». In: *Proceedings of the 37th International Conference on Neural Information Processing Systems*. NIPS '23. New Orleans, LA, USA: Curran Associates Inc., 2023.

[SGM19] Emma Strubell, Ananya Ganesh e Andrew Mccallum. «Energy and Policy Considerations for Deep Learning in NLP». In: gen. 2019, pp. 3645–3650. DOI: 10.18653/v1/P19-1355.

[She+23] Toby Shevlane, Sebastian Farquhar, Ben Garfinkel, Mary Phuong, Jess Whittlestone, Jade Leung, Daniel Kokotajlo, Nahema Marchal, Markus Anderljung, Noam Kolt, Lewis Ho, Divya Siddarth, Shahar Avin,

Will Hawkins, Been Kim, Iason Gabriel, Vijay Bolina, Jack Clark, Yoshua Bengio e Allan Dafoe. «Model evaluation for extreme risks». In: (mag. 2023). DOI: 10.48550/arXiv.2305.15324.

[UNI16] UNI - Ente Italiano di Normazione. *UNI 11648:2016 Attività professionali non regolamentate – Project manager – Definizione dei requisiti di conoscenza, abilità e competenza*. Norma tecnica italiana. Milano: Ente Italiano di Normazione (UNI), 2016.

[Wan+23] Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee e Ee-Peng Lim. «Plan-and-Solve Prompting: Improving Zero-Shot Chain-of-Thought Reasoning by Large Language Models». In: (mag. 2023). DOI: 10.48550/arXiv.2305.04091.

[Wei+22] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le e Denny Zhou. «Chain-of-thought prompting elicits reasoning in large language models». In: *Proceedings of the 36th International Conference on Neural Information Processing Systems*. NIPS '22. New Orleans, LA, USA: Curran Associates Inc., 2022. ISBN: 9781713871088.

[WJ95] Michael Wooldridge e Nicholas R. Jennings. «Intelligent agents: theory and practice». In: *The Knowledge Engineering Review* 10.2 (1995), pp. 115–152. DOI: 10.1017/S0269888900008122.

[Wu+22] Xingjiao Wu, Luwei Xiao, Yixuan Sun, Junhang Zhang, Tianlong Ma e Liang He. «A survey of human-in-the-loop for machine learning». In: *Future Generation Computer Systems* 135 (2022), pp. 364–381. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2022.05.014. URL: https://www.sciencedirect.com/science/article/pii/S0167739X22001790.

[Yan+25] Minglai Yang, Ethan Huang, Liang Zhang, Mihai Surdeanu, William Yang Wang e Liangming Pan. «How Is LLM Reasoning Distracted by Irrelevant Context? An Analysis Using a Controlled Benchmark». In: *Proceedings of the 2025 Conference on Empirical Methods in Natural*

Language Processing. A cura di Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose e Violet Peng. Suzhou, China: Association for Computational Linguistics, nov. 2025, pp. 13340–13358. ISBN: 979-8-89176-332-6. DOI: 10.18653/v1/2025.emnlp-main.674. URL: https://aclanthology.org/2025.emnlp-main.674/.

[Yao+23] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan e Yuan Cao. «ReAct: Synergizing Reasoning and Acting in Language Models». In: *The Eleventh International Conference on Learning Representations*. 2023. URL: https://openreview.net/forum?id=WE_vluYUL-X.

[YTÖ22] Burak Yetiştiren, Eray Tüzün e Işık Özsoy. «Assessing the Quality of GitHub Copilot's Code Generation». In: nov. 2022. DOI: 10.1145/3558489.3559072.

[Zho+24] Zixuan Zhou, Xuefei Ning, Ke Hong, Tianyu Fu, Jiaming Xu, Shiyao Li, Yuming Lou, Luning Wang, Zhihang Yuan, Xiuhong Li, Shengen Yan, Guohao Dai, Xiao-Ping Zhang, Yuhan Dong e Yu Wang. «A Survey on Efficient Inference for Large Language Models». In: *ArXiv* abs/2404.14294 (2024). URL: https://api.semanticscholar.org/CorpusID:269293007.

Sitografia

[AI25] Qodo AI. *Windsurf vs Cursor: A Detailed Comparison.* https://qodo.ai/blog/windsurf-vs-cursor. 2025.

[Bui25] Builder.io. *Windsurf vs Cursor: Which AI IDE is Better?* https://www.builder.io/blog/windsurf-vs-cursor. 2025.

[Cli] Cline. *Cline: Open-Source AI Coding Agent.* https://cline.bot/. Ultima visita: 3 dicembre 2025.

[Cod] Codeium. *Windsurf – Official Website.* https://codeium.com/windsurf. Ultima visita: 3 dicembre 2025.

[Cod25] Roo Code. *Roo Code Site.* https://roocode.com/. 2025.

[Cou23] Coursera. *Prototyping Tools.* https://www.coursera.org/articles/prototyping-tools. 2023.

[Dat25] DataCamp. *Windsurf vs Cursor: Which AI IDE Should You Use?* https://www.datacamp.com/blog/windsurf-vs-cursor. 2025.

[Dev25] Netcorp Software Development. *Will AI Replace Programmers? The Future of Coding in the Age of Artificial Intelligence.* https://www.netcorpsoftwaredevelopment.com/article/will-ai-replace-programmers. 2025.

[Dig24] McKinsey Digital. *State of AI in Software Development 2024.* https://www.mckinsey.com/capabilities/quantumblack/our-insights/the-state-of-ai-2024. 2024.

[Doc25] Cline Docs. *Installing Cline & Getting Started.* https://docs.cline.bot/introduction/welcome. 2025.

[Fos25] Ryan Foster. *The Complete Guide to AI in Software Development: Transforming Code Creation in 2025*. https://emmo.ai/blog/ai-in-software-development. 2025.

[Gau23] M. Gaudion. *What is Documentation Drift and How to Avoid It*. https://gaudion.dev/blog/documentation-drift. 2023.

[Git23] GitHub. *Octoverse: The State of Open Source and the Rise of AI*. https://github.blog/news-insights/research/the-state-of-open-source-and-ai/. 2023.

[Gos22] Dan Goslen. *How to Get Software Documentation Right*. https://dangoslen.me/blog/how-to-get-software-documentation-right/. 2022.

[Ins25a] Business Insider. *Perplexity's engineers use 2 AI coding tools, and they've cut development time from days to hours*. https://www.businessinsider.com/perplexity-engineers-ai-tools-cut-development-time-days-hours-2025-7?op=1. 2025.

[Ins25b] Business Insider. *These Are the Most Popular AI Coding Tools Among Engineers*. https://www.businessinsider.com/ai-coding-tools-popular-github-gemini-code-assist-cursor-q-2025-7?op=1. 2025.

[Lab25] Engine Labs. *Cursor AI: An In-Depth Review*. https://blog.enginelabs.ai/cursor-ai-an-in-depth-review. 2025.

[LLM] LLM-Stats. *LLM Comparison Dashboard*. https://llm-stats.com. Ultima visita: 3 dicembre 2025.

[Lov] Lovable.dev. *AI-Powered App Builder*. https://lovable.dev. Ultima visita: 3 dicembre 2025.

[Mar25a] Visual Studio Marketplace. *Cline – AI Coding Agent*. https://marketplace.visualstudio.com/items?itemName=saoudrizwan.claude-dev. 2025.

[Mar25b] Visual Studio Marketplace. *Roo Code - AI Agent*. https://marketplace.visualstudio.com/items?itemName=RooVeterinaryInc.roo-cline. 2025.

[Mic] Microsoft. *Visual Studio Code Extension API Documentation*. https://code.visualstudio.com/api. Ultima visita: 3 dicembre 2025.

[Mil22] Tara Milburn. *Why Effective Onboarding Is Critical To Employee Retention*. https://www.forbes.com/councils/forbesbusinesscouncil/2022/12/02/why-effective-onboarding-is-critical-to-employee-retention/. 2022.

[Opea] OpenAI. *Function Calling and JSON Mode*. https://platform.openai.com/docs/guides/function-calling. Ultima visita: 3 dicembre 2025.

[Opeb] OpenAI. *Safety System Overview*. https://openai.com/it-IT/safety/. Ultima visita: 3 dicembre 2025.

[Phe24] Julia Phelan. *Onboarding New Employees Without Overwhelming Them*. https://hbr.org/2024/04/onboarding-new-employees-without-overwhelming-them. 2024.

[Qub25] Qubika. *A Practical Review of Roo Code*. https://www.qubika.com/blog/roo-code-review. 2025.

[Ser24] Amazon Web Services. *Security Considerations for Data in Generative AI*. https://docs.aws.amazon.com/whitepapers/latest/security-data-generative-ai. 2024.

[sha] shadcn. *shadcn/ui Documentation*. https://ui.shadcn.com. Ultima visita: 3 dicembre 2025.

[Sha23] Inbal Shani. *Survey Reveals AI's Impact on the Developer Experience*. https://github.blog/news-insights/research/survey-reveals-ais-impact-on-the-developer-experience/. 2023.

[Sic24] Anastassiya Sichkarenko. *The State of Developer Ecosystem 2024*. https://www.jetbrains.com/lp/devecosystem-2024. 2024.

[Ver25] The Verge. *Can Cursor AI Automate Programming?* https://www.theverge.com/tech/cursor-ai. 2025.

[Whi24] Stacy Ledesma Whitenight. *Why Prototyping is the Breakthrough Strategy for Innovation.* https://code.likeagirl.io/why-prototyping-is-the-breakthrough-strategy-for-innovation-b5aaa1b12b84. 2024.

[Zap] Zapier. *Windsurf vs Cursor.* https://zapier.com/blog/windsurf-vs-cursor. Ultima visita: 3 dicembre 2025.

Ringraziamenti

Scrivi qui i tuoi ringraziamenti.