ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

Dipartimento di Informatica – Scienza e Ingegneria
Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

Evoluzione architetturale e migrazione tecnologica di un'applicazione web AI-based per la traduzione e l'analisi di testi in lingue classiche: il caso Teseo

Relatore:
Prof.
Roberto Girau

Presentata da:
Francesco Carlucci

Correlatore:
Dott.
Edoardo Procino

Sessione Marzo 2026
Anno Accademico 2024/2025

Indice

Elenco delle figure

Elenco delle tabelle

Introduzione

L'integrazione dell'intelligenza artificiale nel settore educativo rappresenta una delle trasformazioni più significative degli ultimi anni. I progressi nel campo dei Large Language Models (LLM) hanno aperto possibilità inedite per lo sviluppo di strumenti didattici personalizzati, capaci di adattarsi alle esigenze degli studenti e di affrontare domini disciplinari sempre più specialistici. Tuttavia, mentre le applicazioni AI-based si sono diffuse rapidamente in ambiti come l'apprendimento delle lingue moderne e le discipline STEM, il supporto alle discipline umanistiche classiche - in particolare lo studio del latino e del greco antico - resta ancora largamente inesplorato.

Le lingue classiche presentano sfide peculiari per i sistemi di intelligenza artificiale: una morfologia eccezionalmente ricca, una sintassi flessibile in cui l'ordine delle parole non segue schemi rigidi, un'elevata polisemia e la necessità di conoscenze extratestuali per una corretta interpretazione. Gli strumenti digitali attualmente disponibili, dai dizionari morfologici ai traduttori generalisti, offrono funzionalità parziali e non integrate, lasciando scoperta l'esigenza di una piattaforma che accompagni lo studente nell'intero processo di analisi e traduzione di un testo classico.

Il presente lavoro di tesi, svolto durante il tirocinio presso Memoraiz, una startup bolognese specializzata in soluzioni AI per l'educazione e la cultura, documenta l'evoluzione architetturale e la migrazione tecnologica di Teseo, un'applicazione web progettata per supportare la traduzione e l'analisi di testi in latino e greco antico attraverso l'utilizzo di LLM. L'obiettivo principale del progetto è stato quello di riprogettare integralmente l'architettura della

piattaforma. In particolare, il lavoro ha previsto un'analisi dell'architettura originale con identificazione delle criticità tecniche e definizione dei requisiti di riprogettazione; la progettazione e implementazione di una nuova architettura scalabile basata su TypeScript, Payload CMS, PostgreSQL e AWS, volta a eliminare il vendor lock-in e migliorare la manutenibilità del sistema; l'integrazione di una pipeline AI orchestrata tramite il framework Mastra, con rilevamento automatico della lingua, esecuzione parallela di traduzione e analisi grammaticale, e streaming progressivo dei risultati; infine, una valutazione sperimentale di 13 modelli della famiglia OpenAI sui tre task fondamentali dell'applicazione, con analisi comparativa in termini di qualità, latenza e costi per la selezione dei modelli di produzione.

La tesi è articolata in tre capitoli, di seguito descritti:

- **Capitolo 1 – Contesto e stato dell'arte**: fornisce i fondamenti teorici degli LLM, analizza lo stato dell'arte delle applicazioni AI in ambito educativo e le sfide specifiche della traduzione e analisi automatica di testi in lingue classiche;

- **Capitolo 2 – Analisi del sistema esistente e requisiti di riprogettazione**: descrive la piattaforma Teseo nella sua configurazione originaria, ne identifica le criticità e formalizza i requisiti funzionali e non funzionali della nuova implementazione;

- **Capitolo 3 – Sviluppo e implementazione della nuova architettura**: presenta la migrazione a TypeScript e Payload CMS, l'ottimizzazione della pipeline AI con orchestrazione tramite Mastra, la migrazione a PostgreSQL, il deployment su AWS e la campagna di valutazione di modelli OpenAI per la selezione dei modelli da usare in produzione.

Capitolo 1

Contesto e stato dell'arte

L'integrazione dell'intelligenza artificiale nel settore educativo sta attraversando una fase di profonda trasformazione, guidata dai recenti progressi nel campo degli LLM. Questi sistemi hanno dimostrato capacità senza precedenti nella comprensione e generazione del linguaggio naturale, rendendo possibile lo sviluppo di applicazioni educative personalizzate, interattive e scalabili che, fino a pochi anni fa, erano impensabili.

Il presente capitolo fornisce il contesto necessario per comprendere il lavoro svolto durante il tirocinio presso Memoraiz, una startup specializzata nello sviluppo di soluzioni AI per l'educazione e la cultura. Dopo aver presentato l'azienda e il suo ecosistema, verranno analizzati i fondamenti teorici degli LLM, lo stato dell'arte delle applicazioni web AI-based in ambito educativo e, infine, le sfide specifiche legate alla traduzione e analisi automatica di testi in lingue classiche, che costituiscono il dominio applicativo del progetto di tesi.

1.1 Contesto aziendale

Memoraiz [1] è una startup di Bologna che sviluppa soluzioni di intelligenza artificiale orientate alla valorizzazione e alla fruizione della conoscenza. L'azienda opera principalmente nei settori educativo, culturale ed editoria-

le, collaborando con piattaforme educative, musei, editori e organizzazioni no profit, ma si occupa anche di sviluppare applicazioni B2C per il proprio pubblico.

La missione dell'azienda è quella di supportare le organizzazioni nella trasformazione del proprio patrimonio informativo in esperienze digitali più accessibili, personalizzate e coinvolgenti.

L'ecosistema tecnologico di Memoraiz si articola su tre componenti principali:

- **La Bussola delle 6C**: un framework strategico che guida le organizzazioni nella definizione della propria strategia di intelligenza artificiale e nella progettazione di soluzioni personalizzate;
- **Moduli AI componibili**: soluzioni modulari che consentono di rigenerare i contenuti esistenti e di creare nuove esperienze digitali facilmente integrabili nei sistemi già in uso;
- **Assistenti AI verticali**: strumenti specializzati progettati per supportare i team interni, costruiti sul contesto specifico di ciascuna organizzazione, a differenza delle soluzioni di intelligenza artificiale generaliste.

Le soluzioni proposte sono caratterizzate da personalizzabilità, scalabilità e capacità di evoluzione nel tempo, grazie a un'architettura leggera e interoperabile che mantiene costantemente l'intervento umano nel ciclo decisionale (*human-in-the-loop*).

1.2 Large Language Models

Gli LLM sono modelli di deep learning caratterizzati da dimensioni senza precedenti, sia in termini di parametri (da miliardi a trilioni) sia di dati di addestramento. Gli LLM sono addestrati su obiettivi generali di predizione linguistica che conferiscono loro svariate capacità: dalla traduzione alla generazione di codice, dal ragionamento logico alla sintesi di documenti complessi.

Queste caratteristiche li rendono particolarmente adatti come base tecnologica per applicazioni educative, dove la necessità di comprensione contestuale, adattabilità e capacità esplicative è fondamentale.

1.2.1 Fondamenti teorici: l'architettura Transformer

La base tecnologica degli LLM moderni è l'architettura *Transformer*, introdotta da Vaswani et al. nel 2017 con il paper "Attention is All You Need" [2]. Questa architettura ha rivoluzionato il campo del Natural Language Processing (NLP) sostituendo le precedenti architetture sequenziali (RNN, LSTM) con un meccanismo completamente basato sull'attenzione.

Meccanismo di Self-Attention

Il cuore dell'architettura Transformer è il meccanismo di *self-attention*, che permette al modello di valutare l'importanza relativa di diverse parti di una sequenza di input quando processa ogni elemento. Formalmente, dato un input rappresentato come sequenza di vettori, il meccanismo di attention calcola:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{1.1}$$

dove Q (query), K (key) e V (value) sono proiezioni lineari dell'input, e d_k è la dimensionalità delle key, utilizzata per la normalizzazione.

Il *multi-head attention* estende questo concetto permettendo al modello di catturare diverse relazioni contestuali in parallelo:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O \tag{1.2}$$

dove ogni $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$.

Architettura completa

L'architettura Transformer originale è composta da un encoder e un decoder, ciascuno formato da stack di layer identici. Ogni layer dell'encoder contiene:

- Un sublayer di multi-head self-attention;
- Un sublayer di feed-forward network fully connected;
- Residual connections e layer normalization attorno a ciascun sublayer;

Il decoder ha una struttura simile ma include un terzo sublayer che applica multi-head attention sull'output dell'encoder.

Gli LLM moderni utilizzano principalmente varianti di questa architettura:

- **Encoder-only** (es. BERT): ottimizzati per task di comprensione del testo;
- **Decoder-only** (es. GPT, LLaMA): ottimizzati per generazione di testo;
- **Encoder-decoder** (es. T5, BART): versatili per task di trasformazione testo-testo;

1.2.2 Evoluzione degli LLM

Prima generazione: modelli pre-addestrati (2018-2019)

- **BERT** (Bidirectional Encoder Representations from Transformers) [3], sviluppato da Google nel 2018, ha introdotto il concetto di pre-training bidirezionale attraverso due task: Masked Language Modeling (MLM) e Next Sentence Prediction (NSP). BERT-base conta 110 milioni di parametri, mentre BERT-large raggiunge i 340 milioni.
- **GPT** (Generative Pre-trained Transformer) [4] di OpenAI ha adottato un approccio decoder-only con pre-training autoregressivo, dimostrando che modelli generalisti potevano essere poi specializzati su task specifici attraverso fine-tuning. GPT conta 117 milioni di parametri.

Seconda generazione: scaling e few-shot learning (2019-2021)

- **GPT-2** [5] (2019, 1.5 miliardi di parametri) ha dimostrato che l'aumento della scala porta a capacità emergenti, mostrando performance sorprendenti su task per cui non era stato esplicitamente addestrato.

- **GPT-3** [6] (2020, 175 miliardi di parametri) ha segnato un cambio di paradigma introducendo il concetto di *in-context learning*: la capacità di eseguire task semplicemente fornendo esempi nel prompt, senza necessità di fine-tuning. GPT-3 ha dimostrato capacità di few-shot e zero-shot learning su un'ampia gamma di task.

- **T5** (Text-to-Text Transfer Transformer) [7] ha unificato diversi task NLP in un formato comune testo-a-testo, raggiungendo 11 miliardi di parametri nella versione più grande.

Terza generazione: modelli conversazionali e allineamento (2022-presente)

- **InstructGPT e ChatGPT** [8] hanno introdotto il Reinforcement Learning from Human Feedback (RLHF), un processo in tre fasi:

  1. Supervised fine-tuning su dimostrazioni umane di alta qualità;
  2. Training di un reward model basato su preferenze umane;
  3. Ottimizzazione del modello linguistico tramite Proximal Policy Optimization (PPO);

  Questo approccio ha drasticamente migliorato l'allineamento dei modelli con le intenzioni umane e la loro utilità come assistenti conversazionali.

- **GPT-4** [9] (2023) rappresenta un ulteriore salto qualitativo, con architettura multimodale (testo e immagini), capacità di ragionamento

avanzate e migliore aderenza alle istruzioni. Sebbene OpenAI non abbia rivelato il numero esatto di parametri, si stima che GPT-4 abbia dimensioni significativamente superiori a GPT-3.

- **LLaMA** (LLM Meta AI) [10] e **LLaMA 2** [11] di Meta hanno democratizzato l'accesso agli LLM fornendo modelli open-source performanti (da 7B a 70B parametri) che possono essere eseguiti su hardware consumer.

- **Claude** (Anthropic) [12], sviluppato con tecniche di Constitutional AI, enfatizza sicurezza ed affidabilità.

- **Gemini** (Google DeepMind) [13] è un modello multimodale nativo progettato per integrare testo, immagini, audio e video.

1.2.3 Principi di funzionamento degli LLM

Pre-training e tokenizzazione

Gli LLM vengono addestrati su dataset di testo di grandi dimensioni (centinaia di miliardi o trilioni di token). Il processo inizia con la **tokenizzazione**, che converte il testo grezzo in sequenze di token. Tecniche comuni includono:

- **Byte-Pair Encoding (BPE)**: utilizzato da GPT, costruisce un vocabolario di subword frequenti;

- **WordPiece**: utilizzato da BERT, simile a BPE;

- **SentencePiece**: agnostico alla lingua, utilizzato da T5 e LLaMA;

L'obiettivo del pre-training per modelli decoder-only è predire il token successivo data la sequenza precedente, massimizzando la log-likelihood:

$$\mathcal{L} = \sum_{i=1}^{n} \log P(x_i | x_1, ..., x_{i-1}; \theta) \tag{1.3}$$

dove θ rappresenta i parametri del modello.

Emergent abilities e scaling laws

La ricerca ha identificato delle *scaling laws* che descrivono come le performance degli LLM migliorano con l'aumentare di tre fattori: dimensione del modello (numero di parametri), dimensione del dataset di training e risorse computazionali utilizzate per l'addestramento [14].

Oltre una certa soglia dimensionale, gli LLM manifestano "*capacità emergenti*" [15]: abilità non presenti in modelli più piccoli e non esplicitamente programmate, come:

- Ragionamento aritmetico multi-step;
- Traduzione tra lingue non viste durante il training;
- Generazione di codice funzionante;
- Chain-of-thought reasoning (ragionamento passo-passo);

Prompt engineering e in-context learning

Gli LLM possono adattare il loro comportamento in base al **prompt**, il testo fornito come input. Tecniche di prompt engineering includono:

- **Zero-shot prompting**: richiesta diretta senza esempi;
- **Few-shot prompting**: inclusione di esempi input-output nel prompt;
- **Chain-of-thought prompting** [16]: richiedere ragionamento esplicito passo-passo;
- **Role prompting**: assegnare un ruolo o persona al modello;
- **Structured prompting**: utilizzare formati specifici (JSON, XML) per output strutturati;

1.2.4 Utilizzo in applicazioni reali

Architetture applicative comuni

Le applicazioni basate su LLM tipicamente adottano una delle seguenti architetture:

1. **Direct API calls**
    - Utilizzo di API fornite da provider (OpenAI, Anthropic, Google);
    - Vantaggi: nessuna gestione infrastrutturale, accesso a modelli all'avanguardia;
    - Svantaggi: costi per token, dipendenza da servizi esterni, limiti di privacy;
2. **Self-hosted models**
    - Deploy di modelli open-source su infrastruttura propria;
    - Vantaggi: controllo completo, privacy, costi fissi;
    - Svantaggi: complessità gestionale, costi hardware;
3. **Hybrid architectures**
    - Combinazione di modelli locali per task semplici e API esterne per task complessi;

Applicazioni per dominio

1. **Assistenti conversazionali e chatbot**
    - Virtual assistants: Siri, Google Assistant, Alexa integrano sempre di più gli LLM;
    - Chatbot: ChatGPT, Claude, Gemini, Microsoft Copilot;
2. **Generazione e editing di contenuti**

- Content creation: generazione di articoli, blog post, copy pubblicitario;
- Code generation: GitHub Copilot, Claude Code, Cursor;

3. **Educazione**

- Generazione contenuti didattici: quiz, esercizi, spiegazioni;
- Feedback su elaborati degli studenti;

1.2.5 Sfide e limitazioni

Nonostante i progressi, gli LLM presentano sfide significative:

Allucinazioni

Gli LLM possono generare informazioni plausibili ma false. Tra le strategie di mitigazione troviamo:

- Retrieval-Augmented Generation (RAG) per collegare le risposte a fonti verificabili;
- *Temperature* bassa per ridurre la creatività nella risposta;
- Fact-checking post-generation;

Costi e sostenibilità

- Training: GPT-3 ha richiesto circa $4.6M di costo computazionale [17];
- Inference: costi per token per modelli API-based;
- Impatto ambientale: carbon footprint significativo;

Bias e equità

Gli LLM possono perpetuare o amplificare bias presenti nei dati di training:

- Bias di genere, razza, cultura;
- Rappresentazione sbilanciata di lingue e domini;
- Necessità di dataset curati e tecniche di de-biasing;

Privacy e sicurezza

- Memorizzazione involontaria di dati sensibili dal training set;
- Prompt injection: manipolazione del comportamento tramite input malevoli;
- Data leakage: rischio di esporre informazioni confidenziali;

1.3 Applicazioni web AI-based in ambito educativo

L'integrazione dell'intelligenza artificiale nel settore educativo ha conosciuto una rapida evoluzione negli ultimi anni, passando da sistemi di raccomandazione relativamente semplici a piattaforme capaci di generare contenuti didattici personalizzati e di adattarsi dinamicamente alle esigenze degli studenti. Questa sezione analizza le principali applicazioni web che sfruttano l'AI per la generazione di contenuti educativi, con particolare attenzione alle loro funzionalità, capacità di integrazione e obiettivi pedagogici.

1.3.1 Evoluzione cronologica e tecnologie

Prima generazione: sistemi di raccomandazione (2015-2019)

Le prime applicazioni di AI in ambito educativo si basavano principalmente su algoritmi di machine learning tradizionale per la personalizzazione dei percorsi di apprendimento.

Duolingo (2016-2019) ha introdotto algoritmi di *spaced repetition* e modelli predittivi per ottimizzare la memorizzazione del vocabolario [18]. Il sistema utilizzava principalmente tecniche di *collaborative filtering* e modelli statistici per adattare gli esercizi, senza generare contenuti ex-novo.

Khan Academy ha implementato sistemi di raccomandazione basati su knowledge tracing per suggerire video e esercizi appropriati al livello dello studente [19], utilizzando reti neurali ricorrenti (RNN) per modellare la progressione dell'apprendimento.

Seconda generazione: AI generativa per contenuti (2020-2022)

L'avvento di modelli di linguaggio pre-addestrati ha segnato un cambio di paradigma.

Quizlet ha integrato funzionalità di generazione automatica di quiz e flashcard attraverso modelli NLP, inizialmente basati su GPT-2 e successivamente su architetture transformer più avanzate [20]. La piattaforma genera domande a risposta multipla, definizioni e associazioni a partire da testi forniti dagli utenti.

Coursera Coach (lanciato nel 2021) utilizza modelli generativi per creare riassunti personalizzati delle lezioni, suggerire risorse aggiuntive e rispondere a domande degli studenti sui contenuti del corso [21]. L'architettura si basa su una combinazione di RAG e fine-tuning di modelli linguistici.

Terza generazione: LLM e tutoring intelligente (2023-presente)

Con l'arrivo di GPT-3.5, GPT-4 e modelli concorrenti, le applicazioni educative hanno raggiunto capacità di generazione di contenuti significativamente

più sofisticate.

Khanmigo (Khan Academy, 2023) rappresenta un assistente AI completo basato su GPT-4, capace di fungere da tutor personalizzato, generare spiegazioni su misura, creare esercizi adattivi e fornire feedback contestualizzato [22]. Il sistema implementa tecniche di *prompt engineering* avanzato e guardrail pedagogici per garantire che le risposte siano appropriate dal punto di vista didattico e non forniscano direttamente le soluzioni agli studenti.

Duolingo Max (2023) ha integrato GPT-4 per offrire due funzionalità principali: "Explain My Answer", che fornisce spiegazioni personalizzate degli errori commessi, e "Roleplay", che genera conversazioni contestualizzate per praticare la lingua [23]. L'integrazione sfrutta il contesto dello studente (livello, lingua madre, errori precedenti) per personalizzare le interazioni.

Scholarcy e **Quillbot** utilizzano modelli transformer per generare riassunti automatici di articoli accademici e testi complessi, con funzionalità di parafrasi e semplificazione del linguaggio [24, 25]. Questi strumenti si rivolgono principalmente a studenti universitari e ricercatori.

1.3.2 Analisi comparativa

La Tabella 1.1 presenta un confronto sintetico delle principali applicazioni analizzate.

1.3.3 Funzionalità e pattern architetturali comuni

Dall'analisi emerge un insieme di funzionalità ricorrenti:

- **Generazione di contenuti didattici**: quiz, esercizi, spiegazioni e riassunti generati automaticamente a partire da materiali di base o da knowledge base predefinite;
- **Personalizzazione adattiva**: utilizzo di dati storici dello studente (performance, errori, preferenze) per calibrare difficoltà e stile dei contenuti;

Tabella 1.1: Confronto tra applicazioni web AI-based per l'educazione

Applicazione	**Funzionalità AI principali**	**Tecnologie utilizzate**	**Integrazione**	**Obiettivi pedagogici**
Duolingo (2016-2019)	Adattamento percorso, spaced repetition	ML tradizionale, RNN	App standalone	Memorizzazione vocabolario, gamification
Khan Academy (pre-2023)	Raccomandazione contenuti, knowledge tracing	RNN, collaborative filtering	Piattaforma integrata	Apprendimento personalizzato matematica/scienze
Quizlet	Generazione quiz, flashcard automatiche	GPT-2/3, Transformer	API, LMS integration	Studio autonomo, preparazione esami
Coursera Coach	Riassunti, Q&A, raccomandazioni	RAG, LLM fine-tuned	Integrata in Coursera	Supporto corsi universitari online
Khanmigo (2023)	Tutoring, generazione esercizi, feedback	GPT-4, prompt engineering	Integrata in Khan Academy	Tutoring socratico, apprendimento attivo
Duolingo Max (2023)	Spiegazioni errori, conversazioni roleplay	GPT-4, context-aware prompting	Integrata in Duolingo	Apprendimento linguistico comunicativo
Scholarcy	Riassunti accademici, estrazione key concepts	Transformer, NER	Plugin browser, API	Lettura critica, ricerca accademica

- **Feedback intelligente**: spiegazioni contestualizzate degli errori che vanno oltre la semplice correzione, fornendo ragionamenti pedagogici;
- **Interazione conversazionale**: interfacce chat-based che simulano l'interazione con un tutor umano;

Dal punto di vista architetturale, la maggior parte delle soluzioni più recenti adotta un approccio basato su:

1. **Retrieval-Augmented Generation (RAG)**: combinazione di database di conoscenza strutturata con capacità generative dei LLM;
2. **Prompt engineering**: progettazione accurata di prompt che includono istruzioni pedagogiche, esempi e vincoli per garantire output appropriati;

3. **Fine-tuning domain-specific**: adattamento di modelli generali su dataset educativi specifici;

4. **Human-in-the-loop**: supervisione umana per validazione contenuti e miglioramento continuo;

1.3.4 Lacune e opportunità: AI per discipline umanistiche classiche

Nonostante la proliferazione di applicazioni AI in ambito educativo, emerge una significativa lacuna nel supporto a discipline umanistiche specialistiche, in particolare gli studi classici. Le applicazioni analizzate si concentrano prevalentemente su apprendimento linguistico moderno (inglese, spagnolo, francese), discipline STEM (matematica, scienze, programmazione) e competenze professionali e business.

Le lingue classiche (latino e greco antico) e l'analisi filologica presentano invece sfide specifiche che le rendono particolarmente complesse per l'applicazione di tecnologie AI:

- **Complessità morfologica**: sistemi di declinazione e coniugazione articolati che richiedono analisi grammaticale profonda;

- **Ambiguità semantica**: polisemia e dipendenza contestuale nella traduzione;

- **Corpus limitato**: quantità di testi digitalizzati inferiore rispetto alle lingue moderne;

- **Competenze specialistiche**: necessità di supportare non solo traduzione ma anche analisi stilistica, metrica, contestualizzazione storica;

Strumenti esistenti offrono dizionari morfologici e testi annotati, ma mancano di capacità generative e di interazione intelligente. Traduttori generalisti

come Google Translate o DeepL mostrano performance limitate su testi classici, non essendo ottimizzati per le specificità linguistiche del latino e greco antico [26].

Questa lacuna rappresenta l'opportunità per lo sviluppo di soluzioni specializzate che combinano modelli linguistici, interfacce che supportano il metodo di analisi filologica tradizionale, e l'integrazione con risorse digitali esistenti.

1.4 Traduzione e analisi automatica di testi in lingue classiche

Avendo evidenziato le criticità delle soluzioni AI per gli studi classici, è necessario esaminare più in dettaglio gli strumenti digitali esistenti per il latino e il greco antico, analizzandone funzionalità, architetture e limiti, al fine di identificare con precisione i gap tecnologici che motivano il lavoro presentato in questa tesi.

1.4.1 Strumenti digitali esistenti

Risorse lessicografiche e morfologiche

Perseus Digital Library [27] rappresenta la più completa risorsa digitale per gli studi classici. Sviluppata dalla Tufts University a partire dal 1987, offre:

- Testi greci e latini in formato digitale con morfologia collegata;
- Dizionari integrati (Lewis & Short per il latino, Liddell-Scott-Jones per il greco);
- Strumento di analisi morfologica *Morpheus* che identifica forme flesse e fornisce lemmi;
- Interfaccia di ricerca e concordanze;

Tuttavia, Perseus funziona principalmente come repository passivo: l'utente deve navigare manualmente tra testo, analisi morfologica e dizionario, senza supporto per traduzione assistita o contestualizzazione automatica.

Collatinus [28] è un lemmatizzatore e analizzatore morfologico open-source per il latino, disponibile sia come applicazione desktop che web. Offre:

- Analisi morfologica di forme latine con indicazione di tutti i possibili parsing;
- Scansione metrica per poesia latina;
- Dizionari integrati (Gaffiot, Lewis & Short);

Analogamente, **Eulexis** [29] fornisce funzionalità simili per il greco antico. Questi strumenti sono preziosi per l'analisi morfologica ma non offrono capacità di traduzione né di comprensione contestuale. **Dizionari digitali per lingue classiche**

Inoltre, negli ultimi anni, diverse case editrici hanno digitalizzato i principali vocabolari di riferimento per latino e greco antico, rendendoli accessibili tramite applicazioni web e mobile.

L'App del Rocci [30] è la versione digitale del celebre "Vocabolario Greco-Italiano" di Lorenzo Rocci, pubblicato dalla Società Editrice Dante Alighieri. Disponibile per web, Android e iOS, offre accesso all'intero contenuto lessicografico (circa 1.500 pagine), ricerca avanzata per lemma greco o traduzione italiana, parsing morfologico di forme flesse, cronologia delle ricerche e funzionamento offline.

Analogamente, Loescher Editore ha sviluppato le versioni digitali di due vocabolari di riferimento: **IL - Vocabolario della lingua latina** [31] di Luigi Castiglioni e Scevola Mariotti, e **GI - Vocabolario della lingua greca** [32] di Franco Montanari. Entrambe le applicazioni, disponibili per web, iOS e Android, offrono funzionalità simili all'App del Rocci (ricerca per lemma e forma flessa, consultazione offline, segnalibri), ma sono accessibili esclusivamente tramite codice di attivazione fornito con l'acquisto del dizionario cartaceo, rappresentando un modello di distribuzione più restrittivo.

Questi strumenti dimostrano il successo della digitalizzazione di risorse lessicografiche tradizionali e la validità del modello freemium[1] anche per strumenti educativi specialistici. Tuttavia, condividono limitazioni comuni:

- Funzionano come strumenti di consultazione passivi, senza traduzione automatica di frasi o periodi completi;
- Non forniscono analisi sintattica contestuale;
- Non integrano supporto per caricamento e analisi di testi completi o immagini;

Traduttori automatici generalisti

I principali traduttori neurali commerciali (**Google Translate**, **DeepL**, **Microsoft Translator**) supportano nominalmente latino e greco antico, ma con risultati problematici.

Studi comparativi [26] hanno evidenziato limitazioni significative:

- **Bassa accuratezza**: BLEU score tipicamente inferiore a 20 per latino e greco, contro 40+ per lingue moderne;
- **Mancanza di sensibilità al contesto**: difficoltà con polisemia e costrutti sintattici complessi;
- **Errori morfologici**: confusione tra casi, modi verbali, generi;
- **Assenza di giustificazione**: nessuna spiegazione delle scelte traduttive;
- **Corpus di training limitato**: principalmente testi biblici e poche opere canoniche;

[1] Il modello *freemium* combina un livello di accesso gratuito con funzionalità a pagamento. Gli utenti possono utilizzare gratuitamente le funzionalità base, mentre features avanzate, maggiori quote di utilizzo o assenza di limitazioni richiedono un abbonamento.

Progetti accademici specializzati

CIRCSE (Computational Infrastructure for Research in Classical Studies and Education) [33] è un progetto dell'Università Cattolica di Milano che sviluppa strumenti NLP per il latino:

- Treebank annotati sintatticamente (Latin Dependency Treebank);
- Tool per ricerche linguistiche avanzate;

Tuttavia, CIRCSE si rivolge principalmente a ricercatori di linguistica computazionale piuttosto che a studenti o docenti, e non fornisce interfacce user-friendly per traduzione assistita.

Classical Language Toolkit (CLTK) [34] è una libreria Python open-source che offre:

- Tokenizzazione e sentence segmentation per lingue classiche
- Lemmatizzazione
- Accesso programmatico a corpora e dizionari

CLTK è una risorsa preziosa per sviluppatori ma richiede competenze di programmazione e non fornisce un'applicazione end-user per studenti.

LatinISE e progetti di shallow parsing tentano di fornire analisi sintattica automatica ma con accuratezza limitata su testi complessi o non canonici.

1.4.2 Sfide tecniche specifiche

Complessità morfologica

Latino e greco antico presentano sistemi morfologici eccezionalmente ricchi:

- **Latino**:
  - 5 declinazioni nominali con 6 casi e 3 generi (maschile, femminile, neutro)

  - 4 coniugazioni verbali più una mista, 3 modi finiti, 4 modi indefiniti e diatesi attiva e passiva;
  - Irregolarità frequenti (verbi deponenti, semideponenti, anomali e difettivi);

- **Greco antico**:
  - 3 declinazioni con 5 casi e 3 generi (maschile, femminile, neutro)
  - 3 numeri grammaticali (singolare, plurale, duale)
  - 4 tempi principali (presente, futuro, perfetto, futuro perfetto), 3 tempi storici (imperfetto, aoristo, piuccheperfetto), 4 modi finiti, 3 modi indefiniti;
  - Aumento temporale e raddoppiamento;
  - Sistema complesso di modi (indicativo, congiuntivo, ottativo, imperativo);
  - Dialetti letterari (ionico-attico, dorico, eolico) con variazioni morfologiche;

Questa complessità comporta:

- **Elevata ambiguità**: una singola forma può avere molteplici parsing (es. latino *amare* può essere infinito presente, imperativo presente passivo, vocativo singolare di *amarus*);
- **Necessità di disambiguazione sintattica**: la selezione del parsing corretto richiede analisi del contesto frasale;

Flessibilità sintattica e ordine delle parole

A differenza delle lingue moderne con ordine prevalentemente SVO (Soggetto-Verbo-Oggetto), latino e greco utilizzano la flessione dei casi per codificare relazioni sintattiche, permettendo ordini delle parole estremamente variabili:

- "Puella rosam amat" (La ragazza ama la rosa);

- "Rosam puella amat" (enfasi sull'oggetto);
- "Amat rosam puella" (enfasi sul verbo);

Tutte e tre le frasi sono grammaticalmente valide e hanno lo stesso significato denotativo ma diverse connotazioni stilistiche. Modelli NLP addestrati principalmente su lingue moderne faticano a gestire questa flessibilità.

Scarsità di dati etichettati

L'addestramento di modelli di machine learning richiede grandi quantità di dati etichettati. Per le lingue classiche:

- **Corpus limitato**: il corpus dei testi latini classici che ci sono arrivati è piuttosto ristretto, soprattutto se confrontato con l'enorme quantità di testi disponibili per lingue moderne come l'inglese;
- **Testi non bilanciati**: predominanza di alcuni autori (Cicerone, Virgilio) e generi (prosa storica, epica), sotto-rappresentazione di altri;
- **Annotazione costosa**: creare treebank sintattici richiede expertise filologico e tempi lunghi;
- **Variabilità diacronica**: testi che coprono oltre 10 secoli (latino arcaico, classico, tardo, medievale) con variazioni linguistiche significative;

I principali treebank disponibili sono:

- Latin Dependency Treebank (LDT): 200k token;
- Ancient Greek Dependency Treebank (AGDT): 350k token;
- PROIEL (Pragmatic Resources in Old Indo-European Languages): testi biblici e patristici;

Queste dimensioni sono ordini di grandezza inferiori rispetto a dataset per lingue moderne (Penn Treebank: 1M token; Universal Dependencies: decine di milioni di token per lingue principali).

Polisemia e dipendenza contestuale

Molte parole latine e greche hanno significati altamente dipendenti dal contesto:

Esempio latino:

- *virtus*: coraggio, virtù morale, eccellenza, valore militare, potere;
- *ratio*: ragione, calcolo, metodo, teoria, rapporto, conto;

Esempio greco:

- λόγος (logos): parola, discorso, ragione, argomento, definizione, proporzione;
- ἀρετή (aretē): eccellenza, virtù, valore;

La disambiguazione richiede:

- Comprensione del contesto letterario e storico;
- Conoscenza delle convenzioni retoriche e stilistiche dell'autore;
- Familiarità con riferimenti intertestuali;

Necessità di conoscenza extralingustica

La comprensione di testi classici richiede frequentemente conoscenze enciclopediche:

- **Riferimenti mitologici**: allusioni a miti, personaggi, eventi;
- **Contesto storico**: eventi, personaggi storici, istituzioni romane/greche;
- **Geografia antica**: nomi di luoghi, popolazioni;
- **Realia**: pratiche culturali, religione, vita quotidiana;
- **Retorica**: figure retoriche, topoi, convenzioni di genere;

1.4.3 Gap nelle soluzioni esistenti

L'analisi degli strumenti disponibili e delle sfide tecniche evidenzia significative lacune:

Assenza di soluzioni integrate AI-based

Non esistono attualmente applicazioni web che combinino:

- Traduzione assistita da LLM ottimizzati per lingue classiche;
- Analisi morfologica e sintattica automatica;
- Interfaccia user-friendly per studenti;

Assenza di supporto per workflow filologico

Il processo tradizionale di traduzione include:

1. Segmentazione sintattica (identificare proposizioni);
2. Analisi morfologica di ogni forma;
3. Identificazione di soggetti, predicati, complementi;
4. Comprensione del significato contestuale;
5. Produzione di traduzione in un corretto italiano;

Nessuno strumento digitale supporta sistematicamente questo workflow, che rimane un processo mentale dello studente non assistito da tecnologia.

1.4.4 Opportunità per soluzioni basate su LLM

L'avvento degli LLM offre nuove possibilità per colmare questi gap:

- **Transfer learning**: modelli pre-addestrati su miliardi di token possono essere specializzati su corpora classici relativamente piccoli tramite fine-tuning o RAG;

- **Few-shot learning**: capacità di generalizzare da pochi esempi, mitigando la scarsità di dati annotati;
- **Integrazione di conoscenza**: attraverso RAG, si possono integrare dizionari, grammatiche e knowledge base specialistiche;

Il lavoro presentato in questa tesi esplora proprio queste opportunità, proponendo un'applicazione web che sfrutta LLM moderni (in particolare modelli della famiglia GPT di OpenAI) per fornire supporto intelligente, personalizzato e pedagogicamente efficace alla traduzione e analisi di testi latini e greci antichi.

L'architettura proposta integra:

- LLM fine-tuned o guidati tramite prompt engineering per traduzione sensibile al contesto
- Interfaccia web interattiva che supporta il workflow filologico

Nei capitoli successivi verrà presentato in dettaglio il design, l'implementazione e la valutazione di questa soluzione.

Capitolo 2

Analisi del sistema esistente e requisiti di riprogettazione

Il presente capitolo descrive il sistema esistente inizialmente, analizzandone funzionalità, architettura e limitazioni che hanno motivato la successiva riprogettazione. Dopo una presentazione generale della piattaforma Teseo e delle sue caratteristiche distintive, vengono esaminate le funzionalità offerte agli utenti e l'architettura tecnologica iniziale. Infine, vengono identificate le criticità del sistema originario ed i requisiti che hanno guidato le scelte progettuali della nuova implementazione.

2.1 Descrizione generale della piattaforma

Il tirocinio svolto presso Memoraiz ha riguardato l'evoluzione di Teseo, un'applicazione web progettata per supportare la traduzione e l'analisi di testi in lingua latina e greca attraverso l'utilizzo di tecniche di intelligenza artificiale. La piattaforma rappresenta uno degli assistenti AI verticali sviluppati dall'azienda, specificamente orientato al dominio degli studi classici.

Teseo si rivolge principalmente a tre categorie di utenti:

- **Studenti di scuola secondaria**: che utilizzano la piattaforma come strumento di supporto allo studio e alla comprensione di testi latini e

greci assegnati come compiti o presenti nei manuali scolastici;

- **Universitari ed appassionati di lingue classiche**: autodidatti o studenti universitari che desiderano approfondire la conoscenza delle lingue classiche in autonomia;

Al momento dell'inizio del tirocinio, la piattaforma contava oltre centomila utenti registrati, con un tasso di crescita mensile significativo e più di cinque mila utenti attivi ogni mese. L'obiettivo principale di Teseo è fornire non semplicemente traduzioni automatiche, ma un'esperienza di apprendimento guidato che supporti lo studente nel processo di comprensione del testo classico, mantenendo un approccio pedagogico coerente con i metodi tradizionali di insegnamento delle lingue antiche. A differenza di traduttori generici, Teseo è stato progettato fin dall'origine per rispettare le specificità morfologiche, sintattiche e semantiche del latino e del greco antico.

2.2 Funzionalità principali

2.2.1 Interfaccia utente e processo di utilizzo

Nella sua configurazione originaria, l'interfaccia principale presentata all'utente era costituita dalla schermata Home (Figura 2.1), dove si potevano processare solo testi di latino ed organizzata secondo un layout a due colonne che rifletteva il workflow tipico di utilizzo della piattaforma.

La colonna principale conteneva:

- Un'area di input testuale dove l'utente poteva inserire o incollare il testo latino da tradurre;

- Un pulsante di azione primaria *"Chiedi a Teseo"* che avviava il processo di traduzione ed analisi del testo;

- Un'area di output dove venivano visualizzati i risultati dell'elaborazione;

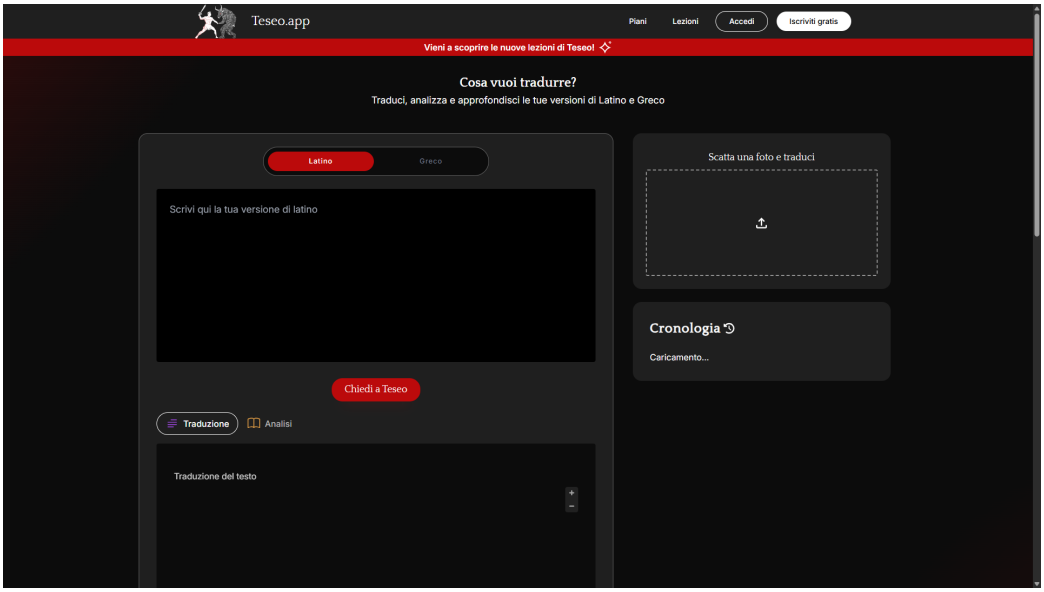

Figura 2.1: Interfaccia della schermata home di Teseo nella versione originale

La colonna laterale destra ospitava funzionalità complementari:

- **Caricamento immagine**: funzionalità che permetteva di fotografare o caricare un'immagine contenente testo latino, da cui il sistema estrapolava il contenuto testuale mediante tecniche di Optical Character Recognition (OCR) prima di procedere all'analisi. Questa feature rispondeva a un'esigenza pratica frequente: gli studenti spesso necessitano di tradurre versioni presenti su libri cartacei e preferiscono evitare la ricopiatura manuale;
- **Cronologia**: elenco delle ultime cinque versioni elaborate dall'utente, con la possibilità di riaprire traduzioni precedenti. Questo permetteva agli studenti di confrontare testi diversi o di riprendere il lavoro su versioni parzialmente completate;

Il workflow di utilizzo tipico si articolava nelle seguenti fasi:

1. L'utente inserisce il testo latino (manualmente o tramite OCR);
2. Il sistema invia il testo al backend per l'elaborazione;

3. Il backend interroga il modello di intelligenza artificiale;

4. I risultati (traduzione e analisi grammaticale) vengono visualizzati nell'area di output;

5. L'elaborazione viene salvata nella cronologia personale dell'utente.

L'applicazione prevedeva due pagine distinte ma funzionalmente identiche per il latino e il greco antico. La separazione rifletteva una scelta progettuale volta a semplificare l'esperienza utente evitando la necessità di selezionare manualmente la lingua di input, che veniva invece implicitamente determinata dalla pagina visitata.

2.2.2 Sistema di gestione crediti e monetizzazione

Data la natura computazionalmente costosa delle operazioni di traduzione e analisi basate su LLM, e considerando i costi associati alle chiamate API ai servizi di AI, era stato implementato un sistema di gestione crediti con le seguenti caratteristiche:

Meccanismo di consumo crediti:

- Ogni operazione di traduzione/analisi consumava un credito specifico per lingua (credito latino o credito greco);

- Ogni immagine caricata per OCR consumava un credito aggiuntivo, indipendentemente dalla lingua del testo estrapolato, riflettendo il costo computazionale del processo di riconoscimento ottico;

- I crediti erano completamente separati tra le due lingue, impedendo di utilizzare i crediti di greco per traduzioni latine e viceversa;

Piano di ricarica e abbonamenti:

- Ogni utente riceveva un numero predefinito di crediti mensili gratuiti;

- I crediti si resettavano ogni trenta giorni a partire dalla data di registrazione individuale dell'utente (non dal primo giorno del mese calendario);
- Era possibile aumentare il massimale mensile sottoscrivendo abbonamenti a pagamento (mensili o annuali) che offrivano quote di crediti più elevate;
- I crediti non utilizzati non venivano accumulati al mese successivo (use-it-or-lose-it policy);

Questo sistema rappresentava un compromesso tra sostenibilità economica della piattaforma e accessibilità per gli studenti, permettendo un utilizzo gratuito per esigenze didattiche moderate mentre richiedeva un contributo per utilizzi intensivi.

2.2.3 Funzionalità secondarie

Oltre alle funzionalità core di traduzione e analisi, la piattaforma offriva tre sezioni aggiuntive accessibili dalla barra di navigazione superiore:

- **Piani**: pagina dedicata alla visualizzazione dei piani di abbonamento disponibili, con confronto dettagliato delle feature offerte da ciascun tier (gratuito, mensile, annuale). Mostrava inoltre il piano corrente dell'utente, i crediti residui e la data del prossimo reset;
- **Lezioni**: sezione contenente materiale didattico complementare sotto forma di video lezioni di grammatica latina e greca. Questi contenuti erano forniti attraverso una partnership con Schoolr, piattaforma specializzata in ripetizioni online. L'integrazione mirava a posizionare Teseo non solo come strumento di traduzione ma come ecosistema completo per l'apprendimento delle lingue classiche;
- **Profilo**: area personale dove l'utente poteva visualizzare le informazioni inserite in fase di registrazione (nome, email), consultare statistiche

di utilizzo (numero totale di traduzioni effettuate) e verificare il numero di traduzioni e analisi residue per il mese corrente.

2.2.4 Tipologie di output generati

L'output prodotto da Teseo in risposta a una richiesta di traduzione comprendeva tipicamente:

- **Traduzione in italiano**: rendering del testo latino/greco in italiano moderno, con attenzione alla naturalezza espressiva pur mantenendo fedeltà al testo originale;
- **Analisi grammaticale**: per ogni parola del testo originale, un oggetto strutturato contenente i seguenti campi:
  - *lemma*: forma base di una parola (come si trova nel dizionario);
  - *categoria grammaticale*: nome, verbo, aggettivo, pronome, avverbio, preposizione, congiunzione, ecc.;
  - *persona grammaticale*: prima, seconda, terza, per verbi e pronomi;
  - *numero grammaticale*: singolare, plurale, duale per il greco;
  - *tempo verbale*: presente, imperfetto, futuro, perfetto, piuccheperfetto, futuro anteriore, ecc.;
  - *modo verbale*: indicativo, congiuntivo, imperativo, infinito, participio, gerundio/gerundivo, ecc.;
  - *diatesi*: attiva, passiva, media per il greco;
  - *genere grammaticale*: maschile, femminile, neutro;
  - *caso*: nominativo, genitivo, dativo, accusativo, ablativo, vocativo, locativo;
  - *grado di comparazione*: positivo, comparativo, superlativo per aggettivi e avverbi;

  - *forza*: per aggettivi greci, tipo di declinazione tematica (forte/debole);
  - *declinazione*: prima, seconda, terza, ecc., per nomi, aggettivi o pronomi;
  - *coniugazione*: prima, seconda, terza, quarta, mista/irregolare per verbi;
  - *significato*: spiegazione testuale della scelta morfologica;

Questo formato di output strutturato, sebbene ricco di informazioni, presentava alcuni problemi:

- **Inconsistenza**: non tutti i campi erano sempre popolati, e il modello AI talvolta ometteva informazioni o utilizzava formati leggermente diversi;
- **Validazione insufficiente**: assenza di schema rigoroso per validare la correttezza dell'output generato dall'AI;

Ma la qualità e il dettaglio di questi output rappresentavano il principale valore aggiunto di Teseo rispetto a traduttori generici, ma dipendevano criticamente dalla qualità dei prompt utilizzati per interrogare i modelli di AI e dalla capacità del sistema di contestualizzare adeguatamente le richieste.

2.3 Architettura iniziale del sistema

2.3.1 Struttura generale

Nella sua configurazione originaria, il sistema Teseo adottava un'architettura client-server tradizionale, articolata su tre componenti principali:

- **Frontend**: applicazione web single-page sviluppata in JavaScript, responsabile dell'interfaccia utente e dell'interazione con l'utente finale;

- **Backend**: server API sviluppato in JavaScript (Node.js), che gestiva la logica applicativa, l'autenticazione, le chiamate ai servizi di AI e la persistenza dei dati;

- **Database**: Firebase Firestore, database NoSQL cloud-hosted fornito da Google, utilizzato per la memorizzazione dei dati utente, della cronologia delle traduzioni e delle configurazioni;

Le tre componenti erano organizzate in repository separate, con deployment indipendente ma coordinato. La Figura 2.2 illustra schematicamente l'architettura originale del sistema.

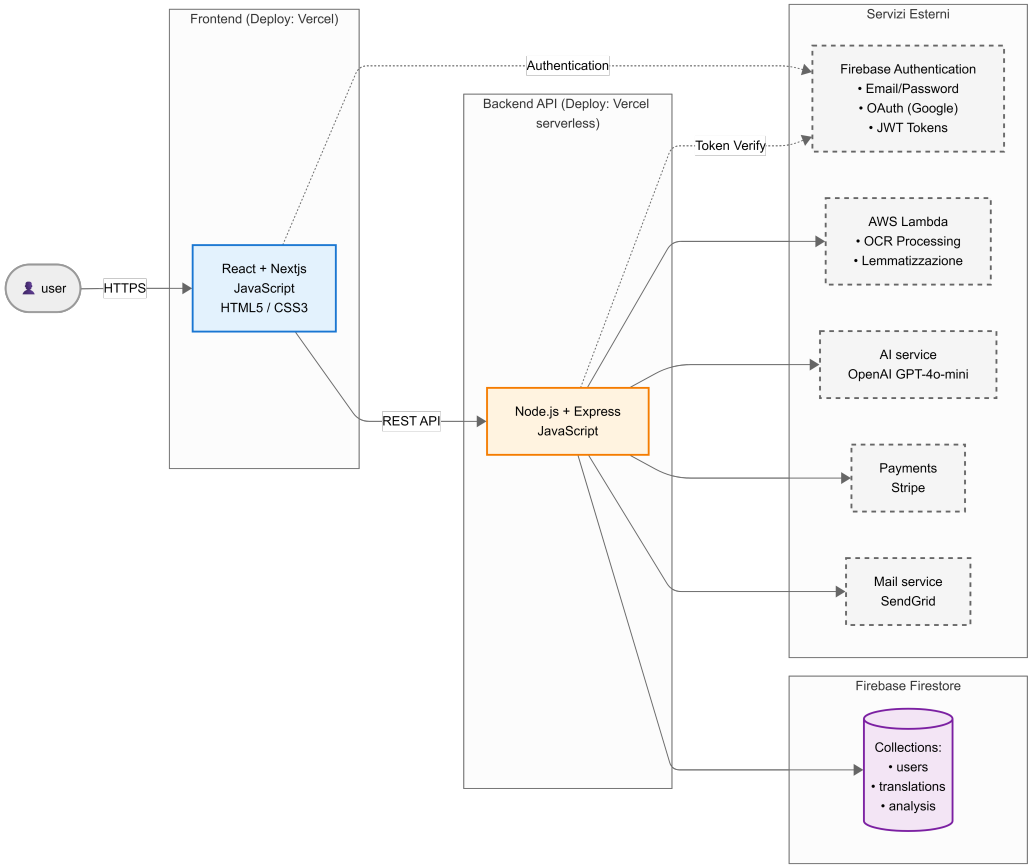

Figura 2.2: Architettura di origine di Teseo

2.3.2 Stack tecnologico

Frontend

Il frontend era implementato utilizzando le seguenti tecnologie:

- **JavaScript (ES6+)**: linguaggio di programmazione utilizzato senza tipizzazione statica, con sintassi moderna ma senza type safety;
- **React**: libreria JavaScript per la costruzione di interfacce utente basate su componenti, utilizzata per gestire lo stato dell'applicazione e il rendering dell'interfaccia;
- **HTML5 e CSS3**: per markup e styling, con utilizzo limitato di preprocessori CSS;
- **REST API**: per comunicazione asincrona con il backend;
- **Firebase Authentication SDK**: per gestione dell'autenticazione lato client.

Il frontend era deployato su **Vercel**, piattaforma di hosting specializzata per applicazioni frontend con supporto nativo per React e deployment automatico da repository Git. Vercel garantiva tempi di caricamento rapidi grazie a una rete CDN globale e ottimizzazioni automatiche per applicazioni React.

Backend

Il backend era costruito su:

- **Node.js**: runtime JavaScript server-side, scelto per omogeneità con il frontend e per l'ampio ecosistema di librerie disponibili;
- **Express.js**: framework web minimale per la gestione di routing e middleware HTTP;

- **Firebase Admin SDK**: per interazioni privilegiate con i servizi Firebase (autenticazione, database, storage);
- **OpenAI API**: SDK ufficiali per l'integrazione con i modelli di linguaggio (OpenAI GPT-4o-mini);
- **AWS Lambda**: funzioni serverless utilizzate per due task specifici:
  - *OCR processing*: elaborazione di immagini caricate dagli utenti per estrazione del testo tramite servizi AWS di computer vision;
  - *Lemmatizzazione*: estrazione di lemmi e analisi morfologica preliminare per arricchire le richieste ai modelli AI;
- **Stripe**: piattaforma di pagamento per gestione abbonamenti, elaborazione pagamenti ricorrenti e one-time, gestione webhook per eventi di pagamento;
- **SendGrid**: servizio di email transazionale per invio di:
  - Email di benvenuto e conferma registrazione;
  - Notifiche di creazione, rinnovo e cancellazione abbonamento;
  - Comunicazioni di servizio agli utenti;

Il backend era deployato su **Vercel** come serverless functions, garantendo scalabilità automatica e deployment semplificato. Le funzioni Lambda AWS operavano come servizi ausiliari invocati dal backend principale tramite chiamate HTTP, creando un'architettura ibrida multi-cloud.

Database e persistenza

Firebase Firestore fungeva da database principale, con le seguenti collezioni:

- **users**: informazioni profilo utente (nome, email, scuola, data registrazione, tipologia di abbonamento, crediti residui, date di rinnovo;);

- **translations**: cronologia traduzioni effettuate, con testo originale, output generato, timestamp;
- **analyses**: cronologia analisi effettuate, con testo originale, output generato in json, timestamp;

La scelta di Firestore rifletteva i vantaggi di un database managed (zero amministrazione, scalabilità automatica, real-time sync) a fronte di limitazioni in termini di query complesse e costi crescenti con il volume di dati.

2.3.3 Flusso di elaborazione

Il flusso di una richiesta di traduzione si articolava nelle seguenti fasi:

1. **Input utente**: l'utente inserisce testo nel frontend (tramite tastiera o caricando un'immagine, che viene processata tramite servizio OCR per estrarne il testo;
2. **Validazione client-side**: il frontend verifica la presenza di crediti residui consultando lo stato locale sincronizzato da Firestore;
3. **Richiesta HTTP**: se i controlli sono superati, il frontend invia richiesta POST al backend includendo il testo da tradurre, lingua target e token di autenticazione;
4. **Autenticazione**: il backend verifica il token Firebase e identifica l'utente;
5. **Verifica crediti**: il backend effettua una verifica server-side dei crediti disponibili (source of truth);
6. **Costruzione prompt**: il backend costruisce due prompt strutturati per il modello AI (uno per l'analisi ed uno per la traduzione), includendo l'input dell'utente e istruzioni specifiche sul formato di output desiderato;

7. **Chiamate API AI**: i prompt vengono inviato al servizio di OpenAI GPT tramite chiamata HTTP in maniera sequenziale;

8. **Parsing risposte**: le risposte del modello vengono parsate ed eventualmente post-processate per garantire consistenza del formato;

9. **Aggiornamento database**:

   - Decremento crediti utente;
   - Salvataggio traduzione in cronologia;

10. **Risposta client**: il backend restituisce l'output al frontend che lo visualizza;

11. **Aggiornamento UI**: il frontend aggiorna la cronologia e il contatore crediti visualizzato;

Questo flusso, pur funzionale, presentava diversi punti di inefficienza e potenziali criticità che verranno analizzati nella sezione successiva.

2.4 Criticità e motivazioni della riprogettazione

L'analisi del sistema esistente ha evidenziato diverse criticità che hanno motivato una riprogettazione sostanziale dell'architettura e dell'implementazione, di seguito descritte.

2.4.1 Limitazioni dell'architettura Firebase

La scelta iniziale di Firebase come piattaforma backend-as-a-service, pur offrendo vantaggi in termini di rapidità di sviluppo e gestione semplificata dell'infrastruttura, presentava limitazioni significative che impedivano l'evoluzione della piattaforma:

- **Vendor lock-in**: la dipendenza stretta dai servizi proprietari di Google rendeva complessa e costosa una eventuale migrazione futura verso altre soluzioni;

- **Limitazioni delle Cloud Functions**: l'esecuzione del backend come Cloud Functions imponeva vincoli stringenti:

  - Cold start significativi (fino a diversi secondi) per funzioni non recentemente utilizzate, degradando l'esperienza utente;
  - Limitata configurabilità delle risorse computazionali (memoria, CPU);

- **Costi crescenti**: il modello di pricing di Firebase Firestore, basato su operazioni di lettura/scrittura e storage, mostrava una crescita non lineare con l'aumento della base utenti. Le operazioni di sincronizzazione real-time, sebbene non strettamente necessarie per il caso d'uso di Teseo, contribuivano a costi evitabili;

- **Difficoltà nel versioning dello schema**: l'assenza di migrazioni strutturate rendeva problematica l'evoluzione dello schema dati, con rischio di inconsistenze tra documenti creati in momenti diversi;

2.4.2 Carenze nella gestione dell'intelligenza artificiale

L'integrazione con i modelli di AI presentava diverse problematiche:

- **Gestione inefficiente delle chiamate API**: ogni richiesta utente generava una singola chiamata sincrona al servizio AI con modelli obsoleti (GPT-4o-mini);

- **Prompt engineering non strutturato**: i prompt inviati ai modelli non seguivano le linee guida odierne: un prompt deve essere corretto, chiaro e ben strutturato;

- **OCR di bassa qualità**: l'utilizzo di AWS Lambda per il processing OCR produceva risultati insoddisfacenti per foto scattate in condizioni di illuminazione non ottimali e la latenza per ricevere il testo estratto era nell'ordine delle decine di secondi;

2.4.3 Problematiche del codice JavaScript

L'utilizzo di JavaScript (senza tipizzazione statica) generava problematiche di manutenibilità e qualità del codice, nonostante l'adozione di React per il frontend:

- **Assenza di type safety**: errori di tipo venivano scoperti solo a runtime, con rischio di bug in produzione che avrebbero potuto essere prevenuti in fase di sviluppo. Props di componenti React non validate, oggetti API con struttura inconsistente e parametri di funzioni ambigui rappresentavano fonti frequenti di errori;

- **Refactoring rischioso**: modifiche al codice esistente erano complicate dall'impossibilità di verificare staticamente che tutti i punti di utilizzo di una funzione o componente fossero aggiornati coerentemente. Il cambio di interfaccia di un componente React o la modifica di una API response richiedeva ricerca manuale di tutte le occorrenze e verifiche manuali;

- **IDE support limitato**: l'autocomplete e la documentazione inline erano significativamente meno efficaci rispetto a quanto possibile con linguaggi tipizzati. Le props dei componenti React, i parametri delle funzioni e le strutture dati restituite dalle API non beneficiavano di IntelliSense[1].accurato;

- **Errori silenti**: problemi come accesso a proprietà undefined, inconsistenze nelle strutture dati tra frontend e backend, o utilizzo scorretto di

[1]Funzionalità di completamento intelligente del codice introdotta da Microsoft negli ambienti di sviluppo integrati (IDE), come Visual Studio Code

API si manifestavano solo a runtime, spesso in produzione dopo essere sfuggiti al testing.

2.4.4 Limitazioni dell'interfaccia utente

L'UI presentava diverse aree di miglioramento:

- **Mancanza di reattività**: alcune operazioni lunghe (caricamento immagini, traduzioni di testi estesi) non fornivano feedback visivo adeguato sullo stato di avanzamento, creando incertezza nell'utente;
- **Visualizzazione output non ottimale**: la presentazione dei risultati era testuale e poco strutturata, rendendo difficile l'identificazione rapida di informazioni specifiche (es. funzione sintattica di una parola particolare);
- **Mobile experience subottimale**: sebbene responsive, l'interfaccia non era ottimizzata per l'uso su smartphone, che rappresentava una percentuale significativa del traffico;

Queste criticità, considerate nel loro complesso, hanno reso evidente la necessità di una riprogettazione sostanziale piuttosto che di interventi incrementali sul sistema esistente. La sezione successiva formalizza i requisiti funzionali e non funzionali che hanno guidato tale riprogettazione.

2.5 Requisiti per la riprogettazione

Sulla base delle criticità identificate e degli obiettivi strategici di evoluzione della piattaforma, sono stati definiti requisiti funzionali e non funzionali che hanno guidato le scelte architetturali e implementative della nuova versione di Teseo.

2.5.1 Requisiti funzionali

RF1: Migrazione a linguaggio tipizzato

Descrizione: L'intera codebase deve essere migrata da JavaScript a TypeScript, introducendo tipizzazione statica completa.

Motivazione: Prevenire errori di tipo a compile-time, migliorare la manutenibilità del codice, facilitare il refactoring e fornire migliore supporto IDE.

Criteri di accettazione:

- Copertura TypeScript al 100% del codice (nessun file .js residuo);
- Strict mode abilitato nella configurazione TypeScript;
- Nessun utilizzo di tipo `any` esplicito senza giustificazione documentata;
- Interfacce TypeScript definite per tutte le entità del dominio, request/response API, e configurazioni.

RF2: Sistema di Content Management

Descrizione: Introdurre un CMS headless che permetta gestione strutturata dei contenuti applicativi senza modifiche al codice.

Motivazione: Separare contenuti da logica applicativa, permettere aggiornamenti rapidi, abilitare gestione da parte di utenti non tecnici.

Criteri di accettazione:

- Configurazione piani di abbonamento via CMS;
- Sistema di permessi per differenziare ruoli amministrativi;
- API REST automaticamente generate dal CMS per accesso ai contenuti;

RF3: Autenticazione modulare

Descrizione: Mantenere Firebase Authentication come provider iniziale ma progettare il sistema in modo che il meccanismo di autenticazione sia sostituibile.

Motivazione: Ridurre il vendor lock-in pur sfruttando l'infrastruttura esistente.

Criteri di accettazione:

- Astrazione del layer di autenticazione tramite interfacce TypeScript;
- Nessuna dipendenza diretta da Firebase SDK nel codice applicativo (solo nel layer di autenticazione);
- Supporto continuo per login email/password e OAuth providers (Google);
- Possibilità di aggiungere nuovi provider senza modifiche al core applicativo.

RF4: Ottimizzazione pipeline AI

Descrizione: Riprogettare completamente il flusso di elaborazione AI introducendo workflow strutturati, OCR avanzato, e prompt engineering sistematico.

Motivazione: Migliorare qualità degli output, ridurre costi, aumentare affidabilità, abilitare personalizzazione.

Criteri di accettazione:

- **OCR migliorato**:
  - Integrazione con servizi OCR avanzati e veloci (Mistral);
  - Accuratezza di riconoscimento almeno del 95% su immagini di qualità media;
- **Workflow AI strutturati**:

  - Utilizzo di framework per orchestrazione workflow (Mastra);
  - Pipeline multi-step;
  - Logging dettagliato di ogni step del workflow per debugging e analytics;

- **Prompt engineering sistematico**:
  - Prompt versionati e memorizzati come configurazioni esterne al codice;
  - Template di prompt parametrizzati (es. variazione drlla lingua da tradurre);
  - Sistema di testing per confrontare l'efficacia di prompt diversi;

- **Gestione robusta delle API AI**:
  - Retry automatico con exponential backoff su errori transitori;
  - Rate limiting applicativo per rispettare quote API;

RF5: Migrazione database

Descrizione: Migrare da Firebase Firestore a database relazionale PostgreSQL, mantenendo integrità dei dati esistenti.

Motivazione: Superare limitazioni di query, ridurre costi, migliorare performance, abilitare analytics complessi.

Criteri di accettazione:

- Schema relazionale normalizzato progettato secondo best practices;
- Script di migrazione per trasferimento dati da Firestore a PostgreSQL;
- Nessuna perdita di dati durante la migrazione;
- Sistema di rollback in caso di problemi critici durante la migrazione;

RF6: Miglioramenti interfaccia utente

Descrizione: Riprogettare l'UI per migliorare usabilità, accessibilità e esperienza utente complessiva.

Motivazione: Ridurre friction nell'uso della piattaforma, aumentare engagement, supportare casi d'uso più complessi.

Criteri di accettazione:

- Feedback visivo su operazioni asincrone (spinner);
- Visualizzazione strutturata dell'output dell'analisi;
- Design responsive ottimizzato per mobile (touch-friendly, layout adattivo);

2.5.2 Requisiti non funzionali

RNF1: Sicurezza

Descrizione: Garantire la protezione dei dati degli utenti e delle comunicazioni attraverso pratiche di sicurezza consolidate a tutti i livelli dell'architettura.

Motivazione: La piattaforma gestisce dati personali di utenti (molti dei quali studenti minorenni), credenziali di accesso e contenuti generati dall'interazione con servizi AI, rendendo la sicurezza un requisito imprescindibile.

Criteri di accettazione:

- Comunicazioni HTTPS obbligatorie con crittografia TLS su tutti gli endpoint;
- Autenticazione basata su token Firebase con validazione server-side su ogni richiesta;
- Gestione dei secrets (API keys, credenziali database, configurazioni sensibili) esclusivamente tramite variabili d'ambiente, senza credenziali hardcoded nel codice sorgente;

- Input validation e sanitization su tutti gli endpoint API esposti;

RNF2: Portabilità e riduzione del vendor lock-in

Descrizione: Progettare l'architettura in modo da minimizzare le dipendenze da fornitori specifici, consentendo la sostituzione di componenti infrastrutturali senza riscritture significative.

Motivazione: L'architettura precedente, fortemente accoppiata all'ecosistema Firebase, rendeva costosa e rischiosa qualsiasi migrazione. La nuova architettura deve evitare di riprodurre lo stesso problema con altri provider.

Criteri di accettazione:

- Nessuna dipendenza diretta da SDK di provider specifici nel codice applicativo core (le integrazioni sono isolate in layer dedicati);
- Database relazionale basato su PostgreSQL, standard aperto e portabile tra diversi provider di hosting;
- Layer di autenticazione astratto, sostituibile senza impatto sulla logica di business;
- Contenuti gestiti tramite CMS headless con API REST standard, disaccoppiato sia dal frontend che dal backend applicativo;

RNF3: Manutenibilità

Descrizione: Garantire che il codice sia facilmente comprensibile, modificabile e estendibile nel tempo, riducendo il costo di evoluzione della piattaforma.

Motivazione: Il progetto è sviluppato da un team ristretto e deve poter evolvere rapidamente. La codebase JavaScript precedente, priva di tipizzazione e con forte accoppiamento tra componenti, rendeva ogni modifica rischiosa e dispendiosa.

Criteri di accettazione:

- Tipizzazione statica completa tramite TypeScript con strict mode abilitato, per prevenzione di errori a compile-time;
- Separazione netta tra layer applicativi (autenticazione, logica di business, accesso ai dati, gestione contenuti);
- Prompt AI versionati e gestiti come configurazione esterna, modificabili senza necessità di rilascio di nuove versioni del codice;
- Struttura modulare che consenta interventi isolati su singoli componenti senza effetti collaterali sul resto del sistema;

RNF4: Usabilità

Descrizione: Offrire un'esperienza utente fluida, intuitiva e accessibile su tutti i dispositivi, con particolare attenzione all'uso da mobile.

Motivazione: Il target primario della piattaforma è composto da studenti che accedono prevalentemente da smartphone. Un'interfaccia poco reattiva o confusa impatta direttamente sull'adozione e sulla retention degli utenti.

Criteri di accettazione:

- Interfaccia responsive e ottimizzata per dispositivi mobile con interazioni touch-friendly;
- Feedback visivo immediato su tutte le operazioni asincrone (spinner, stati di caricamento);
- Tempo di caricamento percepito ridotto tramite lazy loading dei componenti frontend;
- Output dell'analisi AI presentati in formato strutturato e leggibile, non come testo grezzo;

RNF5: Costi

- Costo per utente attivo mensile (MAU): riduzione del 30% rispetto all'architettura Firebase;

- Costo AI per traduzione: ottimizzazione e selezione modelli (target: riduzione 20%);
- Costi infrastrutturali predicibili e monitorabili;

I requisiti elencati hanno guidato le scelte tecnologiche e architetturali descritte nelle sezioni seguenti, bilanciando esigenze di innovazione con vincoli di compatibilità retroattiva e continuità del servizio.

Capitolo 3

Sviluppo e implementazione della nuova architettura

Il presente capitolo descrive le attività di sviluppo e implementazione svolte durante il tirocinio, che hanno portato alla completa riprogettazione dell'architettura di Teseo. Dopo aver identificato nel capitolo precedente le criticità del sistema esistente e i requisiti per la sua evoluzione, vengono ora presentate le soluzioni tecniche adottate, le scelte implementative e i risultati ottenuti attraverso test di performance e analisi dei dati reali di produzione.

3.1 Migrazione di frontend e backend

La riprogettazione del sistema ha richiesto innanzitutto interventi sostanziali sul codice esistente e sull'architettura backend, con l'obiettivo di superare le limitazioni identificate e di porre le fondamenta per un'evoluzione sostenibile della piattaforma.

3.1.1 Migrazione a TypeScript

Innanzitutto, è stata condotta un'attività di migrazione del codice sorgente da JavaScript a TypeScript, che ha interessato sia il frontend che il backend dell'applicazione. Tale intervento si è reso necessario per migliorare signifi-

cativamente la qualità del software, aumentarne la manutenibilità e ridurre la probabilità di errori in fase di sviluppo e di esecuzione.

TypeScript, superset tipizzato di JavaScript, introduce la tipizzazione statica opzionale mantenendo la piena compatibilità con l'ecosistema JavaScript. Il compilatore TypeScript effettua controlli di tipo a compile-time, permettendo di intercettare intere categorie di bug prima dell'esecuzione del codice.

JavaScript, pur essendo un linguaggio flessibile e ampiamente utilizzato nello sviluppo web, presenta una tipizzazione dinamica che può rendere difficile l'individuazione di errori a tempo di sviluppo, specialmente in progetti di dimensioni medio-grandi.

TypeScript, invece, è un superset di JavaScript che introduce la tipizzazione statica e altre funzionalità avanzate, mantenendo al contempo la piena compatibilità con l'ecosistema JavaScript. La migrazione ha permesso di definire in modo esplicito i tipi di variabili, parametri di funzione, valori di ritorno e strutture dati condivise tra frontend e backend, rendendo il codice più chiaro, robusto e auto-documentante.

Dal punto di vista del backend, l'adozione di TypeScript ha consentito una migliore definizione delle interfacce dei servizi, dei modelli di dati e delle API esposte, riducendo errori legati a dati non conformi o a chiamate di funzione errate. Inoltre, il controllo dei tipi in fase di compilazione ha permesso di individuare numerose incongruenze prima dell'esecuzione dell'applicazione, migliorando l'affidabilità complessiva del sistema.

Inoltre, la migrazione ha migliorato l'esperienza di sviluppo del codice in toto grazie a un maggiore supporto da parte degli strumenti di sviluppo (IDE), come l'autocompletamento, la navigazione nel codice e il refactoring assistito.

La migrazione da JavaScript a TypeScript ha rappresentato un passo significativo verso una maggiore qualità del software, contribuendo a rendere l'applicazione più sicura, scalabile e facilmente evolvibile.

3.1.2 Migrazione a Payload CMS

Successivamente, è stata effettuata una migrazione significativa dell'architettura del sistema, che ha coinvolto il passaggio da una soluzione basata su JavaScript e Firebase per la gestione del database e dei servizi backend a una monorepo più strutturata basata su Payload CMS [35]. Questa scelta è stata guidata dall'esigenza di rendere il sistema più flessibile, manutenibile e adatto a evoluzioni future.

Firebase offre una soluzione backend-as-a-service che consente uno sviluppo rapido, ma presenta alcune limitazioni quando l'applicazione cresce in complessità. In particolare, la gestione del database, delle regole di accesso, della logica applicativa e dei dati strutturati può diventare meno chiara e più difficile da mantenere nel lungo periodo. Inoltre, la forte dipendenza da servizi esterni può ridurre il controllo sull'architettura complessiva del sistema.

Payload CMS è un headless CMS open-source basato su Node.js e TypeScript che consente una gestione più esplicita e modulare dei dati e della logica backend. La migrazione ha comportato la ridefinizione dei modelli di dati sotto forma di collezioni e schemi tipizzati, permettendo una rappresentazione più chiara delle entità del dominio applicativo e delle loro relazioni. Questo approccio ha migliorato la coerenza dei dati e ha ridotto il rischio di errori dovuti a strutture non uniformi.

Dal punto di vista architetturale, l'adozione di Payload CMS ha permesso di centralizzare la gestione del backend, includendo validazione dei dati e API generate automaticamente a partire dagli schemi definiti. Questo ha semplificato notevolmente lo sviluppo e la manutenzione del sistema, riducendo la quantità di codice custom necessario rispetto alla soluzione precedente basata su Firebase.

Un ulteriore vantaggio della migrazione è stato il maggiore controllo sul database e sulle operazioni eseguite su di esso. Payload CMS consente l'integrazione diretta con database tradizionali, offrendo una maggiore trasparenza nella gestione delle query, delle transazioni e delle operazioni di persistenza

dei dati, rispetto al modello astratto fornito da Firebase.

Dal punto di vista del frontend, la migrazione ha favorito un'interazione più chiara e tipizzata con il backend, grazie a API ben definite e a una maggiore prevedibilità delle risposte. Questo ha migliorato l'integrazione complessiva del sistema e ha contribuito a ridurre errori di comunicazione tra client e server.

Integrazione con Firebase Authentication

Nonostante la migrazione del backend applicativo, del database e della gestione dei dati verso Payload CMS, si è scelto di mantenere il sistema di autenticazione degli utenti basato su Firebase Authentication. Questa decisione è stata motivata dalla maturità, affidabilità e semplicità di integrazione del servizio di autenticazione offerto da Firebase, già consolidato nel sistema preesistente.

Firebase Authentication fornisce un'infrastruttura completa e sicura per la gestione degli utenti, includendo funzionalità come la gestione delle credenziali, il supporto a diversi provider di autenticazione, la protezione contro accessi non autorizzati e la gestione dei token di sessione. Migrare anche questo componente avrebbe comportato un aumento della complessità senza apportare benefici significativi in termini funzionali.

Di conseguenza, è stata adottata un'architettura ibrida in cui Firebase è responsabile esclusivamente dell'autenticazione, mentre Payload CMS gestisce i dati, il database e la logica applicativa.

Questa scelta ha reso necessaria l'integrazione tra i due sistemi. In particolare, il backend basato su Payload CMS è stato configurato per validare i token di autenticazione emessi da Firebase tramite una strategia custom, garantendo che solo utenti autenticati possano accedere alle risorse protette. In questo modo, l'identità dell'utente viene verificata da Firebase, mentre l'autorizzazione e l'accesso ai dati vengono gestiti a livello applicativo da Payload CMS.

Vantaggi della migrazione

L'adozione di Payload CMS ha prodotto benefici significativi: eliminazione del vendor lock-in di Firebase grazie all'uso di un database standard, type safety end-to-end con generazione automatica dei tipi TypeScript dalle collection, significativa riduzione del codice boilerplate grazie a API REST generate automaticamente, superamento delle limitazioni di query di Firestore (full-text search, join, aggregazioni), admin UI automatica per la gestione dei contenuti, sistema di hook server-side per il controllo granulare della logica di business, gestione nativa delle relazioni tra entità, e consolidamento dell'architettura con riduzione dei servizi esterni da gestire. L'autenticazione resta su Firebase come unica dipendenza residua dall'ecosistema Google.

3.2 Integrazione e ottimizzazione dell'intelligenza artificiale

Una componente centrale del lavoro svolto ha riguardato il ripensamento completo dell'integrazione con i servizi di intelligenza artificiale, con l'obiettivo di migliorare qualità degli output, ridurre latenze, aumentare affidabilità e contenere costi operativi.

3.2.1 Ottimizzazione del riconoscimento ottico dei caratteri

Nel corso del progetto è stata effettuata un'importante ottimizzazione del sistema di Optical Character Recognition (OCR). Inizialmente, il processo di riconoscimento del testo a partire da immagini era demandato a una funzione AWS Lambda, soluzione caratterizzata da tempi di esecuzione relativamente elevati e da una limitata flessibilità in termini di qualità del riconoscimento, in particolare per testi complessi come quelli in lingua latina e greca e prove-

nienti, generalmente, da foto scattate da cellulari con condizioni di luce non sempre ottimale.

Durante il tirocinio, il sistema OCR è stato migrato verso una soluzione basata su MistralAI[36], con risultati significativi su tutti i fronti (i test sono stati effettuati su un campione di 100 immagini, in numero pari tra latino e greco, caricate dagli utenti): la latenza media si è ridotta da 6.2s a 2.4s (-61%), migliorando sensibilmente l'esperienza dell'utente finale, mentre l'accuratezza nel riconoscimento dei caratteri è aumentata dal 94% al 99% per il latino e dall'89% al 96% per il greco, rendendo la soluzione particolarmente adatta al dominio di testi classici. Questo ha consentito di ottenere un input testuale di qualità superiore per le successive fasi di analisi e traduzione basate su modelli di intelligenza artificiale, il tutto con una riduzione dei costi del 35%.

3.2.2 Orchestrazione workflow con Mastra

Un intervento di riprogettazione sostanziale ha riguardato l'implementazione di un workflow strutturato tramite il framework Mastra [37], sostituendo le chiamate API dirette e sincrone con un'architettura orchestrata.

Architettura del workflow

Il nuovo workflow è strutturato in tre step principali (Figura 3.1):

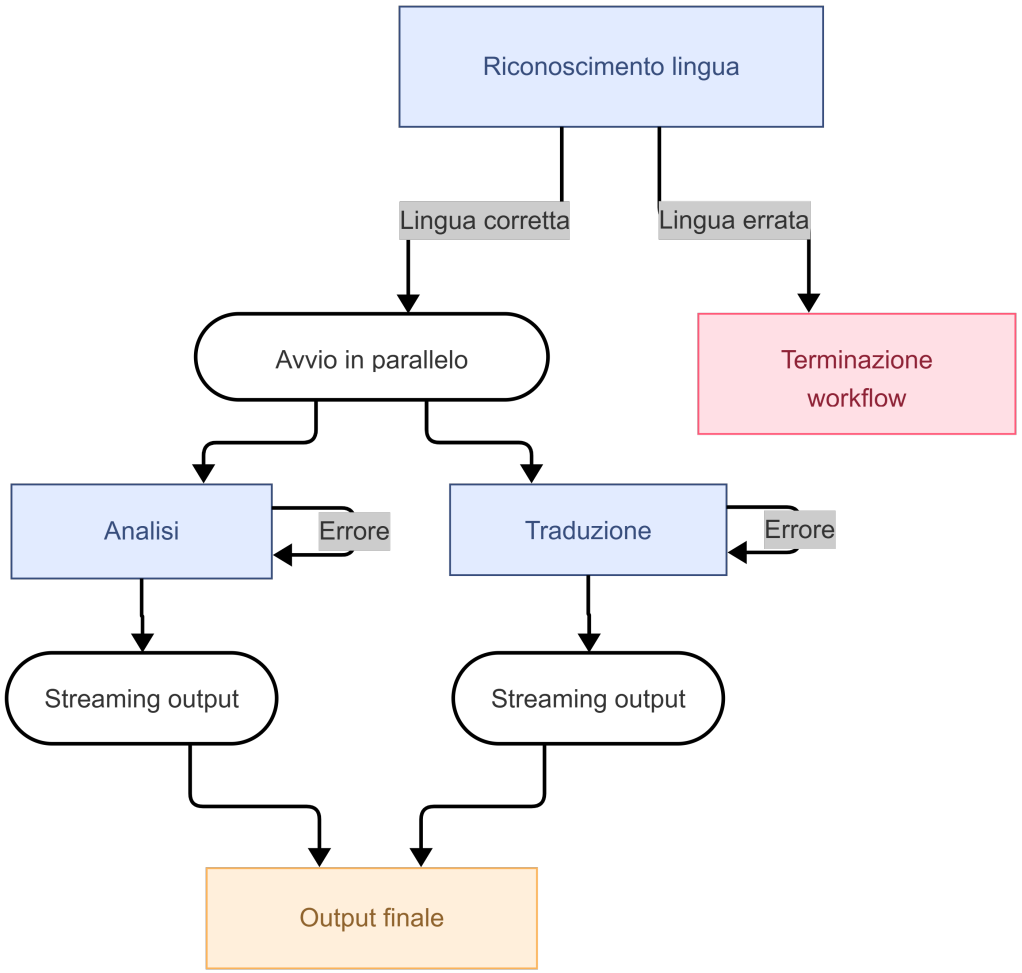

Figura 3.1: Diagramma del workflow di Mastra per processare l'input dell'utente

Step 1: Language Detection - Riconoscimento automatico della lingua (latino o greco), con validazione che sia effettivamente una delle lingue supportate. Se la lingua rilevata non corrisponde a quella dichiarata o risulta sconosciuta, il workflow viene interrotto prevenendo elaborazioni errate e spreco di crediti.

Step 2 e 3: Translation e Analysis (in parallelo) - Una volta validata la lingua, traduzione e analisi grammaticale vengono eseguite in parallelo, dimezzando il tempo totale rispetto all'esecuzione sequenziale. Entrambi gli

step supportano streaming dei risultati in tempo reale tramite Server-Sent Events (SSE). Al termine dello step di analisi, un sistema di verifica controlla la completezza dei risultati, individuando eventuali parole del testo originale non analizzate a causa di errori nel JSON generato dal modello o del raggiungimento del limite massimo di token in output. In tal caso, l'analisi viene automaticamente rieseguita limitatamente alle parole mancanti, garantendo sempre la copertura completa del testo.

Il workflow è configurato con retry automatico (fino a 5 tentativi con exponential backoff) su failure transitori come rate limiting o timeout, aumentando significativamente l'affidabilità percepita.

Streaming in tempo reale

Una delle innovazioni più apprezzate è stata l'introduzione dello streaming progressivo dei risultati. A differenza della versione precedente dove l'utente attendeva diversi secondi vedendo solo un loader, ora traduzione e analisi appaiono progressivamente. L'implementazione utilizza Server-Sent Events (SSE) lato backend e EventSource API lato frontend, permettendo di mostrare risultati parziali con latenza percepita drasticamente ridotta.

Semplificazioni interfaccia

Il rilevamento automatico della lingua ha permesso di rimuovere la pagina /greco; semplificare la gestione crediti unificando i contatori separati per lingua in un unico contatore; fornire feedback immediato in caso di lingua non riconosciuta, evitando così usi inappropriati dell'app.

3.2.3 Prompt Engineering

Un ruolo centrale nell'ottimizzazione delle prestazioni del sistema di intelligenza artificiale è stato svolto dalle attività di prompt engineering. Sono stati sperimentati diversi modelli linguistici, valutandone le prestazioni in termini

di qualità delle traduzioni, accuratezza dell'analisi linguistica, stabilità delle risposte e latenza.

Struttura dei prompt

Parallelamente, sono stati progettati e affinati prompt specifici per il dominio dei testi classici. In particolare, i prompt sono stati strutturati in quattro componenti principali:

1. **System prompt**: definisce ruolo, expertise e principi guida (accuratezza, gestione delle ambiguità, terminologia standard);
2. **Few-shot examples**: 3-5 esempi di input-output per guidare formato e livello di dettaglio;
3. **Task-specific instructions**: istruzioni dettagliate per traduzione o analisi, con specifiche sui campi richiesti;
4. **Output constraints**: vincoli rigorosi sul formato JSON, presenza campi obbligatori;

3.3 Migrazione del database e deployment su infrastruttura cloud

La fase finale ha riguardato la migrazione dei dati a PostgreSQL e il deployment dell'applicazione su infrastruttura AWS scalabile.

3.3.1 Migrazione a PostgreSQL

PostgreSQL [38] è un sistema di gestione di basi di dati relazionali open-source noto per la sua robustezza, conformità agli standard SQL e supporto avanzato a funzionalità come vincoli di integrità, transazioni ACID, relazioni complesse e indici avanzati. La scelta di PostgreSQL è stata motivata da:

supporto nativo di Payload CMS, capacità di query avanzate (join, aggregazioni, full-text search), costi predicibili e scalabilità. L'utilizzo di PostgreSQL in combinazione con Payload CMS ha permesso di definire schemi di dati chiari e coerenti, allineati ai modelli applicativi e alle logiche di business, facilitando la validazione dei dati, la gestione delle relazioni tra entità e l'esecuzione di query complesse in modo efficiente e affidabile.

Lo schema è stato progettato seguendo principi di normalizzazione con tabelle per utenti, traduzioni ed analisi.

Processo di migrazione

La migrazione ha coinvolto oltre 100.000 utenti, 600.000 analisi e altrettante traduzioni, ed è stata eseguita in 3 fasi tramite script implementati manualmente:

1. **Preparazione**: setup del database, sviluppo degli script di migrazione e testing su dati di prova;

2. **Migrazione dati storici**: export bulk da Firestore, trasformazione delle entry e caricamento nel nuovo database. Questa fase è stata eseguita con l'applicazione ancora operativa, migrando tutti i dati accumulati fino a quel momento;

3. **Cutover finale**: l'applicazione è stata messa offline per circa 2 ore tramite pagina di manutenzione, durante le quali sono state migrate le ultime entry generate nel periodo intercorso dalla fase precedente, garantendo che nessun nuovo dato venisse creato durante il trasferimento. Completata la migrazione e verificata l'integrità dei dati, il sistema è stato riavviato sulla nuova infrastruttura.

3.3.2 Deployment su AWS

Il deployment è stato effettuato su Amazon Web Services [39] utilizzando una configurazione moderna e scalabile basata su SST v3 (Ion).

Architettura di deployment

L'architettura di produzione è strutturata come segue:

Componenti AWS:

- **VPC**: Virtual Private Cloud su 2 Availability Zones con subnet private e pubbliche, NAT Gateway per accesso internet in uscita (chiamate API Mistral, Stripe, SendGrid);
- **AWS App Runner**: servizio fully-managed che esegue il container Docker Next.js (2 vCPU, 4 GB RAM). App Runner gestisce automaticamente scaling, health checks, certificati TLS e deployment;
- **Amazon RDS PostgreSQL 17**: database gestito su istanza t4g.micro (development) / t4g.medium (production) in subnet privata, con Multi-AZ deployment, backup automatici giornalieri, encryption at rest e Performance Insights;
- **Amazon ECR**: repository Docker privato che mantiene le ultime 3 immagini del container;
- **VPC Connector**: bridge tra App Runner e VPC per accesso sicuro al database;
- **AWS Secrets Manager**: gestione sicura di credenziali (database password, API keys) con encryption KMS;
- **AWS Certificate Manager**: certificato TLS/SSL per il dominio con auto-renewal;
- **IAM Roles**: ruolo di accesso per pull immagini da ECR, ruolo instance per accesso a Secrets Manager;

Application Runtime:

- Node.js 22 su Alpine Linux containerizzato via Docker;

- Next.js 16 (output: standalone) + PayloadCMS 3;
- Package manager: npm;
- Port: 3000 (esposto da App Runner);

La Figura 3.2 offre una visione d'insieme dell'architettura di produzione, evidenziando le relazioni tra i componenti applicativi, l'infrastruttura AWS e i servizi esterni.

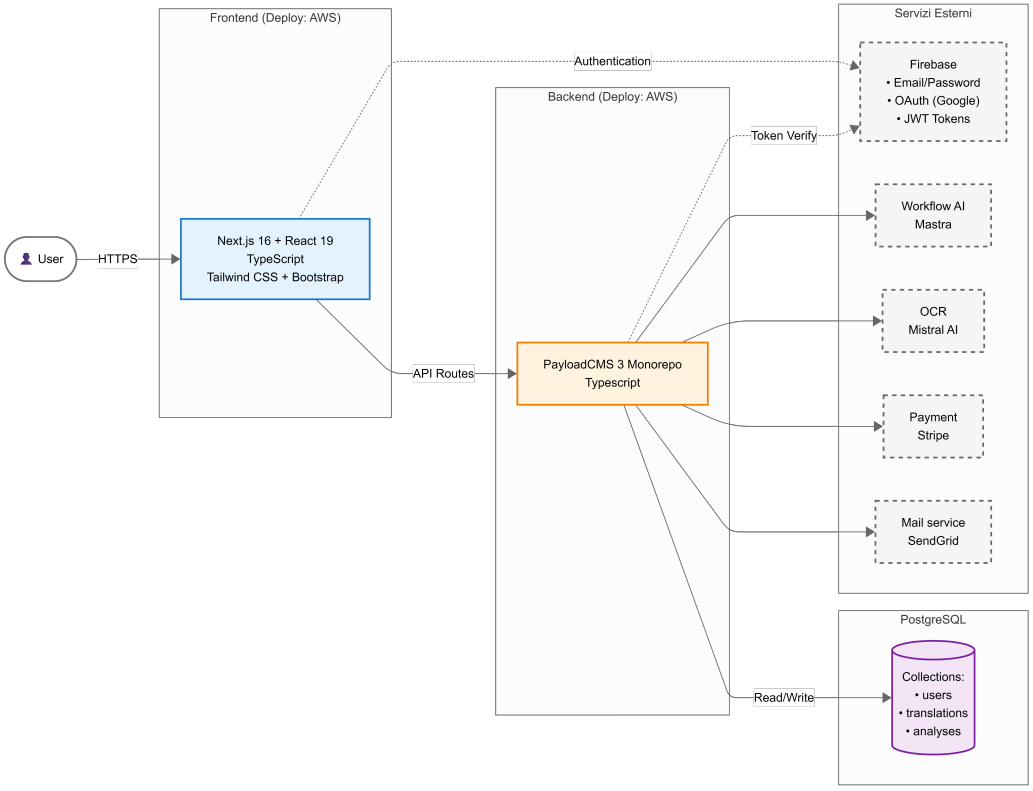

Figura 3.2: Nuova architettura di Teseo

Pipeline di deployment

Il deployment è automatizzato tramite GitHub Actions CI/CD che esegue: build e test del codice, creazione immagine Docker, push su ECR e trigger deployment App Runner. App Runner gestisce automaticamente rolling deployment con health check e rollback su failure.

Vantaggi di App Runner

App Runner offre: auto-scaling basato su richieste concorrenti, zero-downtime deployment, gestione automatica certificati TLS, load balancing integrato, monitoring via CloudWatch, e costi ottimizzati (pay-per-use con nessun addebito quando idle).

3.4 Miglioramenti interfaccia utente tramite AI agentica

Parallelamente alle attività principali di sviluppo backend e di ottimizzazione dell'intelligenza artificiale, sono stati sperimentati alcuni miglioramenti grafici dell'interfaccia utente, con l'obiettivo di valutare possibili evoluzioni dell'esperienza utente dell'applicazione.

Per questa attività è stato utilizzato un approccio sperimentale basato su un agente di intelligenza artificiale, guidato tramite un prompt specifico che descriveva i requisiti grafici e di usabilità dell'interfaccia. L'agente è stato impiegato per proporre modifiche mirate allo stile visivo e all'organizzazione di alcuni elementi dell'interfaccia.

I risultati ottenuti si sono dimostrati complessivamente positivi: le modifiche suggerite hanno migliorato la chiarezza visiva e la fruibilità dell'applicazione senza alterare in modo significativo il suo stile, per non creare disorientamento negli utenti che già la utilizzavano. In seguito a una valutazione manuale, i miglioramenti ritenuti più efficaci sono stati mantenuti e integrati stabilmente nel progetto, vedi 3.3.

Questa attività ha permesso di esplorare in modo pratico l'utilizzo di tecniche di intelligenza artificiale agentica anche in ambiti non strettamente legati alla logica applicativa, evidenziandone il potenziale come strumento di supporto allo sviluppo e al design dell'interfaccia utente.

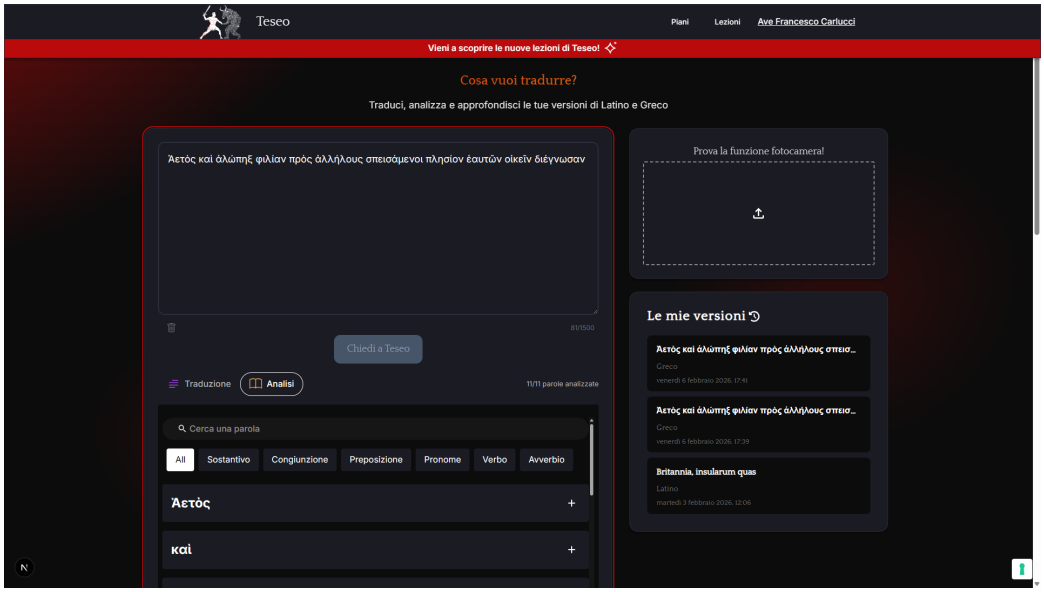

Figura 3.3: Nuova schermata home di Teseo

3.5 Valutazione dei risultati

Al fine di identificare i modelli di intelligenza artificiale più adatti alle esigenze dell'applicazione, è stata condotta una campagna di valutazione sistematica sui tre task fondamentali dell'applicazione: rilevamento della lingua, traduzione e analisi grammaticale. I test sono stati eseguiti su un totale di 13 modelli della famiglia OpenAI, incluse varianti con reasoning effort basso per i modelli che lo supportano (indicati con il suffisso *low*) o prive di suffisso nel caso di reasoning assente, per valutare l'impatto di questo parametro sulle prestazioni. L'ambiente di test utilizzava istanze non di produzione, con ogni test ripetuto 3 volte utilizzando la media come valore rappresentativo, al fine di ridurre l'impatto di variazioni casuali nelle risposte dei modelli e nella latenza di rete.

Per la selezione del modello da utilizzare in produzione sono stati considerati tre criteri principali, in ordine di priorità: accuratezza dell'output (misurata tramite BLEU score per la traduzione e percentuale di correttezza per il rilevamento della lingua), latenza media di risposta e costo per ri-

chiesta. L'obiettivo è stato individuare il miglior compromesso tra qualità e sostenibilità operativa, privilegiando l'accuratezza come requisito primario e utilizzando latenza e costo come fattori discriminanti tra modelli con prestazioni qualitative comparabili.

3.5.1 Dataset di valutazione

Il dataset di test è stato strutturato in due sottoinsiemi distinti, ciascuno progettato per valutare uno specifico aspetto del sistema.

Dataset per il rilevamento della lingua

Per la valutazione dello step di rilevamento della lingua sono stati predisposti 20 campioni di test, suddivisi in quattro categorie:

- 5 brani in latino;
- 5 brani in greco antico;
- 5 brani in altre lingue (italiano, inglese), per verificare che il sistema rigetti correttamente testi in lingue non supportate;
- 5 campioni contenenti tentativi di prompt injection [1];

Dataset per traduzione e analisi

Per la valutazione della traduzione sono stati utilizzati 20 brani reali di autori classici (10 in latino e 10 in greco antico), selezionati per rappresentare diverse tipologie di complessità linguistica. I testi sono stati suddivisi in tre categorie in base alla lunghezza e alla complessità sintattica:

- **Testi brevi**: frasi singole o periodi semplici, rappresentativi dell'uso più comune dell'applicazione da parte degli studenti;

[1] Input progettati per indurre il modello a ignorare le istruzioni di sistema e produrre output non previsti. Questi test verificano la robustezza del modello nel mantenere il comportamento atteso anche in presenza di input malevoli.

- **Testi lunghi**: paragrafi estesi con strutture sintattiche articolate;
- **Testi complessi**: brani con costrutti sintattici avanzati, lessico raro e ambiguità interpretative.

Per la valutazione della qualità delle traduzioni è stata adottata la metrica BLEU (Bilingual Evaluation Understudy), uno standard consolidato nella valutazione automatica delle traduzioni automatiche. Lo score BLEU confronta la traduzione generata dal modello con una o più traduzioni di riferimento, misurando la sovrapposizione di n-grammi su una scala da 0 a 100, dove valori più alti indicano maggiore corrispondenza con il riferimento.

3.5.2 Rilevamento della lingua

Il primo step del workflow è il rilevamento automatico della lingua del testo in ingresso. Questo task, pur essendo concettualmente semplice, è critico per il corretto funzionamento dell'intera pipeline: un errore in questa fase comporta l'avvio di traduzioni e analisi con parametri errati, con conseguente spreco di risorse computazionali e crediti.

Analisi dei risultati ottenuti

La Tabella A.1 riporta i risultati del benchmark di rilevamento della lingua su 20 campioni di test evidenziando che il rilevamento della lingua è un task accessibile alla quasi totalità dei modelli testati: 11 modelli su 13 raggiungono il 100% di accuratezza, classificando correttamente tutte le lingue supportate, rigettando le lingue non supportate e resistendo ai tentativi di prompt injection. Le uniche eccezioni sono gpt-4.1, che classifica erroneamente 1 campione su 20 (95%), e gpt-4.1-nano, che sbaglia 4 campioni su 20 (80%). È significativo che tutti i modelli testati abbiano gestito correttamente i campioni di prompt injection, restituendo in ogni caso una classificazione di lingua non supportata.

Concentrando l'analisi sugli 11 modelli con accuratezza perfetta, la Figura 3.4 evidenzia come latenza e costo non siano correlati linearmente. I modelli

della generazione 4 (gpt-4o-mini, gpt-4o, gpt-4.1-mini) e gpt-5-chat-latest si collocano nella zona ottimale del grafico (data da: latenza massima di 1 secondo e costo inferiore a 0,005$ per il totale delle richieste del test), con latenze inferiori a 800 ms e costi contenuti. Al contrario, i modelli della famiglia gpt-5 con capacità di ragionamento esteso (gpt-5, gpt-5-nano, gpt-5-mini) presentano latenze comprese tra 4 e 7 secondi, un comportamento riconducibile alla generazione interna di una catena di pensiero prima della risposta, un meccanismo sovradimensionato per un task di classificazione come il rilevamento della lingua.

Sul piano dei costi, gpt-5 rappresenta un caso emblematico: con $0,2394 per l'esecuzione di tutte le richieste dei test risulta quasi 200 volte più costoso di gpt-4o-mini ($0,0013), pur offrendo la stessa accuratezza e una latenza 10 volte superiore. Anche all'interno della stessa fascia di latenza, le differenze di costo sono significative: gpt-4o-mini e gpt-4o rispondono in tempi simili (694 ms vs 708 ms), ma gpt-4o costa 17 volte di più.

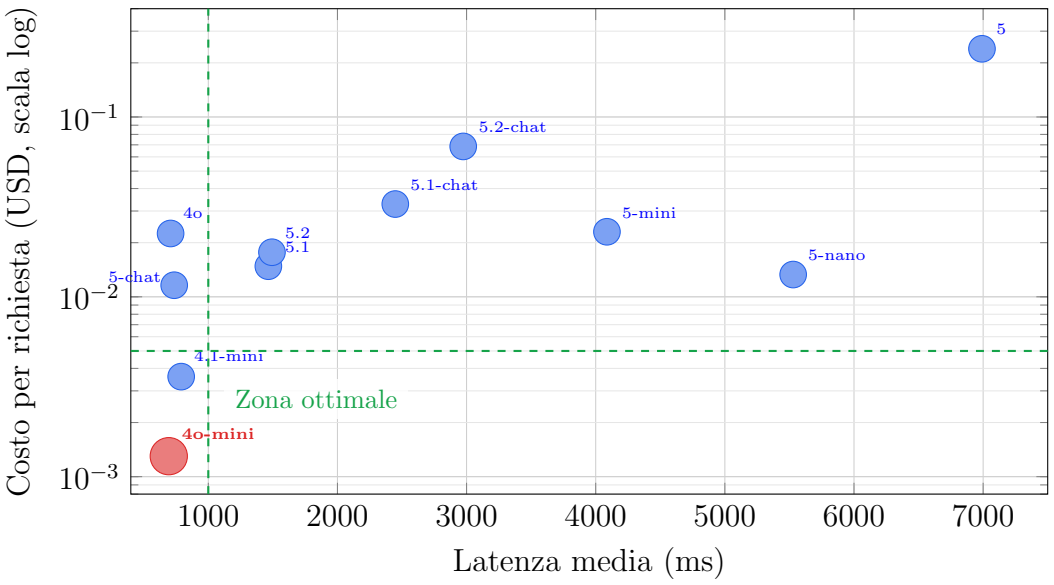

Figura 3.4: Rapporto costo-latenza per i modelli con accuratezza 100% nel rilevamento della lingua

Scelta del modello

Per lo step di rilevamento della lingua, gpt-4o-mini rappresenta la scelta ottimale: come evidenziato dalla Figura 3.4, è l'unico modello a collocarsi nell'angolo inferiore sinistro del grafico, combinando il 100% di accuratezza con la latenza più bassa (694 ms) e il costo minimo ($0,0013). L'utilizzo di modelli più potenti per questo task non apporta alcun beneficio in termini di accuratezza, ma introduce penalità significative in latenza e costi.

3.5.3 Traduzione

La traduzione rappresenta un task più impegnativo rispetto al precdente, richiedendo ai modelli di comprendere strutture sintattiche complesse delle lingue classiche e produrre traduzioni in italiano di qualità. I test sono stati condotti su 20 brani di di difficoltà crescente, sia per il latino che per il greco antico, valutando la qualità tramite punteggio BLEU calcolato rispetto a traduzioni ufficiali tratte da edizioni accademiche consolidate dei brani selezionati. I dati completi dei test effettuati sono riportati in Appendice A.2 A.3 A.4.

Risultati

La Figura 3.5 riporta i punteggi BLEU medi ottenuti da ciascun modello sui testi lunghi, ordinati per prestazione sul latino.

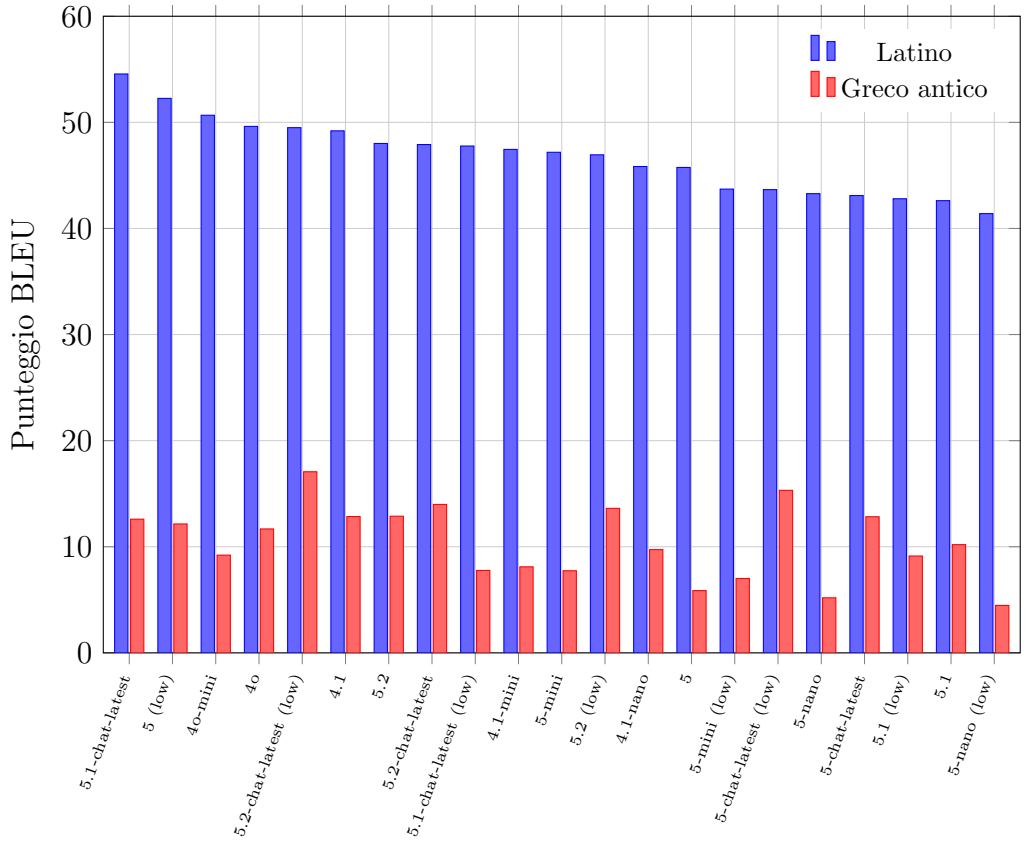

Figura 3.5: Punteggi BLEU per la traduzione di testi: confronto latino vs greco antico per tutti i modelli testati, ordinati per prestazione sul latino

Il dato più evidente è il divario sistematico tra latino e greco antico, che si manifesta in modo consistente su tutti i modelli testati. Per il latino, i punteggi BLEU oscillano tra 41 e 55, un intervallo che indica traduzioni di qualità accettabile per un uso didattico. Per il greco antico, i punteggi raramente superano il valore di 17, con la maggioranza dei modelli al di sotto di 13. Questa differenza è attribuibile alla minore rappresentazione del greco antico nei corpora di addestramento dei modelli e alla maggiore complessità morfologica della lingua greca rispetto al latino, coerentemente con quanto riportato nella letteratura sulla traduzione automatica delle lingue classiche.

Per il latino, gpt-5.1-chat-latest ottiene il punteggio più alto (54,56), seguito da gpt-5 con reasoning effort ridotto (52,26) e gpt-4o-mini (50,67). È

significativo che modelli della generazione 4 come gpt-4o-mini e gpt-4o si collochino nelle prime posizioni, con prestazioni competitive rispetto ai modelli più recenti e costosi. Per il greco, la classifica si ricompone in modo diverso: gpt-5.2-chat-latest con reasoning effort ridotto ottiene il punteggio più alto (17,07), seguito da gpt-5-chat-latest con reasoning effort ridotto (15,32), suggerendo che i modelli della famiglia 5.2 gestiscono meglio il greco antico rispetto alle generazioni precedenti.

Analisi costo-efficacia

Per valutare la sostenibilità economica dei modelli in produzione, i test sui testi complessi sono stati analizzati anche in termini di tempo totale di esecuzione del test e costo complessivo. La Figura 3.6 riporta questi risultati.

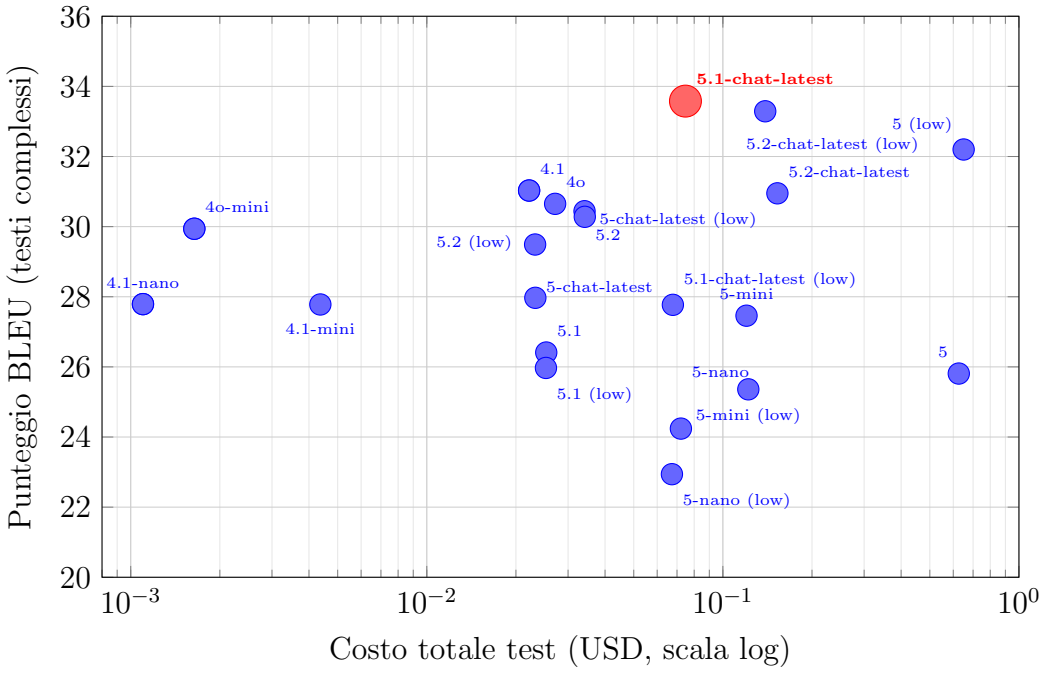

Figura 3.6: Rapporto costo-qualità per la traduzione di testi

L'analisi costo-efficacia rivela che non esiste una correlazione diretta tra costo del modello e qualità della traduzione. Come evidenziato dalla Figura 3.6, gpt-5, il modello più costoso testato ($0,6263), si posiziona nelle ultime

posizioni della classifica (BLEU 25,81), mentre gpt-4o-mini, con un costo 400 volte inferiore ($0,0016), raggiunge un punteggio di 29,94. I modelli con il miglior rapporto qualità-prezzo risultano gpt-4o-mini e gpt-4.1, che combinano punteggi competitivi con costi contenuti.

Sul piano dei tempi di elaborazione (Figura 3.7), emerge una separazione tra alcuni modelli della generazione 5, più lenti a causa del reasoning, e quelli della generazione 4, sempre più veloci.

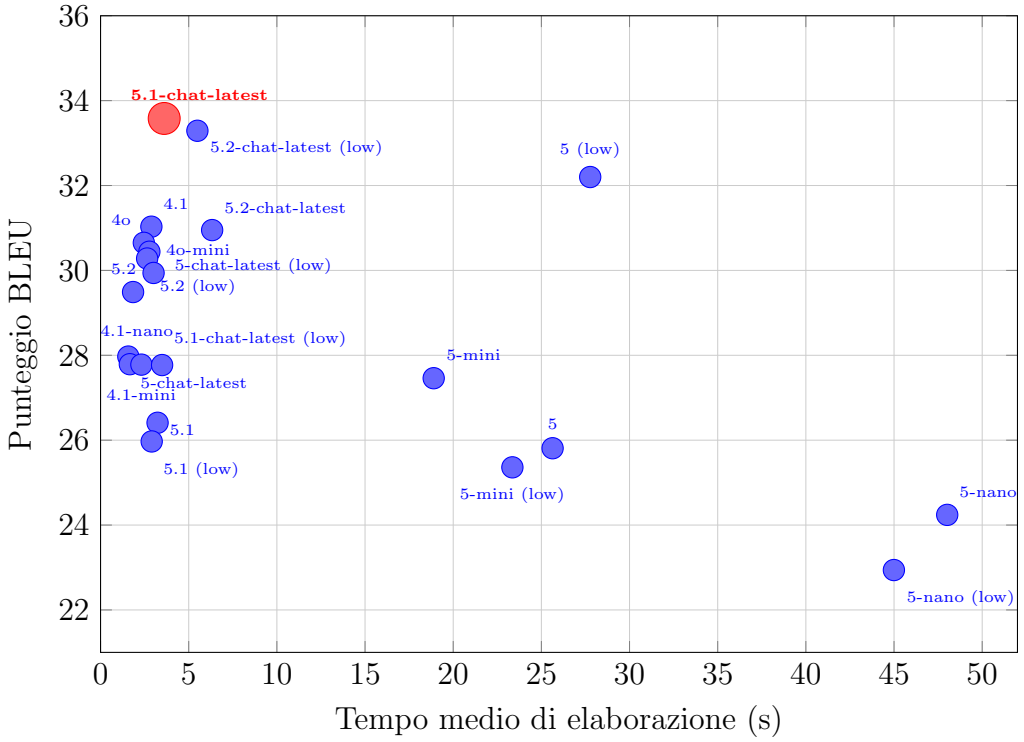

Figura 3.7: Tempo medio di elaborazione per traduzione in funzione del punteggio BLEU

Scelta del modello.

Per lo step di traduzione, gpt-5.1-chat-latest è stato selezionato per la combinazione di miglior punteggio BLEU complessivo, tempo di elaborazione medio contenuto (3,6 s) e costo ragionevole ($0,0748). Rispetto a gpt-4o-mini, il costo aggiuntivo ($0,073 in più per esecuzione) è giustificato dal guadagno

di quasi 4 punti BLEU, un margine significativo per la qualità percepita delle traduzioni in un contesto didattico. Per il greco antico, invece, gpt-5.2-chat-latest con reasoning effort low ottiene il punteggio più alto (17,07 vs 12,60 di gpt-5.1-chat-latest), quindi è stato scelto nonostante un costo leggermente maggiore.

3.5.4 Analisi grammaticale

L'analisi grammaticale è il secondo, e più impegnativo, task eseguito in parallelo alla traduzione nel workflow. Il modello deve produrre un output strutturato in formato JSON contenente, per ogni parola del testo, il lemma, l'analisi morfologica completa e la classificazione grammaticale. A differenza della traduzione, per questo task non è stato possibile condurre una valutazione qualitativa automatizzata, a causa della complessità nella definizione di metriche di riferimento per l'analisi morfologica di lingue classiche e dell'assenza di dataset annotati compatibili con il formato di output del sistema. La correttezza dell'output è stata pertanto verificata manualmente su tutti i modelli testati, riscontrando risultati soddisfacenti per ciascuno di essi. Inoltre, è stata implementata una verifica automatica sulla struttura dell'output JSON, per garantire che il formato rispetti lo schema atteso dal frontend indipendentemente dal modello utilizzato.

La valutazione si è quindi concentrata sulle metriche di tempo di elaborazione e costo, utilizzando gli stessi testi e modelli impiegati per i benchmark di traduzione.

Analisi dei risultati

La Tabella A.5 riporta il tempo medio per singola analisi e il costo totale proiettato sul dataset completo, ordinati per tempo crescente, inoltre, la Figura 3.8 mostra il rapporto tra tempo medio per analisi e costo per ciascun modello.

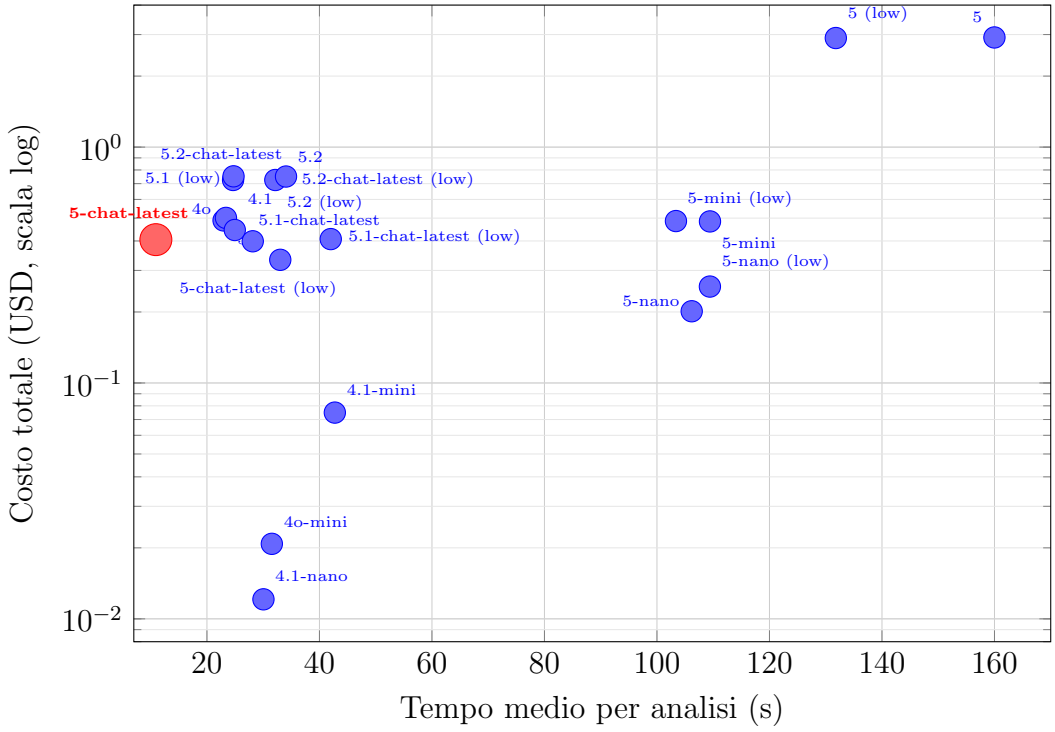

Figura 3.8: Rapporto tempo-costo per l'analisi grammaticale

I risultati confermano il pattern osservato nei benchmark di traduzione, con una separazione netta tra modelli standard e modelli di ragionamento. I modelli standard producono un'analisi in tempi compresi tra 11 e 43 secondi, mentre i modelli di ragionamento richiedono tra 103 e 160 secondi. Il modello più veloce è gpt-5-chat-latest (circa 11 s per analisi), mentre gpt-5 è il più lento (160 s) e il più costoso ($2,92).

Scelta del modello.

Per lo step di analisi grammaticale è stato selezionato gpt-5-chat-latest. Questa scelta è motivata da due fattori principali: la verifica manuale ha confermato la qualità dell'output prodotto e la latenza media, che compensano il costo aggiuntivo rispetto a modelli più economici.

3.5.5 Scelta dei modelli per la produzione

Sulla base dei risultati ottenuti, sono stati selezionati i modelli per ciascuno step del workflow di produzione, bilanciando qualità, latenza e costi, come mostrato in tabella 3.1.

Tabella 3.1: Modelli selezionati per il workflow di produzione

Step	**Modello**	**Motivazione**
Language Detection	gpt-4o-mini	100% accuratezza, latenza minima, basso costo
Traduzione	gpt-5.1-chat-latest	Miglior BLEU complessivo, buon bilanciamento tempi/-costi
Analisi grammaticale	gpt-5-chat-latest	Latenza minima, output corretto, prezzo contenuto

3.5.6 Limitazioni della valutazione

La campagna di test presenta alcune limitazioni che è opportuno esplicitare. Il numero di campioni di test (10 per lingua) è sufficiente per identificare tendenze generali ma non consente un'analisi statisticamente robusta delle differenze tra modelli con punteggi ravvicinati. La metrica BLEU, pur essendo uno standard consolidato, presenta limiti noti nella valutazione delle traduzioni da lingue classiche: traduzioni semanticamente corrette ma con scelte lessicali diverse dal riferimento possono ricevere punteggi bassi. L'assenza di una valutazione qualitativa dell'analisi grammaticale rappresenta un punto aperto che andrà affrontato in future iterazioni. Infine, i benchmark sono stati condotti su istanze non di produzione; le prestazioni in ambiente di produzione potrebbero variare, in particolare per quanto riguarda la latenza.

Conclusioni e sviluppi futuri

Il lavoro presentato in questa tesi ha documentato la riprogettazione completa di Teseo, un'applicazione web AI-based per la traduzione e l'analisi di testi in latino e greco antico, sviluppata durante il tirocinio presso Memoraiz.

L'intervento ha interessato tutti i livelli della piattaforma. La migrazione a TypeScript ha introdotto tipizzazione statica completa, migliorando manutenibilità e robustezza del codice. L'adozione di Payload CMS e la migrazione del database a PostgreSQL hanno ridotto il vendor lock-in legato a Firebase, centralizzato la gestione del backend e reso i costi infrastrutturali più predicibili. Il deployment su AWS con App Runner ha fornito un'architettura scalabile con zero-downtime.

Sul fronte dell'intelligenza artificiale, il sistema OCR migrato a MistralAI ha ridotto la latenza del 61% raggiungendo un'accuratezza del 99% per il latino e del 96% per il greco. L'orchestrazione tramite Mastra ha strutturato il flusso in un workflow con rilevamento automatico della lingua, esecuzione parallela di traduzione e analisi, streaming progressivo e retry automatico. La campagna di valutazione su 13 modelli OpenAI ha guidato la selezione dei modelli di produzione, identificando gpt-4o-mini per il rilevamento della lingua, gpt-5.1-chat-latest per la traduzione e gpt-5-chat-latest per l'analisi grammaticale, bilanciando qualità, latenza e costi.

Poiché la nuova versione è stata rilasciata in produzione da poco tempo, non è ancora possibile disporre di feedback strutturati da parte degli utenti né di dati quantitativi sull'impatto delle modifiche. La validazione dell'efficacia delle scelte progettuali in condizioni reali di utilizzo rappresenta quindi

la fase successiva, che verrà realizzata attraverso l'analisi delle metriche di produzione e indagini qualitative rivolte agli utenti.

Sviluppi futuri

Il lavoro apre diverse direzioni di evoluzione. Sul piano della valutazione, l'adozione di metriche semantiche più sofisticate (BERTScore, COMET) e la costruzione di dataset annotati da esperti consentirebbero di superare i limiti noti della metrica BLEU per le lingue classiche e di valutare accuratamente i risultati prodotti dai vari modelli per l'analisi grammaticale. Sul piano funzionale, l'integrazione di tecniche RAG con dizionari e grammatiche di riferimento e l'introduzione di un'analisi sintattica strutturata permetterebbero di supportare in modo più completo il workflow filologico tradizionale. Infine, l'evoluzione verso un'architettura multi-provider per i modelli AI garantirebbe maggiore resilienza e flessibilità.

In conclusione, il lavoro svolto ha trasformato Teseo da un prototipo funzionale ma limitato a una piattaforma con un'architettura moderna, modulare e predisposta per evoluzioni future, ponendo le fondamenta per una crescita sostenibile sia in termini tecnici che di prodotto.

Appendice A

Dati completi dei benchmark

A.1 Rilevamento della lingua

Tabella A.1: Risultati del benchmark di rilevamento della lingua (20 campioni)

Modello	**Accuratezza**	**Latenza media**	**Costo totale**
gpt-4o-mini	100%	694 ms	$0,0013
gpt-4o	100%	708 ms	$0,0225
gpt-4.1-mini	100%	790 ms	$0,0036
gpt-5-chat-latest	100%	736 ms	$0,0116
gpt-5.1	100%	1465 ms	$0,0148
gpt-5.2	100%	1493 ms	$0,0177
gpt-5.1-chat-latest	100%	2448 ms	$0,0328
gpt-5.2-chat-latest	100%	2974 ms	$0,0686
gpt-5-mini	100%	4087 ms	$0,0230
gpt-5-nano	100%	5529 ms	$0,0133
gpt-5	100%	6991 ms	$0,2394
gpt-4.1	95%	846 ms	$0,0180
gpt-4.1-nano	80%	577 ms	$0,0009

A.2 Traduzione

Test effettuati su 20 campioni reali (10 in latino e 10 in greco) di autori classici.

A.2.1 BLEU score per testi in latino

Tabella A.2: BLEU score per la traduzione dal latino, ordinati per punteggio decrescente

Modello	**BLEU medio latino**
gpt-5.1-chat-latest	54,56
gpt-5 (low)	52,26
gpt-4o-mini	50,67
gpt-4o	49,62
gpt-5.2-chat-latest (low)	49,50
gpt-4.1	49,20
gpt-5.2	48,01
gpt-5.2-chat-latest	47,91
gpt-5.1-chat-latest (low)	47,77
gpt-4.1-mini	47,45
gpt-5-mini	47,18
gpt-5.2 (low)	46,94
gpt-4.1-nano	45,84
gpt-5	45,75
gpt-5-mini (low)	43,71
gpt-5-chat-latest (low)	43,66
gpt-5-nano	43,28
gpt-5-chat-latest	43,10
gpt-5.1 (low)	42,80
gpt-5.1	42,62
gpt-5-nano (low)	41,40

A.2.2 BLEU score per testi in greco antico

Tabella A.3: BLEU score per la traduzione dal greco antico, ordinati per punteggio decrescente

Modello	**BLEU medio greco**
gpt-5.2-chat-latest (low)	17,07
gpt-5-chat-latest (low)	15,32
gpt-5.2-chat-latest	13,99
gpt-5.2 (low)	13,62
gpt-5.2	12,88
gpt-4.1	12,85
gpt-5-chat-latest	12,83
gpt-5.1-chat-latest	12,60
gpt-5 (low)	12,15
gpt-4o	11,68
gpt-5.1	10,20
gpt-4.1-nano	9,73
gpt-4o-mini	9,21
gpt-5.1 (low)	9,13
gpt-4.1-mini	8,11
gpt-5.1-chat-latest (low)	7,77
gpt-5-mini	7,74
gpt-5-mini (low)	7,01
gpt-5	5,87
gpt-5-nano	5,19
gpt-5-nano (low)	4,47

A.2.3 BLEU score, tempi e costi per ciascun modello

Tabella A.4: BLEU score, latenza e costi per la traduzione

Modello	BLEU medio	Latenza media	Costo totale
gpt-5.1-chat-latest	33,58	3609 ms	$0,0748
gpt-5.2-chat-latest (low)	33,29	5492 ms	$0,1390
gpt-5 (low)	32,20	27769 ms	$0,6504
gpt-4.1	31,03	2882 ms	$0,0222
gpt-5.2-chat-latest	30,95	6328 ms	$0,1528
gpt-4o	30,65	2454 ms	$0,0271
gpt-5.2	30,44	2768 ms	$0,0341
gpt-5.2 (low)	30,28	2641 ms	$0,0342
gpt-4o-mini	29,94	3006 ms	$0,0016
gpt-5-chat-latest (low)	29,49	1844 ms	$0,0232
gpt-5-chat-latest	27,97	1574 ms	$0,0233
gpt-4.1-nano	27,79	1661 ms	$0,0011
gpt-4.1-mini	27,78	2316 ms	$0,0044
gpt-5.1-chat-latest (low)	27,77	3490 ms	$0,0678
gpt-5-mini	27,46	18893 ms	$0,1202
gpt-5.1	26,41	3233 ms	$0,0253
gpt-5.1 (low)	25,97	2900 ms	$0,0253
gpt-5	25,81	25636 ms	$0,6263
gpt-5-mini (low)	25,36	23356 ms	$0,1219
gpt-5-nano	24,24	48016 ms	$0,0722
gpt-5-nano (low)	22,94	44992 ms	$0,0673

A.3 Analisi grammaticale

Test effettuati su 20 campioni reali (10 in latino e 10 in greco) di autori classici.

Tabella A.5: Tempo medio per analisi e costo totale dei test sull'analisi grammaticale

Modello	**Latenza media**	**Costo totale**
gpt-5-chat-latest	10,920 s	$0,4052
gpt-5-chat-latest (low)	11,110 s	$0,4032
gpt-5.1 (low)	22,955 s	$0,4888
gpt-5.1	23,380 s	$0,5010
gpt-5.2-chat-latest	24,645 s	$0,7255
gpt-5.2-chat-latest (low)	24,725 s	$0,7519
gpt-4o	24,950 s	$0,4452
gpt-4.1	28,145 s	$0,3988
gpt-4.1-nano	30,045 s	$0,0121
gpt-4o-mini	31,535 s	$0,0208
gpt-5.2 (low)	32,155 s	$0,7254
gpt-5.1-chat-latest (low)	33,035 s	$0,3328
gpt-5.2	34,055 s	$0,7504
gpt-5.1-chat-latest	42,010 s	$0,4074
gpt-4.1-mini	42,735 s	$0,0749
gpt-5-mini (low)	103,375 s	$0,4858
gpt-5-nano	106,175 s	$0,2014
gpt-5-nano (low)	109,410 s	$0,2563
gpt-5-mini	109,425 s	$0,4841
gpt-5 (low)	131,795 s	$2,9000
gpt-5	159,990 s	$2,9163

Bibliografia

[1] "Memoraiz website." `https://www.memoraiz.com/`.

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[3] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[4] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," 2018.

[5] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[6] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[7] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *The Journal of Machine Learning Research*, vol. 21, no. 1, pp. 5485–5551, 2020.

[8] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, *et al.*, "Training language models to follow instructions with human feedback," *Advances in Neural Information Processing Systems*, vol. 35, pp. 27730–27744, 2022.

[9] OpenAI, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[10] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[11] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[12] Y. Bai, S. Kadavath, S. Kundu, A. Askell, J. Kernion, A. Jones, A. Chen, A. Goldie, A. Mirhoseini, C. McKinnon, *et al.*, "Constitutional ai: Harmlessness from ai feedback," *arXiv preprint arXiv:2212.08073*, 2022.

[13] Gemini Team, "Gemini: a family of highly capable multimodal models," *arXiv preprint arXiv:2312.11805*, 2023.

[14] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, "Scaling laws for neural language models," *arXiv preprint arXiv:2001.08361*, 2020.

[15] J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler, *et al.*, "Emergent abilities of large language models," *arXiv preprint arXiv:2206.07682*, 2022.

[16] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou, *et al.*, "Chain-of-thought prompting elicits reasoning in large

language models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 24824–24837, 2022.

[17] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners." `https://arxiv.org/abs/2005.14165`, 2020.

[18] B. Settles *et al.*, "Second language acquisition modeling," *Proceedings of the Thirteenth Workshop on Innovative Use of NLP for Building Educational Applications*, 2018.

[19] C. Piech *et al.*, "Deep knowledge tracing," *Advances in neural information processing systems*, vol. 28, 2015.

[20] Quizlet, "Quizlet's ai-powered learning assistant." `https://quizlet.com/blog/ai-powered-study-tools`, 2021.

[21] Coursera, "Introducing coursera coach: Your ai-powered learning assistant." `https://blog.coursera.org/coursera-coach-leveraging-genai-to-empower-learners/`, 2024.

[22] Khan Academy, "Khanmigo: Khan academy's ai-powered tutor." `https://khanmigo.ai/`, 2023.

[23] Duolingo, "Duolingo max: Ai-powered features." `https://blog.duolingo.com/duolingo-max/`, 2023.

[24] Scholarcy, "Scholarcy: Ai-powered summarization for research articles." `https://www.scholarcy.com`.

[25] QuillBot, "Quillbot: Ai-powered paraphrasing and summarization tool." https://quillbot.com.

[26] K. Gorman *et al.*, "Neural machine translation for low-resource languages: A survey," *ACM Computing Surveys*, 2019.

[27] G. R. Crane, "The perseus project: An interactive curriculum on classical greek civilization," *Educational Technology*, vol. 27, no. 11, pp. 25–32, 1987.

[28] Y. Ouvrard and P. Verker, "Collatinus (11.2): Lemmatiser and morphological analyser for latin texts." https://outils.biblissima.fr/collatinus/, 2023.

[29] P. Verker, "Eulexis (1.1): Lemmatiser for ancient greek texts." https://outils.biblissima.fr/en/eulexis/, 2026.

[30] Società Editrice Dante Alighieri, "L'app del rocci: Vocabolario greco-italiano digitale." https://www.ilrocci.com, 2024.

[31] L. Castiglioni and S. Mariotti, "Il - vocabolario della lingua latina (versione digitale)." https://www.loescher.it/il, 2016.

[32] F. Montanari, "Gi - vocabolario della lingua greca (versione digitale)." https://www.loescher.it/gi, 2025.

[33] M. C. Passarotti and R. Sprugnoli, "The CIRCSE collection of linguistic resources in CLARIN-IT," in *Proceedings of CLARIN Annual Conference 2021*, (Madrid, Spain), pp. 43–46, September 2021. CLARIN Annual Conference 2021, September 27–29.

[34] K. P. Johnson, P. J. Burns, J. Stewart, and T. Cook, "The classical language toolkit: An nlp framework for pre-modern languages," *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: System Demonstrations*, pp. 20–29, 2021.

[35] Payload CMS, "Payload cms: The most powerful typescript headless cms." `https://payloadcms.com`, 2026.

[36] Mistral AI, "Mistral ai: Frontier ai in your hands." `https://mistral.ai`, 2026.

[37] Mastra, "Mastra: Framework for building ai workflows and agents." `https://mastra.ai`, 2026.

[38] PostgreSQL Global Development Group, "Postgresql: The world's most advanced open source relational database." `https://www.postgresql.org`, 2026.

[39] Amazon, "Amazon web services: Cloud computing services." `https://aws.amazon.com`, 2026.