UNIVERSITÀ POLITECNICA DELLE MARCHE
FACOLTÀ DI INGEGNERIA
Dipartimento di Ingegneria dell'Informazione
Corso di Laurea in Ingegneria Informatica e dell'Automazione

TESI DI LAUREA

Progettazione e implementazione di case study relativi alla generazione di testi e di codice mediante l'utilizzo di LLM differenti

Design and implementation of case studies related to text and code generation through the usage of different LLMs

Relatore

Prof. Domenico Ursino

Candidato

Davide Mariotti

ANNO ACCADEMICO 2025-2026

Il computer è la bicicletta della nostra mente.

Steve Jobs

Sommario

Negli ultimi anni l'Intelligenza Artificiale Generativa, e in particolare i Large Language Model, hanno assunto un ruolo sempre più centrale nella produzione automatica di vari contenuti, rendendo necessario comprendere in che misura modelli diversi si comportino in modo simile o divergente a fronte delle stesse richieste.

La presente tesi nasce con l'obiettivo di confrontare quattro modelli, *ChatGPT*, *DeepSeek*, *Claude* e *Gemini*, attraverso due case study distinti. Il primo riguarda la generazione e modifica di testo e si basa sull'adattamento di quattro ricette di pizza a vincoli progressivamente più complessi e, in alcuni casi, contraddittori. Il secondo riguarda la generazione di codice, e consiste nello sviluppo iterativo di un'applicazione con integrazione dell'*API USDA FoodData Central*, a partire da un prompt iniziale comune a tutti i modelli.

Keyword: Intelligenza Artificiale Generativa, Large Language Model, ChatGPT, DeepSeek, Claude, Gemini, generazione di testo, generazione di codice

Indice

Elenco delle figure

Elenco delle tabelle

Introduzione

Negli ultimi anni l'Intelligenza Artificiale Generativa ha conosciuto una diffusione senza precedenti, trasformando il modo in cui vengono prodotti contenuti testuali, immagini, codice e altri tipi di dati. Al centro di questa trasformazione si collocano i Large Language Model (LLM), sistemi in grado di comprendere e generare linguaggio naturale con un livello di coerenza e versatilità tale da renderli utilizzabili in ambiti estremamente diversi tra loro, dalla scrittura assistita alla programmazione, fino a settori più specialistici, come quello nutrizionale. La loro adozione, inizialmente concentrata in contesti sperimentali, è oggi diffusa tanto in ambito professionale quanto in quello quotidiano, complice la facilità con cui è possibile interagire con questi strumenti attraverso semplici richieste in linguaggio naturale.

La rapida diffusione di modelli diversi, sviluppati da aziende differenti, con architetture, dati di addestramento e filosofie di interazione non sempre coincidenti, pone tuttavia un problema pratico e, allo stesso tempo, una domanda di ricerca rilevante: a fronte della stessa richiesta, modelli diversi possono fornire risposte significativamente diverse per qualità, coerenza e affidabilità. Comprendere in che misura questo accada, e in quali condizioni le differenze tra i modelli si accentuino o si attenuino, è quindi un aspetto rilevante sia dal punto di vista della ricerca sia da quello dell'utilizzo pratico di questi strumenti, poiché consente all'utente di orientarsi con maggiore consapevolezza nella scelta del modello più adatto al proprio scopo.

L'obiettivo di questa tesi è confrontare il comportamento di quattro LLM, *ChatGPT*, *DeepSeek*, *Claude* e *Gemini*, attraverso due case study distinti, pensati per sollecitare capacità diverse dei modelli e per osservarne il comportamento non solo in termini di correttezza formale della risposta, ma anche di coerenza, completezza e capacità di gestione della complessità. Il primo case study si concentra sulla generazione e modifica di testo, chiedendo ai modelli di adattare ricette tradizionali di pizza a vincoli nutrizionali progressivamente più numerosi e, in alcuni casi, tra loro contraddittori, al fine di valutare la capacità dei modelli di bilanciare creatività, realizzabilità pratica e coerenza gastronomica. Il secondo case study si concentra sulla generazione di codice, osservando i modelli nello sviluppo iterativo di un'applicazione software che integra dati provenienti da un'API esterna, con l'obiettivo di valutarne non solo la correttezza sintattica del codice prodotto, ma soprattutto la capacità di interpretare correttamente i requisiti, individuare la causa reale dei malfunzionamenti e comunicare in modo trasparente eventuali limiti tecnici. In entrambi i casi il confronto non si limita ad un giudizio complessivo sulla qualità delle risposte, ma cerca di individuare pattern ricorrenti nel comportamento dei singoli modelli, evidenziandone punti di forza e limiti strutturali che emergono.

La presente tesi è composta da quattro capitoli, strutturati come di seguito specificato:

- Nel Capitolo 1 viene introdotto il concetto di Intelligenza Artificiale Generativa, a partire da una panoramica sull'Intelligenza Artificiale, sul Machine Learning e sul Deep Learning, fino ad arrivare alle principali tipologie di modelli generativi e ai relativi benefici e rischi.
- Nel Capitolo 2 vengono presentati i quattro LLM oggetto di studio descrivendone le principali caratteristiche, le funzionalità disponibili e le modalità di interazione.
- Nel Capitolo 3 viene presentato il primo case study, relativo al confronto tra i diversi LLM nella generazione di testo, attraverso quattro scenari di adattamento di ricette a vincoli nutrizionali via via più complessi.
- Nel Capitolo 4 viene presentato il secondo case study, relativo al confronto tra i diversi LLM nella generazione di codice, attraverso lo sviluppo iterativo di un'applicazione software con integrazione dell'*API USDA FoodData Central*.

CAPITOLO 1

Introduzione all'Intelligenza Artificiale Generativa

In questo capitolo introduttivo viene presentato il concetto di Intelligenza Artificiale Generativa analizzandone la struttura e il funzionamento. Questa tecnologia rappresenta una delle evoluzioni più recenti nell'ambito dell'Intelligenza Artificiale e consente di produrre contenuti sotto forma di testi, immagini, codice, audio, video e altri tipi di dati. Grazie alla sua versatilità, sta assumendo un ruolo sempre più rilevante in numerosi ambiti applicativi, dalla comunicazione alla programmazione, fino all'utilizzo in settori specialistici come quello nutrizionale.

Nella prima parte del capitolo viene introdotto in maniera generale il concetto di Intelligenza Artificiale, spiegandone le finalità e gli ambiti applicativi. Successivamente, vengono descritti i concetti di Machine Learning e Deep Learning, fondamentali per comprendere il funzionamento dei modelli generativi. Infine, viene approfondita l'Intelligenza Artificiale Generativa, ne vengono descritti l'architettura, il funzionamento, le principali tipologie di modelli, i benefici e i rischi determinati dal suo utilizzo.

1.1 Definizione di Intelligenza Artificiale

L'Intelligenza Artificiale è un ramo dell'informatica che si occupa dello studio e della progettazione di sistemi hardware e software capaci di svolgere attività che, tradizionalmente, richiedono capacità tipiche dell'intelligenza umana. Tra queste rientrano la comprensione di un testo, l'apprendimento, la risoluzione di problemi, il processo decisionale, la creatività e l'autonomia operativa.

Le applicazioni e dispositivi dotati di Intelligenza Artificiale possono osservare ed identificare gli oggetti, sono in grado di comprendere e rispondere nel linguaggio umano. Inoltre, possono imparare da nuove informazioni ed esperienze oltre che fornire raccomandazioni dettagliate e supporto agli utenti.

1.1.1 Obiettivi e ambiti di applicazione dell'Intelligenza Artificiale

L'obiettivo principale dell'Intelligenza Artificiale è quello di realizzare sistemi capaci di supportare l'essere umano nello svolgimento di attività di varia complessità, automatizzando alcune operazioni e migliorando l'analisi delle informazioni disponibili. In particolare, l'IA permette di elaborare grandi quantità di dati, individuare relazioni e schemi ricorrenti, effettuare previsioni, fornire raccomandazioni e contribuire al processo decisionale.

Ad oggi, molti settori adottano l'utilizzo dell'Intelligenza Artificiale come supporto agli operatori umani. Ne è un esempio l'ambito medico, che impiega sistemi in grado di accedere

a grandi quantità di dati medici per rendere più rapida e precisa la diagnosi, la prevenzione delle malattie e l'individuazione di possibili trattamenti.

Anche il settore industriale e logistico fa uso di sistemi basati sull'Intelligenza Artificiale per pianificare attività e operazioni, monitorarne l'esecuzione e ottimizzare i processi produttivi.

Nel settore del marketing, l'utilizzo dell'Intelligenza Artificiale sta assumendo un ruolo sempre più rilevante. I sistemi sono in grado di analizzare i dati relativi all'utente, come le ricerche recenti, le preferenze e gli acquisti precedenti, al fine di fornire all'utente contenuti, consigli e inserzioni pubblicitarie più coerenti con i suoi interessi.

Ulteriori ambiti di applicazione riguardano i trasporti, la finanza e l'istruzione, ossia tutti quegli ambiti che necessitano di elaborare dati, automatizzare processi o personalizzare i servizi in base alle esigenze dell'utente.

1.1.2 Differenze tra IA Tradizionale e IA Moderna

Nel corso degli anni, l'Intelligenza Artificiale ha subito una profonda evoluzione, modificando il modo in cui i sistemi intelligenti vengono progettati e utilizzati. L'IA tradizionale e l'IA moderna si basano su due approcci differenti sia per modalità di funzionamento sia per capacità di adattamento.

L'IA tradizionale si basa principalmente su regole, algoritmi predefiniti e istruzioni esplicite fornite dall'essere umano. In questi sistemi, il comportamento della macchina viene stabilito in fase di progettazione attraverso una serie di condizioni e procedure da seguire per risolvere problemi specifici. Tale approccio risulta efficace per contesti ben definiti e con regole chiare. Tuttavia, presenta numerosi limiti quando deve affrontare scenari complessi, incerti o non previsti poiché il sistema non è in grado di generalizzare oltre l'insieme delle regole predefinite in fase di progettazione.

L'IA moderna, invece, si fonda sull'apprendimento dai dati. Attraverso tecniche di Machine Learning e Deep Learning, i sistemi sono in grado di individuare autonomamente schemi, relazioni e regolarità all'interno di grandi quantità di informazioni. Questo consente loro di adattarsi a situazioni nuove e di svolgere compiti sempre più complessi.

Questa evoluzione ha rappresentato un passaggio fondamentale per lo sviluppo dell'Intelligenza Artificiale Generativa, che non si limita ad analizzare o classificare dati, ma è in grado di produrre nuovi contenuti a partire dalle informazioni apprese.

1.2 Machine Learning e Deep Learning

1.2.1 Machine Learning

Il Machine Learning è una branca dell'Intelligenza Artificiale che comprende un insieme di tecniche e algoritmi attraverso i quali un sistema è in grado di migliorare le proprie prestazioni nel tempo a partire dall'esperienza e dai dati disponibili. A differenza dei sistemi basati esclusivamente su regole predefinite, un modello di Machine Learning non viene programmato esplicitamente per gestire ogni possibile situazione, ma apprende autonomamente schemi, relazioni e regolarità presenti nei dati.

Alla base dell'apprendimento automatico vi sono algoritmi capaci di analizzare informazioni, individuare pattern ricorrenti e utilizzare tali conoscenze per effettuare previsioni, classificazioni o prendere decisioni. In questo modo, il sistema può adattare il proprio comportamento in funzione dei dati osservati e migliorare progressivamente la qualità dei risultati prodotti.

Lo scopo del Machine Learning è, quindi, quello di consentire a una macchina di apprendere dall'esperienza, generalizzando le informazioni acquisite per applicarle anche a situazioni nuove o non esplicitamente previste in fase di progettazione.

L'apprendimento automatico non trova applicazione soltanto in ambiti altamente specializzati, ma è presente anche in numerosi strumenti di uso quotidiano. Ne sono esempi il riconoscimento vocale degli smartphone, la traduzione automatica e la personalizzazione dei contenuti pubblicitari in base agli interessi dell'utente.

Un ulteriore ambito di applicazione è rappresentato dal settore dei trasporti, in particolare nello sviluppo di sistemi di assistenza alla guida e veicoli autonomi. L'utilizzo combinato di sensori, telecamere, sistemi di localizzazione e algoritmi di apprendimento consente al veicolo di analizzare l'ambiente circostante e supportare decisioni relative a frenata, sterzata, mantenimento della corsia e adattamento della velocità al contesto.

1.2.2 Tecniche di apprendimento

A seconda del tipo di dati disponibili e dell'obiettivo da raggiungere, il Machine Learning può essere suddiviso in diverse metodologie di apprendimento:

- *Apprendimento supervisionato*: in questo tipo di apprendimento, viene fornita al sistema una serie di dati di addestramento etichettati, ossia esempi nei quali sono già presenti sia l'input sia l'output corretto atteso. Il modello analizza questi esempi e impara ad associare determinati dati in ingresso a una specifica risposta in uscita. Un esempio di apprendimento supervisionato è il riconoscimento delle e-mail di spam: il modello viene addestrato su un insieme di e-mail già classificate come "spam" o "non spam".

- *Apprendimento non supervisionato*: in questo caso, i dati forniti al sistema non sono etichettati. Il modello non dispone, quindi, di esempi con una risposta corretta già indicata, ma deve individuare autonomamente strutture, somiglianze, relazioni o gruppi presenti nei dati. Questo tipo di apprendimento permette, quindi, di organizzare le informazioni disponibili senza conoscere in anticipo le categorie da ottenere. Un esempio è il raggruppamento degli utenti di una piattaforma in base ai loro comportamenti o interessi. Il sistema può individuare gruppi di utenti simili tra loro, anche senza sapere in anticipo quali categorie debbano essere create.

- *Apprendimento semi-supervisionato*: questa tecnica rappresenta una soluzione intermedia tra i metodi precedentemente trattati. Il modello viene addestrato utilizzando una piccola quantità di dati etichettati e una quantità maggiore di dati non etichettati. In questo modo, il sistema può sfruttare le informazioni già classificate per orientare l'apprendimento e, allo stesso tempo, utilizzare i dati non etichettati per migliorare la propria capacità di generalizzazione. Un esempio di apprendimento semi-supervisionato è il riconoscimento di immagini: utilizzando le immagini etichettate, il sistema apprende le categorie principali e sfrutta quelle non etichettate per migliorare la precisione complessiva.

- *Apprendimento per rinforzo*: in questa tipologia di apprendimento, un agente interagisce con un ambiente e impara a scegliere le azioni migliori attraverso un sistema di ricompense e penalità. L'obiettivo è massimizzare la ricompensa ottenuta nel tempo, migliorando progressivamente la strategia adottata. A differenza dell'apprendimento supervisionato, non viene fornita direttamente la risposta corretta per ogni situazione, ma il sistema apprende attraverso le conseguenze delle proprie azioni. Un esempio di apprendimento per rinforzo è l'addestramento di un agente virtuale all'interno di un videogioco: l'agente riceve una ricompensa quando compie azioni corrette, come

raggiungere un obiettivo, ed eventualmente una penalità quando commette errori. Nel tempo, il sistema impara quali comportamenti permettono di ottenere risultati migliori.

1.2.3 Deep Learning

Con l'aumento della quantità di dati disponibili e della complessità dei problemi da risolvere, le tecniche tradizionali di Machine Learning hanno mostrato alcuni limiti, soprattutto nei casi in cui era necessario analizzare informazioni molto articolate, come immagini, testi, suoni o grandi insiemi di dati. Da questa esigenza si è sviluppato il Deep Learning, una sottocategoria del Machine Learning basata sull'utilizzo di reti neurali profonde.

Il Deep Learning consente la creazione di modelli di apprendimento organizzati su più livelli, nei quali le informazioni vengono elaborate in modo progressivo. I livelli iniziali tendono a riconoscere caratteristiche più semplici, mentre quelli successivi combinano tali informazioni per individuare rappresentazioni più complesse e astratte. Questo permette al modello di apprendere automaticamente dai dati, riducendo la necessità di definire manualmente le caratteristiche rilevanti per la risoluzione di un problema.

A differenza di alcuni approcci più tradizionali, nei quali l'essere umano deve indicare esplicitamente quali elementi osservare, i sistemi basati su Deep Learning sono in grado di individuare autonomamente schemi, relazioni e caratteristiche significative all'interno dei dati. Durante la fase di addestramento, il modello analizza numerosi esempi e modifica progressivamente i propri parametri interni, in modo da ridurre l'errore e migliorare la qualità delle previsioni o delle classificazioni prodotte.

Gli algoritmi di Deep Learning si basano principalmente sulle reti neurali artificiali, strutture ispirate in modo generale al funzionamento del cervello umano, pur senza riprodurlo fedelmente. Tali reti sono composte da unità di calcolo, chiamate neuroni artificiali o nodi, organizzate in più strati. Ogni strato riceve informazioni dal precedente, le elabora e trasmette il risultato allo strato successivo, fino alla produzione dell'output finale.

Il Deep Learning risulta particolarmente efficace in ambiti caratterizzati da grandi quantità di dati e da informazioni ad alta complessità. Esso trova, infatti, applicazione nel riconoscimento di immagini, nell'elaborazione del linguaggio naturale, nella trascrizione automatica di file audio, nella traduzione automatica, nei sistemi di raccomandazione e in molte altre attività che, in passato, richiedevano necessariamente l'intervento umano.

Un aspetto rilevante dei sistemi moderni basati su Deep Learning è la possibilità di elaborare dati di natura diversa, come testo, immagini, audio e metadati. Questo risulta utile, ad esempio, nei sistemi di e-commerce, dove un prodotto può essere descritto contemporaneamente da immagini, descrizioni testuali, recensioni e informazioni tecniche. L'analisi combinata di queste diverse fonti può migliorare attività come la ricerca dei prodotti, la raccomandazione personalizzata e l'individuazione di contenuti anomali o non corretti.

Nonostante i numerosi vantaggi, il Deep Learning presenta anche alcuni limiti. L'addestramento di modelli profondi richiede spesso grandi quantità di dati, elevate risorse computazionali e tempi di elaborazione significativi. Inoltre, la presenza di molti parametri interni può rendere più difficile comprendere in modo immediato il processo decisionale del modello. Per questo motivo, l'utilizzo del Deep Learning deve essere accompagnato da un'attenta valutazione della qualità dei dati, delle prestazioni ottenute e dell'affidabilità dei risultati prodotti.

1.2.4 Reti neurali

Le reti neurali artificiali rappresentano una delle principali basi del Deep Learning e, più in generale, dei moderni sistemi di Intelligenza Artificiale. Esse sono modelli computazionali ispirati, in modo semplificato, al funzionamento dei neuroni biologici del cervello umano.

Tuttavia, è importante sottolineare che le reti neurali artificiali non riproducono fedelmente il cervello umano, ma ne riprendono alcuni principi generali, come l'elaborazione distribuita dell'informazione e la presenza di connessioni tra unità di calcolo.

Una rete neurale è composta da unità elementari, chiamate neuroni artificiali o nodi, organizzate in livelli. Ogni neurone riceve uno o più valori in ingresso, li elabora attraverso operazioni matematiche e trasmette il risultato ai neuroni del livello successivo. Attraverso questa struttura, la rete è in grado di trasformare progressivamente i dati iniziali in rappresentazioni sempre più complesse, fino alla produzione dell'output finale.

In generale, una rete neurale è composta da tre principali tipologie di livelli. Il primo è il livello di input, che riceve i dati grezzi da elaborare, come valori numerici, parole, pixel di un'immagine o caratteristiche estratte da un insieme di dati. Seguono uno o più livelli nascosti, nei quali avviene la maggior parte dell'elaborazione: ogni nodo combina gli input ricevuti attraverso pesi associati alle connessioni e produce un risultato che viene, poi, inviato allo strato successivo. Infine, il livello di output restituisce il risultato finale del modello, che può essere una classificazione, una previsione numerica o una distribuzione di probabilità.

Due elementi fondamentali nel funzionamento delle reti neurali sono i pesi e i bias. I pesi determinano l'importanza delle connessioni tra i neuroni e stabiliscono quanto un determinato input influenzi il risultato finale. I bias, invece, permettono al modello di adattare meglio la propria risposta, aggiungendo un termine correttivo al calcolo effettuato dal neurone. Durante la fase di addestramento, la rete modifica progressivamente questi parametri con l'obiettivo di ridurre l'errore tra il risultato prodotto e quello atteso.

1.3 L'Intelligenza Artificiale Generativa

Grazie ai progressi nelle tecniche di Deep Learning, si sono affermati dei modelli capaci di generare nuovi contenuti a partire da dati esistenti, aprendo una varietà di nuove possibilità per l'utilizzo dell'Intelligenza Artificiale.

Questi modelli, addestrati su grandi quantità di dati, non si limitano più a restituire informazioni già presenti in archivio, ma generano un nuovo output basandosi su quanto appreso durante la fase di addestramento. La principale innovazione introdotta dall'Intelligenza Artificiale Generativa consiste, quindi, nella capacità del sistema di elaborare una risposta che rispetti il significato, i vincoli e l'obiettivo della richiesta inviata dall'utente.

Negli ultimi anni, l'Intelligenza Artificiale Generativa ha avuto una rapida diffusione grazie al miglioramento delle reti neurali, all'aumento della potenza computazionale disponibile e alla crescente quantità di dati utilizzabili per l'addestramento dei modelli. Strumenti come chatbot, generatori di immagini, assistenti alla programmazione e sistemi di supporto alla scrittura rappresentano esempi concreti di applicazioni basate su questa tecnologia.

1.3.1 Nascita e diffusione dell'IA Generativa

Sebbene l'Intelligenza Artificiale Generativa sia diventata particolarmente nota negli ultimi anni grazie alla diffusione di strumenti come ChatGPT e DALL-E, le sue origini sono molto più lontane. I primi modelli generativi risalgono, infatti, alle fasi iniziali dello sviluppo dell'Intelligenza Artificiale e dell'apprendimento statistico, quando venivano utilizzati modelli matematici per rappresentare sequenze di dati e generare nuove informazioni a partire da regolarità osservate.

I primi approcci erano utilizzati per elaborare dati sequenziali ed effettuare previsioni di elementi successivi all'interno di una sequenza. Seppur molto lontani dalle capacità attuali dei modelli generativi, questi rappresentano i primi tentativi di insegnare ad una macchina a generare dati nuovi basandosi su strutture probabilistiche apprese.

Le prime tecniche di generazione testuale si basavano su modelli linguistici semplici che stimavano la probabilità di una parola in funzione delle parole precedenti. Tali modelli permettevano la generazione di sequenze testuali che presentavano, però, forti limiti nella gestione di frasi lunghe. L'introduzione delle reti neurali ha permesso la produzione di risposte più coerenti al contesto anche per frasi lunghe.

Un altro ambito fondamentale nello sviluppo dell'IA Generativa è stato quello della visione artificiale. Prima della diffusione del Deep Learning la generazione di immagini si basava soprattutto su tecniche progettate manualmente, come la sintesi e la mappatura delle texture. Questi metodi, tuttavia, erano limitati nella capacità di produrre immagini complesse, realistiche e diversificate.

Un importante punto di svolta c'è stato nel 2014 con l'introduzione delle Reti Generative Avversarie (GAN), basate su un meccanismo di competizione tra due reti neurali: una rete generatrice, incaricata di produrre nuovi dati, e una rete discriminatrice, incaricata di distinguere i dati generati da quelli reali. Questo approccio ha favorito lo sviluppo di modelli capaci di generare immagini sempre più realistiche.

Con l'introduzione dei modelli Transformer si è avuta la svolta decisiva. I Transformer utilizzano meccanismi che permettono al modello di analizzare le relazioni tra diverse parti dell'input, anche quando esse sono distanti tra loro. Questa caratteristica ha reso possibile la gestione di sequenze più lunghe e complesse, contribuendo allo sviluppo dei moderni Large Language Models (LLM). Successivamente questa architettura è stata applicata anche alla visione artificiale e ai sistemi multimodali aprendo la strada a sistemi in grado di comprendere richieste testuali e generare contenuti coerenti in diversi formati.

1.3.2 Differenza tra modelli discriminativi e modelli generativi

Nel campo dell'Intelligenza Artificiale è possibile distinguere tra modelli discriminativi e modelli generativi. Questa distinzione riguarda principalmente il modo in cui il modello interpreta i dati e l'obiettivo per cui viene addestrato.

I modelli discriminativi hanno lo scopo di individuare la relazione tra i dati in ingresso e una determinata etichetta di uscita. Essi vengono, quindi, utilizzati soprattutto per compiti di classificazione o previsione. In altre parole, dato un insieme di caratteristiche di input, il modello cerca di determinare a quale classe o categoria appartenga l'elemento analizzato. Un esempio può essere un sistema che, osservando il contenuto di un'e-mail, stabilisce se essa debba essere classificata come "spam" o "non spam".

I modelli generativi, invece, hanno l'obiettivo di apprendere la struttura dei dati e le relazioni presenti al loro interno, in modo da poter generare nuovi esempi simili a quelli osservati durante l'addestramento. A differenza dei modelli discriminativi, che si concentrano principalmente sulla classificazione, i modelli generativi cercano di rappresentare il processo attraverso cui i dati possono essere prodotti. Per questo motivo, sono utilizzati nella generazione di testi, immagini, audio, codice e altri contenuti.

Una differenza fondamentale tra i due modelli consiste, quindi, nel risultato finale prodotto. Il modello discriminativo assegna un'etichetta o fornisce una previsione, mentre un modello generativo produce un nuovo contenuto. Ad esempio, un modello discriminativo può riconoscere se un'immagine rappresenta un gatto o un cane, mentre un modello generativo può creare una nuova immagine realistica di un gatto a partire dalle informazioni apprese.

Può capitare, talvolta, che modelli generativi e discriminativi lavorino insieme, come nel caso già trattato delle Reti Generative Avversarie nelle quali una rete generatrice produce nuovi dati, mentre una rete discriminatrice valuta se tali dati siano reali o generati artificialmente. Tale meccanismo permette al modello generativo di migliorare progressivamente la qualità dei contenuti prodotti.

1.4 Funzionamento dei modelli generativi

Dopo aver introdotto il concetto e la storia dell'Intelligenza Artificiale Generativa, è utile analizzare il processo generale attraverso cui i modelli generativi vengono realizzati e utilizzati. Il funzionamento di questi sistemi non si limita alla semplice produzione di un output, ma comprende diverse fasi: la raccolta dei dati, la loro preparazione, l'addestramento del modello, l'elaborazione dell'input fornito dall'utente e la successiva generazione del risultato con conseguenti verifiche.

Per comprendere meglio tale processo, nelle prossime sezioni verranno analizzate le principali fasi che caratterizzano il funzionamento dei modelli generativi.

1.4.1 Raccolta dei dati

La prima fase del processo consiste nella raccolta di enormi quantità di dati di addestramento; essi possono assumere diversi formati: testi, immagini, audio, codice, pagine web, documenti. La natura dei dati raccolti dipende dall'obiettivo del modello: un modello linguistico, ad esempio, sarà addestrato principalmente su dati testuali, mentre un modello per la generazione di immagini utilizzerà grandi insiemi di immagini e descrizioni associate.

I dati di addestramento rappresentano la base da cui il modello apprende schemi, relazioni e caratteristiche ricorrenti. Per questo motivo, la fase di raccolta è particolarmente importante, poiché la qualità e la varietà delle informazioni disponibili influenzano direttamente le capacità del sistema. Se i dati sono limitati, poco rappresentativi o non coerenti con il problema da affrontare, anche il modello addestrato tenderà a produrre risultati meno affidabili.

In alcuni ambiti, la raccolta dei dati avviene attraverso strumenti specifici. Nei sistemi con guida autonoma, ad esempio, la raccolta avviene attraverso l'uso di strumenti specifici, come sensori e telecamere. In altri casi, i dati provengono da dataset pubblici messi a disposizione da istituti di ricerca, imprese o comunità scientifiche.

1.4.2 Preparazione e qualità dei dati

Dopo la raccolta, i dati devono essere preparati prima di poter essere utilizzati per l'addestramento del modello. Questa fase è fondamentale in quanto i dati grezzi possono contenere errori, duplicati, valori mancanti, formati non uniformi o informazioni non pertinenti.

Per tale motivo, vengono eseguite su di essi operazioni di pulizia, selezione e trasformazione. Un esempio di controllo è la verifica del formato delle date.

Una volta controllati e puliti, i dati vengono trasformati in un formato leggibile dal modello. Nel caso dei modelli linguistici, il testo viene spesso suddiviso in unità più piccole, chiamate token, che possono essere elaborate dal sistema. Nei modelli visivi, invece, le immagini vengono convertite in rappresentazioni numeriche che permettono alla rete neurale di analizzarne le caratteristiche.

Negli ultimi anni, anche i dati sintetici hanno assunto un ruolo rilevante. Essi consistono in dati generati attraverso l'uso dell'Intelligenza Artificiale. Questo approccio risulta molto utile qualora i dati reali siano difficili da ottenere, costosi o soggetti a privacy. Tuttavia, anche questi dati necessitano di attenta valutazione.

1.4.3 Addestramento del modello

Dopo la preparazione dei dati raccolti, viene effettuata la fase di addestramento del modello. Durante questo processo, il sistema analizza i dati disponibili e apprende e le relazioni statistiche, gli schemi ricorrenti e le strutture presenti al loro interno. Se i dati di addestramento assomigliano molto ai problemi del mondo reale di cui si occuperà il modello,

impararne i pattern e le correlazioni consentirà a un modello addestrato di fare previsioni accurate su nuovi dati.

In generale, il dataset viene suddiviso in più parti. Il set di addestramento viene utilizzato per aggiornare i parametri del modello, cioè i valori interni che determinano il modo in cui il sistema elabora le informazioni. Il set di validazione viene impiegato per controllare le prestazioni durante lo sviluppo e per regolare alcune scelte del modello. Infine, il set di test viene utilizzato per valutare le prestazioni finali su dati non utilizzati direttamente durante l'addestramento.

Come già trattato nella Sezione 1.2.2, esistono diverse modalità di apprendimento di un modello. L'addestramento include più fasi: il pre-addestramento su grandi quantità di dati e successive fasi di adattamento a compiti più specifici. Durante l'addestramento, il modello cerca generalmente di ridurre una funzione di perdita, cioè una misura dell'errore commesso rispetto all'obiettivo previsto. In questo modo, il sistema modifica progressivamente i propri parametri interni per migliorare la qualità dei risultati prodotti. Nell'apprendimento per rinforzo, invece, il modello viene guidato da un sistema di ricompense e penalità, con l'obiettivo di massimizzare nel tempo il risultato ottenuto attraverso le proprie azioni.

1.4.4 Input, prompt e generazione dell'output

Una volta completata la fase di addestramento, il modello può essere utilizzato per generare nuovi contenuti. Durante questa fase l'utente fornisce un input, spesso chiamato prompt, che rappresenta la richiesta o l'istruzione da cui il sistema deve partire. Il prompt può essere una domanda, una descrizione, un comando, un testo da completare o un insieme di vincoli da rispettare.

Il modello elabora l'input ricevuto e produce un output sulla base delle informazioni apprese durante l'addestramento. Nel caso dei modelli linguistici, la generazione avviene progressivamente: il sistema produce una sequenza di parole o token cercando di mantenere coerenza con il contesto fornito. In modo analogo, un modello generativo di immagini può produrre un'immagine a partire da una descrizione testuale, mentre un modello per il codice può generare frammenti di programma sulla base delle istruzioni ricevute.

La qualità dell'output dipende da diversi fattori, tra cui la qualità del modello, i dati utilizzati per l'addestramento e la chiarezza del prompt fornito dall'utente. Una richiesta vaga o ambigua può portare a risultati poco precisi, mentre un prompt ben strutturato, con informazioni contestuali e vincoli chiari, può favorire una risposta più coerente e utile.

1.4.5 Attendibilità e validazione dell'output generato

Nonostante le capacità dei modelli generativi moderni, è importante sottolineare che l'output prodotto non può essere considerato automaticamente corretto o affidabile. Questi sistemi generano risultati sulla base di relazioni apprese dai dati, ma non possiedono una reale comprensione del contenuto nello stesso senso umano. Per questo motivo, possono produrre risposte plausibili dal punto di vista formale, ma errate o non verificabili dal punto di vista sostanziale.

Uno dei problemi più noti è rappresentato dalle cosiddette allucinazioni, cioè casi in cui il modello genera informazioni inesatte, inventate o non supportate da fonti reali. Questo fenomeno può verificarsi soprattutto quando il modello non dispone di informazioni sufficienti, quando il prompt è ambiguo o quando il sistema tenta comunque di produrre una risposta anche in assenza di dati affidabili.

Per ridurre questi rischi, è necessario prevedere forme di controllo e validazione dell'output generato. In ambiti tecnici, scientifici, medici o nutrizionali, le informazioni prodotte dovrebbero essere confrontate con fonti attendibili, database ufficiali o valutazioni di esperti.

Il controllo umano rimane, quindi, un elemento fondamentale, soprattutto quando i risultati generati vengono utilizzati per prendere decisioni importanti.

1.5 Principali modelli dell'IA Generativa

L'Intelligenza Artificiale Generativa comprende diverse tipologie di modelli, ciascuno progettato per generare contenuti differenti a seconda del tipo di dati utilizzati e dall'obiettivo da raggiungere. Alcuni modelli sono specializzati nella produzione e comprensione del linguaggio naturale, mentre altri nella generazione di immagini, testi, audio, video o dati sintetici.

Nel corso degli anni, lo sviluppo del Deep Learning ha permesso la realizzazione di modelli sempre più avanzati, capaci di apprendere quantità sempre maggiori di dati e di produrre output coerenti con le richieste fornite dall'utente. Tra i modelli principali troviamo i Large Language Model, i Transformer, i modelli di diffusione e le Reti Generative Avversarie.

Questi modelli, pur svolgendo funzioni diverse, condividono il fatto di apprendere dai dati per la generazione di nuovi contenuti. Nelle prossime sezioni verranno analizzate le principali caratteristiche di ciascun modello, evidenziandone il funzionamento generale e gli ambiti di applicazione.

1.5.1 Modelli Transformer

I modelli Transformer fanno parte della categoria dei modelli autoregressivi, ovvero modelli che effettuano la previsione dell'elemento successivo in una sequenza in base agli elementi precedenti. I primi modelli autoregressivi erano costituiti da reti neurali ricorrenti (RNN) che presentavano, però, criticità nella rilevazione di relazioni tra elementi distanti in una sequenza, e nel calcolo, poiché elaboravano gli elementi in sequenza uno per uno.

Dal loro avvento, i Transformer hanno sempre di più sostituito le RNN grazie a due importanti innovazioni di seguito analizzate.

- *Elaborazione parallela*: tutti gli elementi in una sequenza vengono elaborati contemporaneamente. Ciò permette ai Transformer di essere addestrati in molto meno tempo, soprattutto con set di dati su larga scala.
- *Meccanismi di auto-attenzione*: i Transformer sono in grado di considerare l'importanza relativa di tutti gli elementi di una sequenza durante la loro elaborazione. Tale innovazione consente di catturare relazioni tra elementi distanti in una serie, introducendo il concetto di comprensione testuale che mancava alle RNN.

La capacità di comprendere ed elaborare grandi sequenze di input ha permesso i Transformer di eccellere nei compiti di generazione di testo e traduzione linguistica; proprio per questo oggi sono ampiamente utilizzati da altri modelli di Intelligenza Artificiale Generativa, come i Large Language Model.

1.5.2 Large Language Model

La nascita e lo sviluppo dei Transformer ha permesso la creazione di modelli linguistici di grandi dimensioni chiamati Large Language Model (LLM); questi sono modelli di Deep Learning addestrati su immense quantità di dati e in grado di generare linguaggio naturale e altri contenuti per eseguire un'ampia varietà di attività.

Gli LLM funzionano come gigantesche macchine di previsione su base statistica che prevedono ripetutamente la parola successiva in una sequenza. Essi rappresentano il primo

sistema di Intelligenza Artificiale capace di gestire il linguaggio umano non strutturato su larga scala, consentendo una comunicazione naturale con le macchine. A differenza dei precedenti modelli, che si affidano ad algoritmi per abbinare le parole chiave, gli LLM acquisiscono il contesto, le sfumature e i ragionamenti più profondi.

Gli LLM sono facilmente accessibili al pubblico tramite interfacce come Claude di Anthropic, ChatGPT di Open AI, Copilot di Microsoft o Gemini di Google. Ad oggi, rappresentano il culmine dell'elaborazione del linguaggio naturale.

1.5.3 Modelli di diffusione

Prima di parlare dei modelli di diffusione è necessario introdurre il concetto di rumore. Per "rumore" si intende l'alterazione casuale dei dati. Prendendo in considerazione un'immagine, il rumore è una perturbazione casuale aggiunta ai pixel dell'immagine.

I modelli utilizzano l'inserimento graduale di rumore nei dati di input per poi ripulire il disordine creato in nuovi dati simili. In questo modo, essi generano nuovi dati imparando a trasformare il rumore in dati simili a quelli presenti nei loro set di dati di addestramento.

Questi modelli sono utilizzati soprattutto per la generazione di immagini, per processi di inpainting e outpainting, di modellazione tridimensionale e di rilevamento delle anomalie, grazie alla capacità di apprendere come i dati cambiano nel tempo e quando non si adattano alle relazioni e alle combinazioni stabilite.

1.5.4 Generative Adversarial Networks

Le Reti Generative Avversarie (GAN) sono tra i primi modelli di IA Generativa a mettere due modelli in competizione. Un modello generativo crea l'output, un modello discriminativo lo analizza e ne determina la correttezza. L'obiettivo di questa competizione è quello di ottenere un modello generativo che generi contenuti considerati sempre autentici quando valutati dal discriminatore.

Questi modelli vengono utilizzati per l'elaborazione di informazioni a partire da immagini, la generazione di immagini e il rilevamento di anomalie attraverso il confronto tra un set di dati reali e un set di dati sintetici creati al modello.

1.6 Vantaggi e limiti

La crescente innovazione di questi modelli di IA Generativa ha portato notevoli miglioramenti nella realizzazione di richieste e attività che precedentemente avrebbero richiesto un maggior impiego di tempo e risorse. Non bisogna, però, cadere nell'errore di etichettare qualsiasi informazione generata da questi modelli come veritiera e verificata; gli attuali modelli generativi presentano, infatti, ancora dei limiti.

Nelle prossime sezioni analizzeremo entrambi gli aspetti evidenziando i vantaggi e i rischi che comporta l'uso di modelli generativi.

1.6.1 Vantaggi principali dall'uso dell'IA Generativa

L'innovazione maggiore portata dai modelli generativi risulta, senza dubbio, la riduzione drastica dei tempi di lavoro per la ricerca e la creazione dei contenuti di testo, audio, immagini e codice

Infine, la capacità di elaborare enormi moli di dati in pochi secondi, estraendo le informazioni utili riassumendole e la traduzione rapida dei testi hanno aumentato l'accessibilità e la comprensione delle risorse per gli studenti e i ricercatori.

1.6.2 Rischi e limiti tecnici

Nella Sezione 1.4.5 abbiamo già analizzato le allucinazioni come rischio derivante dalla generazione di output da parte dei modelli. È sempre opportuno verificare l'attendibilità delle informazioni generate anche se presentate assieme a dati, citazioni o fatti storici.

Un altro argomento di discussione è l'accesso a dati protetti da copyright o privati. L'utilizzo di materiale protetto da diritto d'autore o di dati sensibili per l'addestramento dei modelli rischia di generare contenuti manipolati o fughe di informazioni riservate. I sistemi generativi non provano emozioni; si limitano ad eseguire le richieste fatte dall'utente e a prevedere attraverso la combinazione di dati già esistenti. Proprio per questo, talvolta, risultano piatti o privi di empatia.

CAPITOLO 2

Presentazione degli LLM in esame

Nel capitolo precedente sono stati introdotti i principali concetti legati all'Intelligenza Artificiale Generativa, con particolare attenzione ai Large Language Model e al loro ruolo nella produzione automatica di contenuti testuali, visivi e multimodali. Dopo aver descritto il funzionamento generale di questi sistemi, risulta ora necessario analizzare alcuni strumenti concreti attraverso i quali tali modelli vengono resi disponibili agli utenti.

Questo capitolo ha, quindi, lo scopo di presentare gli LLM che saranno successivamente utilizzati nei case study. L'attenzione non è rivolta a un'analisi puramente teorica o architetturale dei modelli, ma alla descrizione delle loro principali caratteristiche, delle funzionalità disponibili e delle modalità con cui l'utente può interagire con essi. Questa scelta è coerente con l'obiettivo generale del lavoro, orientato a valutare il comportamento dei sistemi di IA generativa in contesti applicativi reali.

Gli strumenti presi in esame sono ChatGPT, DeepSeek, Claude e Gemini. Essi rappresentano soluzioni sviluppate da aziende differenti e caratterizzate da approcci diversi: ChatGPT si distingue per la diffusione e l'integrazione di strumenti specializzati, DeepSeek per l'attenzione all'efficienza computazionale e al ragionamento, Claude per l'organizzazione dei modelli e per l'attenzione alla sicurezza e alla chiarezza delle risposte, mentre Gemini per la multimodalità e l'integrazione con l'ecosistema Google.

2.1 Obiettivo dello studio e metodologia di approccio

2.1.1 Obiettivo dello studio

L'obiettivo dello studio è quello di analizzare il comportamento di diversi sistemi basati su Large Language Model in contesti applicativi differenti, al fine di valutarne l'affidabilità, la coerenza e la capacità di rispettare le richieste formulate. In particolare, l'analisi si concentra sugli strumenti ChatGPT, DeepSeek, Claude e Gemini.

Lo studio non ha lo scopo di analizzare nel dettaglio l'architettura interna dei singoli modelli, ma quello di valutarne il comportamento dal punto di vista dell'utente finale. L'attenzione è, quindi, rivolta alla qualità degli output generati, alla completezza delle risposte, alla correttezza delle informazioni fornite e alla capacità dei sistemi di adattarsi alle istruzioni ricevute.

2.1.2 Metodologia di approccio

La metodologia adottata si basa sull'analisi delle risposte generate dai diversi strumenti selezionati, sottoposti a richieste analoghe. Per ciascun case study vengono definiti uno o più prompt, progettati in modo da valutare le specifiche capacità dei modelli. Le risposte

ottenute vengono, successivamente, analizzate sulla base dei criteri qualitativi citati nella sezione precedente.

Tale approccio consente di osservare non solo il risultato finale prodotto dai singoli strumenti, ma anche il modo in cui essi interpretano il compito assegnato. Poiché i case study affrontano contesti applicativi differenti, la valutazione viene adattata alle caratteristiche specifiche di ciascun caso. Tuttavia, viene mantenuto un criterio generale comune, basato sull'analisi dell'affidabilità e dell'utilità pratica delle risposte generate dai modelli.

2.2 ChatGPT

ChatGPT è un potente sistema di dialogo per la generazione di testo. Si tratta di un modello di elaborazione del linguaggio naturale che genera risposte simili a quelle umane agli input degli utenti. ChatGPT si basa sui modelli della famiglia GPT, sviluppati a partire dall'architettura Transformer e successivamente ottimizzati per l'interazione conversazionale. Questo modello viene addestrato su un'enorme quantità di dati conversazionali provenienti da Internet; una volta addestrato, può svolgere diverse attività come la traduzione, la risposta a domande, l'analisi di documenti e la produzione di codice.

Sviluppato dall'azienda americana OpenAI, la sua prima versione di lancio risale al 30 novembre 2022 con il modello GPT-3.5. La sua diffusione fu rapida, raggiungendo numerosi milioni di utenti attivi in soli due mesi, un primato per l'epoca. Nel corso degli anni il servizio si è evoluto, passando dai modelli GPT-4 e GPT-4o, fino ai più recenti GPT-5 e GPT-5.5, integrando diverse funzionalità multimodali sempre più avanzate. La Tabella 2.1 non ha lo scopo di riportare tutte le versioni esistenti, ma di sintetizzare le principali evoluzioni rilevanti per verificarne le variazioni.

versione	data di rilascio	caratteristiche principali
GPT-3.5	Novembre 2022	Modello iniziale basato su InstructGPT. Capace di conversazione e generazione di codice, ma soggetto a frequenti "allucinazioni" e con una finestra di contesto limitata.
GPT-4	Marzo 2023	Modello multimodale (testo e immagini). Notevolmente più potente nel ragionamento logico e nella risoluzione di problemi complessi. Finestra di contesto ampliata.
GPT-4o	Maggio 2024	Versione "Omni". Nativo multimodale per audio, visione e testo in tempo reale. Maggiore velocità di inferenza e costi ridotti per gli sviluppatori.
GPT-4.5	Febbraio 2025	Miglioramenti nella comprensione delle sfumature emotive e nell'intuizione. Maggiore fluidità nella conversazione "umana".
GPT-5	Agosto 2025	Architettura unificata con ragionamento avanzato. Introduzione del selettore di tono dinamico e drastica riduzione degli errori fattuali. Integrazione nativa in sistemi operativi terzi.
GPT-5.5	Aprile 2026	Modello successivo a GPT-5, progettato per attività complesse come coding, ricerca, analisi dei dati e utilizzo di strumenti.

Tabella 2.1: Principali iterazioni delle varie versioni di ChatGPT

2.2.1 Interfaccia di ChatGPT

La Figura 2.1 mostra l'interfaccia iniziale dell'applicazione web. Nella parte centrale è presente un campo di testo, in cui è possibile inserire le richieste da inviare al modello.

Oltre all'input testuale, ChatGPT mette a disposizione anche la possibilità di registrare vocalmente la richiesta, la quale verrà poi trascritta prima di essere elaborata, e l'aggiunta di materiali di riferimento, come, ad esempio, un file PDF o un'immagine. È anche possibile

interagire con il sistema attraverso la conversazione vocale, senza l'utilizzo di messaggi testuali.

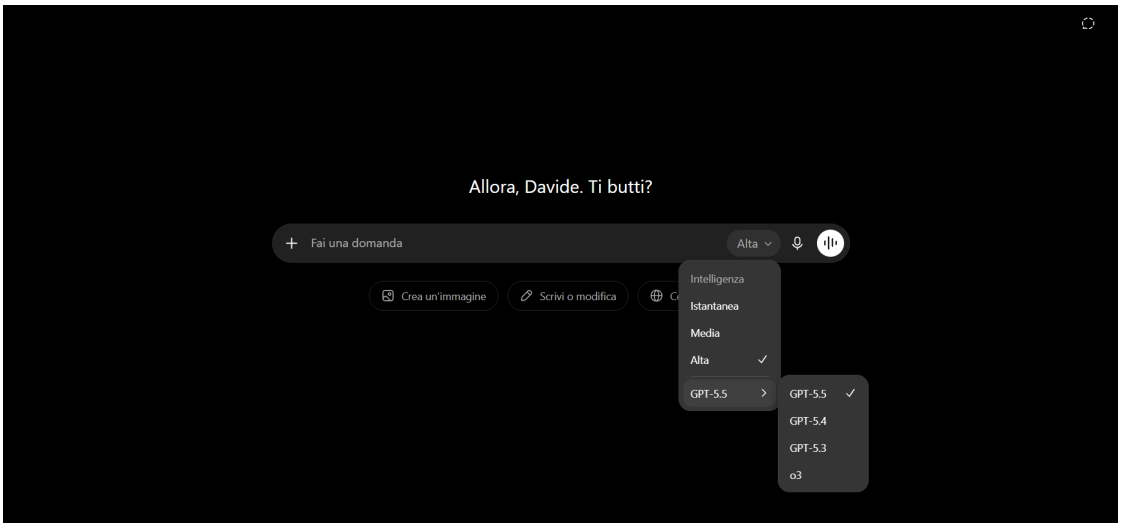


Figura 2.1: Interfaccia di ChatGPT versione web

Prima di iniziare una conversazione, si può selezionare la versione di GPT che si vuole utilizzare. Oltre a ciò, ChatGPT permette di selezionare diverse modalità di risposta o varianti del modello, che possono privilegiare rapidità, ragionamento più approfondito o maggiore accuratezza della risposta.

In base al livello selezionato, la risposta impiegherà un tempo più o meno ampio per essere generata. Questo accade perché il sistema analizza sempre più nel dettaglio le informazioni in suo possesso quanto più alto è il livello di ragionamento selezionato.

Oltre alle funzionalità già mostrate, ChatGPT fornisce la possibilità di integrare strumenti e modelli, sviluppati sempre da OpenAI, specializzati nell'esecuzione di compiti specifici come la generazione di immagini, la scrittura o la ricerca di informazioni online. Tra questi rientrano:

- *Codex*: orientato alla programmazione e all'assistenza nello sviluppo software;
- *DALL-E*: dedicato alla generazione di immagini a partire da descrizioni testuali;
- *Sora*: pensato per la generazione di contenuti video;
- *Whisper*: utilizzato per il riconoscimento automatico del parlato.

Per i case study analizzati è stata utilizzata la versione GPT-5.5, rilasciata il 23 aprile 2026. In particolare, nei casi riguardanti la generazione o la correzione di codice è stato considerato anche il supporto di Codex, con riferimento alla produzione di codice in linguaggio Python.

2.2.2 Codex

Codex è uno strumento orientato all'assistenza nello sviluppo software. A differenza di ChatGPT, che rappresenta un assistente generalista utilizzabile in numerosi contesti, Codex è progettato in modo specifico per attività legate alla programmazione, come la generazione di codice, la correzione di errori, la revisione di file sorgenti e il supporto alla comprensione di una codebase.

Il suo funzionamento può essere interpretato come quello di un agente di coding, cioè un sistema in grado di ricevere istruzioni in linguaggio naturale e trasformarle in operazioni

concrete sul codice. In questo senso, Codex non si limita a produrre frammenti di codice come risposta testuale, ma può essere impiegato anche per analizzare file, proporre modifiche, individuare bug, suggerire refactoring ed eseguire attività di supporto allo sviluppo software.

Nel presente lavoro, Codex viene considerato come strumento complementare a ChatGPT nei case study riguardanti la programmazione. La sua inclusione permette di osservare il comportamento di un sistema di IA generativa non soltanto nella produzione di testo, ma anche in un ambito applicativo specifico, come lo sviluppo di codice.

2.3 DeepSeek

DeepSeek è uno strumento di Intelligenza Artificiale generativa sviluppato dall'azienda cinese DeepSeek. Analogamente a ChatGPT, esso consente all'utente di interagire con un modello linguistico attraverso un'interfaccia conversazionale, formulando richieste in linguaggio naturale e ricevendo risposte generate automaticamente. DeepSeek risulta particolarmente interessante per l'attenzione posta sull'efficienza computazionale e sulla risoluzione di compiti che richiedono ragionamento logico, matematico o di programmazione.

DeepSeek comprende diversi modelli linguistici, tra cui DeepSeek-V3, DeepSeek-R1 e DeepSeek-V4 Preview. DeepSeek ha reso disponibili pubblicamente diversi modelli, report tecnici e risorse di sviluppo, favorendo l'analisi e il riutilizzo da parte della comunità di ricerca e sviluppo. Ciononostante, alcune fonti esterne, come il quotidiano The Guardian, hanno evidenziato possibili limitazioni o forme di censura nelle risposte relative a temi politicamente sensibili legati alla Cina, come Taiwan o il governo cinese.

2.3.1 DeepSeek-V3

Rilasciato nel dicembre 2024, DeepSeek-V3 è un modello linguistico basato sull'architettura Mixture of Experts (MoE). In questo tipo di architettura, il modello è composto da più sottoreti specializzate, dette esperti, e da un meccanismo di instradamento che seleziona quali esperti attivare per elaborare ciascun token.

La caratteristica principale di DeepSeek-V3 è l'efficienza computazionale, cioè la capacità di ottenere buone prestazioni riducendo il costo di calcolo rispetto alla dimensione complessiva del modello. Il technical report indica la presenza di 671 miliardi di parametri totali, di cui 37 miliardi attivati per ciascun token. Questo permette di migliorare l'efficienza durante l'addestramento e l'inferenza, poiché non tutte le componenti del modello vengono utilizzate in ogni singola operazione, ma solo una parte dei parametri viene attivata in base al token e al contesto da elaborare.

2.3.2 DeepSeek-R1

DeepSeek-R1 è un modello orientato al ragionamento. Pubblicato nel gennaio 2025, è stato progettato per gestire compiti nei quali è necessario analizzare un problema, individuare passaggi intermedi e produrre una risposta coerente con il procedimento seguito. Per questo motivo, DeepSeek-R1 risulta particolarmente significativo in ambiti come la risoluzione di esercizi, la scrittura e correzione di codice, l'analisi di problemi tecnici e la valutazione di quesiti logici.

Un ulteriore aspetto rilevante riguarda la disponibilità del modello alla comunità di ricerca e sviluppo. DeepSeek ha, infatti, reso disponibili DeepSeek-R1, DeepSeek-R1-Zero e alcuni modelli distillati, favorendo l'analisi e il riutilizzo del modello in contesti differenti. Questo elemento distingue DeepSeek da altri sistemi maggiormente chiusi e rende R1 un caso interessante nel panorama dei modelli orientati al ragionamento.

2.3.3 DeepSeek-V4 Preview

Rilasciato nell'aprile 2026, DeepSeek-V4 Preview rappresenta la versione più recente della famiglia DeepSeek. Il modello mantiene l'impostazione basata sull'architettura MoE, già presente in DeepSeek-V3, ma introduce un'evoluzione sia nella dimensione del modello sia nelle funzionalità disponibili.

DeepSeek-V4 Preview è stato presentato in due varianti principali: V4-Pro e V4-Flash. La prima è pensata per compiti più complessi e per attività che richiedono maggiore capacità di ragionamento, mentre la seconda privilegia rapidità ed efficienza economica. Inoltre, la serie V4 introduce un contesto fino a un milione di token, rendendola più adatta ad attività che richiedono l'analisi di grandi quantità di informazioni o l'esecuzione di compiti articolati.

2.3.4 Interfaccia di DeepSeek

La Figura 2.2 mostra l'interfaccia iniziale dell'applicazione web. Nella parte centrale è presente un campo di testo, in cui è possibile inserire le richieste da inviare al modello.

Oltre all'input testuale, DeepSeek permette di selezionare diverse modalità operative in base al tipo di attività da svolgere. Esse sono

- *Istantanea*: risponde in modo rapido e diretto, ed è pensata per richieste semplici o spiegazioni brevi.
- *Esperta*: fornisce risposte più approfondite e ragionate, risultando adatta a problemi più complessi, anche se solitamente richiede tempi di elaborazione maggiori.
- *Vision*: permette di lavorare con immagini, screenshot e documenti visivi, consentendo al modello di interpretare il contenuto presente nell'immagine.

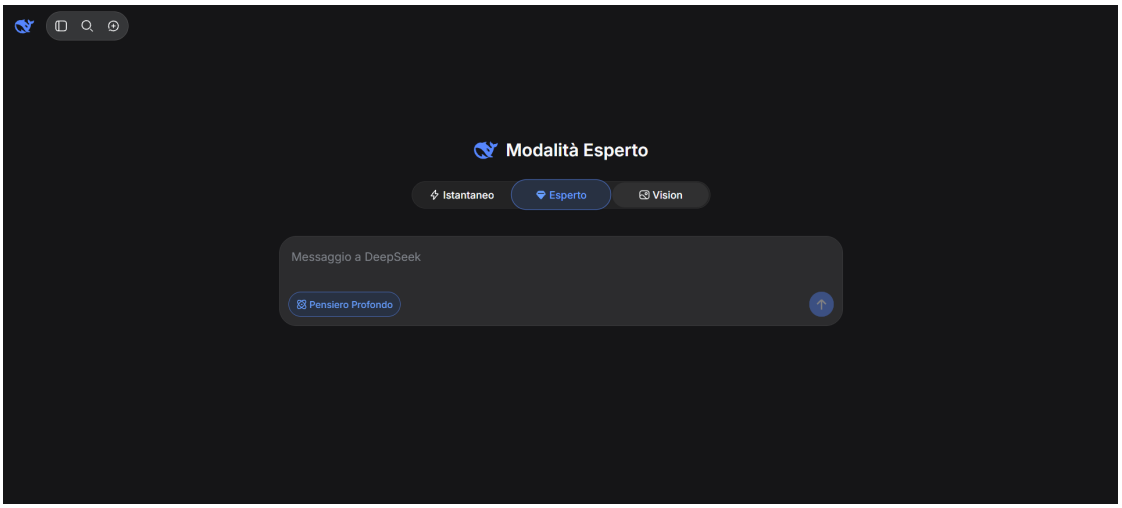


Figura 2.2: Interfaccia di DeepSeek versione web

DeepSeek mette, inoltre, a disposizione alcune funzionalità aggiuntive che possono integrare la modalità operativa scelta. La *Ricerca intelligente* consente al modello di recuperare informazioni online, evitando di basarsi esclusivamente sulle conoscenze interne del modello. Il *Pensiero profondo*, invece, è pensato per favorire un'elaborazione più approfondita della richiesta prima della generazione della risposta. Questa funzionalità risulta utile per problemi complessi, esercizi, codice e richieste che prevdedono più passaggi logici.

Per i case study analizzati, è stata selezionata la modalità *Esperto*, con la funzionalità *Pensiero profondo* attiva.

2.4 Claude

Claude è una famiglia di modelli sviluppata dall'azienda statunitense Anthropic. Claude è principalmente orientato all'elaborazione di testo, documenti e codice. A differenza di altri strumenti, non è però pensato per la generazione di immagini, audio o video.

Questi modelli risultano particolarmente adatti ad attività di scrittura, sintesi, analisi di testi lunghi, programmazione e ragionamento su documenti complessi. Rispetto ad altri modelli di LLM, essi si distinguono per uno stile di risposta generalmente discorsivo e strutturato, spesso orientato alla chiarezza espositiva e alla rielaborazione dei contenuti.

Attraverso la cosiddetta *Costituzione di Claude*, Anthropic descrive un insieme di principi pensati per orientare il comportamento del modello verso risposte utili, sicure e coerenti con determinati valori.

La prima versione di *Claude* è stata rilasciata nel marzo 2023, disponibile inizialmente solo per utenti selezionati da Anthropic. Il primo modello disponibile al pubblico fu *Claude 2*, rilasciato a luglio dello stesso anno. Successivamente, a partire dalla nascita di *Claude 3* nel 2024, l'azienda americana ha iniziato a distribuire tre diversi modelli rappresentati nella Figura 2.3. L'obiettivo di questa varietà di modelli si basa sulla volontà di offrire un diverso equilibrio tra velocità, costo e capacità di ragionamento.

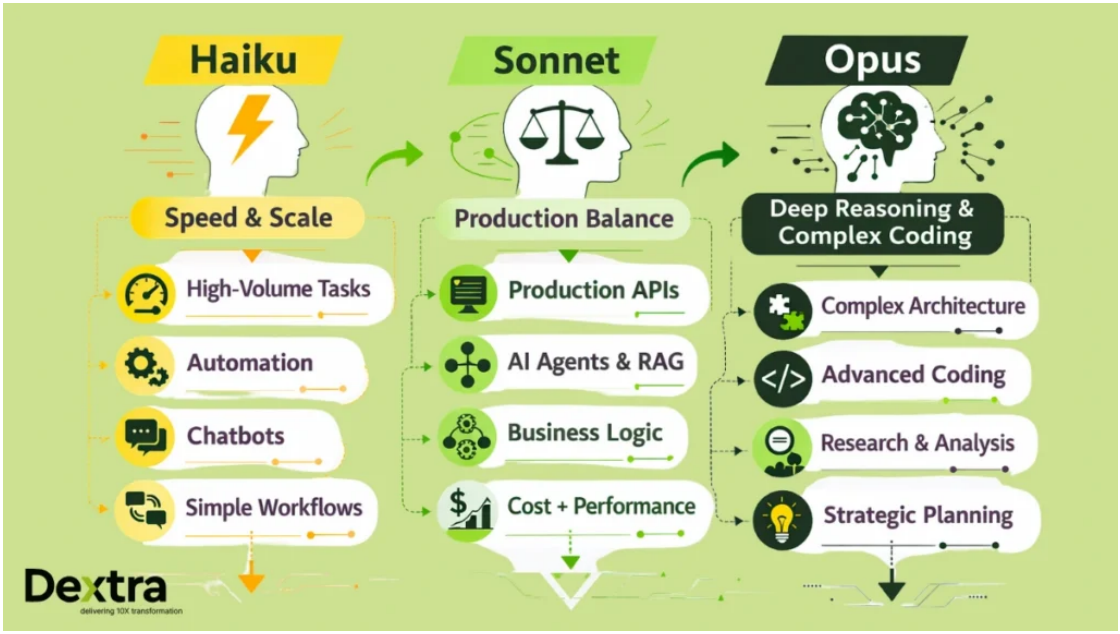


Figura 2.3: I tre principali modelli di Claude

Claude 4 è stato introdotto nel 2025 inizialmente con i modelli Sonnet e Opus; successivamente, la linea Claude 4 è stata ampliata anche con Haiku. Una caratteristica rilevante di questi modelli è la presenza di capacità di *extended thinking*, cioè una modalità che consente al modello di dedicare maggiore elaborazione ai compiti più complessi. In questo modo Claude può fornire risposte più strutturate e ragionate, risultando utile soprattutto nei casi in cui non è sufficiente una semplice risposta immediata.

2.4.1 Claude Haiku

Claude Haiku rappresenta la variante più veloce ed economica tra i modelli disponibili. È pensata per attività che richiedono tempi di risposta ridotti, come conversazioni frequenti,

assistenti virtuali, automazioni semplici e compiti nei quali è importante privilegiare la velocità di elaborazione.

Questi modelli risultano ottimali per applicazioni che richiedono inferenza a bassa latenza, tempo reale e compiti semplici a volume elevato, come riassunti, estrazione dati o traduzione automatica.

Per i costi di utilizzo di Claude, il prezzo viene calcolato per 1 milione di token (MTok). Nella Tabella 2.2 sono riportati i prezzi indicati da Anthropic; un aspetto importante è che il costo dei token di *output* è cinque volte il costo dei token di *input*.

Modello Claude	Input	Output
Claude Haiku 4.5	1 $ / MTok	5 $ / MTok
Claude Sonnet 4.6	3 $ / MTok	15 $ / MTok
Claude Opus 4.8	5 $ / MTok	25 $ / MTok

Tabella 2.2: Costo dei principali modelli Claude per milione di token

Come possiamo notare dalla tabella precedente, *Claude Haiku 4.5* risulta essere il modello di gran lunga più economico tra quelli della famiglia Claude.

2.4.2 Claude Sonnet

Claude Sonnet rappresenta il modello intermedio della famiglia Claude. È progettato per offrire un equilibrio tra la qualità delle risposte, la velocità di generazione e il costo di utilizzo. Per questo motivo può essere considerato il modello più adatto a un impiego generale, poiché riesce a gestire sia richieste semplici sia attività più articolate senza raggiungere i costi computazionali dei modelli più avanzati.

Il modello *Claude Sonnet 4.6*, rilasciato nel febbraio 2026, risulta particolarmente adatto ad attività come la scrittura di testi, la sintesi di documenti, l'analisi di file estesi, la programmazione e il supporto a workflow professionali. Inoltre, nelle API, supporta una finestra di contesto fino a un milione di token in beta, utile per analizzare codebase, contratti lunghi o grandi quantità di documenti.

2.4.3 Claude Opus

Claude Opus è il modello di Claude più grande e capace; il suo nome fa riferimento al concetto di *magnum opus*, termine latino usato per indicare l'opera più grande e pienamente realizzata.

Questo modello è progettato per compiti impegnativi che richiedono elevate capacità di ragionamento, comprensione, pianificazione e autonomia. La priorità non è, quindi, la rapidità o il costo, ma la massima qualità possibile della risposta. Può essere utile per problemi tecnici complessi, analisi scientifiche o matematiche, progettazione software e studio di documenti lunghi.

Rispetto a Sonnet, Opus è pensato per situazioni in cui è necessario ottenere una risposta più approfondita, robusta e ragionata. Per questo motivo il suo impiego risulta più giustificato nei casi in cui la complessità del compito richiede capacità superiori, mentre Sonnet rappresenta una soluzione più bilanciata per un utilizzo frequente e generale.

2.4.4 Interfaccia di Claude

La Figura 2.4 mostra l'interfaccia iniziale dell'applicazione web. Nella parte centrale è presente un campo di testo, in cui è possibile inserire le richieste da inviare al modello.

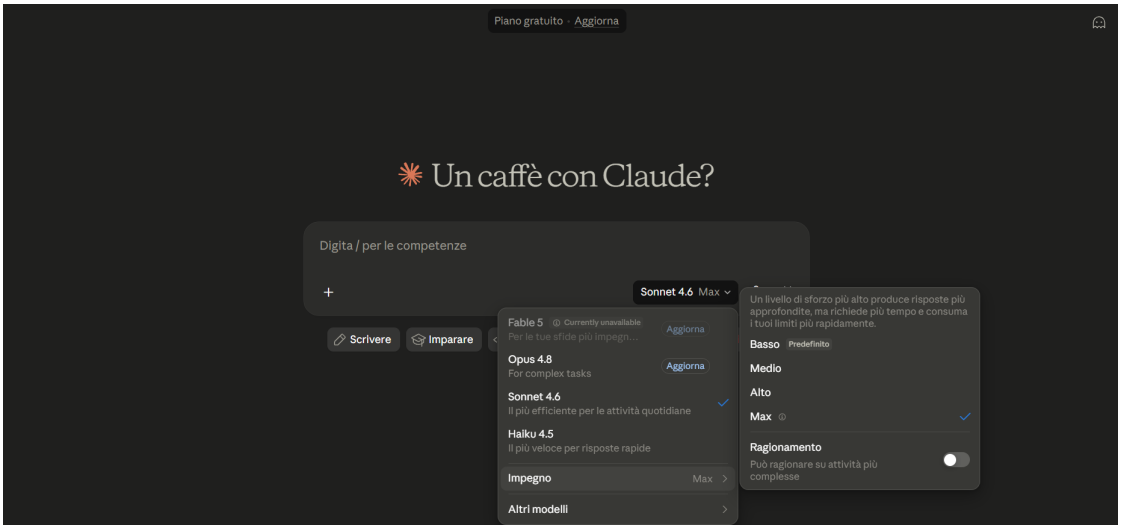


Figura 2.4: Interfaccia di Claude versione web

Claude permette di selezionare il modello che si preferisce, oltre che il livello di ragionamento nel generare la risposta. Un livello di ragionamento più elevato consente generalmente di ottenere risposte più approfondite, ma comporta tempi di elaborazione maggiori e un consumo più rapido dei limiti gratuiti disponibili.

Oltre all'input testuale, Claude mette a disposizione anche la possibilità di registrare vocalmente la richiesta, la quale verrà poi trascritta prima di essere elaborata. È anche possibile interagire con il sistema attraverso la conversazione vocale, senza l'utilizzo di messaggi testuali.

Claude mette, inoltre, a disposizione la possibilità di effettuare ricerche sul web, così da integrare nella risposta informazioni aggiornate provenienti da fonti esterne. È, inoltre, possibile selezionare competenze, connettori e plugin al modello per poter operare al meglio in un determinato contesto.

Infine, Claude propone dei suggerimenti per ambiti specifici con domande preimpostate o suggerite dal modello. Per i case study analizzati, è stato selezionato *Claude Sonnet 4.6* con livello di impegno impostato al massimo.

2.5 Gemini

Gemini è una famiglia di modelli linguistici di grandi dimensioni sviluppati da Google DeepMind. Si tratta di una famiglia di modelli di AI multimodali, progettati per gestire e combinare diverse tipologie di input, tra cui immagini, audio, codice software, testo e video.

Gemini consente all'utente di interagire con il modello attraverso un'interfaccia conversazionale, formulando richieste in linguaggio naturale e ricevendo risposte generate automaticamente. Una delle sue caratteristiche distintive è rappresentata dall'integrazione con l'ecosistema Google, che permette al modello di collegarsi a strumenti come Gmail, Google Drive, Documenti, YouTube, Maps e altre applicazioni dell'ambiente Google.

La prima versione della famiglia, *Gemini 1.0*, è stata presentata nel dicembre 2023 ed era organizzata in tre varianti principali: Ultra, Pro e Nano. Successivamente, Google ha introdotto nuove generazioni del modello, arrivando alla famiglia *Gemini 3*, pubblicata per la prima volta a novembre 2025 con il modello Pro. In questa generazione, l'attenzione è rivolta non soltanto alla generazione di risposte testuali, ma anche al ragionamento avanzato, alla multimodalità, alla programmazione e all'utilizzo di strumenti esterni.

2.5.1 Gemini Pro

Gemini Pro è pensato per attività che richiedono capacità elevate di ragionamento, analisi di informazioni complesse, programmazione e comprensione di contenuti multimodali. Questo modello risulta particolarmente adatto ad attività di studio, ricerca, analisi di documenti, generazione e correzione di codice, interpretazione di immagini, video o audio e supporto a compiti articolati.

L'ultima versione di questo modello risulta essere *Gemini 3.1 Pro*, rilasciato a febbraio 2026. Rispetto a modelli più leggeri, questo modello privilegia la qualità della risposta e la capacità di gestire richieste più complesse, anche quando queste coinvolgono diverse tipologie di dati. Questo lo rende adatto non solo alla semplice generazione testuale, ma anche a contesti in cui il modello deve analizzare un problema, utilizzare strumenti esterni o seguire una sequenza di passaggi logici.

2.5.2 Gemini Flash

Gemini Flash rappresenta un modello orientato alla rapidità e all'efficienza. È pensato per offrire buone prestazioni mantenendo tempi di risposta ridotti e costi inferiori rispetto al modello Pro. L'ultima versione rilasciata risulta essere *Gemini 3.5 Flash* a maggio 2026.

Questo modello può essere utilizzato per attività come sintesi di testi, classificazione di contenuti, supporto conversazionale, analisi di file, automazioni e richieste quotidiane. Rispetto a Gemini 3.1 Pro, Flash è generalmente più indicato nei casi in cui è importante ottenere una risposta rapida, senza richiedere il massimo livello di ragionamento.

2.5.3 Gemini Flash-Lite

Gemini Flash-Lite è una variante ancor più leggera ed economica dei modelli della famiglia Gemini. La sua ultima versione rilasciata risulta essere *Gemini 3.1 Flash-Lite* a marzo 2026.

Questo modello è stato progettato per attività ad alto volume, nelle quali sono importanti bassa latenza, costo ridotto e rapidità di elaborazione. Flash-Lite è adatto a compiti semplici o ripetitivi, come estrazione di dati, classificazione, risposte brevi, automazioni e applicazioni in cui il modello deve essere utilizzato frequentemente. Pur avendo capacità inferiori rispetto al modello Pro, *Gemini Flash-Lite* rappresenta una soluzione utile quando l'efficienza operativa è più importante della massima profondità di ragionamento.

2.5.4 Interfaccia di Google Gemini

La Figura 2.5 mostra l'interfaccia iniziale dell'applicazione web. Nella parte centrale è presente un campo di testo, in cui è possibile inserire le richieste da inviare al modello.

Gemini permette di selezionare il modello che si preferisce, oltre che il livello di ragionamento per generare una risposta. Oltre all'input testuale, Gemini mette a disposizione anche la possibilità di registrare vocalmente la richiesta, la quale verrà poi trascritta prima di essere elaborata; esso, inoltre, consente l'aggiunta di materiali di riferimento, come, ad esempio, un file PDF o un'immagine. Per i case study analizzati, è stato selezionato *Google Gemini 3.1 Pro* con livello di ragionamento impostato su esteso.

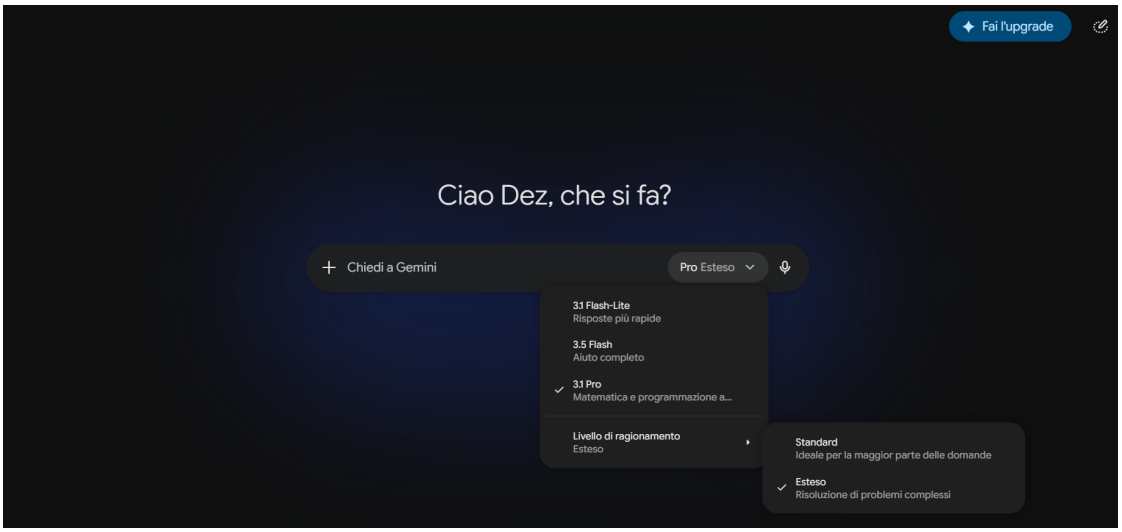


Figura 2.5: Interfaccia di Gemini versione web

CAPITOLO 3

Un case study per confrontare diversi LLM nella generazione di testo

Dopo aver introdotto i vari LLM in esame e aver descritto le loro principali caratteristiche, le funzionalità disponibili e le modalità di interazione, è il momento di entrare nel dettaglio con il primo tipo di case study effettuato.

Nella prima parte del capitolo vengono spiegati l'obiettivo del case study, la metodologia di approccio ad esso e i criteri di valutazione utilizzati per analizzare le risposte che i vari modelli forniranno. Successivamente, viene illustrato l'elenco dei dati raccolti necessari per poter effettuare le richieste. Nella restante parte del capitolo verranno mostrate ed analizzate le risposte dei vari modelli ai diversi prompt iniziali e le richieste che ne conseguiranno.

3.1 Obiettivo e criteri di valutazione

3.1.1 Obiettivo del case study

L'obiettivo di questo case study è analizzare e confrontare il comportamento dei modelli precedentemente introdotti nella generazione di varianti di una ricetta tradizionale. Per effettuare questi studi si è deciso di prendere come riferimento delle ricette per la pizza. La scelta di questi scenari consente di valutare la capacità dei modelli di produrre contenuti pratici, comprensibili e coerenti con un contesto reale, nel quale l'utente richiede modifiche progressive a una preparazione già nota.

In particolare, il test mira a verificare se i modelli sono in grado di proporre varianti della ricetta mantenendo un equilibrio tra creatività e realizzabilità. Una buona risposta, infatti, non dovrebbe limitarsi a elencare ingredienti casuali, ma dovrebbe costruire una proposta coerente dal punto di vista del gusto, delle quantità, del procedimento e dell'effettiva preparazione.

3.1.2 Criteri di valutazione

I criteri utilizzati per la valutazione delle risposte sono i seguenti:

- *Aderenza alla richiesta*: valuta se il modello rispetta il prompt iniziale.
- *Coerenza della ricetta*: considera se ingredienti, abbinamenti e procedimento risultano compatibili tra loro.
- *Completezza della risposta*: verifica la presenza di ingredienti, quantità, passaggi di preparazione ed eventuali consigli utili.

- *Creatività*: considera la capacità del modello di proporre varianti originali mantenendo la coerenza culinaria della ricetta.
- *Chiarezza espositiva*: analizza l'ordine, la leggibilità e la comprensibilità della risposta.
- *Gestione dei vincoli*: valuta se il modello riesce a rispettare limitazioni specifiche, come ingredienti da evitare, preferenze alimentari o richieste di semplificazione.

Nel complesso, questo case study permette, quindi, di analizzare i modelli non solo sulla base della qualità testuale della risposta, ma anche sulla base della loro utilità pratica in un contesto quotidiano. La generazione di varianti per ricette rappresenta infatti un esempio concreto di utilizzo dell'IA generativa, nel quale l'utente non richiede una semplice informazione, ma una proposta personalizzata, modificabile e applicabile nella realtà.

3.2 Ricette di partenza e dati utilizzati

La raccolta dei dati delle ricette di partenza è stata effettuata dal sito GialloZafferano, un portale online che racchiude migliaia di ricette di ogni tipo. Poiché le ricette considerate presentano quantità differenti in termini di numero di porzioni, gli ingredienti e i valori nutrizionali sono stati calcolati per una singola porzione. I valori nutrizionali selezionati sono: Calorie, Proteine, Lipidi, Carboidrati, Sodio e Fibre totali.

Per gli ingredienti indicati come 'q.b.', quando energeticamente rilevanti, è stata adottata una stima standard. In particolare, per l'olio extravergine d'oliva aggiunto in superficie è stato assunto un valore pari a 5 g per pizza, coerente con i disciplinari della pizza napoletana. Gli ingredienti indicati come "q.b." e impiegati esclusivamente nelle fasi tecniche di lavorazione, come le spezie aggiunte dopo la cottura, non sono stati inclusi nel calcolo nutrizionale, in quanto non quantificabili con precisione e non sempre interamente presenti nel prodotto finale.

Di seguito vengono riportate le ricette selezionate con le rispettive quantità degli ingredienti per una porzione:

► Pizza capricciosa (peso complessivo: 585,1 g)

- Farina 0 W300 (166,7 g);
- Acqua (116,7 g);
- Lievito di birra fresco (0,83 g);
- Sale fino (4,17 g);
- Olio extravergine di oliva (10,0 g);
- Pomodori pelati (66,7 g);
- Mozzarella (150.0 g);
- Prosciutto cotto (33,3 g);
- Funghi sott'olio (16,7 g);
- Carciofi sott'olio (13,3 g);
- Olive nere denocciolate (6,7 g)

► Pizza patate e salsiccia (peso complessivo: 591,1 g)

- Farina 0 W300 (166,7 g);
- Acqua (116,7 g);

- Lievito di birra fresco (0,5 g);
- Sale fino (5,0 g);
- Olio extravergine di oliva (5,5 g);
- Patate (66,7 g);
- Salsiccia napoletana (50,0 g);
- Mozzarella di bufala DOP (100,0 g);
- Mozzarella di bufala bocconcini (80,0 g).

► Pizza radicchio, ventricina piccante e primosale (peso complessivo: 618,5 g)

- Farina 0 W300 (175,0 g);
- Acqua (112,5 g);
- Lievito di birra fresco (1,0 g);
- Sale fino (5,0 g);
- Olio extravergine di oliva (15,0 g);
- Primosale (150,0 g);
- Salame ventricina piccante (60,0 g);
- Radicchio (50,0 g);
- Parmigiano Reggiano DOP (25,0 g);
- Ciporre rosse (25,0 g).

► Pizza mortadella e bufala (peso complessivo: 566,5 g)

- Farina 0 W300 (250,0 g);
- Acqua (150,0 g);
- Lievito di birra fresco (0,5 g);
- Sale fino (6,0 g);
- Olio extravergine di oliva (10,0 g);
- Mozzarella di bufala DOP (100,0 g);
- Mortadella (50,0 g).

I dati nutrizionali degli ingredienti sono stati ottenuti consultando le Tabelle di Composizione degli Alimenti del CREA (Centro di Ricerca, Alimenti e Nutrizione). In particolare, per ciascun alimento sono stati presi in considerazione i valori riferiti a 100 g di prodotto, così da rendere omogeneo il confronto tra gli ingredienti utilizzati nella ricetta. Nella Tabella 3.1 vengono riportati i valori nutrizionali di ciascun alimento.

Poiché per alcuni ingredienti non era disponibile una corrispondenza esatta nelle tabelle consultate, sono state adottate alcune approssimazioni, scegliendo le voci nutrizionali e le fonti ritenute più coerenti con l'alimento utilizzato nella ricetta. In particolare:

- a 1 g di sale corrispondono 400 mg di sodio;
- per i valori nutrizionali dei funghi sott'olio è stata presa in considerazione la voce 'funghi coltivati, prataioli, cotti, in padella';
- per i valori nutrizionali dei carciofi sott'olio è stata presa in considerazione la voce 'carciofi, cotti, bolliti';

Alimento	Energia (kcal)	Proteine (g)	Lipidi (g)	Carboidrati (g)	Sodio (mg)	Fibre totali (g)
Farina 0 W300	321	11,5	1,0	69,5	2	2,9
Olio extravergine di oliva	899	-	99,9	trascurabile	-	-
Lievito di birra fresco	70	12,1	0,4	1,1	16	6,9
Pomodori pelati	22	1,2	0,5	3,0	9	0,9
Mozzarella	253	18,7	19,5	0,7	200	-
Prosciutto cotto	138	15,7	7,6	1,7	840	-
Funghi sott'olio	46	7	0,4	1,5	4	4,3
Carciofi sott'olio	44	3,6	0,3	3,3	100	7,4
Olive nere	243	1,6	25,1	0,8	54	3,9
Patate	74	1,8	0,1	16,9	7	1,3
Salsiccia napoletana	304	15,4	26,7	0,6	1100	-
Mozzarella di bufala DOP	257	14,1	21,7	1,4	211	-
Primosale	254	16,47	20,06	1,9	488	-
Salame ventricina piccante	378	24,7	29,7	0,2	1640	-
Radicchio	19	1,4	0,1	1,6	10	3,0
Parmigiano Reggiano DOP	397	32,4	29,7	-	600	-
Cipolla rossa	28	1,0	0,1	5,7	10	1,0
Mortadella	288	15,7	25,0	-	960	-

Tabella 3.1: Valori nutrizionali per 100g di ciascun alimento

- per i valori nutrizionali delle patate è stata presa in considerazione la voce 'senza buccia, cotte, bollite';
- per i valori nutrizionali della salsiccia napoletana è stata presa in considerazione la voce 'salsiccia di suino, fresca';
- per i valori nutrizionali del primosale sono stati utilizzati i dati riportati dalla banca dati alimentare FatSecret;
- per i valori nutrizionali della ventricina è stata presa in considerazione la voce 'salame napoli';

Nella Tabella 3.2 vengono riportati, quindi, i valori nutrizionali corrispondenti per ognuna delle ricette selezionate per questo case study.

Ricetta	Energia (kcal)	Proteine (g)	Lipidi (g)	Carboidrati (g)	Sodio (mg)	Fibre totali (g)
Pizza capricciosa	1095,38	55,10	45,56	120,20	2273,73	7,45
Pizza patate e salsiccia	1248,73	53,51	59,64	129,93	2937,88	5,73
Pizza radicchio, ventricina piccante e primosale	1420,85	68,82	72,15	126,83	3877,16	6,89
Pizza mortadella e bufala	1293,75	50,76	46,69	175,16	3096,08	7,29

Tabella 3.2: Valori nutrizionali per ciascuna ricetta selezionata

In qualsiasi scenario che verrà analizzato nelle sezioni successive, sono stati sottoposti gli stessi prompt a tutti e quattro i modelli di LLM. Questi prompt sono composti dalla ricetta selezionata, dai corrispondenti valori nutrizionali e, successivamente, da una serie di richieste effettuate al modello per poter adattare la ricetta ai vincoli proposti.

3.3 Scenario 1: adattamento della pizza capricciosa

3.3.1 Prompt iniziale

Utente

PIZZA CAPRICCIOSA - Dosi per 1 pizza (peso 585,1g)

- Farina 0 W300 166,7 g
- Acqua 116,7 g
- Lievito di birra fresco 0,83 g
- Sale fino 4,17 g
- Olio extravergine d'oliva 10,0 g
- Pomodori pelati 66,7 g
- Mozzarella 150,0 g
- Prosciutto cotto 33,3 g
- Funghi sott'olio 16,7 g
- Carciofi sott'olio 13,3 g
- Olive nere denocciolate 6,7 g

Valori nutrizionali:

- Energia (kcal): 1095,38
- Proteine (g): 55,10
- Lipidi (g): 45,56
- Carboidrati (g): 120,20
- Sodio (mg): 2273,73
- Fibre totali (g): 7,45

Voglio che la mia pizza, partendo dalla ricetta che ti ho mandato, venga modificata affinché sia senza glutine, vegetariana, con massimo 800 calorie e che il prosciutto cotto rimanga presente.

Risposte al prompt iniziale

ChatGPT

Pizza capricciosa senza glutine vegetariana light

Dosi per 1 pizza

Ingrediente	Quantità
Mix per pizza senza glutine	120 g
Acqua	100 g
Lievito di birra fresco	0,8 g
Sale fino	2,5 g
Olio extravergine d'oliva	3 g
Pomodori pelati	70 g
Mozzarella	60 g
Prosciutto cotto vegetale	33,3 g
Funghi freschi	80 g
Carciofi al naturale	40 g
Olive nere denocciolate	5 g
Origano/basilico	q.b.

Modifiche rispetto alla capricciosa originale

Ricetta originale	Nuova versione
Farina 0 W300 166,7 g	Mix per pizza senza glutine 120 g
Acqua 116,7 g	Acqua 100 g
Sale 4,17 g	Sale 2,5 g
Olio EVO 10 g	Olio EVO 3 g
Mozzarella 150 g	Mozzarella 60 g
Prosciutto cotto 33,3 g	Prosciutto cotto vegetale 33,3 g
Funghi sott'olio 16,7 g	Funghi freschi 80 g
Carciofi sott'olio 13,3 g	Carciofi al naturale 40 g
Olive nere 6,7 g	Olive nere 5 g
Pomodori pelati 66,7 g	Pomodori pelati 70 g

Valori nutrizionali stimati

Valore	Pizza originale	Nuova pizza stimata
Energia	1095,38 kcal	circa 720–780 kcal
Proteine	55,10 g	circa 25–32 g
Lipidi	45,56 g	circa 18–25 g
Carboidrati	120,20 g	circa 95–110 g
Sodio	2273,73 mg	circa 1400–1900 mg
Fibre totali	7,45 g	circa 7–10 g

Perché rispetta i vincoli

- **Senza glutine:** la farina 0 viene sostituita con un mix per pizza senza glutine.
- **Vegetariana:** il prosciutto cotto tradizionale viene sostituito con prosciutto cotto vegetale.
- **Prosciutto presente:** rimane comunque presente un ingrediente con funzione equivalente al prosciutto cotto, cioè il prosciutto cotto vegetale in quantità pari a 33,3 g.
- **Massimo 800 kcal:** la riduzione calorica deriva soprattutto dalla diminuzione della farina, dell'olio e della mozzarella, dall'eliminazione degli ingredienti sott'olio e dall'uso di verdure fresche o al naturale.

Risultato finale
La nuova pizza ha un valore energetico stimato compreso tra circa 720 e 780 kcal. Di conseguenza, rientra nel limite massimo di 800 kcal previsto dalla richiesta.

DeepSeek

Certamente. Considerando che il prosciutto cotto è un ingrediente di origine animale, la pizza non può essere vegetariana. Tuttavia, l'obiettivo principale della richiesta sembra essere la realizzazione di una pizza senza glutine, con prosciutto cotto e con un apporto calorico non superiore a 800 kcal. La proposta mantiene quindi il prosciutto cotto e riduce le calorie agendo sugli altri ingredienti, evitando condimenti sott'olio e utilizzando una mozzarella light.

PIZZA CAPRICCIOSA SENZA GLUTINE CON PROSCIUTTO COTTO
Dosi per 1 pizza – peso circa 505 g

Ingrediente	Quantità
Farina senza glutine, mix per pane/pizza	120 g
Acqua	84 g
Lievito di birra fresco	0,6 g
Sale fino	3 g
Olio extravergine d'oliva	8 g
Pomodori pelati	66,7 g
Prosciutto cotto	33,3 g
Mozzarella light	120 g
Funghi champignon freschi, cotti senza grassi	40 g
Carciofi freschi, cotti senza grassi	30 g
Olive nere denocciolate	6,7 g
Origano, basilico fresco	q.b.

Preparazione

1. Sciogliere il lievito nell'acqua tiepida, quindi incorporare la farina senza glutine, il sale e l'olio.
2. Impastare brevemente fino a ottenere un panetto omogeneo e lasciarlo lievitare coperto per circa 1 ora.
3. Stendere l'impasto su carta forno, distribuire i pomodori pelati schiacciati e cuocere in forno a 250°C per 5–7 minuti.
4. Aggiungere la mozzarella light a pezzetti, il prosciutto cotto, i funghi e i carciofi cotti in padella antiaderente senza grassi, e le olive.
5. Rimettere in forno per 5–6 minuti, fino a quando la mozzarella si sarà sciolta.
6. Profumare con origano e basilico fresco.

Valori nutrizionali stimati
Valori riferiti alla pizza intera

Nutriente	Originale	Versione senza glutine e leggera
Energia	1095,38 kcal	796 kcal
Proteine	55,10 g	36,2 g
Lipidi	45,56 g	24,5 g
Carboidrati	120,20 g	88,3 g
Fibre totali	7,45 g	4,5 g
Sodio	2273,73 mg	∼1600 mg

Questa versione rispetta le condizioni principali: impasto senza glutine, presenza del prosciutto cotto, mozzarella in quantità significativa e apporto calorico inferiore a 800 kcal. La pizza rimane gustosa e ricca grazie alle verdure fresche e al prosciutto cotto, ma risulta molto più leggera rispetto alla ricetta originale.

Claude

Pizza Capricciosa senza glutine, vegetariana con prosciutto, $\leq$ 800 kcal
Piccola nota tecnica prima della ricetta: il prosciutto cotto è di origine animale, quindi una pizza che lo contiene non può essere considerata vegetariana in senso stretto. Procedo interpretando la richiesta come "senza glutine, senza altri ingredienti di origine animale oltre a mozzarella e prosciutto cotto, e con un apporto calorico massimo di 800 kcal", che rappresenta l'unica lettura coerente con i vincoli indicati. Se si intendesse una versione vegetariana stretta, sarebbe sufficiente eliminare il prosciutto cotto dalla ricetta seguente.

Dosi per 1 pizza – peso circa 580 g

Ingrediente	Quantità	Note
Mix farine senza glutine per pizza	150,0 g	Sostituisce la farina 0
Psillio in polvere	3,0 g	Dà struttura all'impasto senza glutine
Acqua	125,0 g	Leggermente aumentata
Lievito di birra fresco	0,83 g	Invariato
Sale fino	4,17 g	Invariato
Olio extravergine d'oliva	10,0 g	Invariato
Pomodori pelati	90,0 g	+23,3 g
Mozzarella	150,0 g	Invariata
Prosciutto cotto	33,3 g	Invariato, come da vincolo
Champignon freschi	50,0 g	Sostituiscono i funghi sott'olio
Carciofi al naturale	30,0 g	Sostituiscono i carciofi sott'olio
Olive nere denocciolate	5,0 g	$-1,7$ g
Origano e basilico fresco	q.b.	Sapore senza incremento calorico rilevante

Valori nutrizionali stimati

Nutriente	Valore	Confronto con l'originale
Energia	$\sim$795 kcal	-300 kcal (-27%)
Proteine	$\sim$48 g	-7 g
Lipidi	$\sim$30 g	$-15,5$ g
Carboidrati	$\sim$95 g	-25 g
Sodio	$\sim$2000 mg	-275 mg
Fibre totali	$\sim$7,7 g	+0,3 g

Logica delle sostituzioni

- **Farina 0 $\rightarrow$ mix senza glutine + psillio**: la farina 0 viene sostituita con 150 g di mix senza glutine e 3 g di psillio. Le farine senza glutine per pizza hanno una densità calorica simile a quella della farina tradizionale, ma assorbono più acqua. Riducendo la quantità di farina e aumentando l'acqua si ottiene un impasto adeguatamente idratato, con un risparmio stimato di circa 50 kcal. Lo psillio è utile perché, in assenza di glutine, contribuisce a dare struttura e masticabilità all'impasto.
- **Funghi sott'olio $\rightarrow$ champignon freschi**: la quantità passa da 16,7 g a 50 g. I funghi freschi hanno un apporto calorico nettamente inferiore rispetto a quelli sott'olio, permettendo di aumentare la quantità, migliorare volume e fibre, e ridurre comunque le calorie.
- **Carciofi sott'olio $\rightarrow$ carciofi al naturale**: la quantità passa da 13,3 g a 30 g. Anche in questo caso vengono eliminati gli effetti calorici dell'olio di conservazione, aumentando la porzione senza appesantire la pizza.

- **Pomodori pelati e olive**: i pomodori pelati vengono aumentati di 23,3 g per bilanciare visivamente la pizza, mentre le olive vengono leggermente ridotte per mantenere la nota saporita caratteristica della capricciosa senza aumentare troppo grassi e sodio.

Il taglio calorico principale deriva dall'eliminazione dell'olio di conservazione di funghi e carciofi. Mozzarella e prosciutto cotto restano invece invariati, come richiesto dai vincoli della ricetta.

Suggerimento di preparazione
L'impasto senza glutine non si lavora come quello tradizionale: tende infatti ad assomigliare più a una pastella densa che a un impasto elastico. Conviene mescolare prima farina senza glutine, psillio e sale a secco, poi sciogliere il lievito nell'acqua tiepida con l'olio, unire gli ingredienti e amalgamare. Dopo un riposo di circa 30 minuti, l'impasto può essere steso direttamente sulla teglia oliata con mani umide o con una spatola.
Gli champignon vanno affettati sottili e saltati per 2–3 minuti in padella antiaderente senza grassi, così da ridurre l'acqua in eccesso. I carciofi devono essere scolati e tamponati. Il prosciutto cotto può essere aggiunto negli ultimi 2 minuti di cottura, per evitare che si secchi.

Attenzione alla contaminazione crociata
In caso di celiachia è necessario verificare che tutti gli ingredienti confezionati, come mozzarella, prosciutto cotto, pomodori pelati, carciofi e olive, riportino la dicitura "senza glutine" o il simbolo della spiga barrata. Alcuni prodotti industriali, in particolare i salumi, possono infatti contenere tracce di glutine o addensanti non adatti.

Gemini

Aspetta un secondo, c'è un piccolo "corto circuito" logico nella richiesta: viene chiesta una pizza vegetariana, ma allo stesso tempo si pretende che il prosciutto cotto rimanga presente. Poiché il prosciutto è carne, le due condizioni si escludono a vicenda.
Per risolvere questo paradosso, l'unica soluzione coerente è utilizzare un prosciutto cotto vegetale, cioè un prodotto plant-based che ne imita consistenza e sapore. Se invece si intendesse utilizzare il prosciutto cotto vero e proprio, la pizza non sarebbe più vegetariana, pur mantenendo un profilo calorico simile.
Per far rientrare l'impasto senza glutine, il "cotto" e il resto degli ingredienti entro il limite massimo di 800 kcal, sono state rimodulate leggermente le porzioni della base e della mozzarella, utilizzando una versione light ed eliminando drasticamente i grassi degli ingredienti sott'olio.

Nuova ricetta: Capricciosa Paradox
Dosi per 1 pizza – massimo 800 kcal

Ingrediente	Quantità	Note / Modifiche apportate
Mix di farine senza glutine	125,0 g	Sostituisce la farina 0, ricalibrata per ridurre le calorie
Acqua	88,0 g	Idratazione al 70%, adatta al senza glutine
Lievito di birra fresco	0,60 g	Proporzionato al nuovo peso della farina
Sale fino	3,10 g	Proporzionato al nuovo peso della farina
Olio extravergine d'oliva	8,0 g	Leggermente ridotto per alleggerire i grassi totali
Pomodori pelati	66,7 g	Invariati
Mozzarella light	110,0 g	Ridotta per non superare le 800 kcal
Prosciutto cotto vegetale	33,3 g	Quantità invariata; rende la pizza vegetariana
Funghi champignon freschi	16,7 g	Sostituiscono i funghi sott'olio
Carciofini al naturale	13,3 g	Sostituiscono i carciofini sott'olio
Olive nere denocciolate	6,7 g	Invariate

Nota

Se si decidesse di utilizzare il vero prosciutto cotto, rinunciando quindi al vincolo vegetariano, sarebbe opportuno scegliere un cotto di alta qualità e rimuovere accuratamente il grasso visibile. In questo modo l'apporto calorico rimarrebbe simile.

Analisi nutrizionale stimata

Nutriente	Valore stimato	Osservazione
Energia	∼756 kcal	Sotto il limite richiesto di 800 kcal
Proteine	∼36 g	Garantite da mozzarella light e prosciutto vegetale
Lipidi	∼24 g	Principalmente legati all'olio EVO della base
Carboidrati	∼95 g	Derivanti soprattutto dal mix senza glutine
Fibre totali	∼7,0 g	Apporto mantenuto dalle verdure e dal mix senza glutine

Confronto delle risposte

In questo primo prompt sono state inserite richieste in contrasto tra loro, in quanto viene richiesto di ottenere una pizza contemporaneamente vegetariana e con il prosciutto cotto ancora presente tra gli ingredienti. Questo scenario consente di valutare la capacità dei modelli di modificare la ricetta riconoscendo e gestendo le richieste contraddittorie. Tutti i modelli fanno notare tale contrasto nelle richieste, ma vengono adottati dei metodi d'approccio differenti:

- *ChatGPT* e *Gemini* adottano un approccio simile. Individuano il conflitto tra i vincoli proposti e scelgono di risolverlo reinterpretando il termine "prosciutto cotto" come un suo equivalente vegetale. In questo modo, i modelli riescono a soddisfare simultaneamente tutti i vincoli richiesti: la pizza diventa senza glutine, vegetariana e inferiore alle 800 kcal mediante una riduzione delle quantità di alcuni ingredienti.
- *DeepSeek* adotta un approccio completamente differente decidendo di non soddisfare contemporaneamente tutti i vincoli. Dopo aver esplicitato l'incompatibilità della richiesta, infatti, produce una ricetta senza glutine e con meno di 800 kcal, mantenendo, però, il prosciutto cotto tradizionale.
- *Claude* segue una strategia intermedia. Anche in questo modello evidenzia immediatamente il conflitto tra la presenza del prosciutto e la definizione di pizza vegetariana, ma, anziché modificare gli ingredienti, propone una propria interpretazione della richiesta. Claude dichiara, infatti, di assumere che l'utente intenda una pizza nella quale gli unici ingredienti di origine animale siano mozzarella e prosciutto cotto, mantenendo quindi quest'ultimo invariato. La criticità principale della risposta consiste nel fatto che il modello decide autonomamente di modificare il significato del vincolo "vegetariana".

3.3.2 Secondo prompt

Utente

PIZZA CAPRICCIOSA - Dosi per 1 pizza (peso 585,1g)

- Farina 0 W300 166,7 g
- Acqua 116,7 g
- Lievito di birra fresco 0,83 g
- Sale fino 4,17 g

- Olio extravergine d'oliva 10,0 g
- Pomodori pelati 66,7 g
- Mozzarella 150,0 g
- Prosciutto cotto 33,3 g
- Funghi sott'olio 16,7 g
- Carciofi sott'olio 13,3 g
- Olive nere denocciolate 6,7 g

Valori nutrizionali:

- Energia (kcal): 1095,38
- Proteine (g): 55,10
- Lipidi (g): 45,56
- Carboidrati (g): 120,20
- Sodio (mg): 2273,73
- Fibre totali (g): 7,45

Voglio che la mia pizza, partendo dalla ricetta che ti ho mandato, venga modificata affinché sia senza glutine, vegetariana, con massimo 800 calorie, che il prosciutto cotto rimanga presente e che l'impasto sia sempre uguale nelle quantità.

Risposte al secondo prompt

ChatGPT

Con tutti i vincoli insieme, la soluzione è possibile solo se per "prosciutto cotto" si intende un prosciutto cotto vegetale.
Se invece si volesse mantenere il prosciutto cotto normale, allora la richiesta sarebbe impossibile, perché una pizza con prosciutto cotto vero non può essere considerata vegetariana.
Di conseguenza, la versione coerente è la seguente.

Pizza capricciosa senza glutine, vegetariana, max 800 kcal

Dosi per 1 pizza

Ingrediente	Quantità
Mix per pizza senza glutine	166,7 g
Acqua	116,7 g
Lievito di birra fresco	0,83 g
Sale fino	4,17 g
Olio extravergine d'oliva	10,0 g
Pomodori pelati	60 g
Mozzarella	15 g
Prosciutto cotto vegetale	33,3 g
Funghi freschi	70 g
Carciofi al naturale, non sott'olio	40 g
Olive nere denocciolate	3 g
Origano/basilico	q.b.

Cosa rimane invariato

Parte della ricetta	Risultato
Quantità dell'impasto	Invariate
Acqua	116,7 g
Lievito	0,83 g
Sale	4,17 g
Olio	10,0 g
Quantità di farina/mix	166,7 g
Prosciutto presente	Sì, ma vegetale

La farina 0 W300 viene sostituita con un mix senza glutine, mantenendo però la stessa quantità, pari a 166,7 g.

Cosa viene modificato

Ricetta originale	Nuova versione
Farina 0 W300 166,7 g	Mix senza glutine 166,7 g
Mozzarella 150 g	Mozzarella 15 g
Prosciutto cotto 33,3 g	Prosciutto cotto vegetale 33,3 g
Funghi sott'olio 16,7 g	Funghi freschi 70 g
Carciofi sott'olio 13,3 g	Carciofi al naturale 40 g
Olive nere 6,7 g	Olive nere 3 g
Pomodoro 66,7 g	Pomodoro 60 g

Valori nutrizionali stimati

Valore	Pizza originale	Nuova versione stimata
Energia	1095,38 kcal	circa 780–800 kcal
Proteine	55,10 g	circa 22–28 g
Lipidi	45,56 g	circa 20–25 g
Carboidrati	120,20 g	circa 120–130 g
Sodio	2273,73 mg	circa 1600–2100 mg
Fibre totali	7,45 g	circa 7–10 g

Attenzione importante sulle 800 kcal
Per rientrare effettivamente nel limite delle 800 kcal, è necessario scegliere:

- un mix per pizza senza glutine con circa 330–340 kcal per 100 g;
- un prosciutto cotto vegetale con circa 140–160 kcal per 100 g;
- carciofi al naturale e non sott'olio;
- funghi freschi e non sott'olio.

Se si utilizza un mix senza glutine più calorico, oppure un affettato vegetale particolarmente ricco di grassi, la pizza potrebbe superare il limite delle 800 kcal.

Risultato finale
La pizza diventa:

- senza glutine, perché la farina 0 viene sostituita con un mix gluten free;
- vegetariana, perché il prosciutto utilizzato è vegetale;
- con impasto invariato nelle quantità;
- con il "prosciutto cotto" ancora presente nella funzione della ricetta;
- con valore energetico stimato intorno a 780–800 kcal.

DeepSeek

Mi dispiace, ma la richiesta presenta alcune contraddizioni interne che la rendono impossibile da soddisfare così com'è. Analizzando i vincoli indicati, emergono infatti alcune incompatibilità.

1. Pizza vegetariana con prosciutto cotto
Il prosciutto cotto è un derivato animale, essendo carne di maiale. Per questo motivo non può essere presente in una pizza vegetariana. Se si desidera ottenere una pizza vegetariana, il prosciutto deve essere eliminato oppure sostituito con un'alternativa vegetale.

2. Impasto senza glutine con le stesse quantità della ricetta originale
L'impasto originale utilizza 166,7 g di farina 0 W300, che contiene glutine. Per ottenere una pizza senza glutine è necessario utilizzare una farina specifica, ad esempio un mix senza glutine, che presenta un diverso assorbimento dell'acqua e caratteristiche nutrizionali differenti. Di conseguenza, non è possibile mantenere la stessa farina e ottenere allo stesso tempo un impasto senza glutine.

3. Calorie massime pari a 800 kcal con impasto originale
Solo l'impasto originale, insieme a mozzarella e pomodoro, supera già il limite calorico indicato. Una stima approssimativa è la seguente:

- Farina 0 W300, 166,7 g: circa 608 kcal;
- Olio extravergine d'oliva, 10 g: circa 90 kcal;
- Mozzarella, 150 g: circa 420 kcal;
- Pomodori pelati, 66,7 g: circa 16 kcal.

Il totale parziale risulta quindi pari a circa 1134 kcal, superando già il limite di 800 kcal prima ancora di aggiungere prosciutto cotto e altri condimenti. Per scendere sotto tale soglia sarebbe necessario ridurre almeno uno tra farina, olio o mozzarella.

Cosa posso fare per te
Per proporre una soluzione reale, è necessario stabilire quali vincoli siano prioritari e quali possano essere modificati. Alcune possibilità sono:

- una pizza senza glutine e vegetariana, quindi senza prosciutto cotto, con massimo 800 kcal;
- una pizza con prosciutto cotto e impasto originale, cercando di ridurre il più possibile le calorie, ma senza rispettare necessariamente il limite di 800 kcal;
- una pizza senza glutine e con prosciutto cotto, accettando però che non sia vegetariana e che l'impasto venga modificato.

Una volta stabiliti gli aspetti prioritari, sarà possibile preparare una ricetta più vicina possibile alla richiesta iniziale, ma coerente e realizzabile.

Claude

Pizza Capricciosa senza glutine, vegetariana con prosciutto, impasto invariato, $\leq$ 800 kcal
Stessa nota tecnica delle volte precedenti: il prosciutto cotto è di origine animale, quindi rigorosamente la pizza non è vegetariana. Interpreto il vincolo come "senza glutine e senza altri ingredienti animali oltre a mozzarella e prosciutto cotto". Se si intendesse una versione vegetariana stretta, sarebbe sufficiente eliminare il prosciutto cotto.
Il punto critico riguarda l'impasto: viene richiesto che rimanga invariato nelle quantità, ma la farina 0 contiene glutine. L'unica lettura coerente è quindi mantenere gli stessi pesi sostituendo la farina 0 con un mix senza glutine. Le proporzioni non cambiano: cambia soltanto la natura della farina.

Dosi per 1 pizza – peso circa 590 g

Ingrediente	Quantità	Note
Mix farine senza glutine per pizza	166,7 g	Quantità invariata, sostituisce la farina 0
Acqua	116,7 g	Invariata, vincolo
Lievito di birra fresco	0,83 g	Invariato, vincolo
Sale fino	4,17 g	Invariato, vincolo
Olio extravergine d'oliva	10,0 g	Invariato, vincolo
Pomodori pelati	90,0 g	+23,3 g
Mozzarella	150,0 g	Invariata
Prosciutto cotto	33,3 g	Invariato, vincolo
Champignon freschi	50,0 g	Sostituiscono i funghi sott'olio
Carciofi al naturale	30,0 g	Sostituiscono i carciofi sott'olio
Olive nere denocciolate	4,0 g	$-2,7$ g
Origano e basilico fresco	q.b.	Sapore senza incremento calorico rilevante

Valori nutrizionali stimati

Nutriente	Valore	Confronto con l'originale
Energia	$\sim$795 kcal	-300 kcal (-27%)
Proteine	$\sim$48 g	-7 g
Lipidi	$\sim$30 g	$-15,5$ g
Carboidrati	$\sim$103 g	-17 g
Sodio	$\sim$2000 mg	-275 mg
Fibre totali	$\sim$7,5 g	circa invariate

Logica delle sostituzioni

- **Farina 0 → mix senza glutine**: la quantità rimane invariata, pari a 166,7 g. I mix senza glutine commerciali per pizza hanno una densità calorica simile alla farina 0, circa 340–360 kcal per 100 g, quindi a parità di peso le calorie dell'impasto restano quasi identiche. Cambia però la natura dell'ingrediente, rendendo la base priva di glutine.
- **Nota tecnica sull'impasto**: a parità di peso, le farine senza glutine assorbono più acqua rispetto alla farina di grano tenero. Con 116,7 g di acqua e 166,7 g di mix senza glutine, l'impasto risulterà più sodo, lavorabile ma poco idratato. Per renderlo più adatto alla cottura si potrebbero aggiungere 2–3 g di psillio in polvere, che aiuta a trattenere l'acqua disponibile e a dare struttura. Lo psillio non viene considerato parte dell'impasto in senso stretto e presenta un apporto calorico trascurabile, pari a circa 3 kcal. Se non si vuole aggiungerlo, l'impasto resterà più compatto e meno alveolato rispetto al classico, ma comunque utilizzabile.
- **Funghi sott'olio → champignon freschi**: la quantità passa da 16,7 g a 50 g. I funghi freschi hanno un apporto calorico nettamente inferiore rispetto a quelli sott'olio, permettendo di aumentare la quantità e ottenere più volume e fibre.
- **Carciofi sott'olio → carciofi al naturale**: la quantità passa da 13,3 g a 30 g. Anche in questo caso viene eliminato l'olio di conservazione, riducendo l'apporto calorico pur aumentando la quantità dell'ingrediente.
- **Pomodori e olive**: i pomodori pelati vengono aumentati di 23,3 g, mentre le olive vengono ridotte di 2,7 g. Questi aggiustamenti servono a bilanciare visivamente la pizza e a mantenere la nota saporita tipica della capricciosa.

Il taglio calorico principale, pari a circa 170 kcal, deriva dall'eliminazione dell'olio di conservazione di funghi e carciofi, che incide molto sul totale pur essendo poco evidente. Impasto, mozzarella e prosciutto cotto restano invece invariati come richiesto dai vincoli.
A questo punto emerge però una criticità: impasto, mozzarella e prosciutto cotto rappresentano già una quota calorica molto elevata. Nonostante ciò, secondo la stima proposta, il totale finale si collocherebbe intorno a 795 kcal, rientrando quindi nel limite massimo di 800 kcal.

Suggerimento di preparazione
L'impasto senza glutine con questa idratazione, pari a circa il 70%, sarà denso. Conviene mescolare farina e sale a secco, sciogliere il lievito nell'acqua tiepida con l'olio, unire gli ingredienti e impastare brevemente. Non essendoci glutine da sviluppare, non è necessario lavorarlo a lungo.
Dopo un riposo di 40–60 minuti coperto, l'impasto può essere steso sulla teglia oliata con le mani umide. Gli champignon vanno affettati sottili e saltati per 2–3 minuti senza grassi, così da ridurre l'acqua in eccesso. I carciofi devono essere scolati e tamponati. Il prosciutto cotto può essere aggiunto negli ultimi 2 minuti di cottura, per evitare che si secchi.

Contaminazione crociata
In caso di celiachia è necessario verificare che mozzarella, prosciutto cotto, pomodori pelati, carciofi e olive riportino la dicitura "senza glutine". Alcuni salumi industriali possono infatti contenere addensanti o tracce di glutine.

Gemini

Questo è il livello definitivo del gioco: un vero e proprio rompicapo matematico e gastronomico. Per rispettare tutti i vincoli indicati, è necessario risolvere due paradossi logici:

- **Impasto identico ma senza glutine:** per mantenere le quantità al grammo, pari a 166,7 g, la farina 0 viene sostituita con un mix di farine senza glutine. Restano invece invariati acqua, olio, sale e lievito. L'impasto da solo, anche a causa dei 10 g di olio extravergine d'oliva, apporta circa 674 kcal.
- **Pizza vegetariana con prosciutto cotto:** poiché il prosciutto è carne, l'unico modo per rispettare entrambi i vincoli è utilizzare un prosciutto cotto vegetale, cioè un prodotto plant-based.

Avendo l'impasto fissato a circa 674 kcal, restano appena 126 kcal per tutti gli altri ingredienti. Per rientrare nel limite massimo di 800 kcal, la mozzarella viene ridotta a 35 g nella versione light e tutti i condimenti sott'olio vengono sostituiti con versioni fresche o al naturale.

Nuova ricetta: Capricciosa Extreme
Senza glutine, vegetariana, massimo 800 kcal

Ingrediente	Quantità	Note / Modifiche
Mix di farine senza glutine	166,7 g	Quantità identica, cambia solo il tipo di farina
Acqua	116,7 g	Invariata, vincolo dell'impasto
Lievito di birra fresco	0,83 g	Invariato, vincolo dell'impasto
Sale fino	4,17 g	Invariato, vincolo dell'impasto
Olio extravergine d'oliva	10,0 g	Invariato, vincolo dell'impasto
Pomodori pelati	66,7 g	Invariati
Prosciutto cotto vegetale	33,3 g	Quantità invariata, versione plant-based
Mozzarella light	35,0 g	Ridotta per non superare il limite calorico
Funghi champignon freschi	16,7 g	Sostituiscono i funghi sott'olio
Carciofini al naturale	13,3 g	Sostituiscono i carciofini sott'olio
Olive nere denocciolate	6,7 g	Invariate

Analisi nutrizionale della ricetta modificata

Nutriente	Valore stimato	Osservazione
Energia	∼798 kcal	Sotto il limite massimo di 800 kcal
Proteine	∼28 g	Ridotte rispetto alla ricetta originale
Lipidi	∼21 g	Quasi totalmente derivanti dall'olio EVO dell'impasto
Carboidrati	∼125 g	Dovuti soprattutto agli amidi del mix senza glutine
Fibre totali	∼8,5 g	Mantenute grazie a verdure e mix senza glutine

Il trucco per la cottura
L'idratazione al 70%, ottenuta con 116,7 g di acqua su 166,7 g di farina, è adatta alle farine senza glutine, che assorbono molti liquidi. L'impasto risulterà molto morbido: conviene stenderlo direttamente sulla teglia unta, aiutandosi con le mani umide.
Dato che la mozzarella è presente in quantità molto ridotta, pari a 35 g, se venisse aggiunta subito rischierebbe di asciugarsi eccessivamente. Per questo motivo è preferibile cuocere inizialmente la base solo con pomodoro, carciofini e olive. Negli ultimi 3 minuti di cottura si può sfornare la pizza, aggiungere il prosciutto vegetale, i funghi freschi tagliati molto sottili e la mozzarella light tritata finemente, quindi rimettere in forno sotto il grill alla massima potenza. In questo modo si cerca di ottenere un risultato visivamente più ricco e filante, nonostante il taglio calorico significativo.

Confronto delle risposte

In questo prompt, rispetto al precedente, è stato introdotto un ulteriore vincolo: oltre a richiedere una pizza senza glutine, vegetariana, con un apporto energetico massimo di 800 kcal e con il prosciutto cotto ancora presente, viene specificato che l'impasto debba mantenere inalterate tutte le quantità degli ingredienti originari. Anche in questo caso, gli approcci adottati dai modelli differiscono notevolmente:

- *ChatGPT* mantiene un approccio orientato alla ricerca di una soluzione pratica, analogamente a quanto osservato nel prompt precedente. Il modello ribadisce innanzitutto che la richiesta può essere soddisfatta soltanto interpretando il prosciutto cotto come una sua variante vegetale. Dal punto di vista del rispetto dei vincoli, la soluzione risulta coerente e rappresenta un tentativo concreto di soddisfare tutte le richieste. Tuttavia, la drastica riduzione della mozzarella compromette in parte l'equilibrio gastronomico della ricetta, trasformando la pizza in una preparazione poco realistica rispetto a una tradizionale capricciosa.

- *DeepSeek* dopo aver analizzato la richiesta, conclude che l'insieme dei vincoli sia incompatibile e decide di non produrre alcuna ricetta, limitandosi a spiegare nel dettaglio le motivazioni della propria decisione.

- *Claude* mantiene l'impostazione osservata nel primo prompt. Nella risposta, però, emergono alcune criticità. Da un lato, continua a non rispettare il requisito della pizza vegetariana; dall'altro, la stima calorica proposta appare poco plausibile. Il modello dichiara, infatti, di mantenere invariati ingredienti particolarmente calorici, stimando comunque un valore energetico finale di circa 795 kcal. Tale risultato risulta difficilmente compatibile con i dati nutrizionali della ricetta originale, suggerendo una sottostima delle calorie effettivamente apportate dagli ingredienti mantenuti invariati.

- *Gemini* adotta lo stesso approccio del prompt precedente. La soluzione proposta prevede l'utilizzo di prosciutto vegetale, una forte riduzione della mozzarella e la sostituzione degli ingredienti sott'olio con equivalenti freschi.

Nel complesso, questo secondo scenario mette in evidenza come l'aumento del numero di vincoli accentui le differenze tra i modelli. Alcuni privilegiano la soddisfazione della richiesta attraverso reinterpretazioni e compromessi, altri preferiscono evidenziare i limiti del problema.

3.3.3 Terzo prompt

Utente

PIZZA CAPRICCIOSA - Dosi per 1 pizza (peso 585,1g)

- Farina 0 W300 166,7 g
- Acqua 116,7 g
- Lievito di birra fresco 0,83 g
- Sale fino 4,17 g
- Olio extravergine d'oliva 10,0 g
- Pomodori pelati 66,7 g
- Mozzarella 150,0 g
- Prosciutto cotto 33,3 g
- Funghi sott'olio 16,7 g
- Carciofi sott'olio 13,3 g
- Olive nere denocciolate 6,7 g

Valori nutrizionali:

- Energia (kcal): 1095,38
- Proteine (g): 55,10
- Lipidi (g): 45,56
- Carboidrati (g): 120,20
- Sodio (mg): 2273,73
- Fibre totali (g): 7,45

Voglio che la mia pizza, partendo dalla ricetta che ti ho mandato, venga modificata affinché sia senza glutine, con almeno 1500 calorie e che impasto, funghi, carciofi e mozzarella siano sempre uguali nelle loro rispettive quantità.

Risposte al terzo prompt

ChatGPT

Pizza capricciosa senza glutine da almeno 1500 kcal

Dosi per 1 pizza

Ingrediente	Quantità
Mix per pizza senza glutine	166,7 g
Acqua	116,7 g
Lievito di birra fresco	0,83 g
Sale fino	4,17 g
Olio extravergine d'oliva	10,0 g
Pomodori pelati	66,7 g
Mozzarella	150,0 g
Prosciutto cotto	33,3 g
Funghi sott'olio	16,7 g
Carciofi sott'olio	13,3 g
Olive nere denocciolate	6,7 g
Scamorza affumicata	90 g
Olio extravergine d'oliva aggiunto a crudo	20 g

Ingredienti mantenuti invariati

Ingrediente/parte	Quantità originale	Nuova quantità
Farina / mix per impasto	166,7 g	166,7 g
Acqua	116,7 g	116,7 g
Lievito	0,83 g	0,83 g
Sale	4,17 g	4,17 g
Olio EVO dell'impasto/ricetta base	10,0 g	10,0 g
Mozzarella	150,0 g	150,0 g
Funghi sott'olio	16,7 g	16,7 g
Carciofi sott'olio	13,3 g	13,3 g

Modifiche fatte

Ricetta originale	Nuova versione
Farina 0 W300 166,7 g	Mix senza glutine 166,7 g
Ricetta da 1095,38 kcal	Ricetta portata sopra 1500 kcal
Nessun ingrediente extra calorico	Aggiunta di scamorza e olio EVO a crudo

Stima calorica
La pizza originale presenta un valore energetico pari a:

1095,38 kcal

A questa base vengono aggiunti due ingredienti ad alta densità calorica:

Ingrediente aggiunto	Quantità	kcal stimate
Scamorza affumicata	90 g	circa 270 kcal
Olio EVO a crudo	20 g	circa 180 kcal

L'aumento calorico stimato è quindi:

270 + 180 = 450 kcal

Di conseguenza:

1095,38 + 450 = circa 1545 kcal

Valori nutrizionali stimati finali

Valore	Pizza originale	Nuova pizza stimata
Energia	1095,38 kcal	circa 1520–1560 kcal
Proteine	55,10 g	circa 75–80 g
Lipidi	45,56 g	circa 80–90 g
Carboidrati	120,20 g	circa 120–130 g
Sodio	2273,73 mg	circa 2700–3200 mg
Fibre totali	7,45 g	circa 7–9 g

Nota importante sul senza glutine
Per essere realmente senza glutine, è necessario utilizzare:

- un mix per pizza certificato senza glutine;
- prosciutto cotto certificato senza glutine;
- scamorza certificata senza glutine;
- funghi, carciofi e olive controllati per eventuali contaminazioni o additivi.

La pizza finale risulta quindi una capricciosa senza glutine ipercalorica, con impasto, mozzarella, funghi e carciofi mantenuti nelle quantità richieste, e con energia stimata intorno a 1545 kcal.

DeepSeek

Per soddisfare la richiesta, cioè ottenere una pizza senza glutine, con almeno 1500 kcal e con le quantità di impasto, mozzarella, funghi e carciofi identiche all'originale, occorre aggiungere ingredienti ad alta densità energetica. La sola sostituzione della farina 0 con una farina senza glutine non è infatti sufficiente, poiché l'impasto senza glutine ha un apporto calorico simile, talvolta leggermente inferiore, rispetto a quello classico.
La soluzione consiste quindi nel mantenere invariati i grammi dell'impasto, della mozzarella, dei funghi sott'olio e dei carciofi sott'olio, aggiungendo calorie extra tramite:

- prosciutto cotto, aumentato rispetto alla ricetta originale;
- olio extravergine d'oliva aggiunto a crudo;
- scaglie di Grana Padano o Parmigiano Reggiano, per aumentare sapore e apporto calorico.

PIZZA CAPRICCIOSA SENZA GLUTINE – ALMENO 1500 KCAL
Dosi per 1 pizza – peso circa 620 g

Ingrediente	Quantità
Impasto, identico nelle quantità	
Farina senza glutine, mix pane/pizza	166,7 g
Acqua	116,7 g
Lievito di birra fresco	0,83 g
Sale fino	4,17 g
Olio extravergine d'oliva	10,0 g
Condimenti fissi, invariati	
Pomodori pelati	66,7 g
Mozzarella fiordilatte o vaccina	150,0 g
Funghi sott'olio	16,7 g
Carciofi sott'olio	13,3 g
Olive nere denocciolate	6,7 g
Ingredienti aggiuntivi per raggiungere almeno 1500 kcal	
Prosciutto cotto a dadini	70,0 g
Olio extravergine d'oliva a crudo	10,0 g
Scaglie di Grana Padano	15,0 g

Valori nutrizionali stimati
Valori riferiti alla pizza intera

Nutriente	Originale	Versione senza glutine $\geq$ 1500 kcal
Energia	1095,38 kcal	1512 kcal
Proteine	55,10 g	62,3 g
Lipidi	45,56 g	69,2 g
Carboidrati	120,20 g	136,8 g
Fibre totali	7,45 g	5,8 g
Sodio	2273,73 mg	$\sim$2450 mg

Perché queste modifiche?

- L'impasto viene reso senza glutine sostituendo la farina 0 con un mix senza glutine, mantenendo però esattamente le stesse quantità di farina, acqua, lievito, sale e olio.
- Mozzarella, funghi e carciofi rimangono presenti nelle quantità originali, rispettando i vincoli indicati.
- L'energia viene portata oltre la soglia richiesta grazie all'aumento del prosciutto cotto, all'aggiunta di olio extravergine d'oliva a crudo e alle scaglie di Grana Padano. Questi ingredienti aggiungono circa 300 kcal complessive senza modificare gli elementi vincolati.

La pizza mantiene quindi il profilo della Capricciosa, arricchito da una maggiore quantità di prosciutto e dalla nota sapida del formaggio stagionato. Il risultato è una versione senza glutine più sostanziosa e calorica, coerente con il limite minimo richiesto di 1500 kcal.

Claude

Pizza Capricciosa senza glutine, $\geq$ 1500 kcal
Vincoli rispettati: pizza senza glutine, impasto, funghi, carciofi e mozzarella invariati nelle quantità, con valore energetico minimo pari a 1500 kcal. Per passare da circa 1095 kcal ad almeno 1500 kcal è necessario aggiungere circa 405 kcal, intervenendo su prosciutto, olive,

pomodori e introducendo alcuni ingredienti extra coerenti con il profilo della pizza capricciosa.

Dosi per 1 pizza – peso circa 720 g

Ingrediente	Quantità	Note
Mix farine senza glutine per pizza	166,7 g	Quantità invariata, sostituisce la farina 0
Acqua	116,7 g	Invariata, vincolo
Lievito di birra fresco	0,83 g	Invariato, vincolo
Sale fino	4,17 g	Invariato, vincolo
Olio extravergine d'oliva	10,0 g	Invariato, vincolo
Pomodori pelati	66,7 g	Invariati
Mozzarella	150,0 g	Invariata, vincolo
Funghi sott'olio	16,7 g	Invariati, vincolo
Carciofi sott'olio	13,3 g	Invariati, vincolo
Prosciutto cotto	60,0 g	+26,7 g
Olive nere denocciolate	20,0 g	+13,3 g
Uovo intero	55,0 g	1 uovo medio, aggiunto come da tradizione capricciosa romana
Scaglie di Grana Padano	25,0 g	A crudo, a fine cottura
Olio EVO a crudo	10,0 g	Aggiunto a fine cottura
Origano e basilico	q.b.	

Valori nutrizionali stimati

Nutriente	Valore	Confronto con l'originale
Energia	∼1510 kcal	+415 kcal (+38%)
Proteine	∼78 g	+23 g
Lipidi	∼76 g	+30 g
Carboidrati	∼123 g	+3 g
Sodio	∼2950 mg	+675 mg
Fibre totali	∼8,2 g	+0,8 g

Logica delle aggiunte

- **Farina 0 → mix senza glutine**: la quantità rimane invariata, pari a 166,7 g. I mix senza glutine per pizza hanno una densità calorica simile alla farina tradizionale, circa 340 kcal per 100 g, quindi le calorie dell'impasto restano sostanzialmente invariate. Dal punto di vista tecnico, con un'idratazione del 70% l'impasto senza glutine risulterà piuttosto denso; eventualmente si possono aggiungere 2–3 g di psillio per migliorarne la struttura, con calorie trascurabili.
- **Prosciutto cotto**: la quantità passa da 33,3 g a 60 g, con un aumento stimato di circa 40 kcal. In questo modo aumentano proteine e sapidità, mantenendo un ingrediente caratteristico della pizza capricciosa.
- **Olive nere**: la quantità passa da 6,7 g a 20 g, con un aumento stimato di circa 30 kcal. Una quantità più generosa di olive resta coerente con il profilo della capricciosa e contribuisce ad aumentare i lipidi.
- **Uovo intero**: l'aggiunta di un uovo medio apporta circa 85 kcal, circa 6 g di proteine e circa 6 g di grassi. Si tratta di un ingrediente coerente con alcune versioni tradizionali della capricciosa, in particolare quella romana.
- **Grana Padano in scaglie**: aggiunto a crudo a fine cottura, apporta circa 98 kcal, circa 9 g di proteine e una nota sapida intensa. Contribuisce ad arricchire il profilo lattiero della pizza senza modificare la quantità di mozzarella, che rimane invariata.
- **Olio extravergine d'oliva a crudo**: l'aggiunta di 10 g di olio EVO a fine cottura apporta circa 88 kcal. Si tratta di una fonte calorica concentrata, coerente dal punto di vista gastronomico, che esalta i sapori e rende la pizza più ricca.

Il totale delle calorie aggiunte è pari a circa 340 kcal dai nuovi topping, più alcuni aggiustamenti minori, portando la pizza a circa 1510 kcal. In questo modo viene superata la soglia minima richiesta di 1500 kcal con un piccolo margine di sicurezza.

Suggerimento di preparazione
L'impasto senza glutine va lavorato come una pastella densa. Dopo un riposo di 40–60 minuti, può essere steso direttamente sulla teglia oliata con le mani umide. Funghi e carciofi sott'olio vanno scolati leggermente per ridurre l'olio in eccesso.
L'uovo va rotto al centro della pizza a metà cottura, dopo circa 5 minuti, per evitare che si rapprenda troppo: l'albume deve risultare cotto, mentre il tuorlo può rimanere ancora morbido. Il prosciutto cotto va aggiunto negli ultimi 2 minuti di cottura, così da non seccarsi. Grana Padano e olio EVO a crudo vanno invece aggiunti solo a pizza sfornata.

Attenzione al sodio
Con circa 2950 mg di sodio, questa pizza supera il limite giornaliero raccomandato dall'OMS, pari a circa 2000 mg. Si tratta di un valore prevedibile per una pizza molto ricca e sapida; in caso di consumo, sarebbe opportuno mantenere gli altri pasti della giornata poveri di sale.

Contaminazione crociata
Per garantire che la pizza sia realmente senza glutine, è necessario verificare che tutti gli ingredienti confezionati, come mozzarella, prosciutto cotto, pomodori pelati, sott'oli e Grana Padano, riportino la dicitura "senza glutine".

Gemini

Visto che i principali elementi della ricetta sono bloccati nelle loro quantità originali, apportando circa 1120 kcal, per raggiungere e superare la soglia delle 1500 kcal è necessario intervenire sulle variabili rimaste libere. In particolare, vengono aumentati nettamente il prosciutto cotto e le olive nere, aggiungendo anche un generoso giro di olio extravergine d'oliva a crudo all'uscita dal forno.

Nuova ricetta: Capricciosa Gluten-Free Energy
Dosi per 1 pizza – circa 1550 kcal

Ingrediente	Quantità	Note / Stato del vincolo
Mix di farine senza glutine	166,7 g	Quantità identica, cambia solo il tipo di farina
Acqua	116,7 g	Invariata, vincolo dell'impasto
Lievito di birra fresco	0,83 g	Invariato, vincolo dell'impasto
Sale fino	4,17 g	Invariato, vincolo dell'impasto
Olio extravergine d'oliva nell'impasto	10,0 g	Invariato, vincolo dell'impasto
Pomodori pelati	66,7 g	Invariati
Mozzarella	150,0 g	Invariata, versione classica intera
Funghi sott'olio	16,7 g	Invariati
Carciofi sott'olio	13,3 g	Invariati
Prosciutto cotto	150,0 g	Aumentato, super farcitura per incrementare le calorie
Olive nere denocciolate	46,7 g	Aumentate, aggiungono grassi e calorie
Olio extravergine d'oliva a crudo	20,0 g	Nuovo elemento, aggiunto a fine cottura

Nota
Il peso complessivo della pizza sale a circa 715 g a causa del surplus di condimenti necessario per raggiungere il target calorico richiesto.

Analisi nutrizionale stimata

Nutriente	Valore stimato	Osservazione
Energia	∼1554 kcal	Target centrato, oltre la soglia minima di 1500 kcal
Proteine	∼75 g	Aumentate grazie al prosciutto e alla mozzarella intera
Lipidi	∼76 g	Derivano da olio a crudo, olive e ingredienti sott'olio
Carboidrati	∼125 g	Dovuti principalmente al mix senza glutine
Fibre totali	∼10,5 g	Incrementate soprattutto dalle olive e dagli ingredienti vegetali

Consigli per la cottura con super-carico
Dato che la pizza risulta molto ricca di ingredienti e l'impasto senza glutine necessita di calore diretto per non rimanere umido al centro, è consigliabile procedere in più fasi.

- **Stesura:** stendere l'impasto senza glutine direttamente sulla teglia ben unta.
- **Prima cottura della base:** infornare la base solo con il pomodoro e i carciofini nella parte più bassa del forno, alla massima temperatura, per circa 10 minuti. Questo permette di ottenere una base più croccante e strutturata, capace di sostenere il peso dei condimenti successivi.
- **Farcitura finale:** sfornare momentaneamente la base, distribuire la mozzarella, i funghi, le olive e il prosciutto cotto, quindi rimettere in forno nella parte medio-alta per altri 4–5 minuti, fino a completa fusione della mozzarella.
- **Aggiunta dell'olio a crudo:** appena la pizza esce dal forno, versare i 20 g di olio extravergine d'oliva a filo su tutta la superficie. Oltre a completare il profilo calorico richiesto, l'olio a crudo esalta i profumi del prosciutto e dei carciofi.

Confronto delle risposte

Nel terzo prompt, viene modificata completamente la natura del problema rispetto ai due prompt precedenti. Tutti i modelli eseguono correttamente la richiesta generando delle ricette che rispettano i vincoli dati. Alcune, però, presentano problemi, come nel caso di *ChatGPT* dove la grande quantità di formaggio e olio aggiunta produce una pizza sensibilmente più ricca di grassi.

DeepSeek mantiene invariati tutti gli ingredienti soggetti ai vincoli e distribuisce l'aumento energetico su più alimenti della ricetta, aggiungendone anche di nuovi.

Claude mantiene un approccio dettagliato. Il modello rispetta i vincoli e arricchisce la ricetta con nuovi alimenti spiegandone ampiamente i motivi. La risposta risulta essere, però, estremamente lunga.

Gemini aumenta in maniera significativa la quantità di alcuni alimenti della ricetta mantenendo una maggiore identità della ricetta originale rispetto alle altre risposte analizzate. La ricetta finale, però risulta estremamente sbilanciata verso il prosciutto cotto, compromettendo l'equilibrio della farcitura della pizza.

Nel complesso, l'assenza di contraddizioni logiche porta tutti i modelli a individuare una soluzione concreta, senza necessità di reinterpretare la richiesta o evidenziarne le criticità. Le differenze emergono soprattutto nella strategia scelta per incrementare il valore energetico della ricetta.

3.4 Scenario 2: adattamento della pizza patate e salsiccia

3.4.1 Prompt iniziale

Utente

PIZZA PATATE E SALSICCIA - Dosi per 1 pizza (peso 585,1g)

- Farina 0 W300 166,7 g
- Acqua 116,7 g
- Lievito di birra fresco 0,5 g
- Sale fino 5,0 g
- Olio extravergine d'oliva 5,5 g
- Patate 66,7 g
- Salsiccia napoletana 50,0 g
- Mozzarella di bufala DOP 100,0 g
- Mozzarella di bufala bocconcini 80,0 g

Valori nutrizionali:

- Energia (kcal): 1248,73
- Proteine (g): 53,51
- Lipidi (g): 59,64
- Carboidrati (g): 129,93
- Sodio (mg): 2937,88
- Fibre totali (g): 5,73

Partendo da quantità e valori nutrizionali dati, voglio che adatti la mia ricetta, pur rimanendo fedele all'idea della pizza, per poterla avere di massimo 700 calorie, senza alimenti per gli intolleranti al lattosio e che le quantità dell'impasto rimangano invariate.

Risposte al prompt iniziale

ChatGPT

Pizza patate e salsiccia adattata
Obiettivo: massimo 700 kcal, senza lattosio, impasto invariato.

Ingrediente	Quantità originale	Nuova quantità
Farina 0	166,7 g	166,7 g
Acqua	116,7 g	116,7 g
Lievito di birra fresco	0,5 g	0,5 g
Sale fino	5,0 g	5,0 g
Olio extravergine d'oliva	5,5 g	5,5 g
Patate	66,7 g	45 g
Salsiccia napoletana	50,0 g	10 g
Mozzarella di bufala DOP	100,0 g	rimossa
Mozzarella di bufala bocconcini	80,0 g	rimossa
Mozzarella senza lattosio/light senza lattosio	–	15 g

Valori nutrizionali stimati della nuova pizza

Valore	Stima nuova pizza
Energia	circa 680–695 kcal
Proteine	circa 24–26 g
Lipidi	circa 11–13 g
Carboidrati	circa 124–127 g
Sodio	circa 2150–2250 mg
Fibre totali	circa 5–6 g

Perché sono state fatte queste modifiche
L'impasto è stato lasciato invariato: farina, acqua, lievito, sale e olio restano uguali rispetto alla ricetta originale.
Il principale taglio calorico deriva dalle seguenti modifiche:

- **Rimozione della mozzarella di bufala**: i 180 g complessivi tra mozzarella di bufala DOP e bocconcini incidono molto sul valore energetico totale e, inoltre, non sono adatti agli intolleranti al lattosio.
- **Sostituzione con poca mozzarella senza lattosio**: viene inserita una piccola quantità di mozzarella senza lattosio, preferibilmente light, per mantenere comunque l'idea della pizza con una componente filante.
- **Riduzione della salsiccia**: la salsiccia napoletana passa da 50 g a 10 g. Si tratta di un ingrediente calorico e ricco di grassi, ma lasciarne una piccola quantità permette di mantenere il richiamo alla pizza patate e salsiccia.
- **Riduzione delle patate**: le patate passano da 66,7 g a 45 g. In questo modo restano presenti nella ricetta, ma senza far superare il limite calorico stabilito.

Ricetta finale consigliata
Pizza patate e salsiccia senza lattosio sotto le 700 kcal

- Farina 0: 166,7 g;
- Acqua: 116,7 g;
- Lievito di birra fresco: 0,5 g;
- Sale fino: 5,0 g;
- Olio extravergine d'oliva: 5,5 g;
- Patate: 45 g;
- Salsiccia napoletana senza lattosio: 10 g;
- Mozzarella senza lattosio light: 15 g;
- Rosmarino, pepe, origano o spezie: a piacere.

La pizza rimane fedele all'idea originale perché conserva patate, salsiccia e una componente filante. Allo stesso tempo, diventa compatibile con l'intolleranza al lattosio e resta sotto le 700 kcal.

DeepSeek

Ecco la ricetta adattata della pizza patate e salsiccia senza lattosio e con massimo 700 kcal, mantenendo invariate le quantità dell'impasto.

Ricetta adattata
Dosi per 1 pizza – peso impasto invariato

Ingrediente	Quantità	Note
Farina 0	166,7 g	Invariato
Acqua	116,7 g	Invariato
Lievito di birra fresco	0,5 g	Invariato
Sale fino	5,0 g	Invariato
Olio extravergine d'oliva	5,5 g	Invariato
Patate	60,0 g	Leggermente ridotte, da 66,7 g
Salsiccia senza lattosio, magra di pollo/tacchino	26,0 g	Invece di 50 g di salsiccia napoletana
Mozzarella di bufala o bocconcini	0 g	Eliminata per assenza di lattosio e riduzione calorica

Nota
La salsiccia deve essere specificamente senza lattosio, verificando in etichetta l'assenza di latte o derivati. Una salsiccia magra di pollo o tacchino ha circa 140 kcal per 100 g.

Valori nutrizionali stimati
Valori riferiti alla pizza intera

Nutriente	Valore
Energia	699,7 kcal
Proteine	∼23,6 g
Lipidi	∼9,2 g
Carboidrati	∼129,5 g
Sodio	∼2000 mg dal sale + salsiccia
Fibre totali	∼5,0 g

Come si ottiene il risultato

- L'impasto rimane identico, con un apporto energetico stimato pari a circa 617 kcal.
- Le patate vengono leggermente ridotte a 60 g, per un apporto stimato di circa 46 kcal.
- La salsiccia viene sostituita con una versione magra e senza lattosio, ridotta a 26 g, per un apporto stimato di circa 36 kcal.
- Tutti i formaggi, quindi mozzarella di bufala e bocconcini, vengono eliminati perché contenenti lattosio e molto calorici.

In questo modo la ricetta resta fedele all'idea della pizza con patate e salsiccia, pur rispettando il limite calorico e l'assenza di lattosio.

Claude

Pizza Patate e Salsiccia – Versione lactose-free sotto le 700 kcal
Ho rivisto la ricetta mantenendo intatto l'impasto, come richiesto, e l'anima del piatto: patate, salsiccia e una base filante. Le modifiche principali riguardano i latticini, sostituiti con alternative senza lattosio, e una riduzione mirata degli ingredienti più calorici.

Ricetta adattata
Dosi per 1 pizza – peso circa 580 g

Ingrediente	Quantità	Note
Farina 0	166,7 g	Invariato
Acqua	116,7 g	Invariato
Lievito di birra fresco	0,5 g	Invariato
Sale fino	5,0 g	Invariato
Olio EVO	5,5 g	Invariato
Patate	66,7 g	Invariato
Salsiccia napoletana	25,0 g	Dimezzata, sbriciolata fine e distribuita bene
Fiordilatte senza lattosio	70,0 g	Sostituisce la mozzarella di bufala DOP
Mozzarella senza lattosio a cubetti	40,0 g	Sostituisce i bocconcini di bufala

Il peso finale stimato è pari a circa 495 g. La riduzione deriva dai topping più grassi e non dall'impasto, che rimane invariato.

Valori nutrizionali stimati

Nutriente	Originale	Adattata	Δ
Energia	1248,73 kcal	∼675 kcal	−574 kcal
Proteine	53,51 g	∼33 g	−20 g
Lipidi	59,64 g	∼25 g	−35 g
Carboidrati	129,93 g	∼128 g	≈
Sodio	2937,88 mg	∼1850 mg	−1090 mg
Fibre totali	5,73 g	∼5,7 g	≈

Logica delle modifiche

- **Lattosio:** la mozzarella di bufala, sia DOP sia in bocconcini, contiene lattosio. Viene quindi sostituita con fiordilatte e mozzarella delattosata, cioè prodotti nei quali l'enzima lattasi ha già scisso il lattosio in glucosio e galattosio. A parità di peso l'apporto calorico è simile, ma il prodotto risulta compatibile con l'intolleranza al lattosio.
- **Calorie:** gli ingredienti più calorici della ricetta originale erano i 180 g complessivi di mozzarella di bufala, pari a circa 530 kcal, e la salsiccia, pari a circa 165 kcal. Per ridurre l'apporto energetico, la quantità complessiva di formaggio viene ridotta da 180 g a 110 g, mentre la salsiccia passa da 50 g a 25 g.
- **Idea della pizza preservata:** i tre elementi principali, cioè patate, salsiccia e formaggio filante, restano presenti. Le patate, pari a 66,7 g, rimangono invariate e contribuiscono a mantenere sazietà e cremosità, compensando parzialmente la riduzione del formaggio.

Consigli di esecuzione

- **Patate:** tagliarle sottili, circa 2 mm, con una mandolina e condirle a crudo con parte dell'olio EVO previsto e un pizzico di rosmarino. In questo modo cuociono direttamente in forno e restano saporite senza aggiungere altri grassi.
- **Salsiccia:** sbriciolarla finemente e distribuirla in piccoli mucchietti su tutta la superficie, così da mantenere alta la percezione di abbondanza anche con metà dose.
- **Formaggio:** aggiungere il fiordilatte senza lattosio negli ultimi 3–4 minuti di cottura, per evitare che rilasci troppa acqua o diventi gommoso.

Se si volesse scendere ulteriormente sotto le 600 kcal, l'unica possibilità senza modificare l'impasto sarebbe ridurre ancora il formaggio, ad esempio arrivando a 80 g totali. Tuttavia, in questo caso la pizza perderebbe parte della sua identità filante.

Gemini

Ecco una versione rivisitata della pizza patate e salsiccia. Per scendere sotto le 700 kcal mantenendo invariata la quantità di impasto, che da sola apporta circa 500 kcal, e azzerando il lattosio, sono stati ottimizzati soprattutto i condimenti.
La mozzarella di bufala e la salsiccia tradizionale sono state sostituite con alternative strategiche, ma comunque saporite, così da non rinunciare del tutto al gusto della combinazione originale.

La nuova ricetta
Dosi per 1 pizza – massimo 700 kcal

Nota sull'impasto: farina, acqua, lievito, sale e olio rimangono identici alla ricetta originale. Il peso totale dell'impasto è pari a circa 294 g.

Ingrediente	Quantità nuova	Nota di adattamento
Farina 0	166,7 g	Invariata
Acqua	116,7 g	Invariata
Lievito di birra fresco	0,5 g	Invariato
Sale fino	5,0 g	Invariato
Olio extravergine d'oliva	5,5 g	Invariato
Patate	60,0 g	Tagliate sottilissime per massima resa visiva
Salsiccia di pollo o tacchino	45,0 g	Sgranata e speziata per ricordare la salsiccia napoletana
Mozzarella delattosata	60,0 g	Sostituisce la bufala per eliminare il lattosio e ridurre i grassi

Nuovi valori nutrizionali stimati
Grazie a questi accorgimenti, l'apporto calorico e i grassi totali vengono quasi dimezzati, mantenendo comunque un buon profilo proteico.

Nutriente	Valore stimato	Osservazione
Energia	∼685 kcal	Obiettivo raggiunto, inferiore a 700 kcal
Proteine	∼38 g	Buon apporto proteico
Lipidi	∼21 g	Netto taglio rispetto ai 59 g iniziali
Carboidrati	∼132 g	Leggermente superiori alla ricetta originale

Il trucco per non perdere il sapore
Poiché la mozzarella di bufala e la salsiccia napoletana originale sono ingredienti molto saporiti, ma anche ricchi di grassi, per evitare che la pizza risulti troppo "dietetica" al palato si possono adottare alcuni accorgimenti.

- **Speziare la salsiccia light:** utilizzando una salsiccia di pollo o tacchino, oppure del macinato magro, è consigliabile condirla prima con pepe nero, semi di finocchietto e un pizzico di aglio in polvere. In questo modo si richiama il sapore tipico della salsiccia napoletana.
- **Rosolare le patate:** le patate possono essere tagliate a fette sottilissime e cotte per pochi minuti in forno o in friggitrice ad aria prima di essere aggiunte sulla pizza. In questo modo acquistano consistenza e sapore senza richiedere altro olio.

Confronto delle risposte

In questo prompt, vengono effettuate delle richieste per adattare una ricetta a nuove esigenze nutrizionali per osservare se i modelli riescono a rispettare i vincoli dati e in che modo scelgono di intervenire sulla ricetta. *ChatGPT* opta per una strategia in cui ogni modifica viene effettuata esclusivamente quando necessaria per soddisfare il vincolo richiesto. *DeepSeek* cerca di mantenere un buon equilibrio tra valore nutrizionale, gusto e aspetto finale

della pizza, evitando modifiche eccessivamente drastiche; seppur non esponga più di tanto il motivo delle sue scelte. *Claude* adotta l'approccio visto anche nello scenario precedente: ogni modifica viene spiegata nel dettaglio giustificandola in ogni tipo di aspetto. La risposta viene completata anche da suggerimenti riguardanti la cottura ed eventuali problemi riscontrabili. *Gemini* espone inizialmente la motivazione delle modifiche applicate agli ingredienti, fornendo una risposta equilibrata tra sintesi e completezza nelle informazioni.

3.4.2 Secondo prompt

Utente

PIZZA PATATE E SALSICCIA - Dosi per 1 pizza (peso 585,1g)

- Farina 0 W300 166,7 g
- Acqua 116,7 g
- Lievito di birra fresco 0,5 g
- Sale fino 5,0 g
- Olio extravergine d'oliva 5,5 g
- Patate 66,7 g
- Salsiccia napoletana 50,0 g
- Mozzarella di bufala DOP 100,0 g
- Mozzarella di bufala bocconcini 80,0 g

Valori nutrizionali:

- Energia (kcal): 1248,73
- Proteine (g): 53,51
- Lipidi (g): 59,64
- Carboidrati (g): 129,93
- Sodio (mg): 2937,88
- Fibre totali (g): 5,73

Partendo da quantità e valori nutrizionali dati, voglio che adatti la mia ricetta, pur rimanendo fedele all'idea della pizza, per poterla avere di almeno 1900 calorie, senza alimenti per gli intolleranti al lattosio e che le quantità di impasto, salsiccia e patate rimangano invariate.

Risposte al secondo prompt

ChatGPT

Pizza patate e salsiccia adattata senza lattosio

Ingrediente	Quantità originale	Nuova quantità
Farina 0	166,7 g	166,7 g
Acqua	116,7 g	116,7 g
Lievito di birra fresco	0,5 g	0,5 g
Sale fino	5,0 g	5,0 g
Olio extravergine d'oliva nell'impasto	5,5 g	5,5 g
Patate	66,7 g	66,7 g
Salsiccia napoletana	50,0 g	50,0 g
Mozzarella di bufala DOP	100,0 g	rimossa
Mozzarella di bufala bocconcini	80,0 g	rimossa
Mozzarella senza lattosio	–	320 g
Scamorza/provola senza lattosio	–	100 g
Olio EVO aggiuntivo da condimento	–	15 g

Valori nutrizionali stimati della nuova pizza

Valore	Stima
Energia	circa 1950–2000 kcal
Proteine	circa 100–105 g
Lipidi	circa 110–120 g
Carboidrati	circa 135 g
Sodio	circa 4000–4300 mg
Fibre totali	circa 5,7 g

Ricetta finale riscritta
Pizza patate e salsiccia senza lattosio – almeno 1900 kcal

- Farina 0: 166,7 g;
- Acqua: 116,7 g;
- Lievito di birra fresco: 0,5 g;
- Sale fino: 5,0 g;
- Olio extravergine d'oliva nell'impasto: 5,5 g;
- Patate: 66,7 g;
- Salsiccia napoletana: 50,0 g;
- Mozzarella senza lattosio: 320 g;
- Scamorza/provola senza lattosio: 100 g;
- Olio extravergine d'oliva aggiuntivo sopra la pizza: 15 g.

Perché questa versione funziona
La mozzarella di bufala originale viene eliminata perché non è adatta agli intolleranti al lattosio, salvo nel caso di prodotti specificamente delattosati. Per mantenere l'idea della pizza originale, vengono lasciati invariati:

- l'impasto;
- le patate;
- la salsiccia.

La componente filante viene sostituita con mozzarella senza lattosio e scamorza o provola senza lattosio. In questo modo la pizza rimane coerente con l'idea di patate, salsiccia e formaggio, ma diventa compatibile con l'intolleranza al lattosio.
L'olio extravergine d'oliva aggiuntivo serve invece a portare la ricetta sopra le 1900 kcal senza modificare l'impasto.

Nota importante
Anche la salsiccia napoletana deve essere scelta controllando l'etichetta, perché alcuni insaccati possono contenere derivati del latte o lattosio come additivo.

DeepSeek

Ricetta finale
Dosi per 1 pizza – peso totale maggiore

Ingrediente	Quantità	Note
Farina 0	166,7 g	Invariato
Acqua	116,7 g	Invariato
Lievito di birra fresco	0,5 g	Invariato
Sale fino	5,0 g	Invariato
Olio extravergine d'oliva nell'impasto	5,5 g	Invariato
Olio extravergine d'oliva aggiunto a crudo	50,0 g	Per aumentare le calorie
Patate	66,7 g	Invariato
Salsiccia napoletana senza lattosio	50,0 g	Invariato
Parmigiano Reggiano DOP grattugiato	150,0 g	Senza lattosio, stagionato >24 mesi
Mozzarella di bufala DOP	eliminata	Contiene lattosio
Mozzarella bocconcini	eliminata	Contiene lattosio

Nota
È necessario verificare che la salsiccia non contenga latte o derivati, spesso utilizzati come leganti. In caso contrario, può essere sostituita con una salsiccia di puro maiale senza lattosio.

Valori nutrizionali stimati
Valori riferiti alla pizza intera

Nutriente	Valore
Energia	∼1909 kcal
Proteine	∼82 g
Lipidi	∼135 g
Carboidrati	∼130 g
Sodio	∼3500 mg, principalmente dal sale e dal Parmigiano
Fibre totali	∼5,7 g

Spiegazione delle modifiche

- **Mozzarelle eliminate**: i 180 g totali di mozzarella vengono rimossi perché contengono lattosio e vengono sostituiti con ingredienti più calorici e senza lattosio.
- **Aggiunta di olio extravergine d'oliva**: i 50 g di olio aggiunti forniscono circa 450 kcal, mantenendo morbidezza e gusto mediterraneo.
- **Aggiunta di Parmigiano Reggiano**: i 150 g di Parmigiano Reggiano DOP, formaggio naturalmente privo di lattosio se stagionato oltre 24 mesi, apportano circa 600 kcal, 40 g di proteine e 45 g di grassi.

Calcolo totale

Componente	Energia stimata
Pizza base, cioè impasto, patate, salsiccia e olio dell'impasto	∼860 kcal
Olio extravergine d'oliva aggiunto	450 kcal
Parmigiano Reggiano	600 kcal
Totale	**∼1910 kcal**

Il totale risulta quindi ampiamente superiore alle 1900 kcal richieste.
L'idea della pizza con patate e salsiccia rimane intatta, arricchita da una generosa quantità di Parmigiano Reggiano e da un filo d'olio extravergine d'oliva a crudo o aggiunto prima della cottura. Il risultato è saporito, filante grazie al Parmigiano sciolto e completamente privo di lattosio.

Claude

Pizza Patate e Salsiccia – Versione "iperproteica/calorica" lactose-free $\geq$ 1900 kcal
Per arrivare ad almeno 1900 kcal senza modificare impasto, salsiccia e patate, si interviene su due elementi principali: olio extravergine d'oliva, caratterizzato da un'elevata densità calorica, pari a circa 9 kcal/g, e formaggi delattosati, utili per aumentare volume, grassi e proteine.
La mozzarella di bufala, contenente lattosio, viene sostituita con alternative senza lattosio. Inoltre, viene aggiunto un elemento cremoso per amplificare l'identità del piatto.

Ricetta adattata
Dosi per 1 pizza

Ingrediente	Quantità	Note
Farina 0	166,7 g	Invariato
Acqua	116,7 g	Invariato
Lievito di birra fresco	0,5 g	Invariato
Sale fino	5,0 g	Invariato
Olio EVO	30,0 g	Aumentato, da 5,5 g a 30 g
Patate	66,7 g	Invariato
Salsiccia napoletana	50,0 g	Invariato
Fiordilatte senza lattosio	180,0 g	Sostituisce la mozzarella di bufala DOP, dose aumentata
Provola affumicata senza lattosio	120,0 g	Sostituisce i bocconcini e aggiunge carattere
Stracchino/crescenza senza lattosio	80,0 g	A fiocchetti dopo cottura, per aumentare la cremosità
Pecorino romano DOP grattugiato	25,0 g	Formaggio stagionato, quasi privo di lattosio

Il peso finale stimato della pizza è pari a circa 1020 g.

Valori nutrizionali stimati

Nutriente	Originale	Adattata	Δ
Energia	1248,73 kcal	∼1955 kcal	+706 kcal
Proteine	53,51 g	∼92 g	+38 g
Lipidi	59,64 g	∼128 g	+68 g
Carboidrati	129,93 g	∼136 g	+6 g
Sodio	2937,88 mg	∼4500 mg	+1560 mg
Fibre totali	5,73 g	∼5,8 g	≈

Logica delle modifiche

- **Vincoli rispettati:** impasto, patate e salsiccia restano nelle dosi originali. Tutte le calorie aggiuntive derivano quindi dall'aumento dei grassi e dall'inserimento di formaggi senza lattosio.

- **Lattosio:** la mozzarella di bufala DOP e i bocconcini contengono lattosio, quindi vengono sostituiti con fiordilatte e provola affumicata delattosati. Questi prodotti mantengono un profilo gustativo simile, ma risultano compatibili con l'intolleranza al lattosio perché il lattosio è stato scisso tramite l'enzima lattasi.
- **Elemento cremoso:** lo stracchino o la crescenza senza lattosio vengono aggiunti per aumentare la cremosità della pizza e renderla più ricca.
- **Pecorino romano DOP:** il pecorino romano stagionato contiene solo tracce di lattosio, generalmente inferiori a 0, 1 g per 100 g, grazie alla lunga stagionatura. Se si vuole essere più rigorosi, può essere sostituito con un pecorino delattosato.

Strategia calorica

- **Olio EVO 30 g:** l'aumento dell'olio apporta circa 220 kcal aggiuntive. Una parte viene utilizzata per condire le patate a crudo con rosmarino, mentre la restante viene aggiunta a filo a fine cottura.
- **Triplice formaggio:** la combinazione di fiordilatte senza lattosio, provola affumicata senza lattosio e stracchino senza lattosio raggiunge circa 380 g complessivi. Il fiordilatte fornisce la base filante, la provola aggiunge una nota affumicata e intensa, mentre lo stracchino contribuisce alla cremosità.
- **Pecorino romano:** la spolverata finale aggiunge circa 100 kcal, aumenta la sapidità e rafforza la percezione gustativa complessiva.

L'identità della pizza viene preservata: resta una pizza patate e salsiccia, ma in una versione più ricca, calorica e festosa, simile a una proposta gourmet da pizzeria contemporanea.

Consigli di esecuzione

- **Patate:** affettarle molto sottili, circa 1–2 mm, e condirle a crudo con parte dell'olio, sale e rosmarino. Vanno poi distribuite direttamente sull'impasto prima del primo formaggio.
- **Stratificazione:** disporre prima fiordilatte e provola sull'impasto, poi patate e salsiccia sbriciolata. Successivamente infornare.
- **Formaggi finali:** stracchino e pecorino vanno aggiunti negli ultimi 60 secondi di cottura, oppure subito dopo lo sfornamento, per mantenere una consistenza cremosa.
- **Olio finale:** il filo d'olio a crudo va aggiunto appena la pizza viene sfornata, non prima, perché contribuisce in modo significativo al profilo aromatico.

Attenzione al sodio
Con circa 4500 mg di sodio, questa pizza supera ampiamente il fabbisogno giornaliero raccomandato. Se questo rappresenta un problema, si può ridurre il pecorino a 10 g e compensare con altri 5 g di olio EVO. In questo modo la pizza rimane sopra le 1900 kcal, ma il contenuto di sodio viene ridotto di circa 700 mg.

Gemini

Accidenti, siamo passati da un estremo all'altro. Se prima l'obiettivo era ridurre il più possibile le calorie, ora l'obiettivo è ottenere un vero e proprio pasto da "bulk" o da maratoneta.
Arrivare ad almeno 1900 kcal mantenendo l'impasto identico, pari a circa 500 kcal, e bloccando anche le quantità di patate e salsiccia, significa intervenire in modo consistente sui formaggi e sui grassi. Inoltre, per rispettare il vincolo del senza lattosio, è necessario eliminare la mozzarella di bufala classica e utilizzare prodotti delattosati o naturalmente privi di lattosio, come alcuni formaggi molto stagionati.

La nuova ricetta

Dosi per 1 pizza – minimo 1900 kcal
Per raggiungere questo obiettivo calorico senza stravolgere la base, viene inserita una generosa dose di mozzarella senza lattosio, un'aggiunta di Provolone Valpadana DOP o Parmigiano Reggiano stagionato, entrambi naturalmente privi di lattosio grazie alla stagionatura, e un incremento strategico dell'olio extravergine d'oliva a crudo in uscita dal forno.

Ingrediente	Quantità nuova	Stato / Nota di adattamento
Farina 0	166,7 g	Invariata
Acqua	116,7 g	Invariata
Lievito di birra fresco	0,5 g	Invariato
Sale fino	5,0 g	Invariato
Olio EVO nell'impasto	5,5 g	Invariato
Patate	66,7 g	Invariate come da richiesta
Salsiccia napoletana	50,0 g	Invariata come da richiesta
Mozzarella senza lattosio	250,0 g	Aumentata per garantire la base filante
Provolone stagionato DOP	120,0 g	Naturalmente senza lattosio, adatto alla fusione con le patate
Olio EVO a crudo post-cottura	35,0 g	Circa 3 cucchiai abbondanti, per aumentare le calorie

Il peso totale della pizza è pari a circa 815 g. Si tratta quindi di una pizza decisamente ricca e pesante, ma l'impasto può reggere il carico se viene steso correttamente.

Nuovi valori nutrizionali stimati
Avendo aumentato drasticamente i formaggi e l'olio, i lipidi e le proteine crescono in modo significativo, permettendo di raggiungere il target calorico richiesto.

Nutriente	Valore stimato	Osservazione
Energia	∼1915 kcal	Obiettivo raggiunto, superiore a 1900 kcal
Proteine	∼112 g	Forte incremento proteico
Lipidi	∼138 g	Aumentati soprattutto da formaggi e olio EVO
Carboidrati	∼133 g	Quasi stabili, dato che impasto e patate non cambiano

Consigli per la cottura con molto condimento
Poiché la pizza contiene una quantità elevata di formaggio, pari a circa 370 g tra mozzarella e provolone, se venisse aggiunto tutto subito si rischierebbe di rendere la pizza troppo umida o di impedire una corretta cottura dell'impasto al centro.

- **Cottura in due tempi:** infornare inizialmente la base dell'impasto solo con le patate tagliate sottili e la salsiccia sgranata.
- **Aggiunta del formaggio:** inserire la mozzarella senza lattosio e il provolone grattugiato o tagliato a cubetti solo negli ultimi 2–3 minuti di cottura, giusto il tempo necessario per farli fondere e creare una leggera crosticina.
- **Tocco finale:** versare i 35 g di olio extravergine d'oliva a filo subito dopo aver sfornato la pizza. L'olio, a contatto con la pizza calda, sprigiona profumo ed esalta il sapore della salsiccia napoletana.

Confronto delle risposte

In questo prompt i modelli vengono messi di fronte a vincoli molto stringenti. Viene, infatti, richiesto un notevole aumento calorico seppur numerosi ingredienti siano vincolati alle loro quantità. *ChatGPT* mantiene il suo approccio orientato al soddisfacimento della richiesta. La sua idea rispetta i vincoli ma la quantità di formaggio aggiunta per alzare calorie e lipidi risulta piuttosto elevata. *DeepSeek* propone un'idea interessante dal punto di vista

nutrizionale; tuttavia la totale sostituzione della mozzarella con il Parmigiano Reggiano modifica l'idea culinaria della ricetta rimuovendo la parte "filante" della mozzarella.

Claude effettua un'ampia analisi sulle modifiche da apportare proponendo nella risposta anche una logica di inserimento in fasi. Dal punto di vista tecnico, risulta essere la risposta più realista seppur le quantità caloriche totali risultino essere estremamente alte. *Gemini* adotta un approccio semplice e diretto, come ChatGPT, in quanto anch'esso aumenta la presenza del formaggio per soddisfare i vincoli.

3.4.3 Terzo prompt

Utente

PIZZA PATATE E SALSICCIA - Dosi per 1 pizza (peso 585,1g)

- Farina 0 W300 166,7 g
- Acqua 116,7 g
- Lievito di birra fresco 0,5 g
- Sale fino 5,0 g
- Olio extravergine d'oliva 5,5 g
- Patate 66,7 g
- Salsiccia napoletana 50,0 g
- Mozzarella di bufala DOP 100,0 g
- Mozzarella di bufala bocconcini 80,0 g

Valori nutrizionali:

- Energia (kcal): 1248,73
- Proteine (g): 53,51
- Lipidi (g): 59,64
- Carboidrati (g): 129,93
- Sodio (mg): 2937,88
- Fibre totali (g): 5,73

Partendo da quantità e valori nutrizionali dati, voglio che adatti la mia ricetta, pur rimanendo fedele all'idea della pizza, per poterla avere estremamente calorica, per intolleranti al glutine e vegetariana.

Risposte al terzo prompt

ChatGPT

Pizza patate e salsiccia vegetariana senza glutine
Versione molto calorica

Ingrediente	Quantità
Mix per pizza senza glutine	166,7 g
Acqua	130 g
Lievito di birra fresco	0,5 g
Sale fino	5,0 g
Olio extravergine d'oliva nell'impasto	5,5 g
Patate	120 g
Salsiccia vegetale senza glutine	100 g
Mozzarella	200 g
Scamorza/provola	120 g
Pesto di basilico senza glutine	40 g
Olio extravergine d'oliva a crudo	15 g
Olive nere	30 g

Valori nutrizionali stimati

Valore	Stima
Energia	circa 2200–2300 kcal
Proteine	circa 85–95 g
Lipidi	circa 125–140 g
Carboidrati	circa 165–180 g
Sodio	circa 3500–4300 mg
Fibre totali	circa 10–14 g

Questa versione risulta molto calorica soprattutto per la presenza di:

- mozzarella;
- scamorza/provola;
- pesto;
- olio extravergine d'oliva;
- salsiccia vegetale;
- aumento della quantità di patate.

Modifiche rispetto alla pizza originale

Ingrediente originale	Modifica
Farina 0	Sostituita con mix per pizza senza glutine
Salsiccia napoletana	Sostituita con salsiccia vegetale senza glutine
Patate 66,7 g	Aumentate a 120 g
Mozzarella di bufala e bocconcini	Sostituite con mozzarella e scamorza/provola
Condimento semplice	Aggiunti pesto, olive e olio EVO a crudo

Nota importante
Per rendere la pizza realmente adatta agli intolleranti al glutine, è necessario scegliere prodotti che riportino chiaramente diciture come "senza glutine" oppure "gluten free".
Occorre prestare particolare attenzione a:

- mix per pizza;
- salsiccia vegetale, poiché alcune versioni possono contenere seitan e quindi glutine;

- pesto pronto;
- formaggi confezionati o affumicati;
- eventuali contaminazioni in cucina.

Per quanto riguarda la parte vegetariana, se si vuole essere più precisi, è opportuno scegliere formaggi prodotti con caglio microbico o vegetale, evitando quelli con caglio animale.

Versione finale scritta bene
Pizza patate e salsiccia vegetariana senza glutine – molto calorica

- Mix per pizza senza glutine: 166,7 g;
- Acqua: 130 g;
- Lievito di birra fresco: 0,5 g;
- Sale fino: 5,0 g;
- Olio EVO nell'impasto: 5,5 g;
- Patate: 120 g;
- Salsiccia vegetale senza glutine: 100 g;
- Mozzarella: 200 g;
- Scamorza/provola: 120 g;
- Pesto di basilico senza glutine: 40 g;
- Olive nere: 30 g;
- Olio EVO a crudo: 15 g.

Questa è una pizza coerente con l'idea originale, perché mantiene il concetto di patate, salsiccia e formaggio, ma diventa vegetariana, senza glutine e molto più calorica.

DeepSeek

Ricetta finale – Pizza senza glutine, vegetariana, ipercalorica

Ingrediente	Quantità	Note
Impasto senza glutine		
Mix per pizza senza glutine	200 g	Tipo Caputo FioreGlut o Schär
Acqua	200 g	Idratazione al 100% per impasto senza glutine
Lievito di birra fresco	6 g	Dose maggiore per lievitazione senza glutine
Sale fino	5 g	
Olio extravergine d'oliva	20 g	Per ottenere un impasto più ricco
Condimento		
Patate	150 g	Aumentate per incrementare le calorie
Salsiccia vegetale senza glutine	150 g	A base di tofu o legumi, aromatizzata al finocchietto
Mozzarella vegetale senza glutine e vegana	200 g	A base di anacardi o cocco, con effetto filante
Formaggio grattugiato vegetariano o vegetale	80 g	Per extra sapore e calorie
Olio extravergine d'oliva a crudo	50 g	Da aggiungere dopo la cottura
Opzionale per calorie extra		
Panna vegetale senza lattosio e senza glutine	100 g	Da spalmare prima dei condimenti

Valori nutrizionali stimati
Valori riferiti alla pizza intera

Nutriente	Valore
Energia	∼2350 kcal
Proteine	∼55 g
Lipidi	∼145 g
Carboidrati	∼210 g
Fibre totali	∼12 g
Sodio	∼3100 mg

Spiegazione delle modifiche

- **Senza glutine:** la farina 0 viene sostituita con un mix specifico senza glutine. Inoltre, vengono aumentate idratazione e quantità di lievito per compensare l'assenza della maglia glutinica.
- **Vegetariana:** la salsiccia napoletana viene sostituita con una salsiccia vegetale senza glutine, verificando sempre l'etichetta. La mozzarella di bufala viene sostituita con una mozzarella vegetale filante, senza glutine e senza derivati animali.
- **Molto calorica:** l'apporto energetico viene aumentato attraverso patate, salsiccia vegetale, mozzarella vegana, formaggio grattugiato vegetale, olio extravergine d'oliva e panna vegetale opzionale.

Incremento calorico degli ingredienti principali

- Patate aumentate a 150 g, con circa 130 kcal extra rispetto alla quantità originale di 66,7 g.
- Salsiccia vegetale, pari a 150 g, con un apporto stimato di circa 300 kcal.

- Mozzarella vegana, pari a 200 g, con un apporto stimato di circa 350 kcal.
- Formaggio grattugiato vegetale, pari a 80 g, con un apporto stimato di circa 350 kcal.
- Olio extravergine d'oliva a crudo, pari a 50 g, con un apporto stimato di circa 450 kcal.
- Panna vegetale opzionale, pari a 100 g, con un apporto stimato di circa 200 kcal.
- Impasto reso più ricco grazie all'aumento dell'olio, da 5,5 g a 20 g, con circa 130 kcal aggiuntive.

Il totale risulta quindi ben oltre le 2000 kcal, mantenendo comunque l'idea della pizza con patate e salsiccia.

Consigli per la preparazione

- L'impasto senza glutine va steso con le mani unte, evitando il mattarello.
- Il forno va preriscaldato a 250 °C con pietra refrattaria.
- Le patate vanno sbollentate a fette sottili per circa 3 minuti prima di essere disposte sulla pizza.
- La salsiccia vegetale può essere rosolata in padella con un filo d'olio e finocchietto.
- Sulla base si può aggiungere prima la panna vegetale, poi patate, salsiccia vegetale, mozzarella vegana e formaggio grattugiato.
- La pizza va cotta per circa 10–12 minuti. A fine cottura si aggiunge l'olio extravergine d'oliva a crudo.

Questa pizza risulta ricca, saporita e adatta a vegetariani e intolleranti al glutine.

Claude

Pizza Patate e "Salsiccia" – Versione gluten-free, vegetariana e molto calorica
I vincoli da rispettare sono tre: eliminare il glutine, rendere la pizza vegetariana e ottenere un profilo molto calorico. La farina 0 viene quindi sostituita con un mix senza glutine, mentre la salsiccia napoletana viene rimpiazzata da un'alternativa vegetale che ne replichi sapore e consistenza.
L'obiettivo è mantenere l'identità del piatto, cioè una base croccante con patate, un elemento speziato simile alla salsiccia e formaggi filanti, aumentando però l'apporto energetico attraverso grassi e formaggi più ricchi.

Ricetta adattata
Dosi per 1 pizza

Ingrediente	Quantità	Note
Mix farine gluten-free per pizza	166,7 g	Sostituisce la farina 0, a base di riso, mais, grano saraceno e amidi
Psillio in polvere	5,0 g	Fondamentale negli impasti senza glutine, perché conferisce elasticità
Acqua	125,0 g	Leggermente aumentata, poiché i mix senza glutine assorbono di più
Lievito di birra fresco	1,0 g	Raddoppiato, perché le farine senza glutine lievitano meno
Sale fino	5,0 g	Invariato
Olio EVO	30,0 g	Aumentato, da 5,5 g a 30 g, tra impasto, condimento delle patate e filo finale
Patate	100,0 g	Leggermente aumentate per incrementare l'apporto calorico
"Salsiccia" vegetale speziata	70,0 g	Senza seitan, a base di soia o piselli, con finocchietto e peperoncino
Mozzarella fiordilatte	150,0 g	Base filante
Gorgonzola dolce DOP	80,0 g	Aggiunge cremosità e ricchezza calorica
Scamorza affumicata	70,0 g	A cubetti, richiama la nota affumicata della salsiccia
Parmigiano Reggiano 30 mesi	25,0 g	Spolverata finale
Pinoli tostati	15,0 g	Finitura croccante e fonte di grassi
Rosmarino fresco	q.b.	Aromatizza le patate

Il peso finale stimato della pizza è pari a circa 840 g.

Nota sulle etichette
La "salsiccia" vegetale deve essere certificata senza glutine, poiché molti prodotti vegetariani o vegani contengono seitan, cioè glutine puro. È preferibile scegliere alternative a base di soia testurizzata, piselli o fave, con dicitura "senza glutine". Anche per gorgonzola e altri formaggi è opportuno verificare l'assenza di contaminazioni.

Valori nutrizionali stimati

Nutriente	Originale	Adattata	Δ
Energia	1248,73 kcal	∼1780 kcal	+530 kcal
Proteine	53,51 g	∼75 g	+22 g
Lipidi	59,64 g	∼108 g	+48 g
Carboidrati	129,93 g	∼138 g	+8 g
Sodio	2937,88 mg	∼3650 mg	+710 mg
Fibre totali	5,73 g	∼9 g	+3 g

Logica delle modifiche

- **Senza glutine:** la farina 0 è incompatibile con una dieta senza glutine, quindi viene sostituita con un mix gluten-free, preferibilmente formulato per pizza. In alternativa, si può usare un blend casalingo composto da farina di riso, mais, grano saraceno e amido di mais. Lo psillio viene aggiunto per restituire elasticità all'impasto, che altrimenti risulterebbe più friabile. Acqua e lievito vengono leggermente aumentati per compensare il diverso assorbimento e la minore forza lievitante.
- **Vegetariana:** la salsiccia napoletana rappresenta uno degli elementi identitari del piatto, quindi la sostituzione deve risultare credibile. Le salsicce vegetali senza glutine, a base di soia o piselli, aromatizzate con finocchietto e peperoncino, possono replicare bene il profilo aromatico. Sbriciolate finemente e distribuite a piccoli mucchietti, mantengono una resa visiva e gustativa simile.

- **Richiamo al sapore originale:** la scamorza affumicata rafforza il richiamo alla componente sapida e intensa della salsiccia, grazie alla nota affumicata.
- **Calorie alte:** l'aumento calorico deriva soprattutto da olio EVO, formaggi grassi e pinoli. L'olio raggiunge 30 g complessivi, mentre la combinazione di fiordilatte, gorgonzola e scamorza arriva a 300 g totali. I pinoli tostati aggiungono grassi, croccantezza e una nota mediterranea.

L'identità della pizza viene quindi preservata: rimane una pizza patate e "salsiccia", filante e ricca, reinterpretata in chiave vegetariana e senza glutine.

Consigli di esecuzione

- **Impasto senza glutine:** idratare farina, psillio e acqua per circa 10 minuti prima di aggiungere lievito e sale. Lo psillio deve formare un gel che sostituisce parzialmente la funzione del glutine. La lievitazione può durare 2–3 ore a circa 26 °C.
- **Stesura:** l'impasto senza glutine non si lavora come una pizza classica. Conviene stenderlo tra due fogli di carta forno con il mattarello, oppure direttamente in teglia oliata.
- **Patate:** affettarle molto sottili, circa 1–2 mm, e condirle a crudo con parte dell'olio, sale e rosmarino. Vanno poi distribuite direttamente sull'impasto prima dei formaggi.
- **Stratificazione:** disporre olio sull'impasto, poi patate, fiordilatte e scamorza, quindi la "salsiccia" vegetale sbriciolata. Cuocere a 250 °C per circa 8–10 minuti. A fine cottura aggiungere gorgonzola a fiocchetti, Parmigiano grattugiato, pinoli tostati e un filo d'olio EVO a crudo.
- **Caglio vegetariano:** se si vuole essere rigorosi sulla vegetarianità, occorre verificare che fiordilatte, scamorza e gorgonzola siano prodotti con caglio microbico. Il Parmigiano Reggiano DOP utilizza caglio animale secondo disciplinare, quindi può essere sostituito con un formaggio grattugiato prodotto con caglio vegetale.

Per superare le 1900 kcal
Per spingere ulteriormente la ricetta oltre le 1900 kcal, si possono aggiungere 20 g di mascarpone a fiocchetti dopo la cottura, pari a circa 85 kcal, e aumentare i pinoli a 25 g, ottenendo circa 60 kcal aggiuntive. In questo modo la pizza supera facilmente quota 1900 kcal, mantenendo comunque un equilibrio gustativo.

Gemini

Ecco la trasformazione della pizza in una versione ipercalorica, totalmente priva di glutine e vegetariana, mantenendo intatta la combinazione di patate e "salsiccia", proposta in versione vegetale.
Per rendere la pizza molto calorica, superando le 2000 kcal, e adattarla ai vincoli richiesti, sono state apportate tre modifiche fondamentali:

- **Impasto senza glutine:** la farina 0 viene sostituita con un mix di farine senza glutine, che richiede un'idratazione leggermente maggiore per mantenere l'impasto soffice e non eccessivamente secco.
- **Salsiccia vegetale:** viene utilizzata una salsiccia 100% vegetale, ad esempio a base di proteine del pisello o della soia, capace di imitare consistenza e sapore speziato della salsiccia tradizionale.
- **Boost calorico:** oltre alla mozzarella di bufala, vengono aggiunti Gorgonzola DOP e noci tritate. Il gorgonzola aumenta cremosità e sapore, mentre le noci aggiungono croccantezza e grassi.

La nuova ricetta ipercalorica, gluten-free e vegetariana
Nota per i vegetariani: è necessario assicurarsi che i formaggi utilizzati siano prodotti con caglio microbico o vegetale.

Ingrediente	Quantità nuova	Ruolo nella ricetta
Mix farina senza glutine	180,0 g	Base per l'impasto gluten-free
Acqua	145,0 g	Maggiore idratazione per farine senza glutine
Lievito di birra fresco	1,0 g	Favorisce la spinta dell'impasto senza glutine
Sale fino	5,0 g	Invariato
Olio extravergine d'oliva	30,0 g	Diviso tra impasto e topping a crudo
Patate	100,0 g	Aumentate per dare più corpo e carboidrati
Salsiccia vegetale	90,0 g	Alternativa vegetale speziata e calorica
Mozzarella di bufala DOP	150,0 g	Aumentata per massima filantezza e cremosità
Gorgonzola DOP	80,0 g	Aggiunta gourmet per legare patate e calorie
Noci sgusciate tritate	30,0 g	Aggiungono croccantezza e grassi

Il peso totale stimato della pizza è pari a circa 810 g. Si tratta quindi di una pizza molto ricca e sostanziosa.

Nuovi valori nutrizionali stimati

Nutriente	Valore stimato	Osservazione
Energia	∼2087 kcal	Obiettivo raggiunto, versione molto calorica
Proteine	∼70,2 g	Apporto elevato da bufala e salsiccia vegetale
Lipidi	∼124,0 g	Derivano da formaggi, olio e frutta secca
Carboidrati	∼165,3 g	Aumentati per impasto e patate
Sodio	∼3052 mg	Valore elevato, legato soprattutto ai formaggi
Fibre totali	∼11,5 g	Incrementate dalla componente vegetale e dalle noci

Consigli per la preparazione e la cottura

- **Lavorazione gluten-free:** gli impasti senza glutine non sviluppano la maglia glutinica, quindi risultano più appiccicosi. Conviene stendere l'impasto direttamente su una teglia ben oliata oppure su carta forno, aiutandosi con le mani unte d'olio.
- **Tempi degli ingredienti:** è consigliabile cuocere inizialmente la base solo con le patate tagliate molto sottili e la salsiccia vegetale sgranata.
- **Topping:** mozzarella di bufala, ben sgocciolata e sfilacciata, e gorgonzola vanno aggiunti solo negli ultimi 3–4 minuti di cottura.
- **Uscita dal forno:** appena sfornata, la pizza può essere guarnita con 30 g di noci tritate e con il giro d'olio EVO a crudo rimasto. Il calore della pizza permette di sprigionare gli aromi delle noci e del gorgonzola fuso.

Confronto delle risposte

I vincoli del prompt impongono che i modelli applichino modifiche all'impasto e al condimento riadattandoli totalmente alle richieste. *ChatGPT* fornisce una risposta equilibrata sostituendo gli alimenti con altre versioni compatibili con i vincoli dati, senza ricercare troppo la soluzione ottimale. *DeepSeek* presta parecchia attenzione all'aumento delle quantità e all'inserimento di nuovi ingredienti per soddisfare la richiesta calorica. La ricetta finale, però, risulta poco aderente all'idea originale della pizza. *Claude* fornisce una spiegazione

estremamente dettagliata riguardo l'utilizzo di alimenti senza glutine e vegetali, aggiungendo alimenti al condimento della pizza senza però perderne l'idea originale. *Gemini* adotta un approccio più semplificato e diretto aggiungendo grassi per aumentare la densità calorica della ricetta, non analizzando nel dettaglio le scelte per la parte vegetale e senza glutine.

3.5 Scenario 3: adattamento della pizza radicchio, ventricina piccante e primosale

3.5.1 Prompt iniziale

Utente

PIZZA RADICCHIO, VENTRICINA PICCANTE E PRIMOSALE - Dosi per 1 pizza (peso 618,5g)

- Farina 0 W300 175,0 g
- Acqua 112,5 g
- Lievito di birra fresco 1,0 g
- Sale fino 5,0 g
- Olio extravergine d'oliva 15,0 g
- Primosale 150,0 g
- Salame ventricina piccante 60,0 g
- Radicchio 50,0 g
- Parmigiano Reggiano DOP 25,0 g
- Cipolle rosse 25,0

Valori nutrizionali:

- Energia (kcal): 1420,85
- Proteine (g): 68,82
- Lipidi (g): 72,15
- Carboidrati (g): 126,83
- Sodio (mg): 3877,16
- Fibre totali (g): 6,89

Partendo da quantità e valori nutrizionali dati, voglio che adatti la mia ricetta, pur rimanendo fedele all'idea della pizza, in modo che vengano ridotti i grassi e il sodio a vantaggio dei carboidrati. La pizza deve essere per gli intolleranti al lattosio. Le quantità dell'impasto devono rimanere le stesse, modifica solo gli ingredienti sopra rimuovendoli o aggiungendone altri.

Risposte al prompt iniziale

ChatGPT

Pizza modificata – senza lattosio, meno grassi e meno sodio
Dosi per 1 pizza – peso totale circa 618,5 g

Impasto invariato

Ingrediente	Quantità
Farina 0	175,0 g
Acqua	112,5 g
Olio extravergine d'oliva	15,0 g
Lievito di birra fresco	1,0 g
Sale fino	5,0 g

Condimento modificato

Ingrediente	Quantità
Mozzarella senza lattosio light	80,0 g
Petto di pollo o tacchino fresco	55,0 g
Patate	100,0 g
Radicchio	50,0 g
Cipolle rosse	25,0 g
Paprika piccante, peperoncino, origano	q.b.

La pizza rimane fedele all'idea originale perché conserva radicchio, cipolla rossa e nota piccante. La parte grassa e salata del salame viene però sostituita da una componente speziata più magra.

Cosa è stato cambiato

Ingredienti rimossi

Ingrediente tolto	Motivo
Primosale 150 g	Molto impattante su grassi e sodio
Ventricina piccante 60 g	Molto ricca di grassi e sodio
Parmigiano 25 g	Grassi e sodio abbastanza alti

Ingredienti aggiunti

Ingrediente aggiunto	Funzione
Mozzarella senza lattosio light	Mantiene la parte formaggiosa, ma con meno grassi
Patate	Aumentano i carboidrati e rendono la pizza più saziante
Pollo/tacchino speziato	Sostituisce il salame come parte saporita e proteica
Paprika/peperoncino	Recupera il carattere piccante della ventricina senza aggiungere sodio

Valori nutrizionali stimati della nuova versione

Valore	Ricetta originale	Nuova versione stimata
Energia	1420,85 kcal	circa 1030 kcal
Proteine	68,82 g	circa 56 g
Lipidi	72,15 g	circa 26 g
Carboidrati	126,83 g	circa 150 g
Sodio	3877,16 mg	circa 2300–2400 mg
Fibre totali	6,89 g	circa 8–9 g

Con questa versione si ottiene quindi:

- una riduzione dei grassi superiore al 60%;

- una riduzione del sodio di circa il 40%;
- un aumento dei carboidrati;
- una pizza adatta agli intolleranti al lattosio, utilizzando mozzarella senza lattosio.

Nota importante
Dato che viene richiesto di non modificare l'impasto, i 5 g di sale restano invariati. Da soli corrispondono già a circa 2000 mg di sodio; di conseguenza, il contenuto di sodio non può diminuire in modo drastico senza intervenire anche sul sale dell'impasto.

Procedimento consigliato
Le patate possono essere lessate o cotte prima, tagliandole a fette sottili. Il pollo o tacchino può essere preparato a striscioline con paprika piccante, peperoncino, pepe e un filo minimo d'acqua o limone, evitando di aggiungere sale.
Sulla pizza si può disporre prima la mozzarella senza lattosio, poi patate, radicchio, cipolla e infine il pollo speziato. In questo modo la pizza rimane ricca, ma risulta molto più equilibrata rispetto alla versione con primosale, ventricina e Parmigiano.

DeepSeek

Ecco la ricetta adattata per ridurre grassi e sodio, aumentare i carboidrati e renderla completamente senza lattosio. L'impasto viene mantenuto identico, quindi farina, acqua, olio, lievito e sale restano invariati. Vengono invece modificati i condimenti, sostituendo gli ingredienti ad alto contenuto di grassi e sodio, come primosale, ventricina e Parmigiano, con alternative vegetali a base di patate, funghi, passata e lievito alimentare.
Il radicchio e la cipolla rossa vengono mantenuti per preservare il carattere della pizza, mentre un pizzico di peperoncino richiama la nota piccante della ventricina.

Nuova ricetta – Pizza Radicchio e Patate
Senza lattosio, a basso contenuto di grassi e sodio
Dosi per 1 pizza – stesso impasto, condimenti modificati

Impasto invariato

- Farina 0: 175,0 g;
- Acqua: 112,5 g;
- Olio extravergine d'oliva: 15,0 g;
- Lievito di birra fresco: 1,0 g;
- Sale fino: 5,0 g.

Condimenti nuovi

- Radicchio: 50,0 g;
- Cipolle rosse: 25,0 g;
- Patate lessate e affettate sottili: 150,0 g;
- Funghi champignon affettati: 50,0 g;
- Passata di pomodoro senza sale aggiunto: 50,0 g;
- Lievito alimentare in scaglie: 10,0 g;
- Peperoncino rosso in polvere o in fiocchi: q.b., circa 1 g;
- Origano e aglio in polvere: a piacere.

Nota
La passata di pomodoro aggiunge umidità e carboidrati, mentre il lievito alimentare dona un sapore "formaggioso" senza grassi né lattosio. Le patate aumentano invece la quota di carboidrati complessi.

Profilo nutrizionale approssimativo
Valori riferiti alla nuova pizza

Nutriente	Nuovo valore	Variazione rispetto all'originale
Energia	∼1240 kcal	−180 kcal
Proteine	∼45 g	−23, 8 g
Lipidi	∼19 g	−53, 2 g
Carboidrati	∼160 g	+33,2 g
Sodio	∼2050 mg	−1827 mg
Fibre totali	∼9 g	+2,1 g

I valori sono stimati. Il sodio deriva quasi esclusivamente dai 5 g di sale presenti nell'impasto, pari a circa 2000 mg, più le tracce naturalmente presenti nei vegetali.

Come soddisfa le richieste

- **Riduzione dei grassi:** vengono eliminati i formaggi, cioè primosale e Parmigiano, e il salame ventricina, ricco di grassi saturi. Al loro posto vengono inseriti patate, funghi e passata, ingredienti con contenuto lipidico molto basso.
- **Riduzione del sodio:** vengono rimossi gli ingredienti più ricchi di sale, come ventricina e formaggi. Inoltre, viene utilizzata passata di pomodoro senza sale aggiunto. Il sodio totale scende da circa 3877 mg a circa 2050 mg.
- **Aumento dei carboidrati:** l'aggiunta di 150 g di patate e 50 g di passata di pomodoro porta i carboidrati da circa 127 g a circa 160 g.
- **Assenza di lattosio:** non sono presenti latticini. Il lievito alimentare è privo di lattosio e conferisce una nota simile a quella del formaggio.
- **Fedeltà all'idea originale:** radicchio e cipolla rimangono presenti. Il peperoncino sostituisce la piccantezza della ventricina, mentre la consistenza complessiva resta quella di una pizza gustosa e bilanciata.

Preparazione

1. Preparare l'impasto come di consueto e lasciarlo lievitare.
2. Lessare le patate con la buccia, poi sbucciarle e affettarle sottili, con uno spessore di circa 3–4 mm.
3. Lavare e tagliare il radicchio a strisce, la cipolla a fettine sottili e i funghi a lamelle.
4. Stendere la pizza e spennellare la base con un velo di passata di pomodoro.
5. Distribuire patate, funghi, radicchio e cipolla. Spolverare con lievito alimentare, peperoncino, origano e aglio.
6. Cuocere in forno ben caldo, a circa 250–280 °C, per 10–12 minuti su pietra refrattaria o teglia.

Il risultato è una pizza più leggera, senza lattosio, con meno grassi e sodio, ma ancora coerente con l'idea della ricetta originale.

Claude

Pizza Radicchio, Ventricina Piccante e "Primosale" vegetale – versione lactose-free
L'impasto viene mantenuto invariato nelle quantità. Le modifiche riguardano soltanto il condimento, con l'obiettivo di ridurre grassi e sodio, eliminare il lattosio e lasciare più spazio percentuale ai carboidrati dell'impasto.

Impasto invariato

Ingrediente	Quantità
Farina 0	175,0 g
Acqua	112,5 g
Olio extravergine d'oliva	15,0 g
Lievito di birra fresco	1,0 g
Sale fino	5,0 g

Condimento riadattato

Ingrediente	Prima	Dopo	Motivo
Primosale	150 g	rimosso	Contiene lattosio, alto in grassi saturi e sodio
"Primosale" vegetale a base di mandorle/anacardi	–	110 g	Cremosità e proteine simili, grassi insaturi, meno sodio
Ventricina piccante	60 g	30 g	Resta come ingrediente-firma, ma viene dimezzata perché molto sodica e grassa
Parmigiano Reggiano DOP	25 g	rimosso	Contiene lattosio in tracce e molto sodio
Lievito alimentare in scaglie	–	6 g	Recupera la nota umami e sapida del Parmigiano, senza lattosio
Radicchio	50 g	90 g	Più volume, fibra e nota amarognola, con calorie trascurabili
Cipolle rosse	25 g	35 g	Dolcezza e croccantezza, per bilanciare il piccante
Pomodorini datterini a spicchi	–	40 g	Aggiungono dolcezza e acidità senza grassi né sodio rilevante

Il peso totale della pizza è pari a circa 624 g, rispetto ai 618,5 g della ricetta originale.

Valori nutrizionali stimati

Nutriente	Originale	Riadattata	Δ
Energia	1420,85 kcal	∼1090 kcal	−23%
Proteine	68,82 g	∼42 g	−39%
Lipidi	72,15 g	∼38 g	−47%
Carboidrati	126,83 g	∼132 g	+4%
Sodio	3877 mg	∼1450 mg	−63%
Fibre totali	6,89 g	∼8,5 g	+23%

Le stime sono indicative: i valori del "primosale" vegetale possono variare in base alla marca, quindi è necessario controllare l'etichetta del prodotto specifico. Inoltre, la quota percentuale dei carboidrati sul totale calorico passa da circa 36% a circa 48%.

Note di preparazione

- **Radicchio:** tagliarlo a listarelle e scottarlo per circa 1 minuto in padella antiaderente senza grassi. In questo modo perde l'amaro più aggressivo e non rilascia troppa acqua in cottura.
- **Topping a crudo dopo cottura:** aggiungere il "primosale" vegetale, la ventricina e la cipolla rossa a fine cottura, con forno spento e pizza appena sfornata. In questo modo il grasso della ventricina non cola sull'impasto e gli aromi restano più intensi.
- **Lievito alimentare:** spolverarlo all'uscita dal forno, così da dare la spinta umami che compensa l'assenza del Parmigiano.
- **Pomodorini:** inserirli in cottura insieme al radicchio scottato.

Perché funziona
I tagli più incisivi riguardano sodio e grassi, ridotti rispettivamente di circa il 63% e il 47%. Il risultato viene ottenuto intervenendo sui tre ingredienti più critici: Parmigiano, primosale vaccino e ventricina.
Il Parmigiano viene rimosso, il primosale vaccino viene sostituito con un'alternativa vegetale meno salata e con un profilo lipidico più orientato verso grassi insaturi, mentre la ventricina viene dimezzata. I carboidrati, in valore assoluto, restano simili perché l'impasto è invariato come richiesto, ma diventano la fonte energetica dominante della pizza, passando da circa il 36% a circa il 48% delle calorie totali.

Gemini

Ecco una versione rielaborata della pizza. Per soddisfare le richieste, il lavoro viene concentrato sul topping, con l'obiettivo di eliminare il lattosio, ridurre drasticamente il sodio evitando salumi lavorati e formaggi stagionati salati, abbassare i grassi e aumentare la componente dei carboidrati complessi tramite l'aggiunta di una crema di legumi.

Ricetta rielaborata
Senza lattosio, basso sodio, grassi ridotti
Le dosi dell'impasto rimangono invariate. Gli ingredienti del condimento vengono invece sostituiti con alternative più leggere e bilanciate.

Ingrediente	Quantità	Note nutrizionali
Impasto, cioè farina, acqua, olio, lievito e sale	308,5 g	Invariato
Crema di cannellini	100,0 g	Apporta carboidrati complessi e fibre; sostituisce la base grassa
Radicchio	100,0 g	Raddoppiato, per dare volume senza aggiungere grassi
Cipolle rosse	50,0 g	Aumentate per aggiungere dolcezza naturale
Petto di pollo affumicato o tacchino	60,0 g	Sostituisce la ventricina, con meno grassi e sodio
Lievito alimentare in scaglie	5,0 g	Sostituisce il Parmigiano, fornendo sapore umami senza lattosio

Modifiche apportate

- **Eliminazione del lattosio:** vengono rimossi il primosale e il Parmigiano Reggiano. Al loro posto viene inserito il lievito alimentare in scaglie, che ha un sapore intenso simile al formaggio, ma è privo di lattosio e povero di sodio.

- **Riduzione dei grassi:** la ventricina, altamente processata e ricca di grassi saturi e sodio, viene sostituita con petto di pollo o tacchino affumicato, più magro. La crema di cannellini funge da base legante al posto di ingredienti più grassi.
- **Riduzione del sodio:** eliminando salume stagionato e formaggi, il contenuto di sodio si riduce in modo significativo. La sapidità viene ottenuta attraverso la cottura del radicchio e della cipolla rossa, che caramellando naturalmente intensificano il sapore senza richiedere sale aggiunto.
- **Aumento dei carboidrati:** l'introduzione dei fagioli cannellini, frullati per ottenere una crema, aumenta la quota di carboidrati complessi, fibre e proteine vegetali. In questo modo la pizza mantiene un buon senso di sazietà, ma con un profilo nutrizionale più equilibrato.

Consigli per la preparazione

- **La base:** frullare i cannellini con un cucchiaio d'acqua e un pizzico di pepe, fino a ottenere una crema da spalmare sulla base prima della cottura.
- **Il topping:** passare radicchio e cipolla in padella antiaderente con pochissima acqua, così da stufarli senza aggiungere olio e ridurre ulteriormente i grassi.
- **Il tocco finale:** aggiungere il lievito alimentare solo dopo la cottura, per preservarne il sapore simile al formaggio.

Confronto delle risposte

In questo scenario vengono inseriti vincoli volti a riequilibrare la composizione nutrizionale della pizza intervenendo esclusivamente sul condimento. Gli approcci adottati e le risposte fornite dai modelli differiscono notevolmente:

- *ChatGPT* propone una soluzione equilibrata. Sostituisce gli ingredienti con un elevato contenuto di grassi e sodio con alimenti più leggeri. Questo permette di mantenere la componente proteica e incrementare i carboidrati. La risposta risulta coerente con i vincoli dati e le modifiche apportate sono spiegate chiaramente. L'unica criticità risiede nella sostituzione della ventricina con la carne bianca, che modifica sensibilmente l'idea originale della pizza.
- *DeepSeek* adotta un approccio orientato al miglioramento nutrizionale. Esso sostituisce, motivando accuratamente ogni scelta, quasi tutti gli alimenti, mantenendo soltanto il radicchio e la cipolla tra gli ingredienti originali. Dal punto di vista nutrizionale è probabilmente la soluzione più efficace, ma è anche quella che si allontana maggiormente dalla ricetta iniziale.
- *Claude* cerca di ottenere un compromesso tra il mantenimento dell'identità della pizza e il miglioramento del profilo nutrizionale. Esso sostituisce gli alimenti ricchi di grassi e sodio con un'alternativa vegetale, o riducendone le quantità, così da conservare almeno in parte il sapore caratteristico della pizza. La criticità principale risiede nella presenza della ventricina, in quanto tale presenza rende la pizza ancora non vegetariana. Inoltre, la stima nutrizionale risulta poco plausibile per i valori del sodio.
- *Gemini* propone una soluzione essenziale. Esso apporta sostituzioni tra gli ingredienti per poter rientrare nelle richieste nutrizionali del prompt, senza approfondire in maniera significativa le implicazioni dovute alle modifiche. La ricetta iniziale, però, viene completamente persa in quanto i nuovi alimenti inseriti non rispecchiano affatto il

gusto iniziale. La risposta, inoltre, appare meno argomentata rispetto a quelle degli altri modelli, ad esempio, non viene presentata nessuna tabella dei nuovi valori nutrizionali della pizza.

Nel complesso, questo scenario mette in evidenza una differenza ricorrente tra i modelli generativi. ChatGPT e Claude mostrano una maggiore attenzione ai vincoli espressi dall'utente, cercando di mantenere un equilibrio tra modifica della ricetta e fedeltà al piatto originale. DeepSeek privilegia invece l'ottimizzazione dell'obiettivo nutrizionale, mentre Gemini propone una soluzione più sintetica, con minore livello di approfondimento tecnico.

3.5.2 Secondo prompt

Utente

PIZZA RADICCHIO, VENTRICINA PICCANTE E PRIMOSALE - Dosi per 1 pizza (peso 618,5g)

- Farina 0 W300 175,0 g
- Acqua 112,5 g
- Lievito di birra fresco 1,0 g
- Sale fino 5,0 g
- Olio extravergine d'oliva 15,0 g
- Primosale 150,0 g
- Salame ventricina piccante 60,0 g
- Radicchio 50,0 g
- Parmigiano Reggiano DOP 25,0 g
- Cipolle rosse 25,0

Valori nutrizionali:

- Energia (kcal): 1420,85
- Proteine (g): 68,82
- Lipidi (g): 72,15
- Carboidrati (g): 126,83
- Sodio (mg): 3877,16
- Fibre totali (g): 6,89

Partendo da quantità e valori nutrizionali dati, voglio che adatti la mia ricetta, pur rimanendo fedele all'idea della pizza, per ottenere un quantitativo energetico di oltre 2000kcal, mantenendo le quantità dell'impasto pressoché invariati e limitando il valore di grassi e sodio, inoltre, la pizza deve essere senza glutine.

Risposte al secondo prompt

ChatGPT

Pizza radicchio, ventricina piccante e primosale – versione senza glutine, >2000 kcal

Impasto quasi invariato

Ingrediente	Quantità
Mix per pizza senza glutine	175 g
Acqua	125–130 g
Olio extravergine d'oliva	15 g
Lievito di birra fresco	1 g
Sale fino	3,5 g

Il sale viene leggermente ridotto da 5 g a 3,5 g, perché altrimenti il sodio rimarrebbe troppo alto. L'acqua viene invece leggermente aumentata, poiché gli impasti senza glutine assorbono più liquidi.

Condimento modificato

Ingrediente	Quantità
Primosale	80 g
Ventricina piccante senza glutine	25 g
Radicchio	80 g
Cipolle rosse	50 g
Patate	450 g
Crema di ceci senza sale aggiunto	250 g
Mais dolce sgocciolato e sciacquato	150 g
Uvetta	50 g
Miele	25 g
Parmigiano Reggiano DOP	10 g
Paprika piccante / peperoncino	q.b.

Valori nutrizionali stimati

Valore	Ricetta originale	Nuova versione stimata
Energia	1420,85 kcal	circa 2050–2150 kcal
Proteine	68,82 g	circa 65–75 g
Lipidi	72,15 g	circa 48–55 g
Carboidrati	126,83 g	circa 330–360 g
Sodio	3877,16 mg	circa 2600–3000 mg
Fibre totali	6,89 g	circa 25–30 g

Rispetto alla pizza iniziale si ottiene quindi:

- energia superiore a 2000 kcal;
- carboidrati molto aumentati;
- grassi più bassi rispetto all'originale;
- sodio ridotto, anche se non bassissimo;
- pizza senza glutine, utilizzando mix e ventricina certificati senza glutine.

Perché questa modifica funziona
La ricetta originale presentava un elevato contenuto di grassi e sodio soprattutto a causa di alcuni ingredienti:

Ingrediente	Problema
Primosale 150 g	Molto sodio e grassi
Ventricina 60 g	Molto sodio e grassi
Parmigiano 25 g	Ingrediente sapido e grasso

Nella nuova versione questi ingredienti non vengono eliminati del tutto, perché altrimenti la pizza perderebbe la propria identità. Tuttavia, vengono trasformati in ingredienti di carattere:

- la ventricina viene ridotta a 25 g, solo per dare piccantezza e sapore;
- il primosale viene ridotto a 80 g, per mantenere la parte casearia;
- il Parmigiano viene ridotto a 10 g, solo come nota finale.

Le calorie vengono invece aumentate attraverso ingredienti più ricchi di carboidrati:

- patate;
- mais;
- ceci;
- uvetta;
- miele.

In questo modo la pizza rimane abbondante e calorica, ma non diventa una bomba di grassi e sale.

Versione finale in forma compatta
Pizza radicchio, ventricina piccante e primosale – senza glutine

Ingrediente	Quantità
Mix per pizza senza glutine	175 g
Acqua	125–130 g
Olio extravergine d'oliva	15 g
Lievito di birra fresco	1 g
Sale fino	3,5 g
Primosale	80 g
Ventricina piccante senza glutine	25 g
Radicchio	80 g
Cipolle rosse	50 g
Patate	450 g
Crema di ceci senza sale aggiunto	250 g
Mais dolce	150 g
Uvetta	50 g
Miele	25 g
Parmigiano Reggiano DOP	10 g

Questa è una pizza molto calorica e senza glutine, ma più bilanciata: il principale aumento energetico deriva dai carboidrati, non dai grassi.

DeepSeek

Nuova ricetta – Pizza Radicchio e Legumi
Senza glutine, alta energia, basso contenuto di grassi e sodio

Impasto senza glutine
Quantità pressoché invariate

Ingrediente	Quantità	Note
Mix per pizza senza glutine	175,0 g	Ad esempio Caputo Fioreglut, Schär o preparato casalingo
Acqua	112,5 g	Invariata
Olio extravergine d'oliva	15,0 g	Invariato
Lievito di birra fresco	1,0 g	Controllare che sia senza glutine
Sale fino	5,0 g	Invariato

Condimenti nuovi
Tutti naturalmente senza glutine

Ingrediente	Quantità
Radicchio tagliato a strisce	50 g
Cipolle rosse affettate sottili	25 g
Patate bollite o al vapore, affettate	500 g
Riso bianco cotto, oppure riso venere ben cotto	300 g
Mais dolce in scatola, sgocciolato	200 g
Ceci cotti, sgocciolati	200 g
Lenticchie cotte, sgocciolate	50 g
Peperoncino rosso in polvere o fresco	q.b., circa 1–2 g
Origano, aglio in polvere, pepe nero	a piacere

Nota
Tutti i legumi e i cereali devono essere ben scolati dall'acqua di conservazione o di cottura, per evitare un eccesso di umidità sulla pizza.

Profilo nutrizionale
Valori approssimativi

Nutriente	Nuovo valore	Originale	Variazione
Energia	∼2085 kcal	1421 kcal	+664 kcal
Proteine	∼68 g	68,8 g	−0,8 g
Lipidi	∼15 g	72,2 g	−57,2 g
Carboidrati	∼398 g	126,8 g	+271 g
Sodio	∼2050 mg	3877 mg	−1827 mg
Fibre totali	∼28 g	6,9 g	+21 g

Il sodio deriva quasi esclusivamente dai 5 g di sale presenti nell'impasto, pari a circa 2000 mg. Vegetali e legumi apportano soltanto tracce di sodio.

Preparazione

1. **Impasto senza glutine:** mescolare il mix per pizza con il lievito sbriciolato, aggiungere l'acqua a temperatura ambiente, l'olio e il sale. Impastare fino a ottenere un composto morbido. L'impasto senza glutine risulta più appiccicoso rispetto a quello tradizionale, poiché non sviluppa la maglia glutinica. Coprire e lasciare lievitare in luogo tiepido per 2–3 ore, oppure fino al raddoppio.

2. **Preparazione dei condimenti:** lessare le patate con la buccia, poi sbucciarle e affettarle a rondelle di circa 3–4 mm. Cuocere riso, ceci e lenticchie, se non si usano prodotti già cotti in scatola, e scolarli molto bene. Lavare e tagliare radicchio e cipolla.
3. **Stesura della pizza:** stendere l'impasto su carta forno, aiutandosi con le mani infarinate con mix senza glutine. La base non deve essere troppo sottile, dato il peso elevato dei condimenti.
4. **Composizione:** distribuire uniformemente uno strato di patate affettate, quindi aggiungere riso, ceci, lenticchie e mais. Completare con radicchio, cipolla, peperoncino, origano, aglio in polvere e pepe nero.
5. **Cottura:** cuocere in forno statico preriscaldato a 250 °C, o alla massima temperatura, con pietra refrattaria o teglia. La cottura richiede circa 12–15 minuti, fino a quando i bordi risultano dorati e i condimenti caldi. Se si usa il forno ventilato, ridurre la temperatura a circa 230 °C.

Perché soddisfa le richieste

- **Energia superiore a 2000 kcal:** l'elevato apporto calorico deriva dall'abbondanza di carboidrati complessi, provenienti da patate, riso, mais e legumi.
- **Assenza di glutine:** la farina 0 viene sostituita con un mix per pizza senza glutine.
- **Riduzione dei grassi:** la ricetta contiene solo i 15 g di olio presenti nell'impasto e non prevede formaggi o salumi.
- **Riduzione del sodio:** il sodio deriva quasi esclusivamente dal sale dell'impasto, poiché non sono presenti insaccati o formaggi salati.
- **Aumento dei carboidrati:** i carboidrati passano da circa 127 g a circa 398 g.
- **Fedeltà all'idea originale:** radicchio e cipolla rimangono presenti, il peperoncino mantiene la nota piccante e la base resta quella di una pizza.

Suggerimento extra
Per ottenere un sapore più simile al formaggio, senza aggiungere lattosio né glutine, si possono inserire 10–15 g di lievito alimentare in scaglie prima della cottura. Questa aggiunta non altera in modo significativo il contenuto di grassi o sodio.
La pizza risulta quindi senza glutine, ricca di energia e basata soprattutto su carboidrati complessi.

Claude

Pizza Radicchio, Ventricina Piccante e "Primosale" – versione senza glutine, oltre 2000 kcal
Impasto gluten-free con quantità sostanzialmente in linea con l'originale; condimento riequilibrato per ridurre grassi saturi e sodio e aumentare l'apporto energetico oltre le 2000 kcal, puntando soprattutto su carboidrati complessi e grassi insaturi di qualità, come olio extravergine d'oliva, frutta secca e avocado.

Impasto senza glutine

Ingrediente	Prima	Dopo	Note
Farina 0	175 g	rimossa	Contiene glutine
Mix farine senza glutine	–	180 g	Sostituisce la farina 0; +5 g per compensare la minor resa elastica
Psillio in polvere	–	4 g	Dà struttura e tenuta all'impasto gluten-free, aumentando anche la fibra
Acqua	112,5 g	125 g	Le farine senza glutine assorbono più acqua
Olio extravergine d'oliva	15 g	12 g	Lieve riduzione
Lievito di birra fresco	1 g	2 g	Le farine senza glutine richiedono una spinta maggiore
Sale fino	5 g	3 g	Ridotto per contenere il sodio

Il peso dell'impasto è pari a circa 326 g, rispetto ai 308,5 g originali. La variazione è quindi contenuta.

Condimento riadattato

Ingrediente	Prima	Dopo	Motivo
Primosale	150 g	130 g	Lieve riduzione: resta protagonista ma con meno sodio e grassi saturi
Ventricina piccante	60 g	35 g	Dimezzata, perché rappresenta una delle principali fonti di sodio e grassi saturi
Parmigiano Reggiano DOP	25 g	rimosso	Molto sodico; sostituito da una nota umami vegetale
Lievito alimentare in scaglie	–	6 g	Recupera la sapidità umami del Parmigiano, senza aggiungere molto sodio
Radicchio	50 g	90 g	Più volume e fibra, con calorie trascurabili
Cipolle rosse	25 g	40 g	Più dolcezza, per bilanciare il piccante
Avocado a fette	–	120 g	A crudo dopo cottura; apporta grassi monoinsaturi, cremosità e circa 190 kcal
Noci tritate grossolanamente	–	30 g	Fonte di grassi polinsaturi, croccantezza e circa 200 kcal
Patate a fettine sottili	–	150 g	Aggiungono carboidrati complessi, circa 115 kcal, e legano gli ingredienti
Olio EVO a crudo	–	10 g	Circa 90 kcal di grassi buoni per raggiungere il target calorico
Miele di castagno	–	15 g	Circa 45 kcal; contrasta il piccante e si abbina al radicchio

Il peso totale della pizza è pari a circa 955 g.

Valori nutrizionali stimati

Nutriente	Originale	Riadattata	Δ
Energia	1420,85 kcal	∼2080 kcal	+46%, obiettivo raggiunto
Proteine	68,82 g	∼58 g	−16%
Lipidi	72,15 g	∼108 g	+50% in valore assoluto, ma con profilo qualitativo migliore
Carboidrati	126,83 g	∼195 g	+54%
Sodio	3877 mg	∼1750 mg	−55%
Fibre totali	6,89 g	∼15 g	+118%

Le stime sono indicative e possono variare in base alla marca dei prodotti utilizzati. I grassi saturi sono stimati intorno a 22 g, contro circa 32 g della ricetta originale, mentre il resto dei lipidi deriva principalmente da olio EVO, avocado e noci.

Nota importante sull'obiettivo di limitare i grassi
Centrare contemporaneamente un apporto energetico superiore a 2000 kcal e una riduzione dei grassi totali è complesso, perché gli ingredienti più calorici sono spesso anche ricchi di lipidi. In questa proposta i grassi totali aumentano in valore assoluto, ma cambia la loro qualità: si riducono i grassi saturi provenienti da salumi e formaggi stagionati, mentre aumentano i grassi insaturi provenienti da olio extravergine d'oliva, avocado e noci.

Gemini

Per adattare la ricetta raggiungendo l'obiettivo delle oltre 2000 kcal, rendendola senza glutine e mantenendo grassi e sodio contenuti, è necessario intervenire in modo strategico sui macronutrienti.
Poiché i grassi, essendo i più densi dal punto di vista calorico, devono essere limitati, l'aumento energetico deve provenire soprattutto da un incremento significativo di carboidrati complessi e proteine magre.

Nuova ricetta
Senza glutine, >2000 kcal, grassi e sodio limitati
La farina 0 viene sostituita con un mix senza glutine e vengono aggiunti alimenti ad alta densità energetica, ma con contenuto lipidico più contenuto.

Ingrediente	Quantità	Note
Mix farina senza glutine	220,0 g	Aumentata per volume e carboidrati
Acqua	140,0 g	Proporzionata alla nuova quantità di farina
Olio extravergine d'oliva	10,0 g	Ridotto rispetto ai 15 g originali
Lievito di birra fresco	1,0 g	Invariato
Sale fino	3,0 g	Ridotto per abbassare il sodio
Hummus di ceci fatto in casa	150,0 g	Base proteica e ricca di carboidrati; sostituisce i formaggi
Petto di pollo grigliato	120,0 g	Proteina magra per aumentare le calorie senza eccesso di grassi
Radicchio	80,0 g	Aumentato per fibra e volume
Cipolle rosse	50,0 g	Aumentate per carboidrati semplici e sapore
Quinoa lessata	60,0 g	Aggiunta per densità calorica da carboidrati

Strategia di bilanciamento

- **Senza glutine:** la farina 0 viene sostituita con un mix per pane o pizza senza glutine. Per compensare la mancanza di glutine e garantire una struttura adeguata, il mix dovrebbe contenere farina di riso, fecola e una piccola quantità di gomma di xantano.

- **Raggiungimento delle oltre 2000 kcal:** l'aumento calorico deriva soprattutto dai carboidrati complessi e dalle proteine magre.
- **Carboidrati:** l'aggiunta di quinoa e l'aumento della quantità di farina base alzano la quota di carboidrati complessi.
- **Proteine:** il petto di pollo, pari a 120 g, apporta una quota proteica elevata senza i grassi saturi della ventricina.
- **Limitazione di grassi e sodio:** vengono eliminati primosale, Parmigiano e ventricina, cioè le principali fonti di sodio, grassi saturi e sale della ricetta originale.
- **Sostituzione dei formaggi:** l'hummus di ceci funge da crema legante. È ricco di carboidrati e proteine vegetali e, se preparato in casa, permette di controllare la quantità di sale, riducendo il sodio rispetto ai formaggi stagionati.

Profilo nutrizionale stimato

Valore	Stima
Calorie	∼2050 kcal
Carboidrati	Aumentati significativamente grazie al mix senza glutine, alla quinoa e ai ceci
Lipidi e sodio	Mantenuti più bassi tramite l'uso di 10 g di olio EVO e l'eliminazione di salumi e formaggi salati

Consigli per il successo

- **Gestione del sodio:** utilizzare erbe aromatiche, come origano, timo e rosmarino, e spezie, come paprika dolce e pepe, per insaporire pollo e hummus. In questo modo si compensa la riduzione del sale senza aggiungere calorie rilevanti.
- **Impasto senza glutine:** l'impasto risulterà più idratato rispetto a quello con glutine. Conviene lavorarlo con le mani leggermente umide, così da stenderlo senza che si attacchi.
- **Cottura:** il petto di pollo deve essere grigliato separatamente e aggiunto a metà cottura sulla pizza, così da non seccarsi troppo.

Questa configurazione permette di raggiungere l'obiettivo energetico richiesto, mantenendo un profilo nutrizionale più controllato rispetto alla ricetta originale.

Confronto delle risposte

In questo prompt viene analizzato come i modelli interpretano in modo diverso il concetto di "limitare" grassi e sodio. *ChatGPT* orienta la ricetta verso un forte aumento dei carboidrati cercando di preservare l'identità della pizza mantenendo, seppur in quantità ridotte, gli ingredienti che ne caratterizzano il sapore. Il prodotto finale, però, risulta una pizza molto distante dall'equilibrio della preparazione originale, lasciando l'impressione di una soluzione costruita principalmente per soddisfare i vincoli numerici piuttosto che quelli culinari.

DeepSeek interpreta il problema quasi esclusivamente come un esercizio di ottimizzazione nutrizionale; l'identità della pizza originale passa in secondo piano rispetto al raggiungimento degli obiettivi quantitativi. Inoltre, la ricetta finale presenta forti criticità nella scelta degli alimenti: riso bianco, ceci e lenticchie non sono ingredienti propriamente adatti ad una pizza.

Claude, a differenza degli altri modelli, spiega che ottenere oltre 2000 kcal limitando i grassi è intrinsecamente complesso, poiché gli alimenti più energetici sono generalmente anche quelli più ricchi di lipidi. Questo tipo di risposta dimostra una maggiore capacità di

ragionamento sul problema piuttosto che una semplice ricerca di una soluzione. Tuttavia, il modello introduce ingredienti ad elevata densità lipidica, e giustifica il tutto distinguendo tra qualità dei grassi e quantità assoluta

Gemini previlegia un equilibrio nutrizionale complessivo. L'aumento calorico avviene attraverso l'inserimento di carboidrati e proteine magre, a discapito di salumi e formaggi che vengono rimossi totalmente. Ciò, comporta nella criticità già analizzata nelle risposte degli altri modelli, ossia la modifica radicale all'identità della pizza. Inoltre, il modello fornisce una giustificazione poco approfondita delle proprie scelte.

3.5.3 Terzo prompt

Utente

PIZZA RADICCHIO, VENTRICINA PICCANTE E PRIMOSALE - Dosi per 1 pizza (peso 618,5g)

- Farina 0 W300 175,0 g
- Acqua 112,5 g
- Lievito di birra fresco 1,0 g
- Sale fino 5,0 g
- Olio extravergine d'oliva 15,0 g
- Primosale 150,0 g
- Salame ventricina piccante 60,0 g
- Radicchio 50,0 g
- Parmigiano Reggiano DOP 25,0 g
- Cipolle rosse 25,0

Valori nutrizionali:

- Energia (kcal): 1420,85
- Proteine (g): 68,82
- Lipidi (g): 72,15
- Carboidrati (g): 126,83
- Sodio (mg): 3877,16
- Fibre totali (g): 6,89

Partendo da quantità e valori nutrizionali dati, voglio che adatti la mia ricetta, pur rimanendo fedele all'idea della pizza, per ottenere un quantitativo energetico di oltre 2000kcal, mantenendo le quantità dell'impasto pressoché invariati e limitando il valore di grassi e sodio, inoltre la pizza deve essere senza glutine. Voglio che il peso totale della pizza pre cottura non sia eccessivo

Risposte al prompt iniziale

ChatGPT

Pizza radicchio, ventricina piccante e primosale – senza glutine, >2000 kcal

Impasto pressoché invariato

Ingrediente	Quantità
Mix per pizza senza glutine	175,0 g
Acqua	125,0 g
Olio extravergine d'oliva	15,0 g
Lievito di birra fresco	1,0 g
Sale fino	3,5 g

Le quantità dell'impasto vengono mantenute quasi uguali alla ricetta originale. L'acqua viene leggermente aumentata perché il mix senza glutine assorbe più liquidi, mentre il sale viene ridotto da 5 g a 3,5 g per abbassare il contenuto di sodio.

Condimento modificato

Ingrediente	Quantità
Primosale	70,0 g
Ventricina piccante certificata senza glutine	20,0 g
Radicchio	70,0 g
Cipolle rosse	35,0 g
Parmigiano Reggiano DOP	10,0 g
Fiocchi di patate senza glutine	60,0 g
Acqua per reidratare i fiocchi	35,0 g
Composta/confettura di fichi o cipolle senza glutine	120,0 g
Miele	60,0 g
Uvetta	80,0 g

Il peso totale pre-cottura stimato è pari a circa 880 g. Non si tratta quindi di una pizza leggerissima, ma per superare le 2000 kcal senza aumentare eccessivamente grassi e sodio, questa soluzione risulta abbastanza compatta. Scendere molto sotto gli 800 g sarebbe difficile senza utilizzare grandi quantità di olio, formaggi o salumi.

Valori nutrizionali stimati

Valore	Ricetta originale	Nuova versione stimata
Energia	1420,85 kcal	circa 2000–2050 kcal
Proteine	68,82 g	circa 45–55 g
Lipidi	72,15 g	circa 35–40 g
Carboidrati	126,83 g	circa 330–350 g
Sodio	3877,16 mg	circa 2200–2600 mg
Fibre totali	6,89 g	circa 12–16 g

Rispetto alla ricetta originale si ottiene quindi:

- energia superiore a 2000 kcal;
- grassi molto più bassi;
- sodio ridotto;
- carboidrati molto aumentati;

- pizza senza glutine;
- peso pre-cottura non eccessivo rispetto all'obiettivo calorico.

Perché questa versione resta fedele alla pizza originale
L'idea della pizza originale era basata sull'abbinamento tra radicchio, ventricina piccante, primosale, cipolla e Parmigiano. In questa versione gli elementi principali vengono mantenuti, ma ridimensionati.

Ingrediente originale	Modifica
Primosale 150 g	Ridotto a 70 g
Ventricina 60 g	Ridotta a 20 g
Parmigiano 25 g	Ridotto a 10 g
Radicchio 50 g	Aumentato a 70 g
Cipolle rosse 25 g	Aumentate a 35 g

La ventricina resta presente, ma diventa soprattutto un ingrediente aromatico: fornisce piccantezza e sapore senza aumentare troppo grassi e sodio.

Da dove arrivano le calorie in più
Le calorie non vengono aumentate attraverso olio, salame o formaggi, ma soprattutto tramite carboidrati.

Ingrediente	Funzione
Fiocchi di patate	Aumentano i carboidrati senza aggiungere troppo peso
Composta di fichi/cipolle	Aggiunge energia e si abbina a radicchio e ventricina
Miele	Aumenta molto le kcal senza aggiungere grassi e sodio
Uvetta	Apporta carboidrati concentrati con peso contenuto

Il risultato è una pizza più agrodolce-piccante, coerente con radicchio, cipolla rossa, primosale e ventricina.

Versione finale pronta da copiare
Pizza radicchio, ventricina piccante e primosale – senza glutine, >2000 kcal

Ingrediente	Quantità
Mix per pizza senza glutine	175,0 g
Acqua	125,0 g
Olio extravergine d'oliva	15,0 g
Lievito di birra fresco	1,0 g
Sale fino	3,5 g
Primosale	70,0 g
Ventricina piccante senza glutine	20,0 g
Radicchio	70,0 g
Cipolle rosse	35,0 g
Parmigiano Reggiano DOP	10,0 g
Fiocchi di patate senza glutine	60,0 g
Acqua per fiocchi di patate	35,0 g
Composta/confettura di fichi o cipolle senza glutine	120,0 g
Miele	60,0 g
Uvetta	80,0 g

Per il gusto, la base può essere composta con una crema di fiocchi di patate, poi primosale, radicchio, cipolla, poca ventricina, uvetta e Parmigiano. A fine cottura si possono aggiungere miele e composta a filo o a cucchiaiate, così da mantenerli più controllati ed evitare che brucino.

DeepSeek

Nuova ricetta – Pizza Radicchio, Riso Soffiato e Ceci
Senza glutine, >2000 kcal

Impasto senza glutine
Quantità pressoché invariate

Ingrediente	Quantità	Note
Mix per pizza senza glutine	175,0 g	Ad esempio Caputo Fioreglut o Schär
Acqua	112,5 g	Invariata
Olio extravergine d'oliva	15,0 g	Invariato
Lievito di birra fresco	1,0 g	Senza glutine
Sale fino	5,0 g	Invariato

Condimenti
Tutti senza glutine e a basso contenuto di sodio

Ingrediente	Quantità
Radicchio tagliato a strisce	50 g
Cipolle rosse affettate sottili	25 g
Patate dolci o comuni, bollite o arrosto	150 g
Ceci cotti senza sale aggiunto, sgocciolati	100 g
Riso soffiato naturale, senza zuccheri	150 g
Olio extravergine d'oliva extra	50 g
Peperoncino rosso in polvere	q.b.
Origano e aglio in polvere	a piacere

Profilo nutrizionale
Valori approssimativi

Nutriente	Nuovo valore	Originale	Variazione
Energia	∼2050 kcal	1421 kcal	+629 kcal
Proteine	∼40 g	68,8 g	−28,8 g
Lipidi	∼65 g	72,2 g	−7,2 g
Carboidrati	∼300 g	126,8 g	+173 g
Sodio	∼2050 mg	3877 mg	−1827 mg
Fibre totali	∼9 g	6,9 g	+2,1 g
Peso totale pre-cottura	∼930 g	618 g	+312 g

Il sodio deriva quasi esclusivamente dai 5 g di sale presenti nell'impasto, pari a circa 2000 mg. I ceci e gli altri vegetali sono senza sale aggiunto. I grassi totali sono stimati intorno a 65 g, quindi inferiori rispetto ai 72 g della ricetta originale, e provengono principalmente dall'olio extravergine d'oliva, ricco di grassi insaturi.

Preparazione

1. **Impasto senza glutine:** mescolare il mix senza glutine, il lievito sbriciolato, l'acqua, 15 g di olio e il sale. Impastare fino a ottenere un composto morbido. L'impasto risulterà appiccicoso, caratteristica normale negli impasti senza glutine. Coprire e lasciare lievitare in luogo tiepido per 2–3 ore, oppure fino al raddoppio.
2. **Preparazione dei condimenti:** lessare le patate, oppure cuocerle al forno, e tagliarle a rondelle. Scolare bene i ceci, preferibilmente scegliendo quelli senza sale aggiunto. Tagliare radicchio e cipolla. Tenere da parte il riso soffiato.

3. **Stesura della pizza:** stendere l'impasto su carta forno, aiutandosi con un po' di mix senza glutine per evitare che si attacchi. Spennellare la superficie con metà dell'olio extra, pari a circa 25 g.
4. **Composizione:** distribuire le patate, i ceci, il radicchio e la cipolla. Cospargere il riso soffiato in modo uniforme. Aggiungere peperoncino, origano e aglio. Infine, versare a filo l'olio extra rimasto, pari a circa 25 g.
5. **Cottura:** cuocere in forno preriscaldato a 250 °C, o alla massima temperatura, con pietra refrattaria o teglia. La cottura richiede circa 12–15 minuti, fino a quando i bordi risultano dorati e il riso soffiato leggermente tostato. Se si utilizza il forno ventilato, ridurre la temperatura a circa 230 °C.

Perché soddisfa le richieste

- **Energia superiore a 2000 kcal:** l'obiettivo viene raggiunto grazie all'olio extra, pari a 50 g e circa 450 kcal, e al riso soffiato, pari a 150 g e circa 570 kcal, uniti all'impasto e agli altri ingredienti.
- **Assenza di glutine:** la farina 0 viene sostituita con un mix specifico senza glutine.
- **Grassi limitati:** i grassi totali sono stimati intorno a 65 g, quindi inferiori rispetto ai 72 g della ricetta originale.
- **Sodio limitato:** il sodio passa da circa 3877 mg a circa 2050 mg, grazie all'eliminazione di formaggi e salame.
- **Peso non eccessivo:** il peso pre-cottura è pari a circa 930 g, quindi gestibile per una pizza grande.
- **Fedeltà all'idea originale:** radicchio, cipolla e peperoncino mantengono il carattere della pizza di partenza. La consistenza rimane quella di una pizza ricca e saporita.

Suggerimento
Per ottenere un sapore ancora più piccante, simile al carattere della ventricina, si può aggiungere anche un pizzico di paprika affumicata o peperoncino in fiocchi.
Il risultato è una pizza senza glutine, ad alta energia e con un contenuto di sodio e grassi più controllato rispetto alla versione originale.

Claude

Pizza Radicchio, Ventricina Piccante e "Primosale" – senza glutine, oltre 2000 kcal
Versione gluten-free, con impasto sostanzialmente invariato nelle quantità, grassi e sodio contenuti rispetto all'originale, energia spinta oltre le 2000 kcal sfruttando ingredienti densi caloricamente ma di peso contenuto, come frutta secca, olio extravergine d'oliva a crudo e miele. In questo modo il peso totale pre-cottura resta abbastanza vicino a quello della pizza originale.

Impasto senza glutine

Ingrediente	Prima	Dopo	Note
Farina 0	175 g	rimossa	Contiene glutine
Mix farine senza glutine	–	175 g	Sostituzione 1:1
Psillio in polvere	–	4 g	Struttura e tenuta dell'impasto gluten-free, con aumento della fibra
Acqua	112,5 g	120 g	Le farine senza glutine assorbono leggermente di più
Olio extravergine d'oliva nell'impasto	15 g	12 g	Lieve riduzione
Lievito di birra fresco	1 g	2 g	Le farine senza glutine richiedono una spinta maggiore
Sale fino	5 g	3 g	Ridotto per contenere il sodio

Il peso dell'impasto è pari a circa 316 g, rispetto ai 308,5 g originali. La variazione risulta quindi trascurabile.

Condimento riadattato

Ingrediente	Prima	Dopo	Motivo
Primosale	150 g	120 g	Resta protagonista, ma viene lievemente ridotto per contenere sodio e grassi saturi
Ventricina piccante	60 g	30 g	Dimezzata, perché rappresenta una delle principali fonti di sodio e grassi saturi
Parmigiano Reggiano DOP	25 g	rimosso	Molto sodico
Lievito alimentare in scaglie	–	5 g	Recupera la nota umami del Parmigiano senza aggiungere molto sodio
Radicchio	50 g	70 g	Più fibra e volume, con calorie trascurabili
Cipolle rosse	25 g	30 g	Bilanciano il piccante
Noci sgusciate tritate	–	40 g	A crudo dopo cottura; circa 260 kcal in soli 40 g, con grassi polinsaturi
Olio EVO a crudo	–	18 g	Circa 160 kcal in 18 g, con grassi monoinsaturi di qualità
Miele di castagno	–	20 g	Circa 60 kcal; contrasta il piccante e si abbina all'amaro del radicchio
Uvetta sultanina	–	20 g	Circa 60 kcal; dolcezza concentrata in poco peso, coerente con radicchio e ventricina

Il peso totale pre-cottura della pizza è pari a circa 675 g, rispetto ai 618,5 g della ricetta originale. L'aumento è quindi di circa il 9%, rimanendo abbastanza contenuto.

Valori nutrizionali stimati

Nutriente	Originale	Riadattata	Δ
Energia	1420,85 kcal	∼2050 kcal	+44%, obiettivo raggiunto
Proteine	68,82 g	∼55 g	−20%
Lipidi totali	72,15 g	∼108 g	+50% in valore assoluto, ma con profilo qualitativo migliore
Carboidrati	126,83 g	∼155 g	+22%
Sodio	3877 mg	∼1650 mg	−57%, obiettivo raggiunto
Fibre totali	6,89 g	∼10 g	+45%

Le stime sono indicative e possono variare in base alla marca dei prodotti utilizzati, in particolare per il primosale, di cui è opportuno controllare l'etichetta nutrizionale. I grassi saturi sono stimati intorno a 20 g, contro circa 32 g della ricetta originale, mentre il resto dei lipidi deriva principalmente da olio extravergine d'oliva e noci.

Nota onesta sull'obiettivo di limitare i grassi
Centrare contemporaneamente un apporto energetico superiore a 2000 kcal, un peso contenuto e grassi assoluti bassi è matematicamente molto difficile. Per raggiungere circa 2050 kcal in una pizza da circa 675 g serve infatti una densità calorica elevata, e i grassi, con 9 kcal/g, sono la via più efficiente in termini di peso.
Per questo motivo i grassi totali aumentano in valore assoluto, ma cambia la loro qualità:

- i grassi saturi calano di circa il 38% rispetto all'originale;
- la maggior parte dei lipidi aggiunti è monoinsatura o polinsatura, proveniente da olio EVO e noci;
- il sodio cala di circa il 57%, centrando pienamente l'obiettivo.

Se si volessero tenere più bassi i grassi totali, accettando però un peso maggiore della pizza, l'alternativa sarebbe aumentare ulteriormente i carboidrati, ad esempio aggiungendo circa 80 g di patate a fettine sottili.

Note di preparazione

- **Radicchio:** tagliarlo a listarelle e scottarlo per circa 1 minuto in padella antiaderente senza grassi, così da smorzare l'amaro e ridurre il rilascio di acqua in cottura.
- **Uvetta:** ammollarla per circa 10 minuti in acqua tiepida, poi strizzarla bene.
- **Cottura:** cuocere a circa 230 °C in forno ventilato per 10–12 minuti. Le farine senza glutine gradiscono una cottura leggermente più lunga e meno aggressiva.
- **Topping a crudo dopo cottura:** aggiungere primosale, ventricina, cipolla, noci, uvetta, lievito alimentare, filo di olio EVO e gocce di miele a forno spento, con la pizza appena sfornata. In questo modo i grassi della ventricina non colano sull'impasto, gli aromi restano vivi e l'olio EVO mantiene meglio il proprio profilo nutrizionale.

Gemini

Nuova ricetta
Senza glutine, >2000 kcal, basso contenuto di sodio e grassi

Ingrediente	Quantità	Note nutrizionali
Mix farine senza glutine, riso/grano saraceno	200,0 g	Carboidrati complessi ad alta energia
Acqua	150,0 g	Idratazione necessaria per farine senza glutine
Olio extravergine d'oliva	10,0 g	Ridotto per controllare i lipidi
Lievito di birra fresco	1,0 g	Invariato
Sale fino	2,0 g	Ridotto drasticamente per contenere il sodio
Crema di mandorle spalmabile	30,0 g	Il segreto calorico: grassi sani e alta densità energetica
Petto di tacchino al naturale	120,0 g	Proteina magra, senza il sodio dei salumi
Radicchio	60,0 g	Invariato
Cipolle rosse	40,0 g	Aumentate per sapore e carboidrati
Semi di zucca/girasole	25,0 g	Densità calorica elevata e buon apporto di micronutrienti
Lievito alimentare in scaglie	15,0 g	Effetto "formaggio" senza sodio né lattosio

Strategia nutrizionale

- **Obiettivo superiore a 2000 kcal:** le mandorle, sotto forma di crema pura o tritate finemente come base, e i semi rappresentano la chiave della ricetta. Forniscono infatti un'elevata densità energetica in poco peso, senza l'elevato apporto di sodio tipico di primosale, Parmigiano e ventricina.
- **Sostituzione della ventricina:** il tacchino sostituisce il salume, garantendo un buon apporto proteico senza il sale aggiunto tipico della stagionatura.
- **Senza glutine:** l'uso di una base di farina di riso e grano saraceno garantisce una buona carica energetica. Il grano saraceno, in particolare, presenta una densità calorica e nutrizionale utile per aumentare l'apporto energetico senza ricorrere a porzioni eccessive.
- **Riduzione di sodio e grassi saturi:** vengono rimossi formaggi e insaccati, cioè primosale, Parmigiano e ventricina, responsabili di gran parte del sodio e dei grassi saturi della ricetta originale.
- **Sapidità alternativa:** il sapore viene costruito attraverso radicchio spadellato con aceto balsamico, senza aggiunta di sale, e lievito alimentare, che apporta una nota sapida e umami naturale.

Peso totale
Con circa 650 g di peso totale pre-cottura, la pizza rimane maneggevole e non eccessiva, ma risulta comunque molto densa dal punto di vista nutrizionale.

Consigli per l'esecuzione

- **La base:** spalmare la crema di mandorle sulla base dell'impasto prima di infornare. Durante la cottura diventerà una crema ricca, utile a sostituire la componente formaggiosa.
- **Il tacchino:** cuocerlo separatamente in padella con erbe aromatiche, come salvia e rosmarino, prima di aggiungerlo alla pizza. In questo modo mantiene meglio succhi e sapore.
- **Sapidità:** poiché il sale viene ridotto a soli 2 g, è utile valorizzare le cipolle rosse stufate lentamente. La loro naturale caramellizzazione crea una dolcezza capace di contrastare l'amaro del radicchio e ridurre la percezione di piatto poco sapido.

Questa ricetta mantiene il concetto della pizza gourmet, ma ne trasforma il profilo nutrizionale per renderlo più energetico e performante.

Confronto delle risposte

Per questo prompt *ChatGPT* adotta un approccio orientato all'ottimizzazione nutrizionale aggiungendo alimenti ricchi di carboidrati e riducendo sensibilmente gli ingredienti iniziali della ricetta. Questa strategia permette di rispettare i vincoli richiesti, ma modifica significativamente il profilo della pizza dando ad essa un gusto agrodolce lontano dalla ricetta originale.

DeepSeek elimina completamente gli alimenti più caratterizzanti della pizza per rimuovere i grassi e il sodio, sostituendoli con patate, ceci e riso soffiato. Il risultato, quindi, rispetta i requisiti nutrizionali ma offre un'idea della pizza totalmente irriconoscibile rispetto alla ricetta di partenza, oltre che ambigua gastronomicamente per la presenza di alimenti come il riso soffiato nel condimento.

Claude propone una soluzione ampiamente argomentata. Esplicita la difficoltà di soddisfare simultaneamente tutti i vincoli e genera una ricetta con olio e frutta secca nel condimento, riducendo le quantità degli alimenti iniziali. L'aspetto più interessante della risposta non è tanto la ricetta finale, quanto la riflessione sul concetto di "limitare i grassi".

Infine, *Gemini*, adotta un'impostazione simile a quella vista per altri modelli, privilegiando il miglioramento del profilo nutrizionale rispetto alla fedeltà della ricetta. Quasi tutti gli ingredienti della pizza originale vengono sostituiti al fine di rispettare i vincoli richiesti; in questo modo l'identità della pizza di partenza viene profondamente modificata.

3.6 Scenario 4: adattamento della pizza mortadella e bufala

3.6.1 Prompt iniziale

Utente

PIZZA MORTADELLA E BUFALA - Dosi per 1 pizza (peso 566.5g)

- Farina 0 W300 250,0 g
- Acqua 150,0 g
- Lievito di birra fresco 0,5 g
- Sale fino 6,0 g
- Olio extravergine d'oliva 10,0 g
- Mozzarella di bufala DOP 100,0 g
- Mortadella 50,0 g

Valori nutrizionali:

- Energia (kcal): 1293,75
- Proteine (g): 50,76
- Lipidi (g): 46,69
- Carboidrati (g): 175,16
- Sodio (mg): 3096,08
- Fibre totali (g): 7,29

Ti ho fornito le quantità e i valori per una pizza bufala e mortadella, ora voglio che applichi a tale ricetta delle modifiche per cui la pizza rimanga con un valore calorico pressoché simile, ma deve essere senza glutine e deve aumentare il numero di proteine a discapito del sodio.

Risposte al prompt iniziale

ChatGPT

Pizza mortadella e bufala senza glutine, più proteica e meno salata
Dosi per 1 pizza – peso circa 600 g

Ingrediente	Quantità nuova	Modifica rispetto all'originale
Mix per pizza senza glutine	210 g	Sostituisce la farina 0
Acqua	170 g	Leggermente più alta, perché il mix senza glutine assorbe di più
Olio extravergine d'oliva	5 g	Ridotto da 10 g per compensare le kcal
Lievito di birra fresco	1,5 g	Leggermente aumentato
Sale fino	2 g	Ridotto da 6 g
Mozzarella di bufala	100 g	Invariata
Mortadella senza glutine	35 g	Ridotta da 50 g
Petto di pollo/tacchino fresco alla piastra, non salato	80 g	Aggiunto per aumentare le proteine con poco sodio

Valori nutrizionali stimati

Valore	Ricetta originale	Ricetta modificata stimata
Energia	1293,75 kcal	circa 1300–1320 kcal
Proteine	50,76 g	circa 54–56 g
Lipidi	46,69 g	circa 44–46 g
Carboidrati	175,16 g	circa 168–172 g
Sodio	3096,08 mg	circa 1500–1650 mg
Fibre totali	7,29 g	circa 5–6 g

Il risultato è coerente con l'obiettivo indicato: le calorie restano quasi uguali, le proteine aumentano e il sodio viene ridotto in modo significativo.
La riduzione più importante deriva dal sale: passando da 6 g a 2 g, si eliminano circa 1600 mg di sodio. Inoltre, riducendo la mortadella da 50 g a 35 g, il contenuto di sodio diminuisce ulteriormente, pur mantenendo il gusto caratteristico della ricetta originale.

Perché questa versione funziona
La farina 0 viene sostituita con un mix senza glutine per pizza, rendendo la base adatta a una dieta senza glutine. Tuttavia, i mix senza glutine hanno spesso un contenuto proteico inferiore rispetto alla farina tradizionale. Per questo motivo viene aggiunto petto di pollo o tacchino fresco, che aumenta l'apporto proteico senza introdurre molto sodio, a differenza di salumi, bresaola, prosciutto o altri affettati proteici.
La mortadella rimane presente, ma in quantità minore: in questo modo la pizza conserva l'identità di mortadella e bufala, senza trasformarsi completamente in un'altra ricetta.

Versione finale da scrivere
Pizza mortadella e bufala senza glutine, più proteica e a ridotto sodio
Dosi per 1 pizza – peso circa 600 g

- Mix per pizza senza glutine: 210 g;
- Acqua: 170 g;
- Olio extravergine d'oliva: 5 g;
- Lievito di birra fresco: 1,5 g;

- Sale fino: 2 g;
- Mozzarella di bufala: 100 g;
- Mortadella senza glutine: 35 g;
- Petto di pollo/tacchino fresco alla piastra non salato: 80 g.

Valore	Stima finale
Energia	circa 1300–1320 kcal
Proteine	circa 54–56 g
Lipidi	circa 44–46 g
Carboidrati	circa 168–172 g
Sodio	circa 1500–1650 mg
Fibre totali	circa 5–6 g

Nota importante
Per rendere la pizza realmente senza glutine, è necessario utilizzare un mix certificato senza glutine e mortadella con dicitura "senza glutine". Anche teglia e utensili devono essere puliti, così da evitare contaminazioni.

DeepSeek

Ecco la ricetta modificata per ottenere una pizza senza glutine, con apporto calorico simile alla ricetta originale, pari a circa 1290 kcal, maggiore contenuto proteico e ridotto contenuto di sodio.

Modifiche apportate

- Sostituzione della farina di grano con un mix senza glutine a più alto contenuto proteico, composto da farina di ceci e farina di riso, con aggiunta di gomma xantana per migliorare la consistenza dell'impasto.
- Riduzione del sale da 6 g a 2 g, così da abbassare in modo significativo il contenuto di sodio.
- Riduzione della mortadella, da 50 g a 23 g, e della mozzarella di bufala, da 100 g a 70 g, per bilanciare le calorie. L'aumento delle proteine viene compensato soprattutto dalla presenza della farina di ceci.

Ingredienti per 1 pizza
Peso totale circa 508 g

Ingrediente	Quantità
Acqua	150,0 g
Farina di riso	120,0 g
Farina di ceci	130,0 g
Gomma xantana	2,0 g
Olio extravergine d'oliva	10,0 g
Lievito di birra fresco	0,5 g
Sale fino	2,0 g
Mozzarella di bufala	70,0 g
Mortadella senza glutine	23,0 g

Valori nutrizionali stimati

Valori riferiti a 1 pizza

Parametro	Valore	Confronto con originale
Energia	∼1292 kcal	Simile, rispetto a 1294 kcal
Proteine	∼51,0 g	In aumento, rispetto a 50,8 g
Lipidi	∼38,8 g	In diminuzione, rispetto a 46,7 g
Carboidrati	∼175,3 g	Simili, rispetto a 175,2 g
Sodio	∼1356 mg	Fortemente ridotto, rispetto a 3096 mg
Fibre totali	∼13,0 g	In aumento, rispetto a 7,3 g

Nota
I valori sono arrotondati e calcolati in base alle tabelle nutrizionali medie degli ingredienti. La gomma xantana non apporta calorie significative.

Istruzioni

1. Mescolare le farine, la gomma xantana e il sale.
2. Sciogliere il lievito nell'acqua tiepida e unire l'olio.
3. Incorporare gli ingredienti liquidi a quelli secchi e impastare. Lasciare lievitare, coperto, per circa 2 ore.
4. Stendere l'impasto senza glutine su carta forno e condire con mozzarella di bufala a pezzi e mortadella a fette.
5. Cuocere in forno statico a 220 °C per circa 12–15 minuti, oppure regolarsi in base al proprio forno.

La pizza risulta quindi con un profilo nutrizionale migliorato: più proteine, meno sodio, assenza di glutine e calorie pressoché invariate rispetto alla versione originale.

Claude

Pizza Mortadella e Bufala – versione gluten-free, high protein, low sodium
Dosi per 1 pizza – peso totale circa 575 g

Ingrediente	Quantità
Acqua	155,0 g
Mix farine senza glutine, ad esempio riso integrale e grano saraceno	180,0 g
Farina di lupino o di soia sgrassata	60,0 g
Psillio in polvere	5,0 g
Olio extravergine d'oliva	10,0 g
Lievito di birra fresco	0,5 g
Sale fino iposodico, con cloruro di potassio	3,0 g
Mozzarella di bufala a ridotto contenuto di sale	110,0 g
Mortadella a ridotto contenuto di sodio	50,0 g

Valori nutrizionali stimati

Parametro	Originale	Riformulata	Δ
Energia	1293,75 kcal	∼1290 kcal	≈
Proteine	50,76 g	∼68 g	+34%
Lipidi	46,69 g	∼48 g	≈
Carboidrati	175,16 g	∼140 g	−20%
Sodio	3096,08 mg	∼1750 mg	−43%
Fibre totali	7,29 g	∼14 g	+90%

Logica delle sostituzioni

- **Farina di lupino o soia sgrassata:** rappresenta l'ingrediente chiave per aumentare il contenuto proteico della pizza. La farina di lupino può arrivare a circa 40 g di proteine per 100 g, contro i circa 10 g della farina 0. Inoltre, è naturalmente senza glutine e contribuisce ad aumentare anche la quota di fibre.
- **Mix riso, grano saraceno e psillio:** questa combinazione permette di ottenere una struttura più adatta a un impasto gluten-free. Lo psillio sostituisce parzialmente la funzione della rete glutinica, evitando una consistenza troppo friabile.
- **Sale iposodico:** la quantità di sale viene ridotta da 6 g a 3 g e parte del cloruro di sodio viene sostituita da cloruro di potassio. Questo passaggio consente, da solo, di ridurre il sodio di circa 1150 mg.
- **Mozzarella di bufala a ridotto sale e mortadella low sodium:** in commercio esistono versioni con circa il 30–40% di sodio in meno rispetto ai prodotti standard, a parità di calorie e proteine. Questi ingredienti rappresentano una leva importante per ridurre il sodio complessivo, poiché salumi e latticini incidono molto sul totale.
- **Idratazione leggermente aumentata:** l'acqua passa a 155 g, perché le farine senza glutine e le farine di legumi assorbono più liquidi rispetto alla farina di grano.

Note pratiche
L'impasto va lasciato lievitare più a lungo a temperatura ambiente, indicativamente per 8–12 ore con 0,5 g di lievito. Va poi steso con le mani inumidite, non con il mattarello, perché l'impasto senza glutine tende a risultare più appiccicoso.
La cottura può essere effettuata a 250 °C per circa 8–10 minuti. La mortadella può essere aggiunta a crudo dopo la cottura, così da preservarne aroma e grassi.

Nota sui valori nutrizionali
I valori sono stime basate su tabelle CREA/USDA e possono variare in base alle marche scelte, soprattutto per i prodotti a ridotto contenuto di sodio.

Gemini

Per trasformare la ricetta in una versione senza glutine e iperproteica, riducendo al contempo il contenuto di sodio, vengono rielaborate le proporzioni degli ingredienti.

Modifiche apportate

- **Farina:** la farina 0 viene sostituita con un mix di farina di riso e farina di lupini. La farina di lupini è utile per aumentare drasticamente il contenuto proteico ed è naturalmente priva di glutine.
- **Sodio:** il sale viene ridotto da 6 g a 3 g. Per compensare la minore sapidità si possono utilizzare erbe aromatiche o spezie. Inoltre, viene suggerito l'uso di mozzarella di bufala ben sgocciolata e mortadella di alta qualità, possibilmente meno salata.

- **Bilanciamento:** la sostituzione permette di mantenere il valore calorico totale vicino a quello originale, spostando però il profilo nutrizionale verso una quota proteica più alta.

Nuova ricetta
Senza glutine, proteica, low sodium

Ingrediente	Quantità
Farina di riso	180 g
Farina di lupini	70 g
Acqua	160 g
Olio extravergine d'oliva	10 g
Lievito di birra fresco	0,5 g
Sale fino	3 g
Mozzarella di bufala	100 g
Mortadella	70 g

L'acqua viene leggermente aumentata per compensare l'assorbimento della farina di lupini.

Stima dei valori nutrizionali

Parametro	Valore stimato	Osservazione
Energia	∼1310 kcal	In linea con la ricetta originale
Proteine	∼75–80 g	Netto aumento rispetto ai 50,76 g originali
Lipidi	∼48 g	Simili alla ricetta originale
Carboidrati	∼150 g	Inferiori rispetto alla versione di partenza
Sodio	∼1800–2000 mg	Forte riduzione rispetto ai 3096 mg originali

Consigli per la riuscita

- **Lavorazione:** gli impasti senza glutine non sviluppano la maglia glutinica. La farina di lupini aiuta la struttura, ma se l'impasto risultasse troppo appiccicoso è preferibile ungere le mani con olio d'oliva invece di aggiungere altra farina.
- **Sapore:** poiché il sale viene ridotto, si possono aggiungere all'impasto origano secco o rosmarino tritato, così da esaltare il gusto senza aumentare il sodio.
- **Lievitazione:** dato che la quantità di lievito è molto bassa, pari a 0,5 g, è consigliabile una maturazione lunga, di almeno 8–12 ore, a temperatura ambiente o in frigorifero. Questo rende la pizza più digeribile e migliora la struttura dell'impasto.

Questa versione mantiene un apporto calorico vicino a quello originale, ma risulta senza glutine, più proteica e con un contenuto di sodio sensibilmente ridotto.

Confronto delle risposte

Per questo prompt, tutti i modelli individuano correttamente le principali fonti di sodio della pizza originale, ossia il sale dell'impasto e della mortadella.

Le differenze tra i modelli emergono, soprattutto, nelle strategie adottate per incrementare il contenuto proteico. *ChatGPT* suggerisce una soluzione che aumenta le proteine, mantiene pressoché invariato il contenuto energetico e riduce sensibilmente il sodio. Tuttavia, l'aggiunta di una nuova fonte proteica modifica parzialmente la natura della pizza originale.

DeepSeek, *Claude* e *Gemini*, invece, adottano un approccio più strutturale, intervenendo direttamente sull'impasto. Tutti propongono infatti l'impiego di farine ricche di proteine

abbinate a farine senza glutine. In questo modo, l'aumento della quota proteica deriva principalmente dalla base della pizza, senza la necessità di aggiungere ulteriori ingredienti nel condimento in modo da lasciare invariata la natura della pizza.

Dal punto di vista del rispetto del prompt, tutti i modelli riescono a mantenere pressoché invariato il contenuto energetico e a ridurre il sodio, ma con risultati differenti nell'aumento delle proteine. *ChatGPT* ottiene un incremento piuttosto contenuto, mentre *DeepSeek* propone un miglioramento limitato facendo affidamento soprattutto sulla farina di ceci. *Claude* e *Gemini*, invece, raggiungono gli incrementi proteici più elevati grazie all'impiego di farine di legumi ad alto contenuto proteico, risultando, quindi, maggiormente aderenti all'obiettivo richiesto.

3.6.2 Secondo prompt

Utente

PIZZA MORTADELLA E BUFALA - Dosi per 1 pizza (peso 566.5g)

- Farina 0 W300 250,0 g
- Acqua 150,0 g
- Lievito di birra fresco 0,5 g
- Sale fino 6,0 g
- Olio extravergine d'oliva 10,0 g
- Mozzarella di bufala DOP 100,0 g
- Mortadella 50,0 g

Valori nutrizionali:

- Energia (kcal): 1293,75
- Proteine (g): 50,76
- Lipidi (g): 46,69
- Carboidrati (g): 175,16
- Sodio (mg): 3096,08
- Fibre totali (g): 7,29

Partendo da questa ricetta, e cercando di mantenere l'idea della pizza, applica delle modifiche affinché la pizza sia di 2000 calorie, mantenendo impasto e mortadella nelle quantità date.

Risposte al secondo prompt

ChatGPT

Pizza mortadella, bufala e pistacchio da circa 2000 kcal

Ingrediente	Quantità originale	Quantità modificata
Acqua	150 g	150 g
Farina 0	250 g	250 g
Olio EVO nell'impasto	10 g	10 g
Lievito di birra fresco	0,5 g	0,5 g
Sale fino	6 g	6 g
Mozzarella di bufala	100 g	250 g
Mortadella	50 g	50 g
Pistacchi non salati/granella di pistacchio	–	30 g
Olio EVO a crudo	–	12 g

Calcolo calorico stimato
La ricetta originale apporta 1293,75 kcal.

Aggiunta	Quantità	kcal stimate
Mozzarella di bufala extra	+150 g	circa 432 kcal
Pistacchi non salati	30 g	circa 169 kcal
Olio EVO a crudo	12 g	circa 108 kcal

Il totale calorico aggiunto è quindi:

$$432 + 169 + 108 = 709 \text{ kcal}$$

Il nuovo totale stimato diventa:

$$1293,75 + 709 = 2002,75 \text{ kcal}$$

La pizza risulta quindi praticamente pari a 2000 kcal.

Valori nutrizionali stimati finali

Valore	Ricetta originale	Ricetta modificata
Energia	1293,75 kcal	circa 2000 kcal
Proteine	50,76 g	circa 82 g
Lipidi	46,69 g	circa 103 g
Carboidrati	175,16 g	circa 184 g
Sodio	3096,08 mg	circa 3650–3800 mg
Fibre totali	7,29 g	circa 10 g

Versione finale pronta
Pizza mortadella, bufala e pistacchio
Dosi per 1 pizza – peso totale circa 758,5 g

- Acqua: 150 g;
- Farina 0: 250 g;
- Olio extravergine d'oliva nell'impasto: 10 g;
- Lievito di birra fresco: 0,5 g;
- Sale fino: 6 g;
- Mozzarella di bufala: 250 g;
- Mortadella: 50 g;

- Pistacchi non salati/granella di pistacchio: 30 g;
- Olio extravergine d'oliva a crudo: 12 g.

Energia stimata: circa 2000 kcal.

Questa modifica resta molto fedele all'idea originale, perché la pizza rimane chiaramente una mortadella e bufala. L'aggiunta del pistacchio la rende però più ricca e più "gourmet", mantenendo un abbinamento coerente con il gusto della mortadella.

DeepSeek

Per ottenere una pizza da circa 2000 calorie mantenendo invariati l'impasto, composto da acqua 150 g, farina 250 g, olio 10 g, lievito 0,5 g e sale 6 g, e mantenendo invariata anche la mortadella a 50 g, è necessario aumentare la quantità di mozzarella di bufala e aggiungere una quota di olio extravergine d'oliva a crudo.

Modifiche alla ricetta originale

Ingrediente	Modifica
Mozzarella di bufala	Da 100 g a 350 g, con aggiunta di 250 g
Olio extravergine d'oliva	Aggiunta di 22 g a crudo, oltre ai 10 g già presenti nell'impasto
Altri ingredienti	Invariati rispetto alla ricetta originale

Il totale dell'olio extravergine d'oliva diventa quindi pari a 32 g, considerando i 10 g presenti nell'impasto e i 22 g aggiunti a crudo.

Nuovi valori nutrizionali stimati
Valori riferiti a 1 pizza

Parametro	Valore
Energia	≈2002 kcal
Proteine	≈65 g
Lipidi	≈72 g
Carboidrati	≈175 g
Sodio	≈3600 mg
Fibre totali	≈7,3 g

Il sodio aumenta rispetto alla ricetta originale a causa della maggiore quantità di mozzarella di bufala.

Nota sul calcolo calorico
Le calorie aggiuntive derivano per circa 510 kcal dalla mozzarella di bufala in più, stimata a circa 2,04 kcal/g secondo i dati originali, e per circa 198 kcal dall'olio extravergine d'oliva aggiunto a crudo.
Il totale calorico aggiunto è quindi:

$$510 + 198 = 708 \text{ kcal}$$

Partendo dalle circa 1294 kcal della ricetta originale, il nuovo totale stimato diventa:

$$1294 + 708 = 2002 \text{ kcal}$$

La pizza raggiunge quindi l'obiettivo di circa 2000 kcal.

Consiglio pratico
Per gestire l'umidità extra dovuta alla maggiore quantità di mozzarella, è consigliabile tamponarla con carta assorbente prima dell'uso. Inoltre, conviene cuocere la pizza su pietra refrattaria o in forno molto caldo, a circa 250 °C per 10–12 minuti.

Claude

Per arrivare a circa 2000 kcal mantenendo invariati impasto e mortadella, è necessario aggiungere circa 700 kcal lavorando soltanto su condimenti e formaggio. Di seguito viene proposta la versione riformulata.

Pizza Mortadella e Bufala – versione 2000 kcal
Dosi per 1 pizza – peso totale circa 736 g

Ingrediente	Quantità	Note
Acqua	150,0 g	Invariata
Farina 0	250,0 g	Invariata
Olio extravergine d'oliva nell'impasto	10,0 g	Invariato
Lievito di birra fresco	0,5 g	Invariato
Sale fino	6,0 g	Invariato
Mortadella	50,0 g	Invariata
Mozzarella di bufala	150,0 g	+50 g
Granella di pistacchi	25,0 g	Nuovo ingrediente
Stracciatella di bufala	40,0 g	Nuovo ingrediente
Olio EVO a crudo	15,0 g	Nuovo ingrediente
Scaglie di Grana Padano	20,0 g	Nuovo ingrediente

Valori nutrizionali stimati

Parametro	Originale	Riformulata	Δ
Energia	1293,75 kcal	∼2000 kcal	+55%
Proteine	50,76 g	∼73 g	+44%
Lipidi	46,69 g	∼107 g	+130%
Carboidrati	175,16 g	∼180 g	≈
Sodio	3096,08 mg	∼3650 mg	+18%
Fibre totali	7,29 g	∼8,5 g	+17%

Logica delle aggiunte
Le circa 700 kcal mancanti vengono distribuite su ingredienti coerenti con una pizza mortadella e bufala, evitando di snaturare la ricetta.

- **Mozzarella di bufala:** l'aumento di 50 g apporta circa 140 kcal e rinforza la base lattica già prevista nella ricetta originale.
- **Stracciatella:** i 40 g aggiunti forniscono circa 110 kcal. Inserita a crudo dopo la cottura, rappresenta una naturale evoluzione gourmet dell'abbinamento bufala e mortadella.
- **Granella di pistacchi:** i 25 g aggiunti apportano circa 150 kcal. Il pistacchio è un abbinamento classico con la mortadella e aggiunge grassi buoni e croccantezza.
- **Olio EVO a crudo:** i 15 g aggiunti forniscono circa 135 kcal. Il filo d'olio a fine cottura valorizza gli ingredienti freschi.
- **Grana Padano in scaglie:** i 20 g aggiunti apportano circa 80 kcal. Viene usato come chiusura sapida, mantenendo una quantità contenuta per non aumentare ulteriormente il sodio.

Note pratiche
La mozzarella di bufala va utilizzata in cottura dopo essere stata ben sgocciolata. A forno spento si possono aggiungere mortadella, stracciatella, pistacchi, scaglie di Grana Padano e un filo d'olio extravergine d'oliva.

In questo modo i salumi e i latticini freschi non vengono esposti direttamente al calore intenso e conservano meglio aroma, consistenza e qualità organolettiche.

Nota sui valori nutrizionali
I valori sono stime basate su tabelle CREA. Il dato più variabile è il sodio, che dipende molto dalla marca di mortadella e formaggi utilizzati.

Gemini

Per portare il valore energetico della ricetta a circa 2000 kcal, mantenendo invariate le quantità dell'impasto, quindi farina, acqua, olio, lievito e sale, e mantenendo invariata anche la mortadella a 50 g, è necessario intervenire in modo consistente sulla farcitura.
Il metodo più efficace per aumentare le calorie senza alterare la struttura dell'impasto o la tipologia di pizza è aggiungere ingredienti densi dal punto di vista nutrizionale, come grassi buoni e proteine, che si abbinano bene al connubio mortadella e bufala.

Strategia di modifica

- **Impasto e mortadella:** mantenuti invariati.
- **Mozzarella di bufala:** aumentata, per incrementare grassi e proteine.
- **Aggiunte:** introduzione di granella di pistacchio e ricotta vaccina, classici abbinamenti gourmet per la mortadella, utili ad aumentare drasticamente le calorie in modo coerente con la ricetta.

Ricetta modificata
Target 2000 kcal

Ingrediente	Quantità
Farina 0	250 g
Acqua	150 g
Olio EVO	10 g
Lievito di birra	0,5 g
Sale fino	6 g
Mozzarella di bufala	180 g
Mortadella	50 g
Ricotta vaccina	100 g
Granella di pistacchio	40 g

La mozzarella di bufala viene aumentata da 100 g a 180 g.

Analisi nutrizionale stimata

Valore	Quantità approssimativa
Energia	∼2010 kcal
Proteine	∼82 g
Lipidi	∼115 g
Carboidrati	∼182 g
Sodio	∼3400 mg

Note sulla modifica

- **Perché la ricotta:** la ricotta è una buona fonte proteica e aggiunge cremosità alla pizza. Può essere disposta a ciuffi dopo la cottura oppure inserita sulla base prima di infornare.

- **Il ruolo del pistacchio:** oltre a fornire una componente croccante coerente con una pizza mortadella e bufala, la frutta secca è molto densa dal punto di vista calorico grazie alla presenza di grassi monoinsaturi. Questo permette di raggiungere le 2000 kcal senza aumentare eccessivamente il volume complessivo della pizza.
- **Gestione del sodio:** poiché mortadella e bufala sono ingredienti naturalmente sapidi, l'aggiunta della ricotta, più povera di sodio, contribuisce a mantenere il gusto equilibrato nonostante l'aumento calorico complessivo.

Questa versione mantiene l'identità della pizza originale, ma la rende più ricca e calorica attraverso ingredienti coerenti con l'abbinamento mortadella, bufala e pistacchio.

Confronto delle risposte

In questo prompt, tutti i modelli adottano una strategia molto simile, poiché i vincoli imposti sono limitati: l'impasto e la quantità di mortadella devono rimanere invariati, mentre l'unico obiettivo è aumentare il valore energetico fino a circa 2000 kcal. Di conseguenza, nessun modello modifica la struttura della ricetta, intervenendo esclusivamente sulla farcitura, ma riuscendo a preservare l'identità della pizza originale.

Le principali differenze riguardano il grado di elaborazione della ricetta. *ChatGPT* propone una soluzione essenziale, limitandosi ad aumentare la mortadella e ad aggiungere pistacchi e olio. *DeepSeek* adotta un approccio ancora più minimale, incrementando quasi esclusivamente la mozzarella e l'olio, senza introdurre nuovi ingredienti. *Claude* e *Gemini*, invece, elaborano maggiormente la ricetta arricchendola di nuovi ingredienti.

3.6.3 Terzo prompt

Utente

PIZZA MORTADELLA E BUFALA - Dosi per 1 pizza (peso 566.5g)

- Farina 0 W300 250,0 g
- Acqua 150,0 g
- Lievito di birra fresco 0,5 g
- Sale fino 6,0 g
- Olio extravergine d'oliva 10,0 g
- Mozzarella di bufala DOP 100,0 g
- Mortadella 50,0 g

Valori nutrizionali:

- Energia (kcal): 1293,75
- Proteine (g): 50,76
- Lipidi (g): 46,69
- Carboidrati (g): 175,16
- Sodio (mg): 3096,08
- Fibre totali (g): 7,29

Partendo da questa ricetta, e cercando di mantenere l'idea della pizza, applica delle modifiche affinché la pizza sia di almeno 2500 calorie, mantenendo la mozzarella di bufala e la mortadella nelle quantità date e evitando di avere un contenuto di carboidrati inferiore di 100g e un limitato valore di lipidi. La pizza deve essere senza glutine.

Risposte al terzo prompt

ChatGPT

Pizza mortadella, bufala, patate e miele senza glutine
Dosi per 1 pizza molto grande / focaccia alta

Ingrediente	Quantità
Mix per pizza senza glutine	480 g
Acqua	410 g
Olio extravergine d'oliva	5 g
Lievito di birra fresco	2 g
Sale fino	4 g
Mozzarella di bufala	100 g
Mortadella senza glutine	50 g
Patate lesse o al forno	200 g
Miele	60 g

Il peso totale prima della cottura è pari a circa 1311 g.
La pizza diventa molto più grande rispetto alla ricetta originale, ma questo aumento è praticamente inevitabile: per arrivare a 2500 kcal senza alzare troppo i grassi, è necessario aumentare in modo consistente la componente glucidica.

Valori nutrizionali stimati

Valore	Ricetta originale	Ricetta modificata
Energia	1293,75 kcal	circa 2500–2520 kcal
Proteine	50,76 g	circa 58–62 g
Lipidi	46,69 g	circa 50–53 g
Carboidrati	175,16 g	circa 445–455 g
Sodio	3096,08 mg	circa 2400–2800 mg
Fibre totali	7,29 g	circa 18–20 g

I vincoli richiesti risultano quindi rispettati:

- almeno 2500 kcal;
- mozzarella di bufala mantenuta a 100 g;
- mortadella mantenuta a 50 g;
- carboidrati superiori a 100 g;
- ricetta senza glutine;
- lipidi abbastanza limitati rispetto alle calorie totali.

Infatti, pur arrivando a circa 2500 kcal, i grassi restano intorno a 50–53 g, evitando un aumento eccessivo come accadrebbe utilizzando burrata, crema di pistacchio o grandi quantità di olio.

Ricetta finale pronta da scrivere
Pizza mortadella, bufala, patate e miele senza glutine
Dosi per 1 pizza molto grande / focaccia alta

- Mix per pizza senza glutine: 480 g;
- Acqua: 410 g;
- Olio extravergine d'oliva: 5 g;
- Lievito di birra fresco: 2 g;
- Sale fino: 4 g;
- Mozzarella di bufala: 100 g;
- Mortadella senza glutine: 50 g;
- Patate lesse o al forno: 200 g;
- Miele: 60 g.

Valore	Stima finale
Energia	circa 2500–2520 kcal
Proteine	circa 58–62 g
Lipidi	circa 50–53 g
Carboidrati	circa 445–455 g
Sodio	circa 2400–2800 mg
Fibre totali	circa 18–20 g

Perché questa modifica ha senso
La pizza rimane fedele all'idea originale perché mortadella e bufala restano protagoniste e vengono mantenute nelle stesse quantità. Le patate rendono la pizza più sostanziosa senza aggiungere grassi rilevanti, mentre il miele è coerente con una pizza bianca in stile gourmet a base di mortadella e bufala.

Nota sul senza glutine
Per rendere la pizza realmente senza glutine, è necessario utilizzare:

- mix per pizza certificato senza glutine;
- mortadella con dicitura "senza glutine";
- piano di lavoro, teglia e utensili non contaminati da farine contenenti glutine.

DeepSeek

Per ottenere una pizza senza glutine da almeno 2500 kcal, mantenendo invariate le quantità di mozzarella di bufala (100 g) e mortadella (50 g), è stata modificata la ricetta aumentando principalmente la quantità di impasto e di olio extravergine d'oliva. I carboidrati risultano ampiamente superiori a 100 g, mentre i lipidi vengono volutamente incrementati per raggiungere il target energetico richiesto.

Modifiche apportate

- Sostituzione della farina di grano con un mix per pizza senza glutine aumentato a 400 g;
- aumento dell'acqua a 240 g per mantenere un'idratazione pari a circa il 60%;
- aumento dell'olio extravergine d'oliva a 70 g;
- incremento del lievito di birra fresco a 1 g;
- sale aumentato a 8 g (facoltativo, per motivi di gusto).

Ingredienti per 1 pizza senza glutine

Ingrediente	Quantità
Acqua	240,0 g
Farina senza glutine (mix per pizza con gomma xantana)	400,0 g
Olio extravergine d'oliva	70,0 g
Lievito di birra fresco	1,0 g
Sale fino	8,0 g
Mozzarella di bufala	100,0 g
Mortadella (senza glutine)	50,0 g

Peso totale approssimativo: circa 860 g.

Valori nutrizionali stimati

Parametro	Valore	Note
Energia	∼2500 kcal	Obiettivo raggiunto
Proteine	∼55 g	Derivano principalmente dalla farina, dalla bufala e dalla mortadella
Lipidi	∼102 g	Elevati soprattutto per l'olio EVO e la mozzarella
Carboidrati	∼300 g	Ben superiori a 100 g
Sodio	∼2000 mg	Valore stimato
Fibre	∼10–12 g	Variabili in funzione del mix utilizzato

I valori sono arrotondati e basati su tabelle nutrizionali medie: farina senza glutine ∼360 kcal/100 g, olio EVO 900 kcal/100 g, mozzarella di bufala ∼280 kcal/100 g e mortadella ∼300 kcal/100 g.

Istruzioni per la preparazione

1. Mescolare il mix di farine senza glutine con il sale.
2. Sciogliere il lievito nell'acqua tiepida e aggiungere l'olio extravergine d'oliva.
3. Unire i liquidi agli ingredienti secchi e impastare fino a ottenere un composto omogeneo. L'impasto senza glutine risulterà naturalmente più morbido e appiccicoso.
4. Coprire e lasciare lievitare in luogo tiepido per circa 2–3 ore; in alternativa è possibile effettuare una lievitazione più lunga in frigorifero.
5. Stendere l'impasto su carta forno aiutandosi con una piccola quantità di farina senza glutine.
6. Distribuire la mozzarella di bufala ben scolata e la mortadella a pezzi.
7. Cuocere in forno statico preriscaldato a 220–230 °C per circa 15–18 minuti, controllando la doratura.

La pizza mantiene l'identità della ricetta originale, con mortadella e bufala come ingredienti principali, risultando però completamente senza glutine e con un apporto energetico superiore a 2500 kcal, caratterizzato da un elevato contenuto di carboidrati e lipidi.

Claude

Per superare le 2500 kcal mantenendo vincolate le quantità di mozzarella di bufala e mortadella, con carboidrati $\geq$ 100 g e lipidi limitati, cioè non aumentati artificialmente con olio o burro, è necessario lavorare soprattutto sulla massa dell'impasto e su ingredienti densi ma equilibrati.

Di seguito viene proposta la riformulazione.

Pizza Mortadella e Bufala – versione $\geq$ 2500 kcal, gluten-free, lipidi contenuti
Dosi per 1 pizza – peso totale circa 1180 g

Ingrediente	Quantità	Note
Acqua	290,0 g	Proporzionata al nuovo mix
Mix farine senza glutine	400,0 g	Sostituisce la farina 0, con aumento del 60%
Farina di riso integrale	80,0 g	Extra carboidrati e fibre
Psillio in polvere	10,0 g	Struttura dell'impasto gluten-free
Olio extravergine d'oliva nell'impasto	10,0 g	Invariato
Lievito di birra fresco	1,0 g	Adeguato alla massa
Sale fino	8,0 g	Proporzionato
Mozzarella di bufala	100,0 g	Invariata
Mortadella	50,0 g	Invariata
Patate a fettine sottili	150,0 g	Base sotto la mozzarella
Granella di pistacchi	35,0 g	Abbinamento classico
Stracciatella di bufala	50,0 g	A crudo

Valori nutrizionali stimati

Parametro	Originale	Riformulata	Δ
Energia	1293,75 kcal	∼2530 kcal	+96%
Proteine	50,76 g	∼75 g	+48%
Lipidi	46,69 g	∼78 g	Contenuti rispetto al target energetico
Carboidrati	175,16 g	∼370 g	Ben sopra 100 g
Sodio	3096,08 mg	∼3450 mg	+11%
Fibre totali	7,29 g	∼22 g	+200%

Logica delle scelte
Il vincolo principale riguarda il contenimento dei lipidi: viene quindi scartato l'approccio basato sull'aumento eccessivo di olio e formaggi grassi. Le calorie derivano principalmente da carboidrati complessi e da una quota mirata di proteine e grassi di qualità.

- **Mix senza glutine a 400 g e 80 g di riso integrale:** rappresenta la leva principale della riformulazione. Questa componente apporta da sola circa 1700 kcal, prevalentemente da carboidrati, rispettando il vincolo sui lipidi e portando i carboidrati molto sopra la soglia minima richiesta.
- **Psillio 10 g:** con una quantità così elevata di farine senza glutine, lo psillio serve a dare elasticità e tenuta in cottura, evitando una base troppo fragile.
- **Patate a fettine 150 g:** richiamano la classica pizza con patate italiana. Aggiungono circa 115 kcal sotto forma di carboidrati, senza aumentare i lipidi, e funzionano da base sotto la mozzarella.
- **Pistacchi 35 g:** apportano circa 210 kcal e rappresentano un abbinamento classico con la mortadella. Costituiscono la principale fonte lipidica aggiunta, ma con grassi di buona qualità.
- **Stracciatella 50 g:** mantiene l'identità legata alla bufala senza appesantire eccessivamente la ricetta. Risulta meno densa lipidicamente rispetto a scelte come gorgonzola o mascarpone.

- **Olio nell'impasto invariato:** l'olio extravergine d'oliva resta pari a 10 g. Non viene aggiunto olio a crudo, scelta deliberata per tenere i lipidi sotto controllo.

Note pratiche
L'impasto presenta un'idratazione di circa il 72%. Per gestire la massa elevata di farine senza glutine, è consigliata una lievitazione lunga, indicativamente 12 ore a temperatura ambiente con 1 g di lievito.
La pizza va stesa su una teglia ampia, almeno 35×40 cm. Questa quantità d'impasto produce infatti una pizza spessa, simile a una pizza alla pala o a una teglia romana.

Sequenza di cottura
Cuocere la base con patate e mozzarella di bufala a 250 °C per circa 12–14 minuti. A forno spento aggiungere mortadella, stracciatella e granella di pistacchi.

Le circa 2530 kcal stimate sono coerenti con una pizza pensata come pasto unico molto abbondante o come preparazione da condividere. Come porzione singola standard, risulta notevolmente superiore al fabbisogno energetico di un pasto comune.

Gemini

Per ottenere una pizza senza glutine che superi le 2500 kcal, mantenendo invariate le quantità di mortadella, pari a 50 g, e mozzarella di bufala, pari a 100 g, è necessario intervenire sull'impasto e sull'aggiunta di ingredienti complementari ad alta densità energetica, come grassi buoni e proteine.
Poiché viene richiesto un valore di lipidi non limitato, quindi un apporto lipidico elevato, vengono integrati ingredienti come farina di mandorle e formaggi stagionati, capaci di aumentare notevolmente il profilo calorico e lipidico della pizza.

Ricetta modificata
Target >2500 kcal, gluten-free

Ingrediente	Quantità	Note nutrizionali
Farina di riso	200 g	Carboidrati di base
Farina di mandorle	120 g	Aumenta drasticamente lipidi e calorie
Fecola di patate	50 g	Aiuta la struttura dell'impasto senza glutine
Acqua	180 g	Liquido per l'impasto
Olio extravergine d'oliva	40 g	Aumenta l'apporto lipidico
Lievito di birra fresco	0,5 g	Invariato
Sale fino	6 g	Invariato
Mozzarella di bufala	100 g	Invariata
Mortadella	50 g	Invariata
Parmigiano Reggiano	60 g	Aggiunto per aumentare calorie e proteine
Granella di pistacchio	50 g	Aggiunta per lipidi e gusto

Analisi nutrizionale stimata

Parametro	Valore stimato
Energia	∼2550–2600 kcal
Carboidrati	∼185 g
Lipidi	∼160 g
Proteine	∼85 g
Sodio	∼3200 mg

I carboidrati risultano superiori a 100 g, rispettando quindi il vincolo richiesto. I lipidi, invece, sono volutamente elevati, in linea con l'obiettivo di ottenere una pizza ipercalorica.

Considerazioni tecniche

- **Struttura senza glutine:** la combinazione di farina di riso, fecola di patate e farina di mandorle permette di ottenere un impasto molto energetico. La farina di mandorle apporta grassi e una consistenza più ricca, adatta a una pizza ipercalorica.
- **Gestione dei lipidi:** l'uso generoso di olio extravergine d'oliva nell'impasto e l'aggiunta di Parmigiano Reggiano sulla superficie permettono di superare la soglia delle 2500 kcal senza aumentare le quantità di mortadella e bufala, mantenute fisse come richiesto.
- **Consiglio gourmet:** per valorizzare il gusto, si può spennellare il bordo della pizza con un mix di olio e una parte del Parmigiano prima della cottura. In questo modo si crea una crosticina croccante e saporita, capace di esaltare il contrasto con la morbidezza della bufala e della mortadella aggiunte a crudo dopo la cottura.
- **Lavorazione dell'impasto:** la gestione di un impasto senza glutine con un'elevata quantità di grassi, derivanti da mandorle e olio, può risultare delicata. È quindi preferibile stendere l'impasto con il dorso di un cucchiaio unto d'olio, invece che con il mattarello, per evitare che si rompa.

Questa versione mantiene l'idea originale della pizza mortadella e bufala, ma la trasforma in una variante senza glutine e fortemente ipercalorica, con un profilo lipidico volutamente elevato.

Confronto delle risposte

In questo prompt, i quattro modelli mostrano differenze abbastanza nette soprattutto nella gestione dei vincoli e nel livello di coerenza complessiva. In generale, tutte le risposte rispettano i vincoli principali della richiesta. Tuttavia, la parte più problematica riguarda il controllo dei lipidi, che non viene sempre interpretato in modo coerente: in alcuni casi viene rispettato solo parzialmente, in altri viene chiaramente superato.

Claude risulta fornire la risposta più affidabile perché mantiene meglio l'equilibrio tra i vincoli nutrizionali e la struttura della ricetta. Le modifiche introdotte sono più ragionate e la proposta rimane coerente sia dal punto di vista gastronomico sia da quello tecnico, con una descrizione anche più completa e ordinata.

DeepSeek e *Gemini* sono, invece, più sbilanciati: essi riescono a raggiungere l'obiettivo calorico, ma lo fanno aumentando di molto i lipidi. Gemini è anche il più creativo, ma si allontana maggiormente dall'idea di "pizza" tradizionale. *ChatGPT* si colloca in una posizione intermedia: la ricetta è coerente e comprensibile, ma meno rigorosa nella gestione dei vincoli rispetto a Claude.

3.7 Considerazioni finali sul case study

La lettura trasversale dei quattro scenari e dei relativi prompt permette di individuare pattern ricorrenti nel comportamento di ciascun modello, che vanno oltre i singoli giudizi caso per caso: la qualità delle risposte dipende dalla complessità dei vincoli, non dalla bontà del modello in assoluto. Quando i prompt prevedono un solo obiettivo chiaro, ad esempio di aumentare le calorie mantenendo impasto e mortadella invariati, le differenze tra i modelli si riducono sensibilmente e tutti producono risultati accettabili. È nella gestione di vincoli multipli, sovrapposti o tra loro contraddittori che emergono le differenze più significative.

Quando la richiesta conteneva vincoli incompatibili, come la pizza "vegetariana con prosciutto cotto", i modelli hanno adottato approcci radicalmente diversi: *ChatGPT* ha risolto il problema silenziosamente, adattando la ricetta senza segnalare l'incongruenza; *Claude* e *Gemini* hanno esplicitato la contraddizione e proposto un'interpretazione coerente; *DeepSeek*, in più di un caso, ha preferito non generare alcuna ricetta, richiedendo, invece, un chiarimento sulle priorità. Nessuno dei tre approcci è intrinsecamente sbagliato, ma riflettono filosofie di interazione molto diverse, con implicazioni pratiche rilevanti per l'utente.

La tendenza all'ottimizzazione numerica a scapito della coerenza culinaria è un limite condiviso, ma con intensità diversa. All'aumentare del numero di vincoli, infatti, tutti i modelli tendono a trattare la ricetta come un problema matematico: si raggiunge il target calorico, si rispetta la soglia di sodio, si eliminano i macronutrienti fuori limite. Il risultato è spesso una ricetta tecnicamente corretta ma gastronomicamente discutibile, con ingredienti come riso soffiato, ceci, lenticchie, crema di mandorle, miele sulla ventricina, non propriamente adatti al contesto di una pizza. Questo limite è particolarmente marcato in *DeepSeek*, che, in diversi scenari, produce soluzioni nutrizionalmente valide ma culinariamente poco plausibili.

La completezza e il dettaglio delle risposte non coincidono necessariamente con la loro accuratezza. *Claude* produce sistematicamente le risposte più estese, con logiche delle sostituzioni, note tecniche sulla cottura e avvertenze sulla contaminazione crociata. Tuttavia, in almeno un caso, nel secondo prompt dello Scenario 1, la stima calorica dichiarata appare difficilmente compatibile con gli ingredienti mantenuti invariati, il che suggerisce che lunghezza e struttura non siano garanzia di correttezza numerica. *Gemini* è il modello che propone soluzioni più originali, ma è anche quello che più spesso si allontana dall'identità del piatto di partenza.

In sintesi, questo case study evidenzia che i modelli generativi sono strumenti efficaci per la rielaborazione di ricette entro vincoli definiti, ma rivelano limiti strutturali quando i vincoli diventano numerosi, contraddittori o richiedono un equilibrio tra ragionamento numerico e giudizio qualitativo. La mancanza di un modello di valutazione del gusto e della coerenza gastronomica rimane il limite più trasversale e difficilmente superabile nel contesto attuale.

CAPITOLO 4

Un case study per confrontare diversi LLM nella generazione di codice

In questo capitolo viene presentato un case study finalizzato al confronto di diversi LLM nella generazione di codice. L'obiettivo è valutare la capacità dei modelli di produrre soluzioni corrette e aderenti ai requisiti richiesti, analizzandone sia la qualità del codice generato sia il comportamento in presenza di richieste con differenti livelli di complessità.

Nella prima parte sono illustrati l'obiettivo dello studio, la metodologia adottata e l'API utilizzata per ottenere le risposte. Successivamente, vengono analizzate le risposte fornite da ciascun modello, evidenziandone le eventuali imperfezioni e le principali differenze rispetto alle richieste formulate.

4.1 Obiettivo e criteri di valutazione

4.1.1 Obiettivi del case study

L'obiettivo di questo case study è valutare le capacità dei diversi modelli LLM selezionati nella generazione di codice, analizzandone il comportamento in risposta ad una serie di richieste formulate secondo criteri uniformi. Il confronto non si limita alla sola correttezza sintattica del codice, ma include anche la capacità dei modelli di interpretare correttamente i requisiti, proporre soluzioni coerenti e produrre implementazioni funzionali rispetto alle specifiche richieste.

Per garantire un confronto il più possibile oggettivo, tutti i modelli sono stati sottoposti al medesimo prompt iniziale. Le risposte ottenute sono state analizzate secondo criteri qualitativi e funzionali, evidenziando eventuali errori, omissioni o comportamenti peculiari emersi durante l'esecuzione dei test.

4.1.2 Criteri di valutazione

L'analisi delle risposte generate dai diversi modelli è stata condotta mediante una valutazione qualitativa, prendendo in esame gli aspetti ritenuti maggiormente significativi ai fini dello sviluppo dell'applicazione. In particolare, sono stati considerati i seguenti criteri:

- *Correttezza funzionale*: verifica della corretta esecuzione del codice prodotto e dell'assenza di errori che ne impediscano il funzionamento.
- *Aderenza ai requisiti*: valutazione della capacità del modello di rispettare le richieste del prompt.

- *Capacità di correzione*: analisi del comportamento del modello a seguito delle richieste di modifica o della segnalazione di errori, verificando se le problematiche individuate vengono effettivamente risolte.
- *Robustezza della soluzione*: analisi della gestione di casi particolari, input non validi e situazioni che potrebbero compromettere il corretto funzionamento del codice.

4.2 Metodologia sperimentale

Per confrontare in maniera omogenea le capacità dei diversi modelli, è stata definita una metodologia sperimentale comune basata sullo sviluppo progressivo della medesima applicazione. A tutti i modelli è stato fornito lo stesso prompt iniziale, nel quale veniva richiesta la realizzazione di un'interfaccia grafica in Python per il calcolo dei valori nutrizionali degli alimenti.

Una volta ottenuta la prima versione del codice, questa è stata eseguita in ambiente locale al fine di verificarne il corretto funzionamento. Gli eventuali errori riscontrati durante l'esecuzione, così come le funzionalità mancanti o i miglioramenti desiderati, sono stati riportati al modello attraverso prompt successivi. In questo modo è stato possibile osservare non solo la qualità della soluzione iniziale, ma anche la capacità del modello di correggere errori, integrare nuove funzionalità e mantenere la coerenza del progetto nel corso dello sviluppo.

Il prodotto finale desiderato consiste in un'interfaccia che permetta di inserire un'elenco di alimenti con la rispettiva quantità al fine di poter calcolare i valori nutrizionali complessivi tenendo conto di tutti gli alimenti considerati.

4.2.1 Prompt iniziale

Di seguito, viene riportato il prompt iniziale fornito a ciascun modello.

Utente

Crea un calcolatore in Python con un'interfaccia utente grafica utilizzando la libreria CustomTkinter. L'interfaccia deve essere divisa in due parti:

- Lato sinistro:
  - Casella di testo chiamata "Ingrediente" dove posso inserire un qualsiasi alimento
  - Casella di testo chiamata "Quantità" associata all'ingrediente inserito, quantità espressa in grammi
  - Possibilità di inserire un numero di alimenti fino ad un massimo di 15
  - Pulsante alla fine di tutti gli ingredienti chiamato "Calcola valori nutrizionali"
- Lato destro:
  - Tabella contenente il contenuto totale dei vari alimenti per i seguenti valori nutrizionali: Calorie, Proteine, Lipidi, Sodio, Fibre

CustomTkinter è una libreria Python per la creazione di interfacce utente personalizzate. Questa libreria consiste in un'estensione con grafica moderna della libreria Tkinter, la libreria standard di Python per la realizzazione di interfacce grafiche (GUI).

4.2.2 USDA FoodData Central

Dopo aver verificato la creazione di interfacce grafiche funzionali alle richieste, a ciascun modello è stato specificato di utilizzare un'API per l'estrazione dei valori nutrizionali di ciascuno degli alimenti selezionati.

L'API USDA FoodData Central è un'interfaccia web messa a disposizione dallo United States Department of Agriculture (USDA) che permette l'accesso tramite programma a un ampio database di dati nutrizionali sugli alimenti. L'API consente di effettuare ricerche per alimento e di ottenere informazioni dettagliate relative a macronutrienti, micronutrienti e composizione degli alimenti, risultando utile per applicazioni di analisi nutrizionale e supporto alla generazione di diete o ricette.

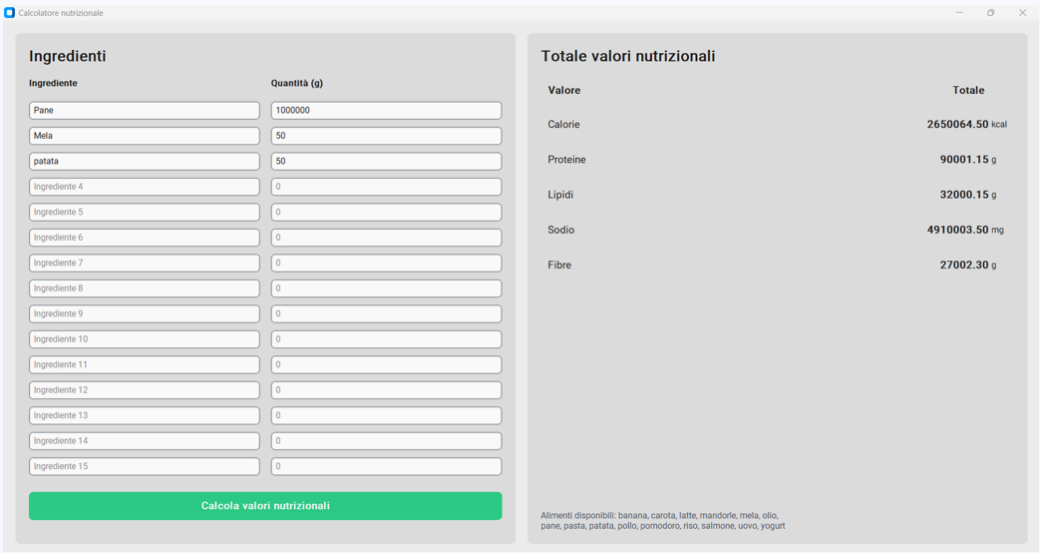
4.3 Analisi dei risultati

4.3.1 ChatGPT

Prompt iniziale

In seguito, viene mostrata l'interfaccia ottenuta dall'esecuzione del codice prodotto da *ChatGPT*.

ChatGPT



In risposta al prompt iniziale, ChatGPT genera un'applicazione pienamente funzionante. L'interfaccia rispetta la struttura richiesta, suddividendo la finestra in due sezioni: una dedicata all'inserimento degli ingredienti e una destinata alla visualizzazione dei valori nutrizionali complessivi. Per consentire il funzionamento dell'applicazione, in assenza di una sorgente dati esterna, il modello introduce un database locale contenente un insieme di alimenti con i rispettivi valori nutrizionali.

Dal punto di vista della robustezza, il codice implementa alcuni controlli sugli input inseriti dall'utente. Nel caso in cui venga specificato un alimento senza la relativa quantità, oppure una quantità senza il corrispondente alimento, l'applicazione interrompe il calcolo e visualizza un messaggio di errore che invita a completare entrambi i campi. In questa situazione i valori nutrizionali complessivi non vengono aggiornati, evitando la produzione di risultati incompleti o non coerenti.

Un ulteriore controllo riguarda la presenza dell'alimento nel database nutrizionale locale. Qualora l'utente inserisca un alimento non presente nel dizionario interno, oppure utilizzi una denominazione diversa da quella memorizzata (ad esempio il plurale anziché il singolare), l'applicazione visualizza un messaggio di avviso che segnala l'impossibilità di effettuare il calcolo.

Nel Listato 4.1 viene riportato il codice generato:

```
import customtkinter as ctk
from tkinter import messagebox

NUTRITION_PER_100G = {
    "mela": {"Calorie": 52, "Proteine": 0.3, "Lipidi": 0.2, "Sodio": 1, "Fibre": 2.4},
    "banana": {"Calorie": 89, "Proteine": 1.1, "Lipidi": 0.3, "Sodio": 1, "Fibre": 2.6},
    "riso": {"Calorie": 130, "Proteine": 2.7, "Lipidi": 0.3, "Sodio": 1, "Fibre": 0.4},
    "pasta": {"Calorie": 131, "Proteine": 5.0, "Lipidi": 1.1, "Sodio": 6, "Fibre": 1.8},
    "pollo": {"Calorie": 165, "Proteine": 31.0, "Lipidi": 3.6, "Sodio": 74, "Fibre": 0.0},
    "salmone": {"Calorie": 208, "Proteine": 20.0, "Lipidi": 13.0, "Sodio": 59, "Fibre":
        0.0},
    "uovo": {"Calorie": 155, "Proteine": 13.0, "Lipidi": 11.0, "Sodio": 124, "Fibre": 0.0},
    "latte": {"Calorie": 42, "Proteine": 3.4, "Lipidi": 1.0, "Sodio": 44, "Fibre": 0.0},
    "yogurt": {"Calorie": 61, "Proteine": 3.5, "Lipidi": 3.3, "Sodio": 46, "Fibre": 0.0},
    "pane": {"Calorie": 265, "Proteine": 9.0, "Lipidi": 3.2, "Sodio": 491, "Fibre": 2.7},
    "patata": {"Calorie": 77, "Proteine": 2.0, "Lipidi": 0.1, "Sodio": 6, "Fibre": 2.2},
    "pomodoro": {"Calorie": 18, "Proteine": 0.9, "Lipidi": 0.2, "Sodio": 5, "Fibre": 1.2},
    "carota": {"Calorie": 41, "Proteine": 0.9, "Lipidi": 0.2, "Sodio": 69, "Fibre": 2.8},
    "olio": {"Calorie": 884, "Proteine": 0.0, "Lipidi": 100.0, "Sodio": 2, "Fibre": 0.0},
    "mandorle": {"Calorie": 579, "Proteine": 21.2, "Lipidi": 49.9, "Sodio": 1, "Fibre":
        12.5},
}

NUTRIENTS = ("Calorie", "Proteine", "Lipidi", "Sodio", "Fibre")


class NutritionCalculator(ctk.CTk):
    def __init__(self):
        super().__init__()
        self.title("Calcolatore nutrizionale")
        self.geometry("980x620")
        self.minsize(840, 540)

        ctk.set_appearance_mode("light")
        ctk.set_default_color_theme("green")

        self.ingredient_entries = []
        self.quantity_entries = []
        self.result_vars = {nutrient: ctk.StringVar(value="0.00") for nutrient in NUTRIENTS
            }

        self._build_layout()

    def _build_layout(self):
        self.grid_columnconfigure(0, weight=1, uniform="main")
        self.grid_columnconfigure(1, weight=1, uniform="main")
        self.grid_rowconfigure(0, weight=1)

        input_frame = ctk.CTkFrame(self, corner_radius=8)
        input_frame.grid(row=0, column=0, padx=(18, 9), pady=18, sticky="nsew")
        input_frame.grid_columnconfigure((0, 1), weight=1)

        results_frame = ctk.CTkFrame(self, corner_radius=8)
        results_frame.grid(row=0, column=1, padx=(9, 18), pady=18, sticky="nsew")
        results_frame.grid_columnconfigure(0, weight=1)
        results_frame.grid_rowconfigure(1, weight=1)

        ctk.CTkLabel(
            results_frame,
            text="Totale valori nutrizionali",
            font=ctk.CTkFont(size=22, weight="bold"),
        ).grid(row=0, column=0, padx=20, pady=(20, 12), sticky="w")

```

```
61          self._build_input_area(input_frame)
62          self._build_results_table(results_frame)
63
64      def _build_results_table(self, parent):
65          table_frame = ctk.CTkFrame(parent, fg_color="transparent")
66          table_frame.grid(row=1, column=0, padx=20, pady=(0, 20), sticky="nsew")
67          table_frame.grid_columnconfigure(0, weight=1)
68          table_frame.grid_columnconfigure(1, weight=1)
69
70          header_style = ctk.CTkFont(size=15, weight="bold")
71          ctk.CTkLabel(table_frame, text="Valore", font=header_style).grid(
72              row=0, column=0, padx=10, pady=10, sticky="w"
73          )
74          ctk.CTkLabel(table_frame, text="Totale", font=header_style).grid(
75              row=0, column=1, padx=10, pady=10, sticky="e"
76          )
77
78          for row, nutrient in enumerate(NUTRIENTS, start=1):
79              unit = "kcal" if nutrient == "Calorie" else "mg" if nutrient == "Sodio" else "g
                    "
80
81              ctk.CTkLabel(table_frame, text=nutrient, font=ctk.CTkFont(size=15)).grid(
82                  row=row, column=0, padx=10, pady=12, sticky="w"
83              )
84              ctk.CTkLabel(
85                  table_frame,
86                  textvariable=self.result_vars[nutrient],
87                  font=ctk.CTkFont(size=15, weight="bold"),
88              ).grid(row=row, column=1, padx=(10, 4), pady=12, sticky="e")
89              ctk.CTkLabel(table_frame, text=unit, font=ctk.CTkFont(size=13)).grid(
90                  row=row, column=2, padx=(0, 10), pady=12, sticky="w"
91              )
92
93          ctk.CTkLabel(
94              parent,
95              text="Alimenti disponibili: " + ", ".join(sorted(NUTRITION_PER_100G)),
96              wraplength=380,
97              justify="left",
98              text_color="#4b5563",
99          ).grid(row=2, column=0, padx=20, pady=(0, 20), sticky="w")
100
101     def _build_input_area(self, parent):
102         ctk.CTkLabel(
103             parent,
104             text="Ingredienti",
105             font=ctk.CTkFont(size=22, weight="bold"),
106         ).grid(row=0, column=0, columnspan=2, padx=20, pady=(20, 12), sticky="w")
107
108         ctk.CTkLabel(parent, text="Ingrediente", font=ctk.CTkFont(weight="bold")).grid(
109             row=1, column=0, padx=(20, 8), pady=(0, 8), sticky="w"
110         )
111         ctk.CTkLabel(parent, text="Quantita' (g)", font=ctk.CTkFont(weight="bold")).grid(
112             row=1, column=1, padx=(8, 20), pady=(0, 8), sticky="w"
113         )
114
115         for index in range(15):
116             ingredient_entry = ctk.CTkEntry(parent, placeholder_text=f"Ingrediente {index +
                    1}")
117             ingredient_entry.grid(row=index + 2, column=0, padx=(20, 8), pady=5, sticky="ew
                    ")
118
119             quantity_entry = ctk.CTkEntry(parent, placeholder_text="0")
120             quantity_entry.grid(row=index + 2, column=1, padx=(8, 20), pady=5, sticky="ew")
121
122             self.ingredient_entries.append(ingredient_entry)
123             self.quantity_entries.append(quantity_entry)
124
125         ctk.CTkButton(
126             parent,
127             text="Calcola valori nutrizionali",
128             command=self.calculate_totals,
129             height=42,
```

```
            font=ctk.CTkFont(size=15, weight="bold"),
        ).grid(row=17, column=0, columnspan=2, padx=20, pady=(18, 20), sticky="ew")

    def calculate_totals(self):
        totals = {nutrient: 0.0 for nutrient in NUTRIENTS}
        unknown_ingredients = []

        for ingredient_entry, quantity_entry in zip(self.ingredient_entries, self.
            quantity_entries):
            ingredient = ingredient_entry.get().strip().lower()
            quantity_text = quantity_entry.get().strip().replace(",", ".")

            if not ingredient and not quantity_text:
                continue

            if not ingredient or not quantity_text:
                messagebox.showerror("Dati incompleti", "Compila sia Ingrediente sia
                    Quantita'.")
                return

            try:
                quantity = float(quantity_text)
            except ValueError:
                messagebox.showerror("Quantita' non valida", f"La quantita' di '{ingredient
                    }' non e' un numero.")
                return

            if quantity < 0:
                messagebox.showerror("Quantita' non valida", "Inserisci quantita' maggiori
                    o uguali a zero.")
                return

            nutrition = NUTRITION_PER_100G.get(ingredient)
            if nutrition is None:
                unknown_ingredients.append(ingredient)
                continue

            multiplier = quantity / 100
            for nutrient in NUTRIENTS:
                totals[nutrient] += nutrition[nutrient] * multiplier

        for nutrient, total in totals.items():
            self.result_vars[nutrient].set(f"{total:.2f}")

        if unknown_ingredients:
            messagebox.showwarning(
                "Ingredienti non trovati",
                "Non ho trovato questi alimenti nel dizionario locale: "
                + ", ".join(sorted(set(unknown_ingredients)))
                + ". Puoi aggiungerli in NUTRITION_PER_100G.",
            )

if __name__ == "__main__":
    app = NutritionCalculator()
    app.mainloop()
```

Listing 4.1: Implementazione di un calcolatore nutrizionale

Spiegazione generica del codice

Nel codice è gestita sia la parte relativa alla creazione di un database locale, sia la parte dedicata all'interfaccia, sia la logica dell'applicazione.

- `NUTRITION_PER_100G` è un dizionario Python nel quale ogni chiave rappresenta un alimento. Per ogni alimento vengono memorizzati i valori nutrizionali riferiti a 100 g.

- La classe `NutritionCalculator` identifica un approccio che segue la programmazione orientata agli oggetti, permettendo di mantenere separati dati, interfaccia grafica e logica di calcolo.
- Il metodo `_build_layout()` suddivide la finestra in due colonne utilizzando il *Grid Layout Manager*.
- Il metodo `_build_results_table()` costruisce la tabella dei valori nutrizionali. Per ogni nutriente, viene creata una riga composta da: nome del nutriente, valore totale, unità di misura.
- Il metodo `_build_input_area()` crea le intestazioni delle colonne, le coppie di campi di testo e il pulsante di calcolo dei valori nutrizionali complessivi.
- Il metodo `calculate_totals()` rappresenta il cuore dell'applicazione. Esso legge ciascuna coppia ingrediente-quantità, verifica la validità dei dati inseriti nelle caselle di testo, ricerca l'alimento dentro il database locale e, infine, calcola i valori dei nutrienti in relazione alla quantità inserita. Alla fine, aggiorna i valori visualizzati nella tabella.

Nella prima implementazione, ChatGPT definisce all'interno del codice un dizionario Python contenente i valori nutrizionali di un insieme limitato di alimenti. Tali valori sono codificati direttamente nel programma e non sono associati ad alcuna fonte dati esplicitamente indicata dal modello. Sebbene i valori risultino plausibili e coerenti con quelli comunemente riportati nelle principali banche dati nutrizionali, l'assenza di un collegamento a una sorgente ufficiale limita la tracciabilità e l'aggiornabilità delle informazioni utilizzate.

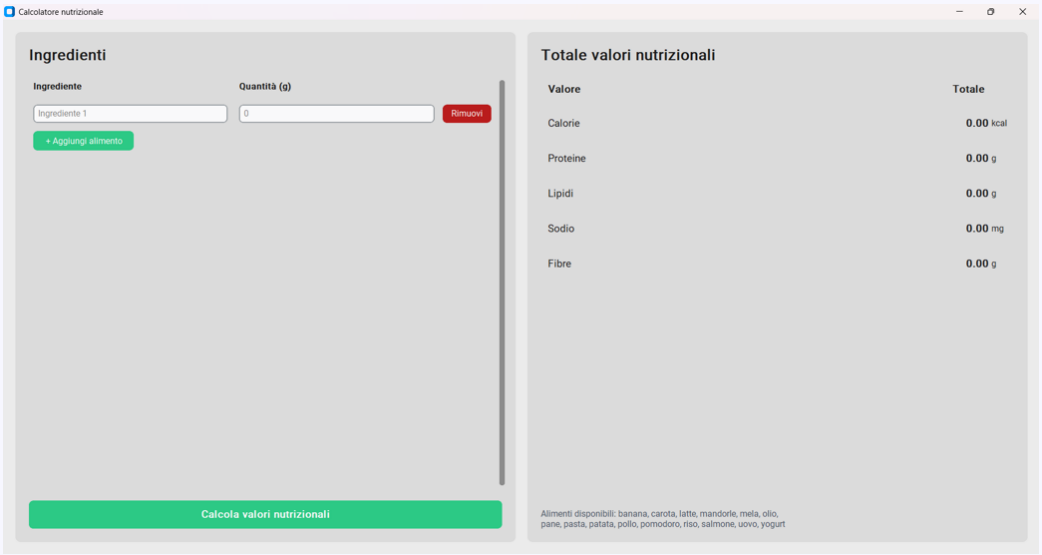
Secondo prompt

Utente

Non voglio avere nello schermo già tutte e 15 le righe, ogni riga nuova deve apparirmi dopo aver cliccato un pulsante "+ aggiungi alimento" così da avere più pulizia visiva. Aggiungimi accanto ad ogni riga per l'inserimento di un alimento la possibilità di rimuovere tale riga. Gli arrotondamenti devono essere effettuati per eccesso.

A seguito delle richieste effettuate, il modello modifica l'approccio iniziale eliminando la visualizzazione simultanea delle quindici righe di inserimento. Viene introdotto un sistema dinamico che permette all'utente di aggiungere o rimuovere progressivamente nuovi ingredienti mediante appositi pulsanti. Inoltre, viene introdotta una verifica per la corretta approssimatizzazione dei valori ottenuti.

ChatGPT



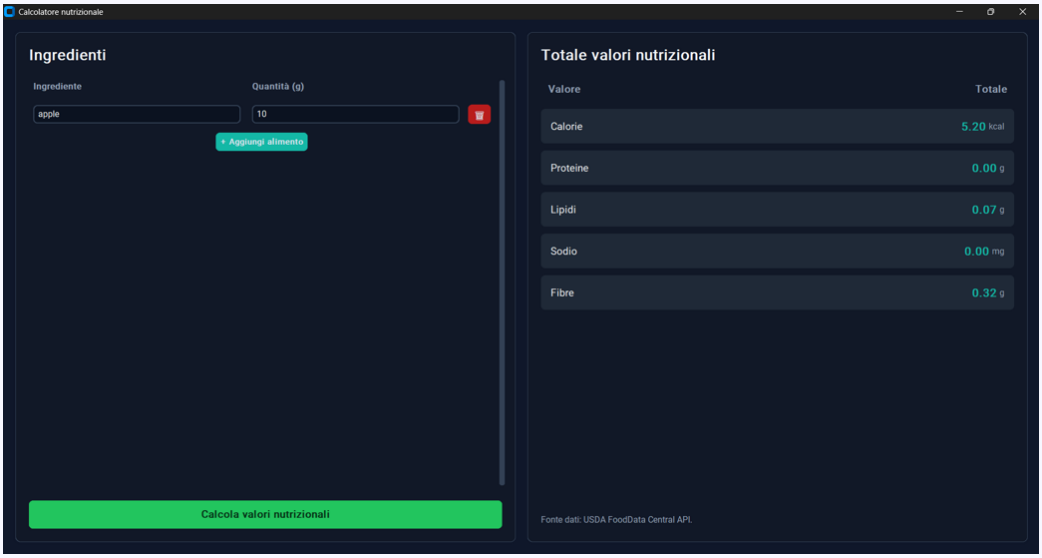
Terzo prompt

Utente

Implementa il collegamento alla *USDA FoodData Central API* per l'estrazione dei valori nutrizionali degli alimenti inseriti rimuovendo quindi il database interno che avevi creato all'inizio. La chiave API è Hola3CZ5B8PwOBJx02Lay9b6ki9E3CZu4cGHbNxH, tutti i valori nutrizionali devono essere estratti solo e solamente dalla *USDA FoodData Central API*.

Il cambiamento più significativo riguarda la sostituzione del database locale con l'API USDA FoodData Central. L'applicazione passa, quindi, da una struttura statica ad una soluzione basata su interrogazioni in tempo reale, consentendo il recupero dei valori nutrizionali direttamente dal database USDA.

ChatGPT



L'integrazione risulta corretta dal punto di vista funzionale; tuttavia, durante i test emergono alcune ambiguità nella ricerca degli alimenti. Termini generici, come, ad esempio, "mushroom", possono infatti restituire numerose varianti dello stesso alimento, rendendo necessarie ulteriori modifiche al sistema di ricerca.

Nel Listato 4.2 viene riportato il codice modificato:

```
1
2  import math
3  import json
4  import urllib.error
5  import urllib.parse
6  import urllib.request
7  import customtkinter as ctk
8  from tkinter import messagebox
9
10 NUTRIENTS = ("Calorie", "Proteine", "Lipidi", "Sodio", "Fibre")
11 MAX_INGREDIENTS = 15
12 USDA_API_KEY = "Hola3CZ5B8PwOBJx02Lay9b6ki9E3CZu4cGHbNxH"
13 USDA_SEARCH_URL = "https://api.nal.usda.gov/fdc/v1/foods/search"
14 USDA_NUTRIENTS = {
15     "Calorie": "1008",
16     "Proteine": "1003",
17     "Lipidi": "1004",
18     "Sodio": "1093",
19     "Fibre": "1079",
20 }
21
22 COLORS = {
23     "background": "#0f172a",
24     "panel": "#111827",
25     "panel_light": "#1f2937",
26     "input": "#0b1220",
27     "border": "#334155",
28     "text": "#f8fafc",
29     "muted": "#94a3b8",
30     "accent": "#14b8a6",
31     "accent_hover": "#0f766e",
32     "primary": "#22c55e",
33     "primary_hover": "#16a34a",
34     "danger": "#ef4444",
35     "danger_hover": "#dc2626",
36 }
37
38
39 class NutritionCalculator(ctk.CTk):
40     def __init__(self):
41         super().__init__()
42         self.title("Calcolatore nutrizionale")
43         self.geometry("980x620")
44         self.minsize(840, 540)
45
46         ctk.set_appearance_mode("dark")
47         ctk.set_default_color_theme("dark-blue")
48         self.configure(fg_color=COLORS["background"])
49
50         self.ingredient_entries = []
51         self.quantity_entries = []
52         self.remove_buttons = []
53         self.result_vars = {nutrient: ctk.StringVar(value="0.00") for nutrient in NUTRIENTS
               }
54         self.status_var = ctk.StringVar(value="Dati nutrizionali estratti solo da USDA
               FoodData Central API.")
55         self.usda_cache = {}
56
57         self._build_layout()
58
59     def _build_layout(self):
60         self.grid_columnconfigure(0, weight=1, uniform="main")
61         self.grid_columnconfigure(1, weight=1, uniform="main")
62         self.grid_rowconfigure(0, weight=1)
63
```

```
64          input_frame = ctk.CTkFrame(
65              self,
66              corner_radius=8,
67              fg_color=COLORS["panel"],
68              border_color=COLORS["border"],
69              border_width=1,
70          )
71          input_frame.grid(row=0, column=0, padx=(18, 9), pady=18, sticky="nsew")
72          input_frame.grid_columnconfigure((0, 1), weight=1)
73          input_frame.grid_rowconfigure(2, weight=1)
74
75          results_frame = ctk.CTkFrame(
76              self,
77              corner_radius=8,
78              fg_color=COLORS["panel"],
79              border_color=COLORS["border"],
80              border_width=1,
81          )
82          results_frame.grid(row=0, column=1, padx=(9, 18), pady=18, sticky="nsew")
83          results_frame.grid_columnconfigure(0, weight=1)
84          results_frame.grid_rowconfigure(1, weight=1)
85
86          ctk.CTkLabel(
87              results_frame,
88              text="Totale valori nutrizionali",
89              font=ctk.CTkFont(size=22, weight="bold"),
90              text_color=COLORS["text"],
91          ).grid(row=0, column=0, padx=20, pady=(20, 12), sticky="w")
92
93          self._build_input_area(input_frame)
94          self._build_results_table(results_frame)
95
96      def _build_results_table(self, parent):
97          table_frame = ctk.CTkFrame(parent, fg_color="transparent")
98          table_frame.grid(row=1, column=0, padx=20, pady=(0, 20), sticky="nsew")
99          table_frame.grid_columnconfigure(0, weight=1)
100         table_frame.grid_columnconfigure(1, weight=1)
101
102         header_style = ctk.CTkFont(size=15, weight="bold")
103         ctk.CTkLabel(
104             table_frame,
105             text="Valore",
106             font=header_style,
107             text_color=COLORS["muted"],
108         ).grid(
109             row=0, column=0, padx=10, pady=10, sticky="w"
110         )
111         ctk.CTkLabel(
112             table_frame,
113             text="Totale",
114             font=header_style,
115             text_color=COLORS["muted"],
116         ).grid(row=0, column=1, columnspan=2, padx=10, pady=10, sticky="e"
117         )
118
119         for row, nutrient in enumerate(NUTRIENTS, start=1):
120             unit = "kcal" if nutrient == "Calorie" else "mg" if nutrient == "Sodio" else "g
                    "
121
122             row_frame = ctk.CTkFrame(table_frame, fg_color=COLORS["panel_light"],
                        corner_radius=6)
123             row_frame.grid(row=row, column=0, columnspan=3, padx=0, pady=5, sticky="ew")
124             row_frame.grid_columnconfigure(0, weight=1)
125
126             ctk.CTkLabel(
127                 row_frame,
128                 text=nutrient,
129                 font=ctk.CTkFont(size=15),
130                 text_color=COLORS["text"],
131             ).grid(
132                 row=0, column=0, padx=14, pady=12, sticky="w"
133             )
```

```
134             ctk.CTkLabel(
135                 row_frame,
136                 textvariable=self.result_vars[nutrient],
137                 font=ctk.CTkFont(size=17, weight="bold"),
138                 text_color=COLORS["accent"],
139             ).grid(row=0, column=1, padx=(10, 4), pady=12, sticky="e")
140             ctk.CTkLabel(
141                 row_frame,
142                 text=unit,
143                 font=ctk.CTkFont(size=13),
144                 text_color=COLORS["muted"],
145             ).grid(
146                 row=0, column=2, padx=(0, 14), pady=12, sticky="w"
147             )
148 
149         ctk.CTkLabel(
150             parent,
151             textvariable=self.status_var,
152             wraplength=380,
153             justify="left",
154             text_color=COLORS["muted"],
155         ).grid(row=2, column=0, padx=20, pady=(0, 20), sticky="w")
156 
157     def _build_input_area(self, parent):
158         ctk.CTkLabel(
159             parent,
160             text="Ingredienti",
161             font=ctk.CTkFont(size=22, weight="bold"),
162             text_color=COLORS["text"],
163         ).grid(row=0, column=0, columnspan=2, padx=20, pady=(20, 12), sticky="w")
164 
165         self.rows_frame = ctk.CTkScrollableFrame(parent, fg_color="transparent",
                scrollbar_button_color=COLORS["border"])
166         self.rows_frame.grid(row=2, column=0, columnspan=2, padx=12, pady=(0, 10), sticky="
                nsew")
167         self.rows_frame.grid_columnconfigure(0, weight=1)
168         self.rows_frame.grid_columnconfigure(1, weight=1)
169         self.rows_frame.grid_columnconfigure(2, weight=0)
170 
171         ctk.CTkLabel(
172             self.rows_frame,
173             text="Ingrediente",
174             font=ctk.CTkFont(weight="bold"),
175             text_color=COLORS["muted"],
176         ).grid(
177             row=0, column=0, padx=(8, 8), pady=(0, 8), sticky="w"
178         )
179         ctk.CTkLabel(
180             self.rows_frame,
181             text="Quantita' (g)",
182             font=ctk.CTkFont(weight="bold"),
183             text_color=COLORS["muted"],
184         ).grid(
185             row=0, column=1, padx=(8, 8), pady=(0, 8), sticky="w"
186         )
187 
188         self.add_button = ctk.CTkButton(
189             self.rows_frame,
190             text="+ Aggiungi alimento",
191             command=self.add_ingredient_row,
192             width=132,
193             height=28,
194             corner_radius=6,
195             fg_color=COLORS["accent"],
196             hover_color=COLORS["accent_hover"],
197             font=ctk.CTkFont(size=12, weight="bold"),
198         )
199 
200         ctk.CTkButton(
201             parent,
202             text="Calcola valori nutrizionali",
203             command=self.calculate_totals,
```

```
            height=42,
            corner_radius=6,
            fg_color=COLORS["primary"],
            hover_color=COLORS["primary_hover"],
            text_color="#052e16",
            font=ctk.CTkFont(size=15, weight="bold"),
        ).grid(row=3, column=0, columnspan=2, padx=20, pady=(0, 20), sticky="ew")

        self.add_ingredient_row()

    def add_ingredient_row(self):
        if len(self.ingredient_entries) >= MAX_INGREDIENTS:
            return

        row = len(self.ingredient_entries)
        ingredient_entry = ctk.CTkEntry(
            self.rows_frame,
            placeholder_text=f"Ingrediente {row}",
            fg_color=COLORS["input"],
            border_color=COLORS["border"],
            text_color=COLORS["text"],
            placeholder_text_color=COLORS["muted"],
        )

        quantity_entry = ctk.CTkEntry(
            self.rows_frame,
            placeholder_text="0",
            fg_color=COLORS["input"],
            border_color=COLORS["border"],
            text_color=COLORS["text"],
            placeholder_text_color=COLORS["muted"],
        )

        remove_button = ctk.CTkButton(
            self.rows_frame,
            text="\U0001F5D1",
            width=34,
            height=30,
            corner_radius=6,
            fg_color="#b91c1c",
            hover_color="#991b1b",
            font=ctk.CTkFont(size=15),
            command=lambda entry=ingredient_entry: self.remove_ingredient_row(entry),
        )

        self.ingredient_entries.append(ingredient_entry)
        self.quantity_entries.append(quantity_entry)
        self.remove_buttons.append(remove_button)

        self._refresh_ingredient_rows()

    def remove_ingredient_row(self, ingredient_entry):
        if ingredient_entry not in self.ingredient_entries:
            return

        index = self.ingredient_entries.index(ingredient_entry)
        widgets = (
            self.ingredient_entries.pop(index),
            self.quantity_entries.pop(index),
            self.remove_buttons.pop(index),
        )

        for widget in widgets:
            widget.destroy()

        if not self.ingredient_entries:
            self.add_ingredient_row()
            return

        self._refresh_ingredient_rows()

    def _refresh_ingredient_rows(self):
```

```
276         self.add_button.grid_forget()
277
278         for index, (ingredient_entry, quantity_entry, remove_button) in enumerate(
279             zip(self.ingredient_entries, self.quantity_entries, self.remove_buttons),
280             start=1,
281         ):
282             ingredient_entry.configure(placeholder_text=f"Ingrediente {index}")
283             ingredient_entry.grid(row=index, column=0, padx=(8, 8), pady=5, sticky="ew")
284             quantity_entry.grid(row=index, column=1, padx=(8, 8), pady=5, sticky="ew")
285             remove_button.grid(row=index, column=2, padx=(4, 8), pady=5, sticky="e")
286
287         add_row = len(self.ingredient_entries) + 1
288         self.add_button.grid(row=add_row, column=0, columnspan=3, padx=(8, 8), pady=(6, 4))
289
290         if len(self.ingredient_entries) == MAX_INGREDIENTS:
291             self.add_button.configure(state="disabled", text="Limite 15")
292         else:
293             self.add_button.configure(state="normal", text="+ Aggiungi alimento")
294
295     def calculate_totals(self):
296         totals = {nutrient: 0.0 for nutrient in NUTRIENTS}
297         unknown_ingredients = []
298         api_errors = []
299
300         self.status_var.set("Calcolo in corso: interrogazione USDA FoodData Central API..."
                )
301         self.update_idletasks()
302
303         for ingredient_entry, quantity_entry in zip(self.ingredient_entries, self.
                quantity_entries):
304             ingredient = ingredient_entry.get().strip().lower()
305             quantity_text = quantity_entry.get().strip().replace(",", ".")
306
307             if not ingredient and not quantity_text:
308                 continue
309
310             if not ingredient or not quantity_text:
311                 messagebox.showerror("Dati incompleti", "Compila sia Ingrediente sia
                        Quantita'.")
312                 self.status_var.set("Calcolo interrotto: dati incompleti.")
313                 return
314
315             try:
316                 quantity = float(quantity_text)
317             except ValueError:
318                 messagebox.showerror("Quantita' non valida", f"La quantita' di '{ingredient
                        }' non e' un numero.")
319                 self.status_var.set("Calcolo interrotto: quantita non numerica.")
320                 return
321
322             if quantity < 0:
323                 messagebox.showerror("Quantita' non valida", "Inserisci quantita' maggiori
                        o uguali a zero.")
324                 self.status_var.set("Calcolo interrotto: quantita non valida.")
325                 return
326
327             try:
328                 nutrition = self.fetch_usda_nutrition(ingredient)
329             except RuntimeError as error:
330                 api_errors.append(str(error))
331                 continue
332
333             if not nutrition:
334                 unknown_ingredients.append(ingredient)
335                 continue
336
337             multiplier = quantity / 100
338             for nutrient in NUTRIENTS:
339                 totals[nutrient] += nutrition.get(nutrient, 0.0) * multiplier
340
341         for nutrient, total in totals.items():
342             self.result_vars[nutrient].set(f"{self._round_up(total):.2f}")
```

```
343
344         self.status_var.set("Fonte dati: USDA FoodData Central API.")
345
346         if api_errors:
347             messagebox.showerror("Errore USDA FoodData Central", "\n".join(api_errors[:3]))
348             self.status_var.set("Errore durante la richiesta a USDA FoodData Central API.")
349             return
350
351         if unknown_ingredients:
352             messagebox.showwarning(
353                 "Ingredienti non trovati",
354                 "Non ho trovato questi alimenti tramite USDA FoodData Central API: "
355                 + ", ".join(sorted(set(unknown_ingredients)))
356             )
357
358     def fetch_usda_nutrition(self, ingredient):
359         if ingredient in self.usda_cache:
360             return self.usda_cache[ingredient]
361
362         query = urllib.parse.urlencode({"api_key": USDA_API_KEY})
363         payload = json.dumps(
364             {
365                 "query": ingredient,
366                 "pageSize": 1,
367                 "requireAllWords": False,
368             }
369         ).encode("utf-8")
370         request = urllib.request.Request(
371             f"{USDA_SEARCH_URL}?{query}",
372             data=payload,
373             headers={"Content-Type": "application/json"},
374             method="POST",
375         )
376
377         try:
378             with urllib.request.urlopen(request, timeout=15) as response:
379                 data = json.loads(response.read().decode("utf-8"))
380         except urllib.error.HTTPError as error:
381             raise RuntimeError(f"USDA API ha risposto con errore HTTP {error.code}.") from
                    error
382         except urllib.error.URLError as error:
383             raise RuntimeError(f"Impossibile collegarsi alla USDA API: {error.reason}.")
                    from error
384         except TimeoutError as error:
385             raise RuntimeError("La richiesta alla USDA API ha superato il tempo massimo.")
                    from error
386
387         foods = data.get("foods", [])
388         if not foods:
389             self.usda_cache[ingredient] = None
390             return None
391
392         nutrition = self.extract_nutrients_from_usda_food(foods[0])
393         self.usda_cache[ingredient] = nutrition
394         return nutrition
395
396     @staticmethod
397     def extract_nutrients_from_usda_food(food):
398         nutrition = {nutrient: 0.0 for nutrient in NUTRIENTS}
399
400         for food_nutrient in food.get("foodNutrients", []):
401             nutrient_id = str(food_nutrient.get("nutrientId", "")).strip()
402             nutrient_value = food_nutrient.get("value")
403
404             for nutrient_name, usda_id in USDA_NUTRIENTS.items():
405                 if nutrient_id == usda_id and nutrient_value is not None:
406                     nutrition[nutrient_name] = float(nutrient_value)
407
408         return nutrition
409
410     @staticmethod
411     def _round_up(value):
```

```
412         return math.ceil((value - 1e-9) * 100) / 100
413
414
415 if __name__ == "__main__":
416     app = NutritionCalculator()
417     app.mainloop()
```

Listing 4.2: Implementazione di un calcolatore nutrizionale

Spiegazione generica del codice

Nel codice viene introdotto il dizionario `COLORS` che definisce tutte le tonalità utilizzate dall'applicazione come sfondo, campi di input, colori del testo, pulsanti e messaggi di stato.

Attraverso il metodo `fetch_usda_nutrition()` viene interrogato direttamente il database USDA per il recupero dei dati nutrizionali degli alimenti inseriti per poi, successivamente, effettuare il calcolo proporzionale rispetto ai grammi inseriti. Il metodo costruisce e invia una richiesta HTTP al server USDA mediante il metodo *POST*. Successivamente, viene ricevuta una risposta JSON, vengono estratti i nutrienti e gli stessi vengono salvati nella cache locale `usda_cache`.

L'introduzione di questa cache ha la funzionalità di ridurre il numero di richieste HTTP e il tempo di risposta, poiché se un alimento salvato al suo interno dovesse essere richiesto, il programma evita una nuova connessione e recupera immediatamente il risultato dalla cache.

Una volta ricevuta la risposta dal server, attraverso il metodo `extract_nutrients_from_usda_food()` viene analizzato il contenuto del campo `foodNutrients`. Per ogni nutriente restituito viene confrontato il relativo identificativo numerico con quelli definiti nel dizionario `USDA_NUTRIENTS`. Quando viene trovata una corrispondenza, il valore viene copiato nel dizionario interno utilizzato dall'applicazione. In questo modo il formato dei dati provenienti dall'API viene convertito nel formato richiesto dal programma.

Quarto prompt

Utente

Aggiungi un sistema di autocomplete per l'inserimento degli alimenti: mentre l'utente digita nel campo alimento, dopo almeno 2 o 3 caratteri, l'app deve interrogare la *USDA FoodData Central API* usando l'endpoint foods/search.
L'utente deve poter selezionare solo uno degli alimenti suggeriti dalla USDA API. Ordina i suggerimenti secondo il seguente criterio di priorità:

1. Foundation
2. SR Legacy
3. Survey (FNDDS)
4. Branded

Per ciascun gruppo privilegia gli alimenti semplici e generici. Ad esempio, quando l'utente ricerca `"chicken breast"`, i primi risultati mostrati dovranno essere:

- Chicken breast, raw
- Chicken breast, roasted
- Chicken breast, grilled

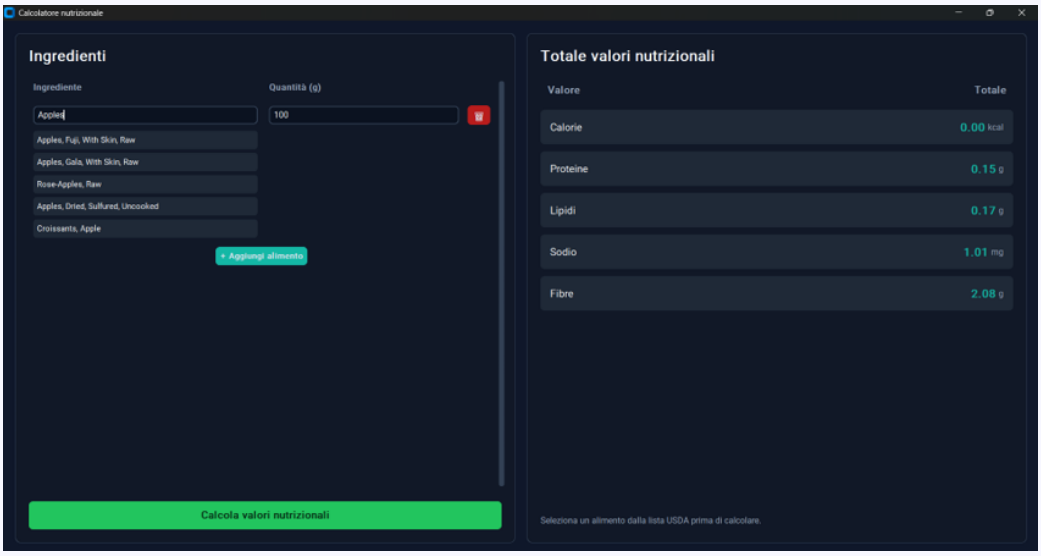
Successivamente potranno essere mostrati:

- Rotisserie
- Breaded

- Commercial products

Mantieni comunque il valore originale di `fdcId` per il calcolo dei valori nutrizionali. Non mostrare all'utente la sorgente da cui provengono i dati (ad esempio *Foundation*, *SR Legacy*, *Survey* o *Branded*), ma esclusivamente la descrizione dell'alimento.
Se l'utente non seleziona un alimento valido dalla lista, mostra un errore.

ChatGPT



Su richiesta dell'utente, ChatGPT implementa un sistema di autocompletamento basato sull'endpoint foods/search dell'API USDA. I suggerimenti vengono successivamente raffinati privilegiando gli alimenti generici (Foundation, SR Legacy e Survey) rispetto ai prodotti commerciali, migliorando la qualità complessiva dei risultati mostrati all'utente.

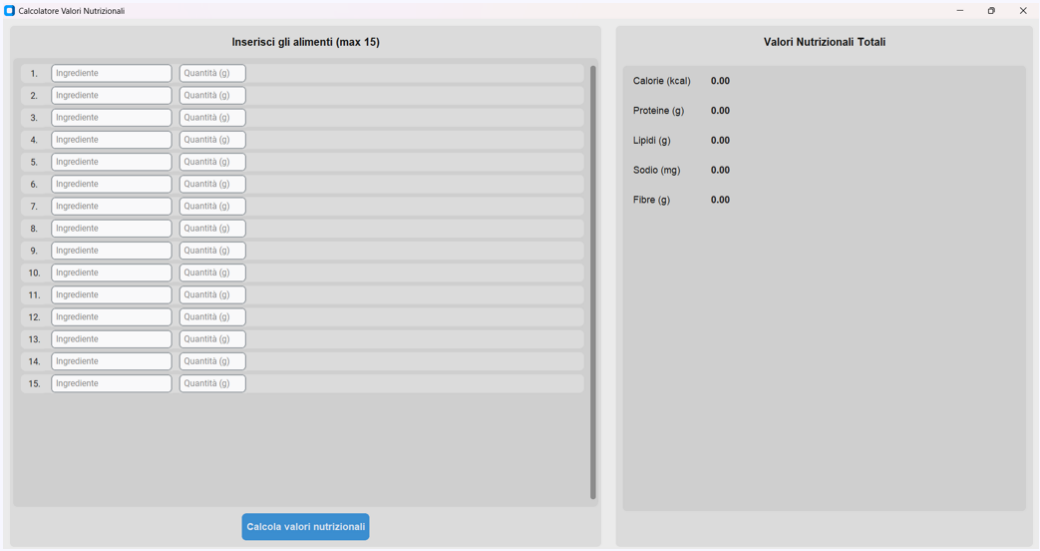
L'applicazione non si limita più a cercare alimenti via API, ma introduce anche autocompletamento intelligente, selezione vincolata dei cibi USDA, caching delle ricerche e ranking dei risultati. Di fatto si passa da un'applicazione didattica a un piccolo sistema software strutturato.

4.3.2 DeepSeek

Prompt iniziale

In seguito, viene mostrata l'interfaccia mostrata all'esecuzione del codice prodotto da *DeepSeek*.

DeepSeek



In risposta al prompt iniziale, DeepSeek produce un'applicazione funzionante che integra un database interno di alimenti con relativi valori nutrizionali, utilizzati per il calcolo automatico dei parametri richiesti. L'interfaccia consente l'inserimento degli alimenti e restituisce un output coerente con le operazioni richieste, includendo il calcolo dei valori aggregati.

Tuttavia, emergono alcune criticità nella gestione degli input non validi e nell'interazione dell'interfaccia grafica. In particolare, quando viene inserito un alimento non presente nel database, il sistema segnala correttamente l'errore all'utente, ma non esclude l'input dal processo di calcolo, mantenendo i dati precedenti e non effettuando il ricalcolo basandosi esclusivamente sugli elementi validi.

Dal punto di vista dell'interfaccia utente, non è prevista una visualizzazione della lista degli alimenti disponibili nel database, riducendo la trasparenza del sistema e aumentando la probabilità di errori in fase di input. Inoltre, si osservano variazioni dinamiche nella dimensione delle colonne dell'interfaccia in base alla lunghezza dei messaggi di stato, fenomeno che influisce negativamente sulla stabilità visiva del layout.

Nel Listato 4.3 viene riportato il codice generato:

```
1  import customtkinter as ctk
2
3  # ============================================================
4  # Database alimenti (valori per 100 g)
5  # Fonti: dati nutrizionali medi (CREA, USDA, tabelle di composizione)
6  # ============================================================
7  FOOD_DB = {
8      "pasta":        {"calorie": 350, "proteine": 12.0, "lipidi": 1.5, "sodio": 5,    "fibre
           ": 3.0},
9      "riso":         {"calorie": 360, "proteine": 7.0,  "lipidi": 0.6, "sodio": 5,    "fibre
           ": 1.0},
10     "pane":         {"calorie": 265, "proteine": 9.0,  "lipidi": 3.2, "sodio": 490,  "fibre
           ": 2.7},
11     "latte":        {"calorie": 61,  "proteine": 3.2,  "lipidi": 3.3, "sodio": 44,   "fibre
           ": 0.0},
12     "uova":         {"calorie": 143, "proteine": 12.6, "lipidi": 9.5, "sodio": 142,  "fibre
           ": 0.0},
13     "pollo":        {"calorie": 165, "proteine": 31.0, "lipidi": 3.6, "sodio": 74,   "fibre
           ": 0.0},
14     "salmone":      {"calorie": 208, "proteine": 20.0, "lipidi": 13.0,"sodio": 59,   "fibre
           ": 0.0},
15     "mela":         {"calorie": 52,  "proteine": 0.3,  "lipidi": 0.2, "sodio": 1,    "fibre
           ": 2.4},
```

```
    "pomodoro":     {"calorie": 18,  "proteine": 0.9,  "lipidi": 0.2, "sodio": 5,    "fibre": 1.2},
    "formaggio":    {"calorie": 431, "proteine": 38.0, "lipidi": 29.0,"sodio": 1529, "fibre": 0.0},
    "olio d'oliva": {"calorie": 884, "proteine": 0.0,  "lipidi": 100.0,"sodio": 0,   "fibre": 0.0},
    "broccoli":     {"calorie": 34,  "proteine": 2.8,  "lipidi": 0.4, "sodio": 33,   "fibre": 2.6},
    "patate":       {"calorie": 77,  "proteine": 2.0,  "lipidi": 0.1, "sodio": 6,    "fibre": 1.8},
    "yogurt":       {"calorie": 63,  "proteine": 5.3,  "lipidi": 3.3, "sodio": 46,   "fibre": 0.0}
}


class NutritionCalculator(ctk.CTk):
    def __init__(self):
        super().__init__()
        self.title("Calcolatore Valori Nutrizionali")
        self.geometry("950x650")
        ctk.set_appearance_mode("system")
        ctk.set_default_color_theme("blue")

        # Layout principale: due colonne
        self.left_frame = ctk.CTkFrame(self, width=450)
        self.left_frame.pack(side="left", fill="both", expand=True, padx=10, pady=10)

        self.right_frame = ctk.CTkFrame(self, width=400)
        self.right_frame.pack(side="right", fill="both", expand=True, padx=10, pady=10)

        self.build_left_panel()
        self.build_right_panel()

    def build_left_panel(self):
        """Crea l'area di inserimento ingredienti."""
        # Titolo
        ctk.CTkLabel(
            self.left_frame,
            text="Inserisci gli alimenti (max 15)",
            font=("Arial", 16, "bold")
        ).pack(pady=(10, 5))

        # Area scorrevole per le righe
        self.scroll_frame = ctk.CTkScrollableFrame(self.left_frame, width=420, height=420)
        self.scroll_frame.pack(fill="both", expand=True, padx=5, pady=5)

        # Liste per conservare i widget di input
        self.ingredient_entries = []
        self.quantity_entries = []

        # Creazione fissa di 15 righe
        for i in range(15):
            row = ctk.CTkFrame(self.scroll_frame)
            row.pack(fill="x", padx=5, pady=2)

            # Etichetta numerica
            ctk.CTkLabel(row, text=f"{i+1}.", width=30).pack(side="left", padx=5)

            # Campo ingrediente
            ing_entry = ctk.CTkEntry(row, placeholder_text="Ingrediente", width=180)
            ing_entry.pack(side="left", padx=5)

            # Campo quantita'
            qty\_entry = ctk.CTkEntry(row, placeholder_text="Quantita' (g)", width=100)
            qty_entry.pack(side="left", padx=5)

            self.ingredient_entries.append(ing_entry)
            self.quantity_entries.append(qty_entry)

        # Pulsante di calcolo
        self.calc_button = ctk.CTkButton(
            self.left_frame,
```

```
            text="Calcola valori nutrizionali",
            command=self.calculate,
            height=40,
            font=("Arial", 14, "bold")
        )
        self.calc_button.pack(pady=(5, 10))

    def build_right_panel(self):
        """Crea la tabella dei risultati."""
        ctk.CTkLabel(
            self.right_frame,
            text="Valori Nutrizionali Totali",
            font=("Arial", 16, "bold")
        ).pack(pady=10)

        # Contenitore della tabella
        table_frame = ctk.CTkFrame(self.right_frame)
        table_frame.pack(fill="both", expand=True, padx=10, pady=10)

        # Intestazioni e valori (etichette che aggiorneremo)
        nutrients = [
            ("Calorie (kcal)", "calorie"),
            ("Proteine (g)",   "proteine"),
            ("Lipidi (g)",     "lipidi"),
            ("Sodio (mg)",     "sodio"),
            ("Fibre (g)",      "fibre")
        ]

        self.value_labels = {}

        for i, (name, key) in enumerate(nutrients):
            lbl_name = ctk.CTkLabel(table_frame, text=name, font=("Arial", 14))
            lbl_name.grid(row=i, column=0, padx=15, pady=8, sticky="w")

            lbl_value = ctk.CTkLabel(table_frame, text="0.00", font=("Arial", 14, "bold"))
            lbl_value.grid(row=i, column=1, padx=15, pady=8, sticky="e")

            self.value_labels[key] = lbl_value

        # Etichetta per messaggi di errore o conferma
        self.message_label = ctk.CTkLabel(self.right_frame, text="", text_color="red")
        self.message_label.pack(pady=(5, 10))

    def calculate(self):
        """Legge gli input, cerca gli alimenti e aggiorna i totali."""
        # Reset messaggio
        self.message_label.configure(text="")
        totals = {"calorie": 0.0, "proteine": 0.0, "lipidi": 0.0, "sodio": 0.0, "fibre":
            0.0}
        found_any = False

        # Scansione delle 15 righe
        for ing_entry, qty_entry in zip(self.ingredient_entries, self.quantity_entries):
            ing = ing_entry.get().strip()
            qty_str = qty_entry.get().strip()

            # Salta righe completamente vuote
            if not ing and not qty_str:
                continue

            # Entrambi i campi devono essere compilati
            if not ing or not qty_str:
                self.message_label.configure(text="Completa entrambi i campi per ogni
                    alimento inserito.")
                return

            # Validazione quantita'
            try:
                qty = float(qty_str)
                if qty <= 0:
                    self.message_label.configure(text="La quantita' deve essere un numero
                        positivo.")
```

```
                    return
                except ValueError:
                    self.message_label.configure(text="Inserisci una quantita' numerica valida (grammi).")
                    return

                # Ricerca alimento nel database (case-insensitive)
                food = FOOD_DB.get(ing.lower())
                if food is None:
                    self.message_label.configure(text=f"Alimento '{ing}' non presente nel database.")
                    return

                # Aggiunge il contributo proporzionale alla quantita'
                factor = qty / 100.0
                for key in totals:
                    totals[key] += food[key] * factor

                found_any = True

        if not found_any:
            self.message_label.configure(text="Inserisci almeno un alimento per eseguire il calcolo.")
            return

        # Aggiorna le etichette con i valori calcolati
        for key, lbl in self.value_labels.items():
            if key == "sodio":
                lbl.configure(text=f"{totals[key]:.1f}")   # sodio in mg, un decimale
            else:
                lbl.configure(text=f"{totals[key]:.2f}")

        self.message_label.configure(text="Calcolo completato con successo.", text_color="green")


if __name__ == "__main__":
    app = NutritionCalculator()
    app.mainloop()
```

Listing 4.3: Implementazione di un calcolatore nutrizionale

Spiegazione generica del codice

L'architettura del codice si basa principalmente su due elementi: un database statico degli alimenti e una classe che implementa l'interfaccia grafica e la logica applicativa.

Nella prima parte del codice viene definito il dizionario `FOOD_DB`, che rappresenta un database locale contenente i valori nutrizionali per 100 g di alcuni alimenti.

Il metodo `build_left_panel()` costruisce l'area destinata all'inserimento degli alimenti. Vengono generate 15 righe, ciascuna composta da un'etichetta numerica, un campo di testo per il nome dell'alimento e un campo di testo per la quantità espressa in grammi. I riferimenti ai campi vengono salvati in due liste, permettendo successivamente di recuperarne facilmente il contenuto durante il calcolo.

Il metodo `build_right_panel()` realizza una semplice tabella composta da due colonne: il nome del nutriente e il valore calcolato.

L'intera logica applicativa è concentrata nel metodo `calculate()`. In esso, vengono analizzate sequenzialmente le 15 righe disponibili, si verifica se il nome dell'alimento e la quantità siano entrambi presenti, viene ricercato l'alimento all'interno del database locale e viene calcolato il contributo nutrizionale proporzionale alla quantità inserita; infine, vengono aggiornati i valori complessivi.

Secondo prompt

Utente

Non voglio avere sullo schermo già tutte e 15 le righe, ogni riga nuova deve apparirmi dopo aver cliccato un pulsante "+ aggiungi alimento" così da avere più pulizia visiva. Aggiungimi accanto ad ogni riga per l'inserimento di un alimento la possibilità di rimuovere tale riga. Il tema dell'interfaccia deve essere tendente a colori scuri. Gli arrotondamenti devono essere effettuati per eccesso.
Inoltre, apportami le seguenti modifiche:

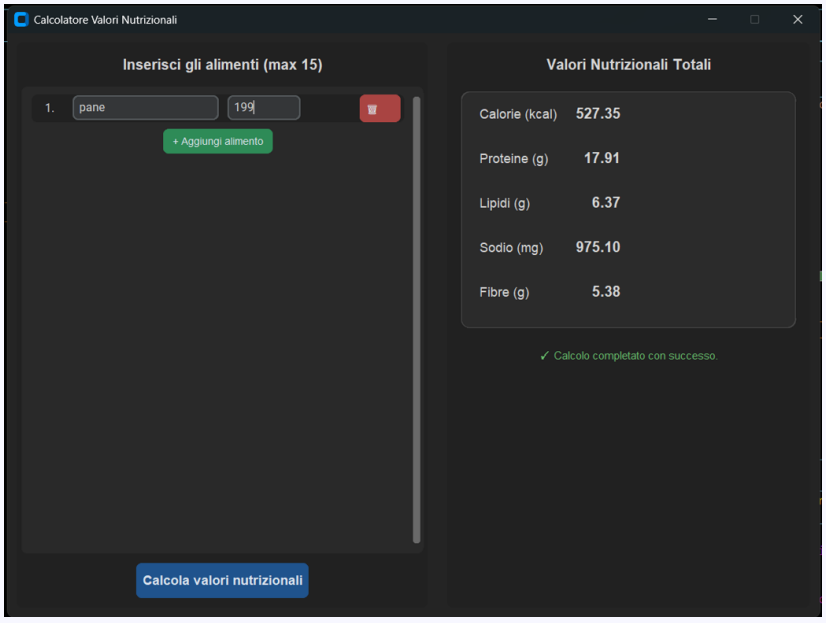
- Quando cerco di effettuare il calcolo totale dei valori nutrizionali, ma per qualsiasi motivo (alimento non presente nel database, quantità inserita non valida, ecc) il procedimento di esso fallisce, i valori dei valori nutrizionali complessivi deve azzerarsi.
- Le dimensioni delle due colonne devono rimanere fisse in lunghezza e larghezza.

Le modifiche richieste sono state implementate correttamente. In particolare, l'interfaccia è stata resa più ordinata introducendo l'aggiunta dinamica delle righe dedicate agli alimenti tramite un pulsante dedicato, evitando la visualizzazione iniziale di tutte le righe disponibili. È stata, inoltre, aggiunta la possibilità di eliminare singolarmente ciascuna riga inserita, migliorando la flessibilità di utilizzo dell'applicazione.

Dal punto di vista grafico, il modello ha applicato un tema con predominanza di colori scuri e ha mantenuto fisse le dimensioni delle due colonne dell'interfaccia, eliminando le variazioni di layout osservate nella versione precedente durante la visualizzazione dei messaggi di stato.

Anche la logica applicativa è stata corretta. In caso di errore durante il calcolo dei valori nutrizionali complessivi, dovuto ad esempio alla presenza di alimenti non riconosciuti o a quantità non valide, i valori calcolati vengono ora azzerati, evitando che rimangano visualizzati risultati ottenuti da elaborazioni precedenti. È stato, inoltre, implementato l'arrotondamento per eccesso dei valori restituiti, in accordo con quanto richiesto nel prompt.

DeepSeek



Terzo prompt

Utente

Implementa il collegamento alla *USDA FoodData Central API* per l'estrazione dei valori nutrizionali degli alimenti inseriti, rimuovendo, quindi, il database interno creato in precedenza. La chiave API da utilizzare è `Hola3CZ5B8PwOBJx02Lay9b6ki9E3CZu4cGHbNxH`. Tutti i valori nutrizionali devono essere ottenuti esclusivamente dalla *USDA FoodData Central API*.
Ordina i suggerimenti secondo il seguente criterio di priorità:

1. Foundation
2. SR Legacy
3. Survey (FNDDS)
4. Branded

Per ciascun gruppo privilegia gli alimenti semplici e generici. Ad esempio, quando l'utente ricerca `"chicken breast"`, i primi risultati mostrati dovranno essere:

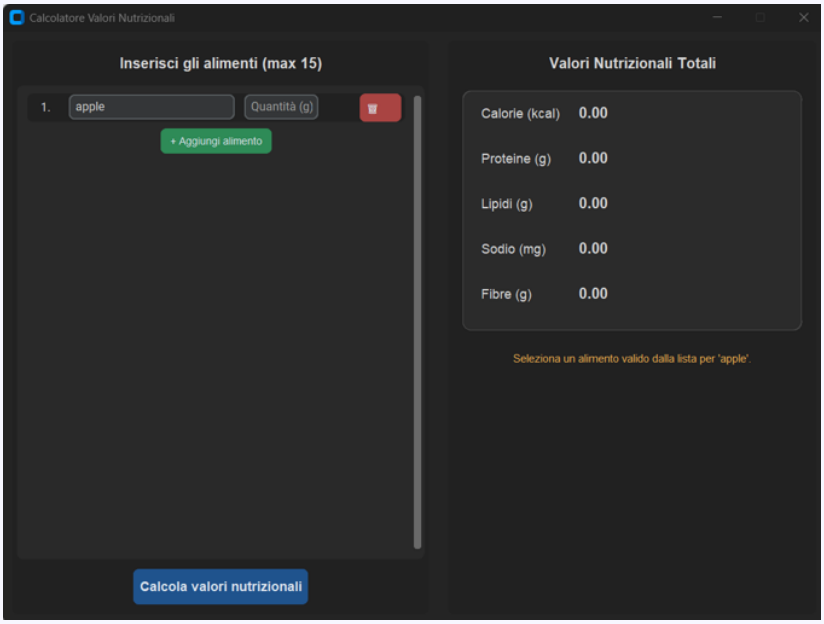
- Chicken breast, raw
- Chicken breast, roasted
- Chicken breast, grilled

Successivamente potranno essere mostrati:

- Rotisserie
- Breaded
- Commercial products

Mantieni comunque il valore originale di `fdcId` per il calcolo dei valori nutrizionali. Non mostrare all'utente la sorgente da cui provengono i dati (ad esempio *Foundation*, *SR Legacy*, *Survey* o *Branded*), ma esclusivamente la descrizione dell'alimento.

DeepSeek



Il modello ha recepito le richieste e ha sostituito il database locale con chiamate alla *USDA*

FoodData Central API, implementando anche un meccanismo di ricerca con suggerimenti. Tuttavia, durante i test sono emerse criticità che ne compromettono il corretto funzionamento.

In particolare, al primo inserimento di un alimento la lista dei risultati viene visualizzata correttamente, ma la selezione di uno degli elementi proposti non viene registrata dall'interfaccia. Di conseguenza, l'alimento non viene associato al relativo identificativo *fdcId* e il calcolo dei valori nutrizionali non può essere completato.

Successivamente, effettuando una nuova ricerca, l'applicazione genera un errore HTTP 400 (Bad Request) durante la richiesta alla *USDA FoodData Central API*, interrompendo completamente il funzionamento del sistema di ricerca e impedendo ulteriori interrogazioni. Tale comportamento evidenzia una gestione non corretta delle richieste verso il servizio remoto e dello stato dell'interfaccia dopo il primo tentativo di ricerca.

Nel Listato 4.4 viene riportato il codice generato:

```
import customtkinter as ctk
import math
import requests
import threading
import time

# ============================================================
# CONFIGURAZIONE API USDA
# ============================================================
USDA_API_KEY = "Hola3CZ5B8PwOBJx02Lay9b6ki9E3CZu4cGHbNxH"
USDA_SEARCH_URL = "https://api.nal.usda.gov/fdc/v1/foods/search"
USDA_FOOD_URL = "https://api.nal.usda.gov/fdc/v1/food/{}"

# DataType prioritari (dal piu' al meno preferito)
DATA_TYPE_PRIORITY = {
    "Foundation": 1,
    "SR Legacy": 2,
    "Survey (FNDDS)": 3,
    "Branded": 4
}

# Parole chiave che abbassano la priorita' (alimenti poco generici)
COMPLEX_KEYWORDS = [
    "rotisserie", "breaded", "commercial", "with", "cooked",
    "brand", "frozen", "canned", "ready-to-eat", "instant",
    "prepared", "formulated", "baby food", "infant formula"
]


class SuggestionPopup(ctk.CTkToplevel):
    """Finestra popup per i suggerimenti dell'autocompletamento."""
    def __init__(self, parent, entry, commit_callback):
        super().__init__(parent)
        self.entry = entry
        self.commit_callback = commit_callback
        self.overrideredirect(True)  # nessuna barra del titolo
        self.attributes("-topmost", True)

        self.configure(fg_color="#2b2b2b")
        self.grid_columnconfigure(0, weight=1)

        # Area scorrevole con i suggerimenti
        self.scroll = ctk.CTkScrollableFrame(self, width=300, height=150)
        self.scroll.grid(row=0, column=0, sticky="nsew")

        self.suggestions = []  # memorizza i dati (fdcId, description)

    def set_suggestions(self, items):
        """Popola il popup con i suggerimenti e lo posiziona sotto l'entry."""
        # Pulisce i widget precedenti
        for w in self.scroll.winfo_children():
            w.destroy()
        self.suggestions = items

```

```
        if not items:
            self.withdraw()
            return

        for idx, item in enumerate(items):
            btn = ctk.CTkButton(
                self.scroll,
                text=item["description"],
                anchor="w",
                fg_color="transparent",
                hover_color="#3a3a3a",
                corner_radius=0,
                command=lambda i=idx: self.select(i)
            )
            btn.pack(fill="x", padx=2, pady=1)

        # Posizionamento
        x = self.entry.winfo_rootx()
        y = self.entry.winfo_rooty() + self.entry.winfo_height() + 2
        self.geometry(f"+{x}+{y}")
        self.deiconify()

    def select(self, idx):
        """Seleziona un suggerimento e richiama la callback."""
        item = self.suggestions[idx]
        self.commit_callback(item["fdcId"], item["description"])
        self.withdraw()


class NutritionCalculator(ctk.CTk):
    def __init__(self):
        super().__init__()
        self.title("Calcolatore Valori Nutrizionali")
        self.geometry("890x640")
        self.resizable(False, False)

        ctk.set_appearance_mode("dark")
        ctk.set_default_color_theme("dark-blue")

        self.left_frame = ctk.CTkFrame(self, width=450, height=600)
        self.left_frame.pack(side="left", fill="y", expand=False, padx=10, pady=10)
        self.left_frame.pack_propagate(False)

        self.right_frame = ctk.CTkFrame(self, width=400, height=600)
        self.right_frame.pack(side="right", fill="y", expand=False, padx=10, pady=10)
        self.right_frame.pack_propagate(False)

        # Oggetti per l'autocompletamento
        self.active_popup = None
        self.search_timer = None
        self.current_entry = None
        self.current_row = None

        self.build_left_panel()
        self.build_right_panel()

    # ------------------------------------------------------------
    # COSTRUZIONE INTERFACCIA
    # ------------------------------------------------------------
    def build_left_panel(self):
        ctk.CTkLabel(self.left_frame, text="Inserisci gli alimenti (max 15)",
                     font=("Arial", 16, "bold")).pack(pady=(10, 5))

        self.scroll_frame = ctk.CTkScrollableFrame(self.left_frame, width=420, height=400)
        self.scroll_frame.pack(fill="both", expand=True, padx=5, pady=5)

        self.ingredient_entries = []
        self.quantity_entries = []
        self.fdc_ids = []           # memorizza l'fdcId selezionato per ogni riga
        self.row_frames = []
        self.number_labels = []

```

```
127         bottom_frame = ctk.CTkFrame(self.scroll_frame, fg_color="transparent")
128         bottom_frame.pack(side="bottom", fill="x", padx=0, pady=(5, 10))
129 
130         self.add_button = ctk.CTkButton(
131             bottom_frame, text="+ Aggiungi alimento",
132             command=self.add_ingredient_row, width=120, height=28,
133             font=("Arial", 11), fg_color="#2e8b57", hover_color="#1f6d3d"
134         )
135         self.add_button.pack(expand=True)
136 
137         self.calc_button = ctk.CTkButton(
138             self.left_frame, text="Calcola valori nutrizionali",
139             command=self.calculate, height=38,
140             font=("Arial", 14, "bold"), fg_color="#1f538d", hover_color="#14375e"
141         )
142         self.calc_button.pack(pady=(5, 10))
143 
144     def add_ingredient_row(self):
145         if len(self.ingredient_entries) >= 15:
146             self.add_button.configure(state="disabled", text="Massimo 15 alimenti")
147             return
148 
149         row_frame = ctk.CTkFrame(self.scroll_frame)
150         row_frame.pack(fill="x", padx=5, pady=2, before=self.add_button.master)
151 
152         num = len(self.ingredient_entries) + 1
153         num_label = ctk.CTkLabel(row_frame, text=f"{num}.", width=30)
154         num_label.pack(side="left", padx=5)
155 
156         ing_entry = ctk.CTkEntry(row_frame, placeholder_text="Cerca alimento...", width
                =180)
157         ing_entry.pack(side="left", padx=5)
158 
159         qty_entry = ctk.CTkEntry(row_frame, placeholder_text="Quantita' (g)", width=80)
160         qty_entry.pack(side="left", padx=5)
161 
162         remove_btn = ctk.CTkButton(
163             row_frame, text="X", width=30, height=30,
164             fg_color="#a94442", hover_color="#843534",
165             command=lambda f=row_frame: self.remove_row(f)
166         )
167         remove_btn.pack(side="right", padx=5)
168 
169         self.ingredient_entries.append(ing_entry)
170         self.quantity_entries.append(qty_entry)
171         self.fdc_ids.append(None)          # nessun alimento selezionato
172         self.row_frames.append(row_frame)
173         self.number_labels.append(num_label)
174 
175         # Associa gli eventi per l'autocompletamento
176         row_idx = len(self.ingredient_entries) - 1
177         ing_entry.bind("<KeyRelease>", lambda e, idx=row_idx: self.on_key_release(idx))
178         ing_entry.bind("<FocusOut>", lambda e: self.hide_popup())
179 
180         if len(self.ingredient_entries) == 15:
181             self.add_button.configure(state="disabled", text="Massimo 15 alimenti")
182 
183     def remove_row(self, frame):
184         try:
185             idx = self.row_frames.index(frame)
186         except ValueError:
187             return
188 
189         del self.ingredient_entries[idx]
190         del self.quantity_entries[idx]
191         del self.fdc_ids[idx]
192         del self.row_frames[idx]
193         del self.number_labels[idx]
194         frame.destroy()
195 
196         for i in range(idx, len(self.number_labels)):
197             self.number_labels[i].configure(text=f"{i+1}.")
```

```
198
199         if len(self.ingredient_entries) < 15:
200             self.add_button.configure(state="normal", text="+ Aggiungi alimento")
201
202     # ------------------------------------------------------------
203     # AUTOCOMPLETAMENTO
204     # ------------------------------------------------------------
205     def on_key_release(self, idx):
206         """Avvia la ricerca con un debounce di 500 ms."""
207         if self.search_timer is not None:
208             self.after_cancel(self.search_timer)
209
210         self.current_row = idx
211         self.current_entry = self.ingredient_entries[idx]
212
213         text = self.current_entry.get().strip()
214         if len(text) < 2:
215             self.hide_popup()
216             return
217
218         self.search_timer = self.after(500, lambda: self.search_food(text))
219
220     def search_food(self, query):
221         """Esegue la ricerca USDA in un thread separato."""
222         def worker():
223             try:
224                 params = {
225                     "query": query,
226                     "api_key": USDA_API_KEY,
227                     "pageSize": 25,           # risultati sufficienti per filtrare
228                     "dataType": ["Foundation", "SR Legacy", "Survey (FNDDS)", "Branded"]
229                 }
230                 resp = requests.get(USDA_SEARCH_URL, params=params, timeout=10)
231                 resp.raise_for_status()
232                 data = resp.json()
233                 foods = data.get("foods", [])
234
235                 # Ordina secondo i criteri richiesti
236                 sorted_foods = self.sort_results(foods)
237
238                 # Prepara i suggerimenti (massimo 10)
239                 suggestions = []
240                 for f in sorted_foods[:10]:
241                     suggestions.append({
242                         "fdcId": f["fdcId"],
243                         "description": f["description"]
244                     })
245
246                 self.after(0, lambda: self.show_suggestions(suggestions))
247             except Exception as e:
248                 # In caso di errore, semplicemente non mostra suggerimenti
249                 print(f"Errore ricerca USDA: {e}")
250                 self.after(0, self.hide_popup)
251
252         threading.Thread(target=worker, daemon=True).start()
253
254     def sort_results(self, foods):
255         """Ordina i risultati per dataType e poi per 'semplicita'."""
256         # Calcola un punteggio per ogni alimento
257         scored = []
258         for f in foods:
259             dt = f.get("dataType", "")
260             base = DATA_TYPE_PRIORITY.get(dt, 99)
261
262             # Penalita' per descrizioni lunghe o contenenti parole complesse
263             desc = f.get("description", "")
264             words = desc.lower().split()
265             word_penalty = max(0, len(words) - 3) * 0.5
266             complexity = 0
267             for kw in COMPLEX_KEYWORDS:
268                 if kw in desc.lower():
269                     complexity += 1
```

```
270             score = base + word_penalty + complexity * 2
271             scored.append((score, f))
272 
273         scored.sort(key=lambda x: x[0])
274         return [item[1] for item in scored]
275 
276     def show_suggestions(self, suggestions):
277         """Crea o aggiorna il popup dei suggerimenti."""
278         if not self.current_entry:
279             return
280 
281         # Chiude il popup precedente se esiste
282         self.hide_popup()
283 
284         if not suggestions:
285             return
286 
287         self.active_popup = SuggestionPopup(
288             self, self.current_entry,
289             self.on_food_selected
290         )
291         self.active_popup.set_suggestions(suggestions)
292 
293     def hide_popup(self):
294         if self.active_popup:
295             self.active_popup.destroy()
296             self.active_popup = None
297 
298     def on_food_selected(self, fdc_id, description):
299         """Callback quando l'utente seleziona un alimento dal popup."""
300         if self.current_row is not None and self.current_entry:
301             self.current_entry.delete(0, "end")
302             self.current_entry.insert(0, description)
303             self.fdc_ids[self.current_row] = fdc_id
304         self.hide_popup()
305 
306     # ------------------------------------------------------------
307     # PANNELLO DESTRO (RISULTATI)
308     # ------------------------------------------------------------
309     def build_right_panel(self):
310         ctk.CTkLabel(self.right_frame, text="Valori Nutrizionali Totali",
311                      font=("Arial", 16, "bold")).pack(pady=(10, 15))
312 
313         table_box = ctk.CTkFrame(self.right_frame, fg_color="#2b2b2b",
314                                  corner_radius=10, border_width=1,
315                                  border_color="#4a4a4a")
316         table_box.pack(fill="x", padx=15, pady=0, ipady=10)
317 
318         nutrients = [
319             ("Calorie (kcal)", "calorie"),
320             ("Proteine (g)",   "proteine"),
321             ("Lipidi (g)",     "lipidi"),
322             ("Sodio (mg)",     "sodio"),
323             ("Fibre (g)",      "fibre")
324         ]
325         self.value_labels = {}
326         for i, (name, key) in enumerate(nutrients):
327             lbl_name = ctk.CTkLabel(table_box, text=name, font=("Arial", 14), anchor="w")
328             lbl_name.grid(row=i, column=0, padx=(20, 10), pady=10, sticky="w")
329             lbl_value = ctk.CTkLabel(table_box, text="0.00", font=("Arial", 16, "bold"),
                                         anchor="e")
330             lbl_value.grid(row=i, column=1, padx=(10, 20), pady=10, sticky="e")
331             self.value_labels[key] = lbl_value
332 
333         self.message_label = ctk.CTkLabel(self.right_frame, text="",
334                                           text_color="#66bb6a", font=("Arial", 12))
335         self.message_label.pack(pady=(15, 5))
336 
337     def reset_totals(self):
338         for lbl in self.value_labels.values():
339             lbl.configure(text="0.00")
340 
```

```
341     # ------------------------------------------------------------
342     # CALCOLO
343     # ------------------------------------------------------------
344     def calculate(self):
345         self.reset_totals()
346         self.message_label.configure(text="")
347 
348         totals = {"calorie": 0.0, "proteine": 0.0, "lipidi": 0.0, "sodio": 0.0, "fibre":
                0.0}
349         found_any = False
350 
351         for idx, (ing_entry, qty_entry) in enumerate(zip(self.ingredient_entries, self.
                quantity_entries)):
352             ing_text = ing_entry.get().strip()
353             qty_str = qty_entry.get().strip()
354 
355             if not ing_text and not qty_str:
356                 continue
357 
358             if not ing_text or not qty_str:
359                 self.message_label.configure(
360                     text="Completa entrambi i campi per ogni alimento inserito.",
361                     text_color="#f0ad4e"
362                 )
363                 return
364 
365             try:
366                 qty = float(qty_str)
367                 if qty <= 0:
368                     self.message_label.configure(
369                         text="La quantita' deve essere un numero positivo.",
370                         text_color="#f0ad4e"
371                     )
372                     return
373             except ValueError:
374                 self.message_label.configure(
375                     text="Inserisci una quantita' numerica valida (grammi).",
376                     text_color="#f0ad4e"
377                 )
378                 return
379 
380             # Recupera l'fdcId selezionato
381             fdc_id = self.fdc_ids[idx]
382             if fdc_id is None:
383                 self.message_label.configure(
384                     text=f"Seleziona un alimento valido dalla lista per '{ing_text}'.",
385                     text_color="#f0ad4e"
386                 )
387                 return
388 
389             # Ottiene i nutrienti per 100g dall'API
390             nutrients = self.fetch_nutrients(fdc_id)
391             if nutrients is None:
392                 self.message_label.configure(
393                     text=f"Impossibile recuperare i dati nutrizionali per '{ing_text}'.",
394                     text_color="#f0ad4e"
395                 )
396                 return
397 
398             factor = qty / 100.0
399             for key in totals:
400                 totals[key] += nutrients.get(key, 0.0) * factor
401 
402             found_any = True
403 
404         if not found_any:
405             self.message_label.configure(
406                 text="Inserisci almeno un alimento per eseguire il calcolo.",
407                 text_color="#f0ad4e"
408             )
409             return
410 
```

```
411         # Arrotondamento per eccesso ai centesimi
412         for key in totals:
413             totals[key] = math.ceil(totals[key] * 100) / 100
414 
415         for key, lbl in self.value_labels.items():
416             lbl.configure(text=f"{totals[key]:.2f}")
417 
418         self.message_label.configure(
419             text="Calcolo completato con successo.",
420             text_color="#66bb6a"
421         )
422 
423     def fetch_nutrients(self, fdc_id):
424         """Recupera i nutrienti per 100g dell'alimento specificato."""
425         try:
426             url = USDA_FOOD_URL.format(fdc_id)
427             params = {"api_key": USDA_API_KEY}
428             resp = requests.get(url, params=params, timeout=10)
429             resp.raise_for_status()
430             data = resp.json()
431 
432             # Mappa id nutrienti USDA -> nostre chiavi
433             # Energy (kcal) = 1008, Protein = 1003, Total lipid = 1004,
434             # Sodium = 1093, Fiber = 1079
435             target_nutrients = {
436                 1008: "calorie",
437                 1003: "proteine",
438                 1004: "lipidi",
439                 1093: "sodio",
440                 1079: "fibre"
441             }
442             result = {"calorie": 0.0, "proteine": 0.0, "lipidi": 0.0, "sodio": 0.0, "fibre"
                    : 0.0}
443 
444             for nut in data.get("foodNutrients", []):
445                 nut_id = nut.get("nutrientId")
446                 if nut_id in target_nutrients:
447                     key = target_nutrients[nut_id]
448                     # Il valore e' per 100g se disponibile, altrimenti 0
449                     value = nut.get("value", 0.0)
450                     result[key] = float(value)
451 
452             return result
453         except Exception as e:
454             print(f"Errore nel recupero nutrienti per fdcId {fdc_id}: {e}")
455             return None
456 
457 
458 if __name__ == "__main__":
459     app = NutritionCalculator()
460     app.mainloop()
```

Listing 4.4: Implementazione di un calcolatore nutrizionale

Spiegazione generica del codice

Nella parte iniziale del codice vengono definite le costanti necessarie per la comunicazione con il servizio USDA e due strutture ausiliarie per l'assegnazione della priorità delle diverse tipologie di alimenti restituite. Ciò permette di migliorare la qualità dei risultati proposti all'utente privilegiando alimenti generici rispetto ai prodotti industriali.

Una delle principali innovazioni introdotte è il sistema di ricerca assistita degli alimenti. Durante la digitazione del nome dell'alimento, viene attivato il metodo `on_key_release()`, che permette alla ricerca, eseguita con il metodo `search_food()`, di essere avviata soltanto dopo che l'utente ha interrotto la digitazione, evitando un numero eccessivo di richieste al server.

I risultati ottenuti vengono successivamente elaborati dal metodo `sort_results()` che ne decide l'ordinamento nella finestra popup `SuggestionPopup`.

Quarto prompt

Utente

Ho un problema nell'estrazione dei valori nutrizionali degli alimenti inseriti, ecco cosa ottengo nel terminale quando invio la richiesta di calcolo:

```
[RICERCA] Trovati 25 alimenti, mostrati 10

[FETCH NUTRIENTS] Nutrienti trovati:
{'calorie': 0.0, 'proteine': 0.0, 'lipidi': 0.0,
 'sodio': 0.0, 'fibre': 0.0}

[CALCOLO] Totali finali:
{'calorie': 0.0, 'proteine': 0.0, 'lipidi': 0.0,
 'sodio': 0.0, 'fibre': 0.0}
```

Dopo numerose iterazioni in cui erano stati corretti principalmente gli effetti del problema, DeepSeek individua la causa del malfunzionamento nell'errata interpretazione della struttura della risposta JSON restituita dalla *USDA FoodData Central API*. L'errore non era quindi riconducibile alle richieste HTTP o alla gestione dell'interfaccia grafica, bensì al modo in cui venivano estratti i dati nutrizionali dalla risposta dell'API.

Sebbene il sistema non risulti ancora completamente stabile, il debugging si sposta finalmente dal livello superficiale dei messaggi di errore all'analisi della logica di elaborazione dei dati, consentendo di porre le basi per la risoluzione definitiva del problema.

Quinto prompt

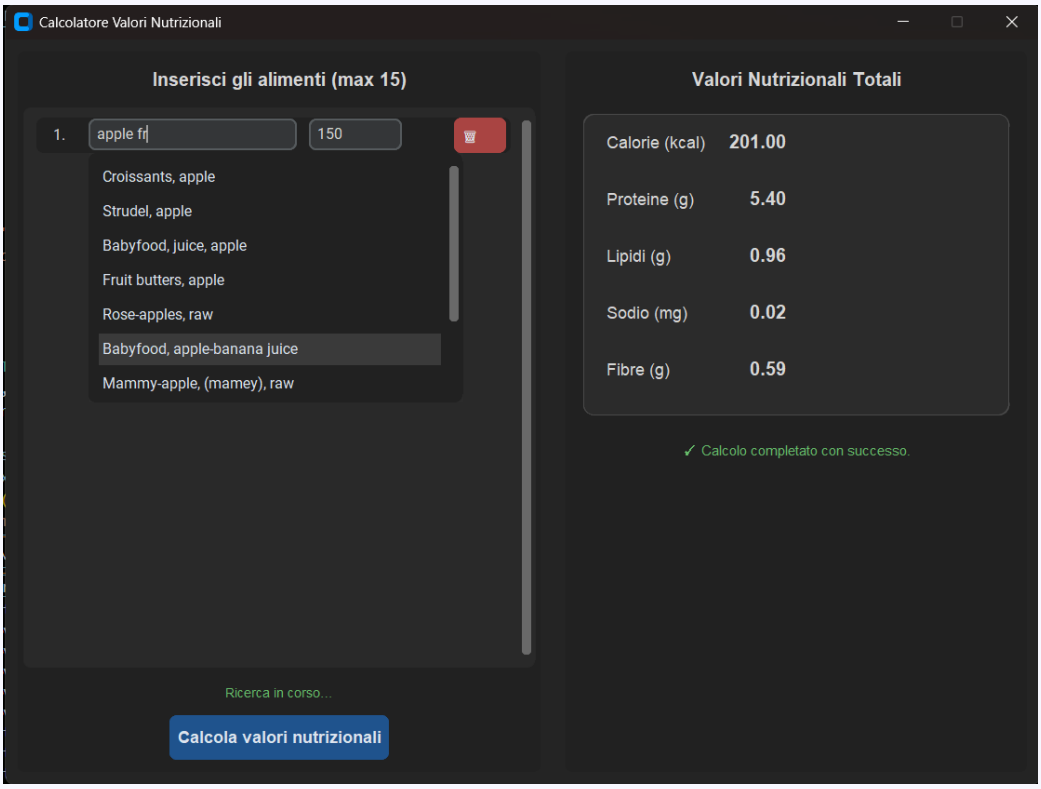
Utente

Continuano ad esserci imprecisioni nel momento in cui viene mandata la ricerca riguardo i caratteri inseriti nella casella di testo: a volte funziona e altre no a prescindere che sia una parola completa o una parte dell'alimento che sto cercando. Voglio che mi risolvi questo problema in modo che, qualsiasi sia l'input, il sistema riesca a trovarmi delle risposte senza darmi il solito error 400; non è possibile che sullo stesso input una volta funzioni e una volta no.

In questa fase vengono risolte anche le ultime criticità legate all'autocompletamento e alle ricerche parziali, eliminando i malfunzionamenti residui che si manifestavano durante l'interazione con la *USDA FoodData Central API*. Il risultato è un'applicazione complessivamente stabile, in grado di recuperare correttamente gli alimenti e i relativi valori nutrizionali attraverso il servizio esterno.

L'evoluzione osservata evidenzia come DeepSeek sia in grado di migliorare progressivamente il software attraverso iterazioni successive, arrivando, infine, a una soluzione funzionante. Tuttavia, il percorso seguito dal modello è caratterizzato da numerosi tentativi intermedi e dall'introduzione di regressioni che hanno richiesto ulteriori cicli di correzione.

DeepSeek



4.3.3 Claude

Prompt iniziale

Il modello ha generato una struttura complessivamente coerente con la richiesta, implementando correttamente la suddivisione dell'interfaccia e la logica di inserimento multiplo degli alimenti. Tuttavia, durante la generazione del codice è stata introdotta autonomamente una dipendenza da un'API esterna per il reperimento dei valori nutrizionali, non prevista dal prompt, al posto della creazione di un database locale.

Questa scelta ha modificato implicitamente l'architettura iniziale del sistema, introducendo un livello di complessità non richiesto e rendendo necessaria la gestione di chiavi di accesso e richieste HTTP, non contemplate nella specifica.

Dal punto di vista esecutivo, il codice ha, inoltre, evidenziato un errore nella configurazione del layout grafico. In particolare, è stato utilizzato in modo non corretto il parametro sticky del sistema grid, assegnando il valore "center", non compatibile con le specifiche della libreria Tkinter. L'esecuzione del programma è, quindi, fallita con la seguente eccezione:

Claude

```
_tkinter.TclError: bad stickyness value "center": must be a string containing n, e, s, and/or w
```

Secondo prompt

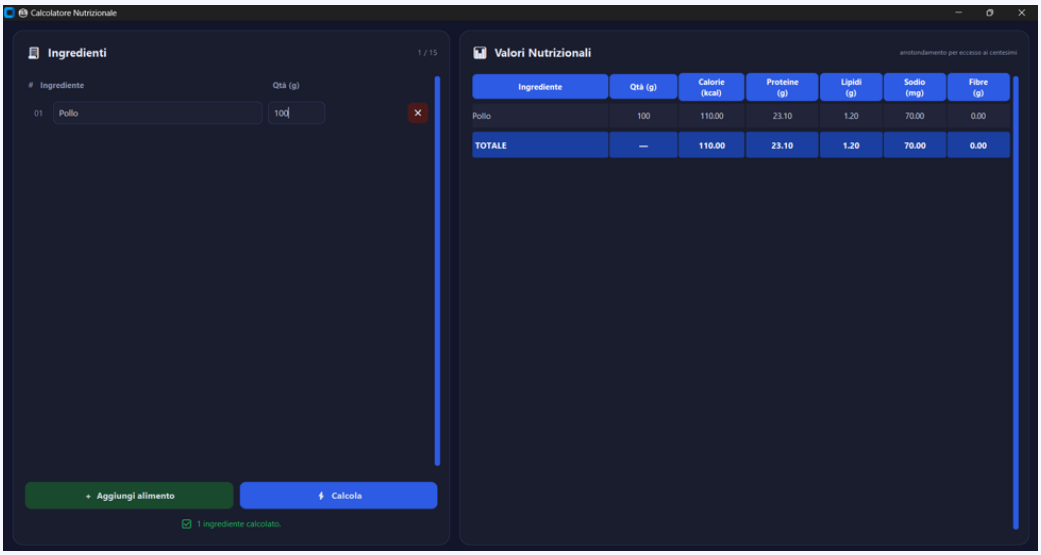
Tra il primo e il secondo prompt, il codice generato è stato oggetto di una fase intermedia di correzione necessaria per renderne possibile l'esecuzione. Una volta risolto tale problema e ottenuta una versione funzionante dell'applicazione, si è proceduto con un nuovo intervento

progettuale volto a modificare l'architettura del sistema, spostando la gestione dei dati nutrizionali da una possibile fonte esterna a una soluzione locale.

Utente

Il calcolo fallisce poiché non è stata inserita alcuna API key. Crea un database interno con alcuni ingredienti scelti da te e i rispettivi valori nutrizionali per 100g e basati su essi. Inoltre, non voglio avere visualizzate già tutte e 15 le righe; ogni riga nuova deve apparirmi dopo aver cliccato un pulsante "+ aggiungi alimento" così da avere più pulizia visiva. Gli arrotondamenti devono essere effettuati per eccesso fino ai centesimi. Aggiungimi accanto ad ogni riga per l'inserimento di un alimento la possibilità di rimuovere tale riga.

Claude



Il modello ha ristrutturato l'applicazione introducendo una gestione dinamica dei componenti dell'interfaccia, in cui le righe di input vengono generate solo su richiesta dell'utente.

È stata, inoltre, implementata la logica di gestione di un database interno per gli alimenti, insieme al calcolo aggregato dei valori nutrizionali complessivi. Il modello ha, quindi, modificato non solo la parte grafica, ma anche la struttura dati sottostante dell'applicazione.

A differenza delle soluzioni generate dai modelli analizzati in precedenza, Claude non si limita a visualizzare esclusivamente i valori nutrizionali complessivi, ma mostra anche il contributo di ciascun alimento in funzione della quantità inserita. Per ogni riga vengono, infatti, calcolati e visualizzati i corrispondenti valori nutrizionali.

Tuttavia, sono emerse alcune criticità nella gestione dello stato dell'interfaccia e nella validazione degli input, in particolare nei casi di valori non validi o incompleti, che potevano influenzare il corretto aggiornamento dei risultati.

Nel Listato 4.5 viene riportato il codice generato:

```
1  import math
2  import customtkinter as ctk
3
4  # Theme
5  ctk.set_appearance_mode("dark")
6  ctk.set_default_color_theme("blue")
7
8  COLORS = {
9      "bg_panel":      "#1a1d2e",
10     "bg_row_even":   "#22253a",
11     "bg_row_odd":    "#1e2133",
12     "bg_header":     "#2d5be3",
```

```
    "bg_total":     "#1b3fa0",
    "accent":       "#2d5be3",
    "accent_hover": "#3d6ef3",
    "add_btn":      "#1a4a2e",
    "add_btn_hov":  "#215c38",
    "del_btn":      "#4a1a1a",
    "del_btn_hov":  "#6b2020",
    "success":      "#22c55e",
    "warning":      "#f59e0b",
    "error":        "#ef4444",
    "text_primary": "#e8eaf6",
    "text_secondary":"#8892b0",
    "border":       "#2d3561",
}

# Database nutrizionale (valori per 100 g)
# Chiavi in minuscolo per ricerca case-insensitive
# Struttura: { nome: (calorie_kcal, proteine_g, lipidi_g, sodio_mg, fibre_g) }
NUTRIENT_DB: dict[str, tuple[float, float, float, float, float]] = {
    # Carni e pesce
    "pollo petto":          (110.0,  23.1,  1.2,   70.0,  0.0),
    "pollo":                (110.0,  23.1,  1.2,   70.0,  0.0),
    "manzo macinato":       (254.0,  17.2, 20.0,   72.0,  0.0),
    "manzo":                (250.0,  26.0, 15.0,   60.0,  0.0),
    "salmone":              (208.0,  20.4, 13.4,   59.0,  0.0),
    "tonno in scatola":     (116.0,  25.5,  0.8,  400.0,  0.0),
    "tonno":                (144.0,  23.3,  4.9,   47.0,  0.0),
    "merluzzo":             ( 82.0,  17.8,  0.7,   54.0,  0.0),
    "prosciutto cotto":     (136.0,  18.4,  5.9,  950.0,  0.0),
    "prosciutto crudo":     (145.0,  25.0,  4.8, 2700.0,  0.0),
    "pancetta":             (460.0,  13.0, 45.0, 1717.0,  0.0),
    "uovo intero":          (155.0,  13.0, 11.0,  124.0,  0.0),
    "uova":                 (155.0,  13.0, 11.0,  124.0,  0.0),
    "albume":               ( 52.0,  10.9,  0.2,  166.0,  0.0),
    # Cereali e derivati
    "riso bianco cotto":    (130.0,   2.7,  0.3,    1.0,  0.4),
    "riso":                 (130.0,   2.7,  0.3,    1.0,  0.4),
    "pasta cotta":          (157.0,   5.8,  0.9,    1.0,  1.8),
    "pasta":                (157.0,   5.8,  0.9,    1.0,  1.8),
    "pane bianco":          (265.0,   8.9,  3.2,  491.0,  2.7),
    "pane integrale":       (247.0,   9.0,  3.5,  400.0,  6.8),
    "farina 00":            (364.0,  10.5,  0.9,    2.0,  2.7),
    "avena":                (389.0,  17.0,  7.0,    2.0, 10.6),
    "fiocchi di avena":     (389.0,  17.0,  7.0,    2.0, 10.6),
    "quinoa cotta":         (120.0,   4.4,  1.9,    7.0,  2.8),
    "farro cotto":          (150.0,   6.0,  1.0,    5.0,  3.5),
    "mais":                 ( 86.0,   3.3,  1.4,   15.0,  2.7),
    # Legumi
    "ceci cotti":           (164.0,   8.9,  2.6,    7.0,  7.6),
    "lenticchie cotte":     (116.0,   9.0,  0.4,    2.0,  7.9),
    "fagioli borlotti":     (133.0,   8.7,  0.5,    5.0,  6.0),
    "fagioli cannellini":   (120.0,   8.3,  0.4,    5.0,  5.5),
    "soia":                 (446.0,  36.5, 19.9,    2.0,  9.3),
    "piselli":              ( 81.0,   5.4,  0.4,    5.0,  5.1),
    "edamame":              (122.0,  10.9,  5.2,   63.0,  5.2),
    # Latticini
    "latte intero":         ( 61.0,   3.2,  3.3,   43.0,  0.0),
    "latte":                ( 61.0,   3.2,  3.3,   43.0,  0.0),
    "latte scremato":       ( 34.0,   3.4,  0.1,   52.0,  0.0),
    "yogurt bianco":        ( 59.0,   3.5,  3.3,   36.0,  0.0),
    "yogurt greco":         ( 97.0,   9.0,  5.0,   36.0,  0.0),
    "mozzarella":           (254.0,  17.1, 19.9,  466.0,  0.0),
    "parmigiano":           (392.0,  33.0, 28.0,  614.0,  0.0),
    "grana padano":         (384.0,  33.0, 26.0,  650.0,  0.0),
    "ricotta":              (146.0,   9.5, 10.9,   84.0,  0.0),
    "burro":                (717.0,   0.9, 81.0,  576.0,  0.0),
    # Verdure
    "broccoli":             ( 34.0,   2.8,  0.4,   33.0,  2.6),
    "spinaci":              ( 23.0,   2.9,  0.4,   79.0,  2.2),
    "zucchine":             ( 17.0,   1.2,  0.3,    8.0,  1.0),
    "carote":               ( 41.0,   0.9,  0.2,   69.0,  2.8),
    "pomodori":             ( 18.0,   0.9,  0.2,    5.0,  1.2),
```

```
    "pomodoro":             ( 18.0,   0.9,  0.2,    5.0,  1.2),
    "insalata":             ( 15.0,   1.4,  0.2,   28.0,  1.3),
    "lattuga":              ( 15.0,   1.4,  0.2,   28.0,  1.3),
    "cetriolo":             ( 16.0,   0.7,  0.1,    2.0,  0.5),
    "peperone":             ( 31.0,   1.0,  0.3,    4.0,  2.1),
    "melanzane":            ( 25.0,   1.0,  0.2,    2.0,  3.0),
    "cipolla":              ( 40.0,   1.1,  0.1,    4.0,  1.7),
    "aglio":                (149.0,   6.4,  0.5,   17.0,  2.1),
    "patate":               ( 77.0,   2.0,  0.1,    6.0,  2.2),
    "funghi":               ( 22.0,   3.1,  0.3,    5.0,  1.0),
    "cavolfiore":           ( 25.0,   1.9,  0.3,   30.0,  2.0),
    "cavolo":               ( 25.0,   1.3,  0.1,   18.0,  2.5),
    "asparagi":             ( 20.0,   2.2,  0.1,    2.0,  2.1),
    "carciofi":             ( 47.0,   3.3,  0.2,   94.0,  5.4),
    # Frutta
    "mela":                 ( 52.0,   0.3,  0.2,    1.0,  2.4),
    "banana":               ( 89.0,   1.1,  0.3,    1.0,  2.6),
    "arancia":              ( 47.0,   0.9,  0.1,    0.0,  2.4),
    "fragole":              ( 32.0,   0.7,  0.3,    1.0,  2.0),
    "uva":                  ( 69.0,   0.7,  0.2,    2.0,  0.9),
    "kiwi":                 ( 61.0,   1.1,  0.5,    3.0,  3.0),
    "pera":                 ( 57.0,   0.4,  0.1,    1.0,  3.1),
    "pesca":                ( 39.0,   0.9,  0.3,    0.0,  1.5),
    "anguria":              ( 30.0,   0.6,  0.2,    1.0,  0.4),
    "limone":               ( 29.0,   1.1,  0.3,    2.0,  2.8),
    "avocado":              (160.0,   2.0, 14.7,    7.0,  6.7),
    # Grassi e condimenti
    "olio di oliva":        (884.0,   0.0, 100.0,   0.0,  0.0),
    "olio":                 (884.0,   0.0, 100.0,   0.0,  0.0),
    "olio evo":             (884.0,   0.0, 100.0,   0.0,  0.0),
    "sale":                 (  0.0,   0.0,  0.0, 38758.0, 0.0),
    "zucchero":             (387.0,   0.0,  0.0,    1.0,  0.0),
    "miele":                (304.0,   0.3,  0.0,    4.0,  0.2),
    "aceto":                ( 21.0,   0.0,  0.0,    5.0,  0.0),
    # Frutta secca e semi
    "mandorle":             (579.0,  21.2, 49.9,    1.0, 12.5),
    "noci":                 (654.0,  15.2, 65.2,    2.0,  6.7),
    "nocciole":             (628.0,  15.0, 60.8,    0.0,  9.7),
    "arachidi":             (567.0,  25.8, 49.2,   18.0,  8.5),
    "semi di chia":         (486.0,  16.5, 30.7,   16.0, 34.4),
    "semi di lino":         (534.0,  18.3, 42.2,   30.0, 27.3),
    "semi di girasole":     (584.0,  20.8, 51.5,    9.0,  8.6),
    # Altro
    "cioccolato fondente":  (598.0,   5.5, 42.6,   20.0, 10.9),
    "cioccolato":           (535.0,   7.8, 31.0,   79.0,  3.4),
    "nutella":              (539.0,   6.3, 30.9,   57.0,  3.4),
    "tofu":                 ( 76.0,   8.0,  4.8,    7.0,  0.3),
    "seitan":               (370.0,  75.0,  1.9,  120.0,  0.6),
}


def ceil2(value: float) -> float:
    """Arrotondamento per eccesso ai centesimi."""
    return math.ceil(value * 100) / 100


def lookup(name: str, qty_g: float) -> dict | None:
    """Cerca l'ingrediente nel DB e scala i valori alla quantita' indicata."""
    key = name.strip().lower()
    # Ricerca esatta
    if key in NUTRIENT_DB:
        data = NUTRIENT_DB[key]
        factor = qty_g / 100.0
        return {
            "name":     name.strip(),
            "quantity": qty_g,
            "calories": ceil2(data[0] * factor),
            "proteins": ceil2(data[1] * factor),
            "lipids":   ceil2(data[2] * factor),
            "sodium":   ceil2(data[3] * factor),
            "fibers":   ceil2(data[4] * factor),
        }
```

```
157     # Ricerca parziale (il nome del DB e' contenuto nell'input o viceversa)
158     for db_key, data in NUTRIENT_DB.items():
159         if db_key in key or key in db_key:
160             factor = qty_g / 100.0
161             return {
162                 "name":     name.strip(),
163                 "quantity": qty_g,
164                 "calories": ceil2(data[0] * factor),
165                 "proteins": ceil2(data[1] * factor),
166                 "lipids":   ceil2(data[2] * factor),
167                 "sodium":   ceil2(data[3] * factor),
168                 "fibers":   ceil2(data[4] * factor),
169             }
170     return None
171 
172 
173 # Main Application
174 class NutritionalCalculator(ctk.CTk):
175     def __init__(self):
176         super().__init__()
177         self.title("Calcolatore Nutrizionale")
178         self.geometry("1280x780")
179         self.minsize(1000, 600)
180         self.configure(fg_color="#13152a")
181 
182         # Each item: (row_frame, ing_entry, qty_entry)
183         self.ingredient_rows: list[tuple[ctk.CTkFrame, ctk.CTkEntry, ctk.CTkEntry]] = []
184 
185         self._build_layout()
186 
187     # Layout
188     def _build_layout(self):
189         self.grid_columnconfigure(0, weight=4, minsize=380)
190         self.grid_columnconfigure(1, weight=6, minsize=460)
191         self.grid_rowconfigure(0, weight=1)
192         self._build_left_panel()
193         self._build_right_panel()
194 
195     # Left panel
196     def _build_left_panel(self):
197         outer = ctk.CTkFrame(self, fg_color=COLORS["bg_panel"],
198                              corner_radius=16, border_width=1,
199                              border_color=COLORS["border"])
200         outer.grid(row=0, column=0, padx=(14, 7), pady=14, sticky="nsew")
201         outer.grid_rowconfigure(1, weight=1)
202         outer.grid_columnconfigure(0, weight=1)
203 
204         # Title
205         title_bar = ctk.CTkFrame(outer, fg_color="transparent")
206         title_bar.grid(row=0, column=0, sticky="ew", padx=18, pady=(18, 6))
207         ctk.CTkLabel(
208             title_bar, text="Ingredienti",
209             font=ctk.CTkFont("Segoe UI", 17, "bold"),
210             text_color=COLORS["text_primary"]
211         ).pack(side="left")
212         self.count_lbl = ctk.CTkLabel(
213             title_bar, text="0 / 15",
214             font=ctk.CTkFont("Segoe UI", 11),
215             text_color=COLORS["text_secondary"]
216         )
217         self.count_lbl.pack(side="right")
218 
219         # Scrollable list
220         self.scroll = ctk.CTkScrollableFrame(
221             outer, fg_color="transparent",
222             scrollbar_button_color=COLORS["accent"],
223             scrollbar_button_hover_color=COLORS["accent_hover"]
224         )
225         self.scroll.grid(row=1, column=0, sticky="nsew", padx=10, pady=4)
226         self.scroll.grid_columnconfigure(0, weight=0)   # numero
227         self.scroll.grid_columnconfigure(1, weight=3)   # ingrediente
228         self.scroll.grid_columnconfigure(2, weight=2)   # quantita'
```

```
229         self.scroll.grid_columnconfigure(3, weight=0)   # elimina
230
231         # Column headers
232         for col, txt in enumerate(["#", "Ingrediente", "Qta' (g)", ""]):
233             ctk.CTkLabel(
234                 self.scroll, text=txt,
235                 font=ctk.CTkFont("Segoe UI", 12, "bold"),
236                 text_color=COLORS["text_secondary"]
237             ).grid(row=0, column=col, padx=(6, 4), pady=(4, 8), sticky="w")
238
239         # Bottom buttons
240         bottom = ctk.CTkFrame(outer, fg_color="transparent")
241         bottom.grid(row=2, column=0, sticky="ew", padx=18, pady=(8, 18))
242         bottom.grid_columnconfigure(0, weight=1)
243         bottom.grid_columnconfigure(1, weight=1)
244
245         self.add_btn = ctk.CTkButton(
246             bottom, text="Aggiungi alimento",
247             font=ctk.CTkFont("Segoe UI", 13, "bold"),
248             height=40, corner_radius=10,
249             fg_color=COLORS["add_btn"],
250             hover_color=COLORS["add_btn_hov"],
251             command=self._add_row
252         )
253         self.add_btn.grid(row=0, column=0, sticky="ew", padx=(0, 5), pady=(0, 8))
254
255         self.calc_btn = ctk.CTkButton(
256             bottom, text="Calcola",
257             font=ctk.CTkFont("Segoe UI", 13, "bold"),
258             height=40, corner_radius=10,
259             fg_color=COLORS["accent"],
260             hover_color=COLORS["accent_hover"],
261             command=self._on_calculate
262         )
263         self.calc_btn.grid(row=0, column=1, sticky="ew", padx=(5, 0), pady=(0, 8))
264
265         self.status_lbl = ctk.CTkLabel(
266             bottom, text="Aggiungi ingredienti per iniziare.",
267             font=ctk.CTkFont("Segoe UI", 12),
268             text_color=COLORS["text_secondary"]
269         )
270         self.status_lbl.grid(row=1, column=0, columnspan=2)
271
272         # Start with one empty row
273         self._add_row()
274
275     def _add_row(self):
276         if len(self.ingredient_rows) >= 15:
277             self._set_status("Massimo 15 ingredienti raggiunto.", COLORS["warning"])
278             return
279
280         idx = len(self.ingredient_rows)
281         grid_row = idx + 1   # row 0 is the header
282
283         row_frame = ctk.CTkFrame(self.scroll, fg_color="transparent")
284         row_frame.grid(row=grid_row, column=0, columnspan=4,
285                        sticky="ew", padx=2, pady=2)
286         row_frame.grid_columnconfigure(0, weight=0)
287         row_frame.grid_columnconfigure(1, weight=3)
288         row_frame.grid_columnconfigure(2, weight=2)
289         row_frame.grid_columnconfigure(3, weight=0)
290
291         num_lbl = ctk.CTkLabel(
292             row_frame, text=f"{idx + 1:02d}",
293             font=ctk.CTkFont("Segoe UI", 12),
294             text_color=COLORS["text_secondary"], width=30
295         )
296         num_lbl.grid(row=0, column=0, padx=(4, 2), sticky="e")
297
298         ing_entry = ctk.CTkEntry(
299             row_frame,
300             placeholder_text="es. Riso, Pollo...",
```

```
            font=ctk.CTkFont("Segoe UI", 13),
            fg_color="#1e2133", border_color=COLORS["border"],
            text_color=COLORS["text_primary"],
            placeholder_text_color=COLORS["text_secondary"],
            border_width=1, corner_radius=8, height=34
        )
        ing_entry.grid(row=0, column=1, padx=4, sticky="ew")

        qty_entry = ctk.CTkEntry(
            row_frame,
            placeholder_text="g",
            font=ctk.CTkFont("Segoe UI", 13),
            fg_color="#1e2133", border_color=COLORS["border"],
            text_color=COLORS["text_primary"],
            placeholder_text_color=COLORS["text_secondary"],
            border_width=1, corner_radius=8, height=34, width=85
        )
        qty_entry.grid(row=0, column=2, padx=4, sticky="w")

        del_btn = ctk.CTkButton(
            row_frame, text="X", width=30, height=30,
            font=ctk.CTkFont("Segoe UI", 12, "bold"),
            fg_color=COLORS["del_btn"],
            hover_color=COLORS["del_btn_hov"],
            corner_radius=8,
            command=lambda rf=row_frame: self._remove_row(rf)
        )
        del_btn.grid(row=0, column=3, padx=(4, 2))

        self.ingredient_rows.append((row_frame, ing_entry, qty_entry))
        self._refresh_numbers()

        if len(self.ingredient_rows) >= 15:
            self.add_btn.configure(state="disabled")

    def _remove_row(self, row_frame: ctk.CTkFrame):
        for i, (rf, _, _) in enumerate(self.ingredient_rows):
            if rf is row_frame:
                self.ingredient_rows.pop(i)
                rf.destroy()
                break
        self.add_btn.configure(state="normal")
        self._refresh_numbers()
        self._set_status("", COLORS["text_secondary"])

    def _refresh_numbers(self):
        """Aggiorna i numeri progressivi dopo aggiunta/rimozione."""
        self.count_lbl.configure(text=f"{len(self.ingredient_rows)} / 15")
        for i, (rf, _, _) in enumerate(self.ingredient_rows):
            # Il primo widget figlio nella colonna 0 e' il num_lbl
            for child in rf.winfo_children():
                info = child.grid_info()
                if info.get("column") == 0:
                    child.configure(text=f"{i + 1:02d}")
                    break

    # Right panel
    def _build_right_panel(self):
        outer = ctk.CTkFrame(self, fg_color=COLORS["bg_panel"],
                             corner_radius=16, border_width=1,
                             border_color=COLORS["border"])
        outer.grid(row=0, column=1, padx=(7, 14), pady=14, sticky="nsew")
        outer.grid_rowconfigure(1, weight=1)
        outer.grid_columnconfigure(0, weight=1)

        title_bar = ctk.CTkFrame(outer, fg_color="transparent")
        title_bar.grid(row=0, column=0, sticky="ew", padx=18, pady=(18, 10))
        ctk.CTkLabel(
            title_bar, text="Valori Nutrizionali",
            font=ctk.CTkFont("Segoe UI", 17, "bold"),
            text_color=COLORS["text_primary"]
        ).pack(side="left")
```

```
373         ctk.CTkLabel(
374             title_bar, text="arrotondamento per eccesso ai centesimi",
375             font=ctk.CTkFont("Segoe UI", 10),
376             text_color=COLORS["text_secondary"]
377         ).pack(side="right")
378 
379         self.table_scroll = ctk.CTkScrollableFrame(
380             outer, fg_color="transparent",
381             scrollbar_button_color=COLORS["accent"],
382             scrollbar_button_hover_color=COLORS["accent_hover"]
383         )
384         self.table_scroll.grid(row=1, column=0, sticky="nsew", padx=12, pady=(0, 12))
385 
386         for c, w in enumerate([3, 1, 1, 1, 1, 1, 1]):
387             self.table_scroll.grid_columnconfigure(c, weight=w, minsize=70)
388 
389         self._draw_table_header()
390         self._draw_placeholder()
391 
392     def _draw_table_header(self):
393         headers = ["Ingrediente", "Qta' (g)",
394                    "Calorie\n(kcal)", "Proteine\n(g)",
395                    "Lipidi\n(g)", "Sodio\n(mg)", "Fibre\n(g)"]
396         for col, txt in enumerate(headers):
397             ctk.CTkLabel(
398                 self.table_scroll, text=txt,
399                 font=ctk.CTkFont("Segoe UI", 12, "bold"),
400                 fg_color=COLORS["bg_header"],
401                 text_color="white",
402                 corner_radius=6, justify="center"
403             ).grid(row=0, column=col, padx=1, pady=(0, 3),
404                    sticky="ew", ipadx=6, ipady=6)
405 
406     def _draw_placeholder(self):
407         self._placeholder = ctk.CTkLabel(
408             self.table_scroll,
409             text="Inserisci ingredienti e premi \"Calcola\"",
410             font=ctk.CTkFont("Segoe UI", 13),
411             text_color=COLORS["text_secondary"],
412             justify="center"
413         )
414         self._placeholder.grid(row=1, column=0, columnspan=7, pady=70, sticky="ew")
415 
416     # Calculation
417     def _on_calculate(self):
418         ingredients = []
419         not_found = []
420 
421         for _, ing_e, qty_e in self.ingredient_rows:
422             name = ing_e.get().strip()
423             qty  = qty_e.get().strip()
424             if not name and not qty:
425                 continue
426             if not name:
427                 self._set_status("Inserisci il nome di tutti gli ingredienti.", COLORS["
                        warning"])
428                 return
429             if not qty:
430                 self._set_status(f"Quantita' mancante per \"{name}\".", COLORS["warning"])
431                 return
432             try:
433                 qty_val = float(qty.replace(",", "."))
434             except ValueError:
435                 self._set_status(f"Quantita' non valida per \"{name}\".", COLORS["warning"
                        ])
436                 return
437             if qty_val <= 0:
438                 self._set_status(f"La quantita' di \"{name}\" deve essere > 0.", COLORS["
                        warning"])
439                 return
440 
441             result = lookup(name, qty_val)
```

```
            if result is None:
                not_found.append(name)
            else:
                ingredients.append(result)

        if not ingredients and not not_found:
            self._set_status("Inserisci almeno un ingrediente.", COLORS["warning"])
            return

        if not_found:
            missing = ", ".join(not_found)
            self._set_status(f"Non trovati nel DB: {missing}", COLORS["warning"])
            if not ingredients:
                return

        self._render_results(ingredients, not_found)

    # Rendering
    def _render_results(self, rows: list[dict], not_found: list[str]):
        # Clear previous table content (keep header row=0)
        for w in self.table_scroll.winfo_children():
            w.destroy()
        self._draw_table_header()

        totals = {"calories": 0.0, "proteins": 0.0,
                  "lipids": 0.0, "sodium": 0.0, "fibers": 0.0}

        for i, row in enumerate(rows):
            bg = COLORS["bg_row_even"] if i % 2 == 0 else COLORS["bg_row_odd"]
            for key in totals:
                totals[key] += row[key]

            values = [
                row["name"],
                f"{row['quantity']:.0f}",
                f"{row['calories']:.2f}",
                f"{row['proteins']:.2f}",
                f"{row['lipids']:.2f}",
                f"{row['sodium']:.2f}",
                f"{row['fibers']:.2f}",
            ]
            for col, val in enumerate(values):
                ctk.CTkLabel(
                    self.table_scroll, text=val,
                    font=ctk.CTkFont("Segoe UI", 12),
                    fg_color=bg,
                    text_color=COLORS["text_primary"],
                    corner_radius=0,
                    anchor="w" if col == 0 else "center",
                    justify="left" if col == 0 else "center"
                ).grid(row=i + 1, column=col, padx=1, pady=1,
                       sticky="ew", ipadx=8, ipady=5)

        # Ceil totals as well
        for key in totals:
            totals[key] = ceil2(totals[key])

        # Totals row
        total_row = len(rows) + 1
        total_vals = [
            "TOTALE",
            f"{totals['calories']:.2f}",
            f"{totals['proteins']:.2f}",
            f"{totals['lipids']:.2f}",
            f"{totals['sodium']:.2f}",
            f"{totals['fibers']:.2f}",
        ]
        for col, val in enumerate(total_vals):
            ctk.CTkLabel(
                self.table_scroll, text=val,
                font=ctk.CTkFont("Segoe UI", 13, "bold"),
                fg_color=COLORS["bg_total"],
```

```
                text_color="white",
                corner_radius=4,
                anchor="w" if col == 0 else "center",
                justify="left" if col == 0 else "center"
            ).grid(row=total_row, column=col, padx=1, pady=(5, 2),
                   sticky="ew", ipadx=8, ipady=7)

        msg = f"{len(rows)} ingredient{'e' if len(rows)==1 else 'i'} calcolat{'o' if len(
            rows)==1 else 'i'}."
        if not_found:
            msg += f"  Non trovati: {', '.join(not_found)}"
        self._set_status(msg, COLORS["success"] if not not_found else COLORS["warning"])

    # Helpers
    def _set_status(self, text: str, color: str):
        self.status_lbl.configure(text=text, text_color=color)


# Entry point
if __name__ == "__main__":
    app = NutritionalCalculator()
    app.mainloop()
```

Listing 4.5: Implementazione di un calcolatore nutrizionale

Spiegazione generica del codice

Nella prima parte del codice viene implementato il dizionario `NUTRIENT_DB`, i temi dell'applicazione e le librerie necessarie al suo funzionamento.

Successivamente, sono definite le due funzioni principali:

- `ceil2()`, che esegue l'arrotondamento per eccesso dei dati ottenuti;
- `lookup()`, che ricerca un alimento all'interno del database locale.

L'interfaccia dell'applicazione è costruita all'interno dei metodi `_build_left_panel()` e `_build_right_panel()`, assieme ai metodi per la gestione delle righe per l'inserimento degli alimenti. Il processo di acquisizione e verifica dei dati, invece, viene gestito dal metodo `_on_calculate()`. Infine, la rappresentazione grafica dei dati è gestita da `_render_results()`.

Terzo prompt

Utente

Correggimi i seguenti errori:

- mi deve essere mostrata anche la quantità totale nella colonna associata ad essa;
- quando cerco di effettuare il calcolo totale dei valori nutrizionali, ma per qualsiasi motivo (alimento non presente nel database, quantità inserita non valida, ecc) il procedimento di esso fallisce, i valori nutrizionali complessivi devono azzerarsi.

Inoltre, implementa il collegamento alla *USDA FoodData Central API* per l'estrazione dei valori nutrizionali degli alimenti inseriti, rimuovendo quindi il database interno creato in precedenza. La chiave API da utilizzare è `Hola3CZ5B8PwOBJx02Lay9b6ki9E3CZu4cGHbNxH`. Tutti i valori nutrizionali devono essere ottenuti esclusivamente dalla *USDA FoodData Central API*. Ordina i suggerimenti secondo il seguente criterio di priorità:

1. Foundation
2. SR Legacy
3. Survey (FNDDS)

4. Branded

Per ciascun gruppo privilegia gli alimenti semplici e generici. Ad esempio, quando l'utente ricerca `"chicken breast"`, i primi risultati mostrati dovranno essere:

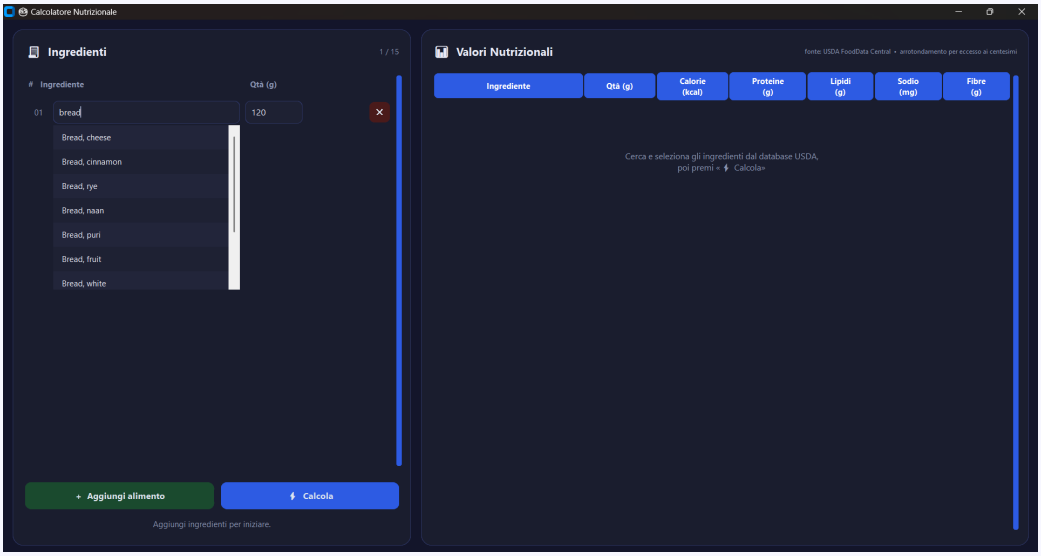
- Chicken breast, raw
- Chicken breast, roasted
- Chicken breast, grilled

Successivamente potranno essere mostrati:

- Rotisserie
- Breaded
- Commercial products

Mantieni comunque il valore originale di `fdcId` per il calcolo dei valori nutrizionali. Non mostrare all'utente la sorgente da cui provengono i dati (ad esempio *Foundation*, *SR Legacy*, *Survey* o *Branded*), ma esclusivamente la descrizione dell'alimento.

Claude



Sebbene il modello abbia implementato correttamente il collegamento all'API e la logica generale di ricerca, sono emerse alcune criticità sia dal punto di vista dell'interfaccia utente sia nella qualità dei risultati restituiti.

La lista dei suggerimenti presentava anomalie durante la navigazione: lo scorrimento tramite rotella del mouse non risultava correttamente gestito e, in alcuni casi, il tentativo di utilizzare la barra di scorrimento provocava la chiusura della lista dei risultati. Inoltre, durante l'esecuzione delle interrogazioni all'API non veniva fornito alcun riscontro visivo all'utente, rendendo poco chiaro se la ricerca fosse ancora in corso oppure conclusa.

Sono state, inoltre, riscontrate criticità nell'algoritmo di ricerca implementato. Per alcune query semplici, come "egg", il sistema restituiva risultati coerenti, mentre per altre, come "apple", i suggerimenti risultavano costituiti prevalentemente da prodotti commerciali appartenenti alla catena "Applebee's", anziché da alimenti generici come richiesto nelle specifiche del prompt. Analogamente, per alcuni termini di ricerca comuni il sistema non restituiva alcun suggerimento, evidenziando una gestione non ottimale delle interrogazioni verso la USDA FoodData Central API.

Le problematiche riscontrate hanno evidenziato che, sebbene il modello fosse riuscito a implementare l'integrazione con il servizio esterno, la logica di ordinamento e selezione dei risultati non rispettava completamente i criteri richiesti. Per questo motivo si è reso necessario formulare ulteriori prompt finalizzati al miglioramento dell'algoritmo di ricerca e dell'esperienza d'uso dell'interfaccia.

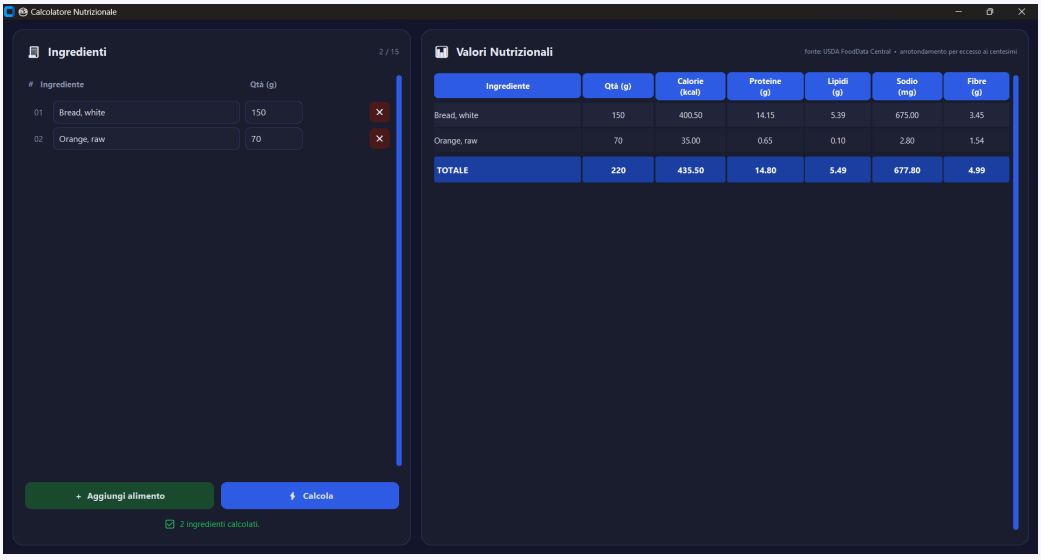
Quarto prompt

Utente

Apportami le seguenti modifiche:

- la rotella del mouse non funziona correttamente allo scorrimento
- mentre si stanno cercando gli elementi per popolare la lista dei risultati, ci deve essere un messaggio che riporta l'esecuzione della ricerca
- la lista dei suggerimenti viene popolata in modo strano; ad esempio "apple" mi da tutti alimenti della catena "Applebees" e non le mele normali, cosa che è totalmente l'opposto rispetto all'ordine che ti avevo chiesto prima di dare agli elementi che avrebbero popolato la lista dei suggerimenti. Inoltre, spesso, la ricerca di alimenti fallisce e non mi dà risultati anche per gli alimenti più semplici.

Claude



A differenza delle iterazioni precedenti, il modello non si è limitato a modificare il codice, ma ha fornito una spiegazione tecnica del comportamento osservato. In particolare, ha individuato come causa del problema il funzionamento del motore di ricerca della USDA FoodData Central API, basato su una ricerca full-text che opera su token completi e che, di conseguenza, non restituisce risultati affidabili per prefissi incompleti.

Per superare tale limitazione, il modello ha proposto l'introduzione di una strategia di ricerca basata su wildcard, accompagnata da un meccanismo di fallback automatico verso una ricerca tradizionale nel caso in cui la prima interrogazione non producesse risultati. Tale soluzione è stata implementata mantenendo invariata la logica di ordinamento dei suggerimenti, continuando a privilegiare gli alimenti generici rispetto ai prodotti commerciali.

Le prove di esecuzione hanno confermato il corretto funzionamento della nuova implementazione. Le ricerche tramite prefissi hanno iniziato a restituire suggerimenti coerenti con

l'input dell'utente, migliorando sia la completezza dei risultati sia l'esperienza di utilizzo dell'applicazione.

Claude ha dimostrato non solo la capacità di correggere il codice, ma anche di analizzare il problema, individuarne la causa tecnica e proporre una soluzione coerente con il funzionamento del servizio esterno utilizzato. Nel Listato 4.7 viene riportato il codice generato:

```
import math
import threading
import urllib.request
import urllib.parse
import json
import tkinter as tk
import customtkinter as ctk
from decimal import Decimal

# Theme
ctk.set_appearance_mode("dark")
ctk.set_default_color_theme("blue")

COLORS = {
    "bg_app":         "#13152a",
    "bg_panel":       "#1a1d2e",
    "bg_row_even":    "#22253a",
    "bg_row_odd":     "#1e2133",
    "bg_header":      "#2d5be3",
    "bg_total":       "#1b3fa0",
    "accent":         "#2d5be3",
    "accent_hover":   "#3d6ef3",
    "add_btn":        "#1a4a2e",
    "add_btn_hov":    "#215c38",
    "del_btn":        "#4a1a1a",
    "del_btn_hov":    "#6b2020",
    "success":        "#22c55e",
    "warning":        "#f59e0b",
    "error":          "#ef4444",
    "text_primary":   "#e8eaf6",
    "text_secondary":"#8892b0",
    "border":         "#2d3561",
    "dd_bg":          "#1e2133",
    "dd_even":        "#22253a",
    "dd_hover":       "#2d5be3",
}

# USDA FoodData Central
API_KEY  = "Hola3CZ5B8PwOBJx02Lay9b6ki9E3CZu4cGHbNxH"
BASE_URL = "https://api.nal.usda.gov/fdc/v1"

MAX_SUGGESTIONS = 10

NUTRIENT_IDS: dict[str, set] = {
    "calories": {1008, 208},
    "proteins": {1003, 203},
    "lipids":   {1004, 204},
    "sodium":   {1093, 307},
    "fibers":   {1079, 291},
}

SIMPLE_WORDS  = {"raw", "fresh", "plain", "whole", "cooked", "boiled", "steamed",
                 "roasted", "grilled", "baked", "dried", "skinless"}
COMPLEX_WORDS = {"rotisserie", "breaded", "commercial", "frozen", "restaurant",
                 "fast", "takeout", "ready-to-eat", "seasoned", "prepared", "flavored"}


def _simplicity_key(food: dict) -> tuple:
    """Within a data-type group, sort simpler/more generic foods first."""
    desc = food.get("description", "").lower()
    bad  =  sum(2 for w in COMPLEX_WORDS if w in desc)
    good = -sum(1 for w in SIMPLE_WORDS  if w in desc)
    return (bad + good, len(desc))
```

```
64
65
66  def _add_wildcard(text: str) -> str:
67      """
68      Append * to the last token for prefix matching.
69      'brea' -> 'brea*',  'chicken brea' -> 'chicken brea*'
70      Leaves the string unchanged if it already ends with a space
71      (last word is complete) or is too short.
72      """
73      stripped = text.strip()
74      if not stripped or stripped.endswith(" "):
75          return stripped
76      parts = stripped.split()
77      last  = parts[-1]
78      if len(last) >= 2 and not last.endswith("*"):
79          parts[-1] = last + "*"
80          return " ".join(parts)
81      return stripped
82
83
84  def _fetch_with_fallback(query: str,
85                           data_types: list[str],
86                           target: list[dict]) -> None:
87      """
88      Try the wildcard query first (prefix matching).
89      If that returns no results or raises an exception,
90      automatically retry with the original plain query.
91      This ensures 'brea' -> 'brea*' finds 'bread', 'breadcrumbs', etc.
92      while still working correctly when the wildcard form is unsupported.
93      """
94      query_wild = _add_wildcard(query)
95      attempts   = [query_wild, query] if query_wild != query else [query]
96
97      for q in attempts:
98          params = (
99              [("query", q), ("api_key", API_KEY), ("pageSize", "25")]
100             + [("dataType", dt) for dt in data_types]
101         )
102         url = f"{BASE_URL}/foods/search?" + urllib.parse.urlencode(params)
103         try:
104             with urllib.request.urlopen(url, timeout=10) as r:
105                 foods = json.loads(r.read()).get("foods", [])
106             if foods:
107                 target.extend(foods)
108                 return          # Got results - no need to try the next variant
109         except Exception:
110             continue            # Network / API error - try next variant
111
112
113 def usda_search(query: str) -> list[dict]:
114     """
115     Two parallel requests, kept strictly separate:
116         - Request 1 - Foundation + SR Legacy (scientific / generic foods)
117         - Request 2 - Survey (FNDDS) + Branded (survey meals & commercial products)
118
119     Foundation / SR Legacy results always appear first.
120     Each request tries a wildcard query first, with automatic plain-query fallback.
121     """
122     priority_foods: list[dict] = []
123     fallback_foods: list[dict] = []
124
125     t1 = threading.Thread(
126         target=_fetch_with_fallback,
127         args=(query, ["Foundation", "SR Legacy"], priority_foods),
128         daemon=True
129     )
130     t2 = threading.Thread(
131         target=_fetch_with_fallback,
132         args=(query, ["Survey (FNDDS)", "Branded"], fallback_foods),
133         daemon=True
134     )
135     t1.start(); t2.start()
```

```
    t1.join();  t2.join()

    top_priority = sorted(priority_foods, key=_simplicity_key)[:5]
    top_fallback = sorted(fallback_foods, key=_simplicity_key)[:5]

    # Merge keeping order: Foundation/SR Legacy first, then Survey/Branded
    seen: set[int] = set()
    merged: list[dict] = []
    for food in top_priority + top_fallback:
        fid = food.get("fdcId")
        if fid not in seen:
            seen.add(fid)
            merged.append(food)

    return merged[:MAX_SUGGESTIONS]


def usda_nutrients(fdc_id: int, qty_g: float) -> dict:
    url = f"{BASE_URL}/food/{fdc_id}?api_key={API_KEY}"
    with urllib.request.urlopen(url, timeout=10) as r:
        data = json.loads(r.read())
    raw = {k: 0.0 for k in NUTRIENT_IDS}
    for n in data.get("foodNutrients", []):
        nobj = n.get("nutrient") or {}
        nid  = nobj.get("id") or n.get("nutrientId") or n.get("nutrientNumber")
        amt  = float(n.get("amount") or n.get("value") or 0)
        for key, ids in NUTRIENT_IDS.items():
            if nid in ids:
                raw[key] = amt
                break
    f = qty_g / 100.0
    return {k: math.ceil(v * f * 100) / 100 for k, v in raw.items()}


def fmt_qty(value: float) -> str:
    return format(Decimal(str(value)).normalize(), "f")


# Dropdown
class Dropdown(tk.Toplevel):
    """
    Floating suggestion list.

    Scroll support:
      - Mac / Linux  : events go to the widget under the cursor, so we bind
                       <MouseWheel> / <Button-4> / <Button-5> directly on canvas,
                       inner frame, and every button.
      - Windows      : events go to the *focused* widget (the CTkEntry); the app
                       intercepts them in _redirect_scroll() and forwards them here
                       when the pointer is over this window.
    """
    ROW_H = 38

    def __init__(self, root: ctk.CTk, anchor: ctk.CTkEntry,
                 results: list[dict], on_select):
        super().__init__(root)
        self.overrideredirect(True)
        self.wm_attributes("-topmost", True)
        self.configure(bg=COLORS["border"])
        self._on_select = on_select
        self._canvas: tk.Canvas | None = None
        self._build(anchor, results)

    def _build(self, anchor: ctk.CTkEntry, results: list[dict]) -> None:
        anchor.update_idletasks()
        w = max(anchor.winfo_width(), 300)
        h = min(len(results) * self.ROW_H + 2, 304)
        x = anchor.winfo_rootx()
        y = anchor.winfo_rooty() + anchor.winfo_height() + 2
        self.geometry(f"{w}x{h}+{x}+{y}")

        canvas = tk.Canvas(self, bg=COLORS["dd_bg"], highlightthickness=0, bd=0)
```

```
208         self._canvas = canvas
209
210         if h >= 304:
211             sb = tk.Scrollbar(self, orient="vertical", command=canvas.yview)
212             canvas.configure(yscrollcommand=sb.set)
213             sb.pack(side="right", fill="y")
214         canvas.pack(fill="both", expand=True)
215
216         inner = tk.Frame(canvas, bg=COLORS["dd_bg"])
217         win   = canvas.create_window((0, 0), window=inner, anchor="nw")
218         inner.bind("<Configure>",
219                    lambda e: canvas.configure(scrollregion=canvas.bbox("all")))
220         canvas.bind("<Configure>",
221                     lambda e: canvas.itemconfigure(win, width=e.width))
222
223         # Mac / Linux scroll support: bind to canvas and inner frame
224         self._bind_wheel(canvas)
225         self._bind_wheel(inner)
226
227         for i, food in enumerate(results):
228             desc = food.get("description", "")
229             bg   = COLORS["dd_even"] if i % 2 == 0 else COLORS["dd_bg"]
230             btn  = tk.Button(
231                 inner, text=desc, anchor="w",
232                 bg=bg, fg=COLORS["text_primary"],
233                 activebackground=COLORS["dd_hover"], activeforeground="white",
234                 bd=0, padx=14, pady=8,
235                 font=("Segoe UI", 11), cursor="hand2",
236                 justify="left", wraplength=w - 28,
237                 command=lambda f=food: self._pick(f)
238             )
239             btn.pack(fill="x")
240             btn.bind("<Enter>", lambda e, b=btn:       b.configure(bg=COLORS["dd_hover"]))
241             btn.bind("<Leave>", lambda e, b=btn, c=bg: b.configure(bg=c))
242             self._bind_wheel(btn)  # Mac / Linux: bind to every button too
243
244     def _bind_wheel(self, widget: tk.BaseWidget) -> None:
245         """Bind mouse-wheel events on widget to scroll the canvas (Mac / Linux)."""
246         c = self._canvas
247
248         def _scroll_win(e):
249             c.yview_scroll(int(-1 * (e.delta / 120)), "units")
250             return "break"
251
252         def _scroll_up(e):
253             c.yview_scroll(-1, "units")
254             return "break"
255
256         def _scroll_down(e):
257             c.yview_scroll(1, "units")
258             return "break"
259
260         widget.bind("<MouseWheel>", _scroll_win)
261         widget.bind("<Button-4>",   _scroll_up)
262         widget.bind("<Button-5>",   _scroll_down)
263
264     def _pick(self, food: dict) -> None:
265         self._on_select(food)
266         try:
267             self.destroy()
268         except tk.TclError:
269             pass
270
271
272 # Main Application
273 class NutritionalCalculator(ctk.CTk):
274
275     def __init__(self):
276         super().__init__()
277         self.title("Calcolatore Nutrizionale")
278         self.geometry("1280x780")
279         self.minsize(1000, 600)
```

```
280         self.configure(fg_color=COLORS["bg_app"])
281         self.ingredient_rows: list[dict] = []
282         self._build_layout()
283         # Force-close all dropdowns on any click in the main window
284         self.bind("<Button-1>", lambda e: self._close_all_dropdowns(force=True))
285 
286     # Layout
287     def _build_layout(self) -> None:
288         self.grid_columnconfigure(0, weight=4, minsize=380)
289         self.grid_columnconfigure(1, weight=6, minsize=460)
290         self.grid_rowconfigure(0, weight=1)
291         self._build_left_panel()
292         self._build_right_panel()
293 
294     # Left panel
295     def _build_left_panel(self) -> None:
296         outer = ctk.CTkFrame(self, fg_color=COLORS["bg_panel"],
297                              corner_radius=16, border_width=1,
298                              border_color=COLORS["border"])
299         outer.grid(row=0, column=0, padx=(14, 7), pady=14, sticky="nsew")
300         outer.grid_rowconfigure(1, weight=1)
301         outer.grid_columnconfigure(0, weight=1)
302 
303         hdr = ctk.CTkFrame(outer, fg_color="transparent")
304         hdr.grid(row=0, column=0, sticky="ew", padx=18, pady=(18, 6))
305         ctk.CTkLabel(hdr, text="Ingredienti",
306                      font=ctk.CTkFont("Segoe UI", 17, "bold"),
307                      text_color=COLORS["text_primary"]).pack(side="left")
308         self.count_lbl = ctk.CTkLabel(hdr, text="0 / 15",
309                                       font=ctk.CTkFont("Segoe UI", 11),
310                                       text_color=COLORS["text_secondary"])
311         self.count_lbl.pack(side="right")
312 
313         self.scroll = ctk.CTkScrollableFrame(
314             outer, fg_color="transparent",
315             scrollbar_button_color=COLORS["accent"],
316             scrollbar_button_hover_color=COLORS["accent_hover"]
317         )
318         self.scroll.grid(row=1, column=0, sticky="nsew", padx=10, pady=4)
319         self.scroll.grid_columnconfigure(0, weight=0)
320         self.scroll.grid_columnconfigure(1, weight=3)
321         self.scroll.grid_columnconfigure(2, weight=2)
322         self.scroll.grid_columnconfigure(3, weight=0)
323 
324         for col, txt in enumerate(["#", "Ingrediente", "Qta' (g)", ""]):
325             ctk.CTkLabel(self.scroll, text=txt,
326                          font=ctk.CTkFont("Segoe UI", 12, "bold"),
327                          text_color=COLORS["text_secondary"]
328                          ).grid(row=0, column=col, padx=(6, 4), pady=(4, 8), sticky="w")
329 
330         btm = ctk.CTkFrame(outer, fg_color="transparent")
331         btm.grid(row=2, column=0, sticky="ew", padx=18, pady=(8, 18))
332         btm.grid_columnconfigure(0, weight=1)
333         btm.grid_columnconfigure(1, weight=1)
334 
335         self.add_btn = ctk.CTkButton(
336             btm, text="+  Aggiungi alimento",
337             font=ctk.CTkFont("Segoe UI", 13, "bold"),
338             height=40, corner_radius=10,
339             fg_color=COLORS["add_btn"], hover_color=COLORS["add_btn_hov"],
340             command=self._add_row
341         )
342         self.add_btn.grid(row=0, column=0, sticky="ew", padx=(0, 5), pady=(0, 8))
343 
344         self.calc_btn = ctk.CTkButton(
345             btm, text="Calcola",
346             font=ctk.CTkFont("Segoe UI", 13, "bold"),
347             height=40, corner_radius=10,
348             fg_color=COLORS["accent"], hover_color=COLORS["accent_hover"],
349             command=self._on_calculate
350         )
351         self.calc_btn.grid(row=0, column=1, sticky="ew", padx=(5, 0), pady=(0, 8))
```

```
352
353         self.status_lbl = ctk.CTkLabel(
354             btm, text="Aggiungi ingredienti per iniziare.",
355             font=ctk.CTkFont("Segoe UI", 12),
356             text_color=COLORS["text_secondary"]
357         )
358         self.status_lbl.grid(row=1, column=0, columnspan=2)
359
360         self._add_row()
361
362     # Row management
363     def _add_row(self) -> None:
364         if len(self.ingredient_rows) >= 15:
365             self._set_status("Massimo 15 ingredienti raggiunto.", COLORS["warning"])
366             return
367
368         idx = len(self.ingredient_rows)
369
370         row_frame = ctk.CTkFrame(self.scroll, fg_color="transparent")
371         row_frame.grid(row=idx + 1, column=0, columnspan=4,
372                        sticky="ew", padx=2, pady=2)
373         row_frame.grid_columnconfigure(0, weight=0)
374         row_frame.grid_columnconfigure(1, weight=3)
375         row_frame.grid_columnconfigure(2, weight=2)
376         row_frame.grid_columnconfigure(3, weight=0)
377
378         ctk.CTkLabel(row_frame, text=f"{idx + 1:02d}",
379                      font=ctk.CTkFont("Segoe UI", 12),
380                      text_color=COLORS["text_secondary"], width=30
381                      ).grid(row=0, column=0, padx=(4, 2), sticky="e")
382
383         ing_entry = ctk.CTkEntry(
384             row_frame, placeholder_text="Cerca alimento USDA",
385             font=ctk.CTkFont("Segoe UI", 13),
386             fg_color="#1e2133", border_color=COLORS["border"],
387             text_color=COLORS["text_primary"],
388             placeholder_text_color=COLORS["text_secondary"],
389             border_width=1, corner_radius=8, height=34
390         )
391         ing_entry.grid(row=0, column=1, padx=4, sticky="ew")
392
393         qty_entry = ctk.CTkEntry(
394             row_frame, placeholder_text="g",
395             font=ctk.CTkFont("Segoe UI", 13),
396             fg_color="#1e2133", border_color=COLORS["border"],
397             text_color=COLORS["text_primary"],
398             placeholder_text_color=COLORS["text_secondary"],
399             border_width=1, corner_radius=8, height=34, width=85
400         )
401         qty_entry.grid(row=0, column=2, padx=4, sticky="w")
402
403         row: dict = {
404             "frame":            row_frame,
405             "ing_entry":        ing_entry,
406             "qty_entry":        qty_entry,
407             "fdc_id":           None,
408             "confirmed_text":   "",
409             "debounce_job":     None,
410             "dropdown":         None,
411             "search_indicator": None,
412             "search_token":     0,   # incremented on every keystroke to cancel stale
                    results
413         }
414
415         del_btn = ctk.CTkButton(
416             row_frame, text="X", width=30, height=30,
417             font=ctk.CTkFont("Segoe UI", 12, "bold"),
418             fg_color=COLORS["del_btn"], hover_color=COLORS["del_btn_hov"],
419             corner_radius=8, command=lambda r=row: self._remove_row(r)
420         )
421         del_btn.grid(row=0, column=3, padx=(4, 2))
422
```

```
423         # Entry bindings
424         ing_entry.bind("<KeyRelease>", lambda e, r=row: self._on_key(r))
425         ing_entry.bind("<<Paste>>",    lambda e, r=row: self.after(50, lambda: self._on_key
                (r)))
426         ing_entry.bind("<Escape>",     lambda e, r=row: self._close_dropdown(r, force=True)
                )
427         ing_entry.bind("<Tab>",        lambda e, r=row: self._close_dropdown(r, force=True)
                )
428         ing_entry.bind("<FocusOut>",   lambda e, r=row: self.after(250,
429                                                            lambda: self._close_dropdown(r)))
430
431         # Windows: wheel events land on the focused entry; redirect them to
432         # the dropdown canvas whenever the pointer is inside the dropdown.
433         ing_entry.bind("<MouseWheel>",
434                        lambda e, r=row: self._redirect_scroll(r, e.delta, "win"))
435         ing_entry.bind("<Button-4>",
436                        lambda e, r=row: self._redirect_scroll(r, None, "up"))
437         ing_entry.bind("<Button-5>",
438                        lambda e, r=row: self._redirect_scroll(r, None, "down"))
439
440         self.ingredient_rows.append(row)
441         self._refresh_numbers()
442
443         if len(self.ingredient_rows) >= 15:
444             self.add_btn.configure(state="disabled")
445
446     def _remove_row(self, row: dict) -> None:
447         self._close_dropdown(row, force=True)
448         self._hide_search_indicator(row)
449         if row in self.ingredient_rows:
450             self.ingredient_rows.remove(row)
451         row["frame"].destroy()
452         self.add_btn.configure(state="normal")
453         self._refresh_numbers()
454         self._set_status("", COLORS["text_secondary"])
455
456     def _refresh_numbers(self) -> None:
457         self.count_lbl.configure(text=f"{len(self.ingredient_rows)} / 15")
458         for i, row in enumerate(self.ingredient_rows):
459             for child in row["frame"].winfo_children():
460                 if child.grid_info().get("column") == 0:
461                     child.configure(text=f"{i + 1:02d}")
462                     break
463
464     # Search
465     def _on_key(self, row: dict) -> None:
466         text = row["ing_entry"].get().strip()
467
468         # Invalidate confirmed selection when the user edits the field
469         if row["fdc_id"] and text != row["confirmed_text"]:
470             row["fdc_id"] = None
471             row["confirmed_text"] = ""
472
473         # Cancel previous debounce and advance token (invalidates running searches)
474         if row["debounce_job"]:
475             self.after_cancel(row["debounce_job"])
476             row["debounce_job"] = None
477         row["search_token"] += 1
478
479         if len(text) < 2:
480             self._close_dropdown(row, force=True)
481             self._hide_search_indicator(row)
482             return
483
484         row["debounce_job"] = self.after(420, lambda q=text: self._do_search(row, q))
485
486     def _do_search(self, row: dict, query: str) -> None:
487         token = row["search_token"]
488         self._show_search_indicator(row)   # <- "Ricerca in corso..."
489
490         def _bg() -> None:
491             try:
```

```
                results = usda_search(query)
            except Exception:
                results = []
            self.after(0, lambda r=results, t=token: self._on_search_done(row, r, t))

        threading.Thread(target=_bg, daemon=True).start()

    def _on_search_done(self, row: dict, results: list, token: int) -> None:
        if token != row["search_token"]:
            return  # Stale result from a superseded query - discard
        self._hide_search_indicator(row)
        if results:
            self._show_dropdown(row, results)

    # Search indicator
    def _show_search_indicator(self, row: dict) -> None:
        """Show a small 'Ricerca in corso...' banner below the entry."""
        self._hide_search_indicator(row)
        self._close_dropdown(row, force=True)
        ing = row["ing_entry"]
        try:
            ing.update_idletasks()
            x = ing.winfo_rootx()
            y = ing.winfo_rooty() + ing.winfo_height() + 2
            w = max(ing.winfo_width(), 220)

            ind = tk.Toplevel(self)
            ind.overrideredirect(True)
            ind.wm_attributes("-topmost", True)
            ind.configure(bg=COLORS["border"])
            ind.geometry(f"{w}x34+{x}+{y}")

            tk.Label(
                ind, text="Ricerca in corso...",
                bg=COLORS["dd_bg"], fg=COLORS["text_secondary"],
                font=("Segoe UI", 11), anchor="w"
            ).pack(fill="both", expand=True, padx=1, pady=1)

            row["search_indicator"] = ind
        except Exception:
            pass

    def _hide_search_indicator(self, row: dict) -> None:
        ind = row.get("search_indicator")
        if ind:
            try:
                ind.destroy()
            except tk.TclError:
                pass
            row["search_indicator"] = None

    # Dropdown management
    def _redirect_scroll(self, row: dict, delta, direction: str):
        """
        Windows fix: the focused entry receives wheel events even when the
        pointer is over the dropdown Toplevel. Forward those events to the
        dropdown canvas when the cursor is inside its bounds.
        """
        dd = row.get("dropdown")
        if not dd:
            return None
        try:
            if not dd.winfo_exists():
                return None
            px, py = self.winfo_pointerx(), self.winfo_pointery()
            dx, dy = dd.winfo_rootx(), dd.winfo_rooty()
            dw, dh = dd.winfo_width(), dd.winfo_height()
            if dx <= px < dx + dw and dy <= py < dy + dh:
                c = dd._canvas
                if direction == "win":
                    c.yview_scroll(int(-1 * (delta / 120)), "units")
                elif direction == "up":
```

```
564                     c.yview_scroll(-1, "units")
565                 else:
566                     c.yview_scroll(1, "units")
567                 return "break"
568         except tk.TclError:
569             pass
570         return None
571 
572     def _show_dropdown(self, row: dict, results: list) -> None:
573         self._close_dropdown(row, force=True)
574         if not results:
575             return
576         ing = row["ing_entry"]
577 
578         def on_select(food: dict) -> None:
579             row["fdc_id"]         = food["fdcId"]
580             row["confirmed_text"] = food["description"]
581             row["dropdown"]       = None   # clear before Toplevel.destroy()
582             ing.delete(0, "end")
583             ing.insert(0, food["description"])
584 
585         try:
586             row["dropdown"] = Dropdown(self, ing, results, on_select)
587         except Exception:
588             pass
589 
590     def _close_dropdown(self, row: dict, force: bool = False) -> None:
591         dd = row.get("dropdown")
592         if not dd:
593             return
594 
595         if not force:
596             # While the pointer is inside the dropdown, defer closing so that
597             # scrolling with the mouse wheel never dismisses the list.
598             try:
599                 if dd.winfo_exists():
600                     px, py = self.winfo_pointerx(), self.winfo_pointery()
601                     dx, dy = dd.winfo_rootx(), dd.winfo_rooty()
602                     dw, dh = dd.winfo_width(), dd.winfo_height()
603                     if dx <= px < dx + dw and dy <= py < dy + dh:
604                         # Re-check every 300 ms until the pointer leaves
605                         self.after(300, lambda r=row: self._close_dropdown(r))
606                         return
607             except tk.TclError:
608                 pass
609 
610         try:
611             dd.destroy()
612         except tk.TclError:
613             pass
614         row["dropdown"] = None
615 
616     def _close_all_dropdowns(self, force: bool = False) -> None:
617         for row in self.ingredient_rows:
618             self._close_dropdown(row, force=force)
619 
620     # Right panel
621     def _build_right_panel(self) -> None:
622         outer = ctk.CTkFrame(self, fg_color=COLORS["bg_panel"],
623                              corner_radius=16, border_width=1,
624                              border_color=COLORS["border"])
625         outer.grid(row=0, column=1, padx=(7, 14), pady=14, sticky="nsew")
626         outer.grid_rowconfigure(1, weight=1)
627         outer.grid_columnconfigure(0, weight=1)
628 
629         hdr = ctk.CTkFrame(outer, fg_color="transparent")
630         hdr.grid(row=0, column=0, sticky="ew", padx=18, pady=(18, 10))
631         ctk.CTkLabel(hdr, text="Valori Nutrizionali",
632                      font=ctk.CTkFont("Segoe UI", 17, "bold"),
633                      text_color=COLORS["text_primary"]).pack(side="left")
634         ctk.CTkLabel(
635             hdr,
```

```
636             text="fonte: USDA FoodData Central  -  arrotondamento per eccesso ai centesimi",
637             font=ctk.CTkFont("Segoe UI", 10),
638             text_color=COLORS["text_secondary"]
639         ).pack(side="right")
640 
641         self.table_scroll = ctk.CTkScrollableFrame(
642             outer, fg_color="transparent",
643             scrollbar_button_color=COLORS["accent"],
644             scrollbar_button_hover_color=COLORS["accent_hover"]
645         )
646         self.table_scroll.grid(row=1, column=0, sticky="nsew", padx=12, pady=(0, 12))
647 
648         for c, w in enumerate([3, 1, 1, 1, 1, 1, 1]):
649             self.table_scroll.grid_columnconfigure(c, weight=w, minsize=70)
650 
651         self._draw_table_header()
652         self._draw_placeholder()
653 
654     def _draw_table_header(self) -> None:
655         headers = ["Ingrediente", "Qta' (g)", "Calorie\n(kcal)", "Proteine\n(g)",
656                    "Lipidi\n(g)", "Sodio\n(mg)", "Fibre\n(g)"]
657         for col, txt in enumerate(headers):
658             ctk.CTkLabel(
659                 self.table_scroll, text=txt,
660                 font=ctk.CTkFont("Segoe UI", 12, "bold"),
661                 fg_color=COLORS["bg_header"], text_color="white",
662                 corner_radius=6, justify="center"
663             ).grid(row=0, column=col, padx=1, pady=(0, 3),
664                    sticky="ew", ipadx=6, ipady=6)
665 
666     def _draw_placeholder(self) -> None:
667         ctk.CTkLabel(
668             self.table_scroll,
669             text="Cerca e seleziona gli ingredienti dal database USDA,\npoi premi \"Calcola \"",
670             font=ctk.CTkFont("Segoe UI", 13),
671             text_color=COLORS["text_secondary"], justify="center"
672         ).grid(row=1, column=0, columnspan=7, pady=70, sticky="ew")
673 
674     def _reset_table(self) -> None:
675         for w in self.table_scroll.winfo_children():
676             w.destroy()
677         self._draw_table_header()
678         self._draw_placeholder()
679 
680     # Calculation
681     def _on_calculate(self) -> None:
682         self._close_all_dropdowns(force=True)
683         to_fetch: list[dict] = []
684 
685         for row in self.ingredient_rows:
686             name = row["ing_entry"].get().strip()
687             qty  = row["qty_entry"].get().strip()
688             if not name and not qty:
689                 continue
690             if not name:
691                 self._reset_table()
692                 self._set_status("Inserisci il nome di tutti gli ingredienti.",
693                                  COLORS["warning"])
694                 return
695             if not qty:
696                 self._reset_table()
697                 self._set_status(f"Quantita' mancante per \"{name}\".", COLORS["warning"])
698                 return
699             try:
700                 qty_val = float(qty.replace(",", "."))
701             except ValueError:
702                 self._reset_table()
703                 self._set_status(f"Quantita' non valida per \"{name}\".", COLORS["warning"])
704                 return
```

```
705             if qty_val <= 0:
706                 self._reset_table()
707                 self._set_status(f"La quantita' di \"{name}\" deve essere > 0.",
708                                  COLORS["warning"])
709                 return
710             if not row["fdc_id"]:
711                 self._reset_table()
712                 self._set_status(
713                     f"Seleziona \"{name}\" dall'elenco suggerimenti.", COLORS["warning"])
714                 return
715             to_fetch.append({"name": name, "quantity": qty_val, "fdc_id": row["fdc_id"]})
716 
717         if not to_fetch:
718             self._reset_table()
719             self._set_status("Inserisci almeno un ingrediente.", COLORS["warning"])
720             return
721 
722         self.calc_btn.configure(state="disabled", text="Recupero dati...")
723         self._set_status("Connessione a USDA FoodData Central...", COLORS["warning"])
724         threading.Thread(target=self._fetch_all, args=(to_fetch,), daemon=True).start()
725 
726     def _fetch_all(self, to_fetch: list[dict]) -> None:
727         results, errors = [], []
728         for item in to_fetch:
729             try:
730                 nuts = usda_nutrients(item["fdc_id"], item["quantity"])
731                 results.append({"name": item["name"], "quantity": item["quantity"], **nuts
                        })
732             except Exception:
733                 errors.append(item["name"])
734         self.after(0, lambda: self._finish(results, errors))
735 
736     def _finish(self, results: list, errors: list) -> None:
737         self.calc_btn.configure(state="normal", text="Calcola")
738         if errors:
739             self._reset_table()
740             self._set_status(
741                 f"Errore nel recupero dati per: {', '.join(errors)}", COLORS["error"])
742             return
743         self._render_results(results)
744 
745     # Rendering
746     def _render_results(self, rows: list[dict]) -> None:
747         for w in self.table_scroll.winfo_children():
748             w.destroy()
749         self._draw_table_header()
750 
751         totals = {k: 0.0 for k in
752                   ("quantity", "calories", "proteins", "lipids", "sodium", "fibers")}
753 
754         for i, row in enumerate(rows):
755             bg = COLORS["bg_row_even"] if i % 2 == 0 else COLORS["bg_row_odd"]
756             for key in totals:
757                 totals[key] += row[key]
758 
759             vals = [
760                 row["name"],                fmt_qty(row["quantity"]),
761                 f"{row['calories']:.2f}",   f"{row['proteins']:.2f}",
762                 f"{row['lipids']:.2f}",     f"{row['sodium']:.2f}",
763                 f"{row['fibers']:.2f}",
764             ]
765             for col, val in enumerate(vals):
766                 ctk.CTkLabel(
767                     self.table_scroll, text=val,
768                     font=ctk.CTkFont("Segoe UI", 12),
769                     fg_color=bg, text_color=COLORS["text_primary"],
770                     corner_radius=0,
771                     anchor="w" if col == 0 else "center",
772                     justify="left" if col == 0 else "center"
773                 ).grid(row=i + 1, column=col, padx=1, pady=1,
774                        sticky="ew", ipadx=8, ipady=5)
775 
```

```
        for key in totals:
            totals[key] = math.ceil(totals[key] * 100) / 100

        tv = [
            "TOTALE",                    fmt_qty(totals["quantity"]),
            f"{totals['calories']:.2f}", f"{totals['proteins']:.2f}",
            f"{totals['lipids']:.2f}",   f"{totals['sodium']:.2f}",
            f"{totals['fibers']:.2f}",
        ]
        for col, val in enumerate(tv):
            ctk.CTkLabel(
                self.table_scroll, text=val,
                font=ctk.CTkFont("Segoe UI", 13, "bold"),
                fg_color=COLORS["bg_total"], text_color="white",
                corner_radius=4,
                anchor="w" if col == 0 else "center",
                justify="left" if col == 0 else "center"
            ).grid(row=len(rows) + 1, column=col, padx=1, pady=(5, 2),
                   sticky="ew", ipadx=8, ipady=7)

        n = len(rows)
        self._set_status(
            f"{n} ingredient{'e' if n == 1 else 'i'} calcolat{'o' if n == 1 else 'i'}.",
            COLORS["success"]
        )

    # Helpers
    def _set_status(self, text: str, color: str) -> None:
        self.status_lbl.configure(text=text, text_color=color)


# Entry point
if __name__ == "__main__":
    app = NutritionalCalculator()
    app.mainloop()
```

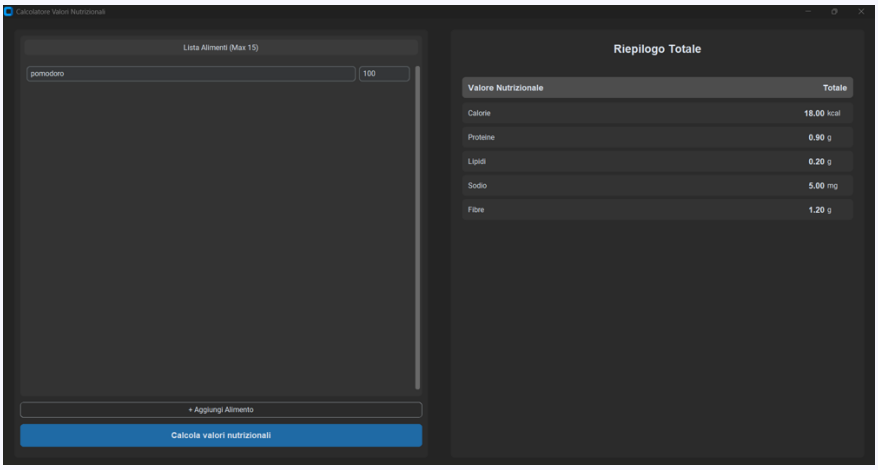
Listing 4.6: Implementazione di un calcolatore nutrizionale

4.3.4 Gemini

Prompt iniziale

In seguito, viene mostrata l'interfaccia restituita nel momento in cui viene eseguito il codice prodotto da *Gemini*.

Gemini



Le prove di esecuzione hanno evidenziato un'elevata robustezza nella gestione degli input. Il sistema verificava la correttezza del tipo di dato inserito, ignorava automaticamente le righe incomplete senza interrompere il calcolo complessivo e azzerava correttamente i risultati nel caso in cui non fosse presente alcun alimento valido. Inoltre, gli arrotondamenti venivano eseguiti correttamente e i risultati ottenuti erano coerenti con i valori presenti nel database interno creato dal modello.

Un ulteriore aspetto rilevante riguarda la gestione degli errori: qualora fossero stati inseriti alimenti non presenti nel database, l'applicazione continuava comunque ad elaborare gli ingredienti validi, informando l'utente mediante un apposito messaggio. Tale comportamento rendeva il sistema particolarmente robusto rispetto a input parzialmente errati. Nel Listato 4.7 viene riportato il codice generato:

```
import customtkinter as ctk
from tkinter import messagebox

# Database locale fittizio (Valori riferiti a 100g di prodotto)
# Per inserire un alimento reale, il nome digitato dovra' corrispondere a queste chiavi.
DATABASE_ALIMENTI = {
    "farina": {"calorie": 364.0, "proteine": 10.0, "lipidi": 1.0, "sodio": 2.0, "fibre": 2.7},
    "acqua": {"calorie": 0.0, "proteine": 0.0, "lipidi": 0.0, "sodio": 0.0, "fibre": 0.0},
    "lievito": {"calorie": 105.0, "proteine": 8.0, "lipidi": 1.9, "sodio": 51.0, "fibre": 5.4},
    "sale": {"calorie": 0.0, "proteine": 0.0, "lipidi": 0.0, "sodio": 38758.0, "fibre": 0.0},
    "pomodoro": {"calorie": 18.0, "proteine": 0.9, "lipidi": 0.2, "sodio": 5.0, "fibre": 1.2},
    "mozzarella": {"calorie": 280.0, "proteine": 28.0, "lipidi": 17.0, "sodio": 160.0, "fibre": 0.0},
    "mozzarella di bufala": {"calorie": 288.0, "proteine": 16.7, "lipidi": 24.4, "sodio": 16.0, "fibre": 0.0},
    "patate": {"calorie": 77.0, "proteine": 2.0, "lipidi": 0.1, "sodio": 6.0, "fibre": 2.2},
    "salsiccia": {"calorie": 346.0, "proteine": 14.0, "lipidi": 31.0, "sodio": 845.0, "fibre": 0.0},
    "mortadella": {"calorie": 288.0, "proteine": 15.7, "lipidi": 25.0, "sodio": 900.0, "fibre": 0.0},
}

class CalcolatoreNutrizionale(ctk.CTk):
    def __init__(self):
        super().__init__()

        self.title("Calcolatore Valori Nutrizionali")
        self.geometry("900x600")
        ctk.set_appearance_mode("dark")  # Opzioni: "system", "dark", "light"
        ctk.set_default_color_theme("blue")

        # Configurazione del layout a 2 colonne
        self.grid_columnconfigure(0, weight=1, uniform="col")
        self.grid_columnconfigure(1, weight=1, uniform="col")
        self.grid_rowconfigure(0, weight=1)

        self.ingredient_rows = []
        self.max_ingredients = 15

        self.crea_lato_sinistro()
        self.crea_lato_destro()

        # Aggiungo la prima riga vuota di default
        self.aggiungi_riga()

    def crea_lato_sinistro(self):
        # Container principale per la colonna sinistra
        self.left_container = ctk.CTkFrame(self)
        self.left_container.grid(row=0, column=0, padx=20, pady=20, sticky="nsew")
        self.left_container.grid_rowconfigure(0, weight=1)
        self.left_container.grid_columnconfigure(0, weight=1)
```

```
48
49          # Frame scrollabile per poter inserire fino a 15 ingredienti
50          self.scroll_frame = ctk.CTkScrollableFrame(self.left_container, label_text="Lista
                Alimenti (Max 15)")
51          self.scroll_frame.grid(row=0, column=0, padx=10, pady=10, sticky="nsew")
52
53          # Pulsante per aggiungere nuove righe
54          self.add_btn = ctk.CTkButton(self.left_container, text="+ Aggiungi Alimento",
                command=self.aggiungi_riga, fg_color="transparent", border_width=1)
55          self.add_btn.grid(row=1, column=0, padx=10, pady=(0, 10), sticky="ew")
56
57          # Pulsante finale per il calcolo
58          self.calc_btn = ctk.CTkButton(self.left_container, text="Calcola valori
                nutrizionali", command=self.calcola_valori, height=40, font=("Arial", 14, "bold
                "))
59          self.calc_btn.grid(row=2, column=0, padx=10, pady=(0, 20), sticky="ew")
60
61      def crea_lato_destro(self):
62          # Container principale per la colonna destra
63          self.right_frame = ctk.CTkFrame(self)
64          self.right_frame.grid(row=0, column=1, padx=20, pady=20, sticky="nsew")
65
66          titolo_tabella = ctk.CTkLabel(self.right_frame, text="Riepilogo Totale", font=("
                Arial", 20, "bold"))
67          titolo_tabella.pack(pady=(20, 30))
68
69          # Intestazione della Tabella
70          header_frame = ctk.CTkFrame(self.right_frame, fg_color=("gray80", "gray30"))
71          header_frame.pack(fill="x", padx=20, pady=5)
72          ctk.CTkLabel(header_frame, text="Valore Nutrizionale", width=150, anchor="w", font
                =("Arial", 14, "bold")).pack(side="left", padx=10, pady=5)
73          ctk.CTkLabel(header_frame, text="Totale", width=100, anchor="e", font=("Arial", 14,
                 "bold")).pack(side="right", padx=10, pady=5)
74
75          # Struttura dati per aggiornare facilmente i testi
76          self.risultati_vars = {
77              "Calorie": {"var": ctk.StringVar(value="0.00"), "unita": "kcal"},
78              "Proteine": {"var": ctk.StringVar(value="0.00"), "unita": "g"},
79              "Lipidi": {"var": ctk.StringVar(value="0.00"), "unita": "g"},
80              "Sodio": {"var": ctk.StringVar(value="0.00"), "unita": "mg"},
81              "Fibre": {"var": ctk.StringVar(value="0.00"), "unita": "g"}
82          }
83
84          # Creazione delle righe della tabella
85          for nutriente, info in self.risultati_vars.items():
86              row_frame = ctk.CTkFrame(self.right_frame)
87              row_frame.pack(fill="x", padx=20, pady=2)
88
89              # Etichetta Nutriente (es. Calorie)
90              ctk.CTkLabel(row_frame, text=nutriente, width=150, anchor="w").pack(side="left"
                    , padx=10, pady=5)
91
92              # Frame per Valore + Unita' di misura allineato a destra
93              val_frame = ctk.CTkFrame(row_frame, fg_color="transparent")
94              val_frame.pack(side="right", padx=10)
95
96              ctk.CTkLabel(val_frame, textvariable=info["var"], width=60, anchor="e", font=("
                    Arial", 14, "bold")).pack(side="left")
97              ctk.CTkLabel(val_frame, text=info["unita"], width=35, anchor="w").pack(side="
                    left", padx=(5, 0))
98
99      def aggiungi_riga(self):
100         if len(self.ingredient_rows) >= self.max_ingredients:
101             return
102
103         row_frame = ctk.CTkFrame(self.scroll_frame, fg_color="transparent")
104         row_frame.pack(fill="x", pady=5, padx=5)
105
106         ing_entry = ctk.CTkEntry(row_frame, placeholder_text="Ingrediente")
107         ing_entry.pack(side="left", fill="x", expand=True, padx=(0, 5))
108
109         qty_entry = ctk.CTkEntry(row_frame, placeholder_text="Quantita' (g)", width=90)
```

```
110         qty_entry.pack(side="left")
111
112         # Salviamo i riferimenti alle entry widget
113         self.ingredient_rows.append((ing_entry, qty_entry))
114
115         # Se abbiamo raggiunto 15 ingredienti, disabilitiamo il pulsante
116         if len(self.ingredient_rows) >= self.max_ingredients:
117             self.add_btn.configure(state="disabled", text="Limite 15 raggiunto")
118
119     def calcola_valori(self):
120         totali = {
121             "Calorie": 0.0,
122             "Proteine": 0.0,
123             "Lipidi": 0.0,
124             "Sodio": 0.0,
125             "Fibre": 0.0
126         }
127
128         ingredienti_non_trovati = []
129
130         for ing_entry, qty_entry in self.ingredient_rows:
131             nome = ing_entry.get().strip().lower()
132             qty_str = qty_entry.get().strip()
133
134             if not nome or not qty_str:
135                 continue
136
137             try:
138                 # Sostituiamo eventuali virgole con il punto per i decimali
139                 quantita = float(qty_str.replace(',', '.'))
140             except ValueError:
141                 messagebox.showwarning("Errore di Inserimento", f"La quantita' inserita per
                        '{nome}' non e' un numero valido.")
142                 return
143
144             if nome in DATABASE_ALIMENTI:
145                 valori = DATABASE_ALIMENTI[nome]
146                 fattore = quantita / 100.0 # Valori base sono calcolati su 100g
147
148                 totali["Calorie"] += valori["calorie"] * fattore
149                 totali["Proteine"] += valori["proteine"] * fattore
150                 totali["Lipidi"] += valori["lipidi"] * fattore
151                 totali["Sodio"] += valori["sodio"] * fattore
152                 totali["Fibre"] += valori["fibre"] * fattore
153             else:
154                 ingredienti_non_trovati.append(nome)
155
156         # Aggiornamento Variabili dell'interfaccia
157         for nutriente, valore_totale in totali.items():
158             self.risultati_vars[nutriente]["var"].set(f"{valore_totale:.2f}")
159
160         # Se qualche alimento non e' presente nel nostro "Database", avvisa l'utente
161         if ingredienti_non_trovati:
162             lista_err = ", ".join(ingredienti_non_trovati)
163             messagebox.showinfo("Alimenti mancanti", f"I seguenti alimenti non sono
                    presenti nel database e sono stati ignorati:\n\n{lista_err}")
164
165 if __name__ == "__main__":
166     app = CalcolatoreNutrizionale()
167     app.mainloop()
```

Listing 4.7: Implementazione di un calcolatore nutrizionale

Spiegazione generica del codice

All'inizio viene definito il dizionario `DATABASE_ALIMENTI` che fungerà da database locale. Al suo interno vi sono contenuti un numero limitato di alimenti con i relativi valori nutrizionali riferiti a 100 grammi di prodotto.

La creazione della finestra principale è gestita dai due metodi `crea_lato_sinistro()`, dove l'utente può inserire gli ingredienti e la quantità espressa in grammi, e `crea_lato_destro()`, dove viene visualizzato un riepilogo dei valori nutrizionali complessivi, comprendente calorie, proteine, lipidi, sodio e fibre. Il modello ha implementato in automatico il metodo `aggiungi_riga()`, che fornisce la possibilità di aggiungere dinamicamente nuovi ingredienti.

Il metodo `calcola_valori()` esegue il calcolo dei valori nutrizionali cercando ogni alimento all'interno del database locale. Poiché tutti i dati sono espressi per 100 grammi di prodotto, il software calcola un fattore di proporzione in base alla quantità inserita dall'utente e utilizza tale fattore per determinare il contributo di ciascun ingrediente ai valori nutrizionali complessivi della ricetta.

Secondo prompt

Utente

Implementa il collegamento alla *USDA FoodData Central API* per l'estrazione dei valori nutrizionali degli alimenti inseriti, rimuovendo quindi il database interno creato in precedenza. La chiave API da utilizzare è `Hola3CZ5B8PwOBJx02Lay9b6ki9E3CZu4cGHbNxH`. Tutti i valori nutrizionali devono essere ottenuti esclusivamente dalla *USDA FoodData Central API*.
Ordina i suggerimenti secondo il seguente criterio di priorità:

1. Foundation
2. SR Legacy
3. Survey (FNDDS)
4. Branded

Per ciascun gruppo privilegia gli alimenti semplici e generici. Ad esempio, quando l'utente ricerca `"chicken breast"`, i primi risultati mostrati dovranno essere:

- Chicken breast, raw
- Chicken breast, roasted
- Chicken breast, grilled

Successivamente potranno essere mostrati:

- Rotisserie
- Breaded
- Commercial products

Mantieni comunque il valore originale di `fdcId` per il calcolo dei valori nutrizionali. Non mostrare all'utente la sorgente da cui provengono i dati (ad esempio *Foundation*, *SR Legacy*, *Survey* o *Branded*), ma esclusivamente la descrizione dell'alimento.

Il modello ha implementato correttamente il collegamento con l'API USDA, sostituendo il database locale con un sistema di interrogazione in tempo reale. È stato, inoltre, introdotto un meccanismo di autocompletamento che consente all'utente di selezionare direttamente l'alimento desiderato tra i risultati restituiti dalla ricerca.

Le prove effettuate hanno mostrato un funzionamento complessivamente corretto dell'integrazione, con una gestione efficace dei suggerimenti. Tuttavia, permanevano alcune limitazioni nell'interfaccia grafica: non era ancora presente il pulsante per la rimozione delle righe e la lista dei suggerimenti non supportava lo scorrimento tramite la rotella del mouse,

rendendo meno agevole la consultazione di elenchi particolarmente lunghi. Nel Listato 4.8 viene riportato il codice generato:

```
import customtkinter as ctk
from tkinter import messagebox
import requests
import threading

API_KEY = "Hola3CZ5B8PwOBJx02Lay9b6ki9E3CZu4cGHbNxH"
USDA_URL = "https://api.nal.usda.gov/fdc/v1/foods/search"

class CalcolatoreNutrizionaleUSDA(ctk.CTk):
    def __init__(self):
        super().__init__()

        self.title("Calcolatore Valori Nutrizionali - USDA FoodData Central")
        self.geometry("1000x650")
        ctk.set_appearance_mode("dark")
        ctk.set_default_color_theme("blue")

        self.grid_columnconfigure(0, weight=1, uniform="col")
        self.grid_columnconfigure(1, weight=1, uniform="col")
        self.grid_rowconfigure(0, weight=1)

        self.ingredient_rows = []
        self.max_ingredients = 15

        self.crea_lato_sinistro()
        self.crea_lato_destro()

        # Inizializza con la prima riga vuota
        self.aggiungi_riga()

    def crea_lato_sinistro(self):
        self.left_container = ctk.CTkFrame(self)
        self.left_container.grid(row=0, column=0, padx=20, pady=20, sticky="nsew")
        self.left_container.grid_rowconfigure(0, weight=1)
        self.left_container.grid_columnconfigure(0, weight=1)

        self.scroll_frame = ctk.CTkScrollableFrame(self.left_container, label_text="Lista
            Alimenti (Cerca in Inglese - Max 15)")
        self.scroll_frame.grid(row=0, column=0, padx=10, pady=10, sticky="nsew")

        self.add_btn = ctk.CTkButton(self.left_container, text="+ Aggiungi Riga", command=
            self.aggiungi_riga, fg_color="transparent", border_width=1)
        self.add_btn.grid(row=1, column=0, padx=10, pady=(0, 10), sticky="ew")

        self.calc_btn = ctk.CTkButton(self.left_container, text="Calcola valori
            nutrizionali", command=self.calcola_valori, height=40, font=("Arial", 14, "bold
            "))
        self.calc_btn.grid(row=2, column=0, padx=10, pady=(0, 20), sticky="ew")

    def crea_lato_destro(self):
        self.right_frame = ctk.CTkFrame(self)
        self.right_frame.grid(row=0, column=1, padx=20, pady=20, sticky="nsew")

        titolo_tabella = ctk.CTkLabel(self.right_frame, text="Riepilogo Totale (USDA)",
            font=("Arial", 20, "bold"))
        titolo_tabella.pack(pady=(20, 30))

        header_frame = ctk.CTkFrame(self.right_frame, fg_color=("gray80", "gray30"))
        header_frame.pack(fill="x", padx=20, pady=5)
        ctk.CTkLabel(header_frame, text="Valore Nutrizionale", width=150, anchor="w", font
            =("Arial", 14, "bold")).pack(side="left", padx=10, pady=5)
        ctk.CTkLabel(header_frame, text="Totale", width=100, anchor="e", font=("Arial", 14,
             "bold")).pack(side="right", padx=10, pady=5)

        self.risultati_vars = {
            "Calorie": {"var": ctk.StringVar(value="0.00"), "unita": "kcal"},
            "Proteine": {"var": ctk.StringVar(value="0.00"), "unita": "g"},
            "Lipidi": {"var": ctk.StringVar(value="0.00"), "unita": "g"},
            "Sodio": {"var": ctk.StringVar(value="0.00"), "unita": "mg"},
```

```
63              "Fibre": {"var": ctk.StringVar(value="0.00"), "unita": "g"}
64          }
65
66          for nutriente, info in self.risultati_vars.items():
67              row_frame = ctk.CTkFrame(self.right_frame)
68              row_frame.pack(fill="x", padx=20, pady=2)
69
70              ctk.CTkLabel(row_frame, text=nutriente, width=150, anchor="w").pack(side="left"
                    , padx=10, pady=5)
71
72              val_frame = ctk.CTkFrame(row_frame, fg_color="transparent")
73              val_frame.pack(side="right", padx=10)
74
75              ctk.CTkLabel(val_frame, textvariable=info["var"], width=60, anchor="e", font=("
                    Arial", 14, "bold")).pack(side="left")
76              ctk.CTkLabel(val_frame, text=info["unita"], width=35, anchor="w").pack(side="
                    left", padx=(5, 0))
77
78      def aggiungi_riga(self):
79          if len(self.ingredient_rows) >= self.max_ingredients:
80              return
81
82          row_frame = ctk.CTkFrame(self.scroll_frame, fg_color="transparent")
83          row_frame.pack(fill="x", pady=5, padx=5)
84
85          # 1. Input di Ricerca
86          search_entry = ctk.CTkEntry(row_frame, placeholder_text="Es. Chicken...", width
                =110)
87          search_entry.pack(side="left", padx=(0, 5))
88
89          # 2. Pulsante per avviare la ricerca
90          search_btn = ctk.CTkButton(row_frame, text="Cerca", width=30)
91          search_btn.pack(side="left", padx=(0, 5))
92
93          # 3. Menu a tendina per visualizzare solo le descrizioni (I risultati)
94          combo = ctk.CTkComboBox(row_frame, values=["---"], state="disabled")
95          combo.pack(side="left", fill="x", expand=True, padx=(0, 5))
96
97          # 4. Input Quantita'
98          qty_entry = ctk.CTkEntry(row_frame, placeholder_text="g", width=60)
99          qty_entry.pack(side="left")
100
101         # Dizionario di gestione per questa specifica riga
102         row_data = {
103             "search_entry": search_entry,
104             "search_btn": search_btn,
105             "combo": combo,
106             "qty_entry": qty_entry,
107             "dati_usda": {}  # Qui memorizziamo fdcId e nutrienti estratti
108         }
109
110         # Assegna i comandi per la ricerca tramite bottone o tasto Invio
111         search_btn.configure(command=lambda r=row_data: self.avvia_ricerca_api(r))
112         search_entry.bind("<Return>", lambda event, r=row_data: self.avvia_ricerca_api(r))
113
114         self.ingredient_rows.append(row_data)
115
116         if len(self.ingredient_rows) >= self.max_ingredients:
117             self.add_btn.configure(state="disabled", text="Limite 15 raggiunto")
118
119     # --- INTEGRAZIONE API USDA ---
120     def avvia_ricerca_api(self, row_data):
121         query = row_data["search_entry"].get().strip()
122         if not query:
123             return
124
125         # Disabilita UI temporaneamente per far capire all'utente che sta caricando
126         row_data["search_btn"].configure(state="disabled")
127         row_data["combo"].configure(state="normal", values=["Ricerca in corso..."])
128         row_data["combo"].set("Ricerca in corso...")
129
130         # Eseguiamo la richiesta web in un thread separato per non freezare l'app
```

```
131         threading.Thread(target=self.chiama_usda, args=(query, row_data), daemon=True).
                start()
132
133     def chiama_usda(self, query, row_data):
134         params = {
135             "api_key": API_KEY,
136             "query": query,
137             "pageSize": 50 # Prendiamo un po' di risultati per poter applicare l'
                    ordinamento avanzato
138         }
139         try:
140             risposta = requests.get(USDA_URL, params=params, timeout=10)
141             risposta.raise_for_status()
142             foods = risposta.json().get("foods", [])
143         except requests.RequestException as e:
144             self.after(0, lambda: self.gestisci_errore_api(row_data, str(e)))
145             return
146
147         # LOGICA DI ORDINAMENTO RICHIESTA DALL'UTENTE
148         # 1. Foundation, 2. SR Legacy, 3. Survey, 4. Branded
149         # A parita' di categoria, predilige lunghezze descrittive piu' corte (alimenti
                semplici e puri)
150         priorita_tipo = {
151             "Foundation": 1,
152             "SR Legacy": 2,
153             "Survey (FNDDS)": 3,
154             "Branded": 4
155         }
156
157         foods_ordinati = sorted(foods, key=lambda f: (
158             priorita_tipo.get(f.get("dataType", "Branded"), 5),
159             len(f.get("description", ""))
160         ))
161
162         nuovi_dati_usda = {}
163         combo_values = []
164
165         for f in foods_ordinati:
166             desc = f.get("description", "").capitalize()
167
168             # Evitare descrizioni doppie identiche nel menu a tendina
169             original_desc = desc
170             counter = 1
171             while desc in nuovi_dati_usda:
172                 desc = f"{original_desc} ({counter})"
173                 counter += 1
174
175             # SALVATAGGIO DEI DATI "DIETRO LE QUINTE"
176             # Manteniamo il fdcId in memoria come richiesto, pronto per essere letto al
                    bisogno
177             nuovi_dati_usda[desc] = {
178                 "fdcId": f.get("fdcId"),
179                 "nutrienti": self.estrai_nutrienti(f.get("foodNutrients", []))
180             }
181             combo_values.append(desc)
182
183         # Aggiorniamo la UI tornando sul Main Thread
184         self.after(0, lambda: self.popola_combo_risultati(row_data, combo_values,
                nuovi_dati_usda))
185
186     def estrai_nutrienti(self, food_nutrients):
187         """Mappa gli ID nutrizionali standard della USDA ai nostri macronutrienti."""
188         valori = {"Calorie": 0.0, "Proteine": 0.0, "Lipidi": 0.0, "Sodio": 0.0, "Fibre":
                0.0}
189
190         for nut in food_nutrients:
191             nid = nut.get("nutrientId")
192             val = float(nut.get("value", 0.0))
193
194             if nid == 1008: valori["Calorie"] = val             # Energy (kcal)
195             elif nid == 1003: valori["Proteine"] = val          # Protein
196             elif nid == 1004: valori["Lipidi"] = val            # Total lipid (fat)
```

```
197             elif nid == 1093: valori["Sodio"] = val             # Sodium, Na
198             elif nid == 1079: valori["Fibre"] = val             # Fiber, total dietary
199
200         return valori
201
202     def popola_combo_risultati(self, row_data, combo_values, nuovi_dati_usda):
203         row_data["search_btn"].configure(state="normal")
204         row_data["dati_usda"] = nuovi_dati_usda
205
206         if combo_values:
207             row_data["combo"].configure(values=combo_values, state="normal")
208             row_data["combo"].set(combo_values[0])
209         else:
210             row_data["combo"].configure(values=["Nessun risultato trovato"], state="
                disabled")
211             row_data["combo"].set("Nessun risultato trovato")
212
213     def gestisci_errore_api(self, row_data, errore):
214         row_data["search_btn"].configure(state="normal")
215         row_data["combo"].configure(values=["Errore di connessione"], state="disabled")
216         row_data["combo"].set("Errore API USDA")
217         messagebox.showerror("Errore Rete", f"Impossibile collegarsi alla USDA FoodData
                Central:\n{errore}")
218
219     # --- CALCOLO FINALE ---
220     def calcola_valori(self):
221         totali = { "Calorie": 0.0, "Proteine": 0.0, "Lipidi": 0.0, "Sodio": 0.0, "Fibre":
                0.0 }
222
223         for indice, row_data in enumerate(self.ingredient_rows):
224             ricerca = row_data["search_entry"].get().strip()
225             selezione = row_data["combo"].get()
226             qty_str = row_data["qty_entry"].get().strip()
227
228             # Salta le righe non compilate in nessun campo
229             if not ricerca and not qty_str and (selezione == "---" or selezione == ""):
230                 continue
231
232             # Gestione Errori per i campi parziali
233             if not ricerca or selezione in ("---", "Nessun risultato trovato", "Ricerca in
                    corso..."):
234                 if qty_str:
235                     messagebox.showwarning("Dato Mancante", f"Alla riga {indice + 1} hai
                            inserito una quantita' ({qty_str}g) ma non hai cercato e
                            selezionato l'ingrediente.")
236                     return
237                 continue
238
239             if not qty_str:
240                 messagebox.showwarning("Dato Mancante", f"Alla riga {indice + 1} hai
                        selezionato '{selezione}' ma manca la quantita'.")
241                 return
242
243             try:
244                 quantita = float(qty_str.replace(',', '.'))
245             except ValueError:
246                 messagebox.showwarning("Errore di Inserimento", f"La quantita' inserita
                        alla riga {indice + 1} non e' un numero valido.")
247                 return
248
249             # ESTRAZIONE DATI MEMORIZZATI NEL DIZIONARIO
250             dati_alimento = row_data["dati_usda"].get(selezione)
251             if not dati_alimento:
252                 continue
253
254             # N.B. I dati dell'API USDA FoodData Central Search sono sempre parametrizzati
                    per 100g di prodotto
255             nutrienti = dati_alimento["nutrienti"]
256
257             # fdc_id = dati_alimento["fdcId"] # -> Il FDC ID e' qui disponibile come
                    richiesto
258
```

```
259             fattore = quantita / 100.0
260
261             totali["Calorie"] += nutrienti["Calorie"] * fattore
262             totali["Proteine"] += nutrienti["Proteine"] * fattore
263             totali["Lipidi"] += nutrienti["Lipidi"] * fattore
264             totali["Sodio"] += nutrienti["Sodio"] * fattore
265             totali["Fibre"] += nutrienti["Fibre"] * fattore
266
267         # Aggiornamento UI della parte destra
268         for nutriente, valore_totale in totali.items():
269             self.risultati_vars[nutriente]["var"].set(f"{valore_totale:.2f}")
270
271 if __name__ == "__main__":
272     app = CalcolatoreNutrizionaleUSDA()
273     app.mainloop()
```

Listing 4.8: Implementazione di un calcolatore nutrizionale

Spiegazione generica del codice

Una delle principali novità introdotte dal modello riguarda la gestione della comunicazione con il servizio remoto. Per evitare che l'interfaccia si blocchi durante l'attesa della risposta del server, le richieste *HTTP* vengono eseguite all'interno di un thread separato.

Il metodo `avvia_ricerca_api()` presenta il punto di ingresso dell'intero flusso. Oltre a verificare che sia stata effettivamente inserita una chiave di ricerca, la funzione aggiorna temporaneamente l'interfaccia grafica, informando l'utente che la richiesta è in fase di elaborazione, e avvia un nuovo thread per la comunicazione con il server.

`chiama_usda()` si occupa della comunicazione con il server. Inizialmente, esso invia una richiesta *HTTP GET* contenente il termine di ricerca inserito dall'utente. Una volta ricevuta la risposta, la funzione estrae l'elenco degli alimenti restituiti dall'API, applica un ordinamento alla lista e prepara le informazioni necessarie per le fasi successive.

L'estrazione dei nutrienti è affidata al metodo `estrai_nutrienti()`, che riceve l'elenco completo dei nutrienti associati ad un alimento e seleziona esclusivamente quelli utilizzati dall'applicazione. Infine, il metodo `popola_combo_risultati()` aggiorna l'interfaccia grafica al termine delle operazioni.

Terzo prompt

Utente

Apportami le seguenti modifiche:

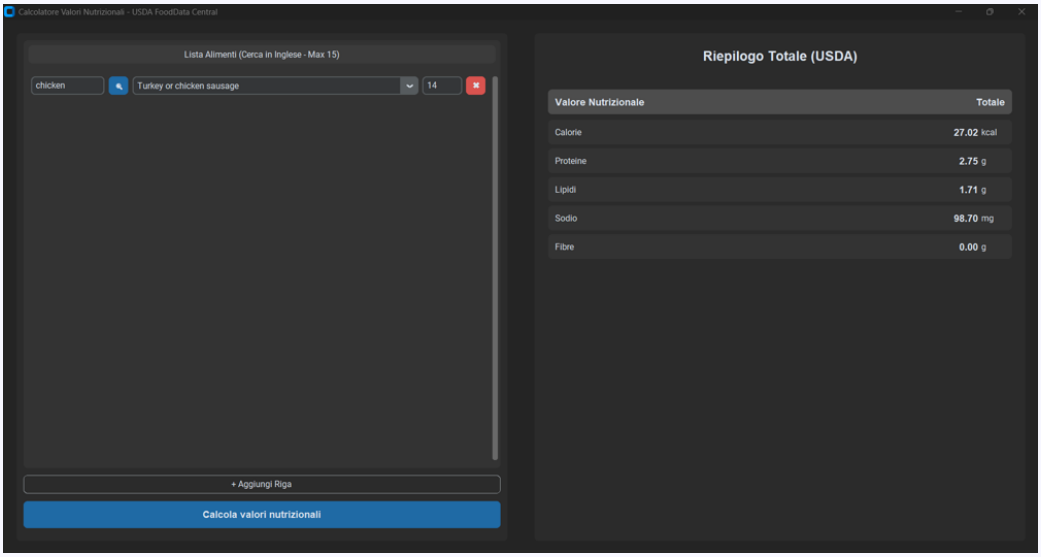
- aggiungimi, per ogni riga di alimento, la possibilità di rimuovere tale riga
- nell'elenco degli alimenti selezionabili in base all'input inserito, voglio poter scorrere su e giù usando la rotella di scorrimento del mouse

Questo prompt è dedicato al perfezionamento dell'interfaccia grafica. Il modello ha implementato correttamente il pulsante di rimozione per ogni riga, completando così la gestione dinamica degli ingredienti. Per quanto riguarda lo scorrimento della lista dei suggerimenti, invece, il comportamento richiesto non è stato ottenuto.

Gemini non ha tentato di implementare una soluzione non funzionante, ma ha motivato tecnicamente l'impossibilità di soddisfare la richiesta. Il modello ha infatti spiegato che il componente utilizzato da CustomTkinter per la visualizzazione dei menu a tendina si basa sul controllo nativo del sistema operativo Windows, il quale non supporta lo scorrimento mediante rotella del mouse per liste molto lunghe.

Le prove effettuate hanno confermato tale limitazione, evidenziando come il comportamento non dipendesse dall'implementazione del codice generato, bensì da un vincolo strutturale del componente grafico sottostante.

Gemini



4.4 Considerazioni finali sul case study

Il case study scelto in questo capitolo permette di osservare la capacità di mantenere la coerenza del progetto nel tempo, di correggere errori in modo strutturale e di gestire la complessità crescente da parte dei modelli.

Claude è l'unico modello che ha prodotto codice non eseguibile alla prima richiesta, a causa di un parametro non valido. Si tratta di un errore preciso e localizzato, risolto con un semplice prompt di correzione. Più indicativo, invece, è il fatto che nello stesso primo prompt, *Claude* abbia introdotto autonomamente una dipendenza dall'API esterna, non richiesta dal prompt, modificando l'architettura del sistema senza che l'utente lo avesse previsto. Questo comportamento non è un vantaggio quando la fase iniziale richiede, invece, semplicità ed eseguibilità immediata.

DeepSeek ha richiesto il percorso più lungo e accidentato. Il modello ha prodotto fin dall'inizio codice funzionante dal punto di vista della struttura grafica, ma con un bug sottile nella gestione degli errori: in presenza di un alimento non riconosciuto, i valori precedenti non venivano azzerati, producendo risultati incoerenti silenziosamente. Nella fase di integrazione con l'API USDA, il malfunzionamento nell'estrazione dei dati ha richiesto numerose iterazioni prima di essere risolto poiché il modello tendeva inizialmente a correggere gli effetti (messaggi di errore, valori a zero) senza individuare la causa (errata interpretazione della struttura della risposta). Il percorso riflette una modalità di debugging a cascata, in cui ogni fix può introdurre nuove regressioni.

La qualità del debugging è il vero discriminante. Quando la ricerca per "apple" restituiva prevalentemente risultati della catena "Applebee's", *Claude* non si è limitato a modificare parametri o soglie, ma ha identificato la causa del malfunzionamento e ha proposto una soluzione strutturale con strategia wildcard e fallback automatico. Questo tipo di risposta, che parte dall'analisi del problema e non dalla correzione del sintomo, rappresenta la differenza più netta osservata nel capitolo tra i modelli.

Gemini ha identificato correttamente che lo scrolling con rotella del mouse nei menù a tendina non è implementabile via codice, ma dipende da una limitazione del componente nativo di Windows. Invece di proporre una soluzione non funzionante o ignorare la richiesta, il modello ha documentato il vincolo tecnico. *ChatGPT* ha mostrato il profilo più lineare. Ha prodotto codice funzionante fin dal primo prompt, ha integrato correttamente l'API, ha implementato caching, threading asincrono e ordinamento dei risultati in modo coerente, senza richiedere cicli di correzione anomali. Non ha aggiunto funzionalità non richieste né ha avuto difficoltà a interpretare specifiche complesse come il criterio di priorità dei risultati.

In sintesi, questo case study evidenzia che la capacità di generare codice sintatticamente corretto è una condizione necessaria ma non sufficiente. Ciò che distingue i modelli in un processo di sviluppo iterativo è la profondità del ragionamento tecnico in fase di debugging, la coerenza nel rispettare le specifiche e la capacità di comunicare i propri limiti in modo trasparente, piuttosto che mascherarli con soluzioni non funzionanti.

Conclusioni

Nella presente tesi abbiamo affrontato il tema del confronto tra diversi Large Language Model, analizzandone il comportamento in due ambiti applicativi distinti, quali la generazione di testo e la generazione di codice. Nella prima parte della tesi abbiamo introdotto il concetto di Intelligenza Artificiale Generativa, ripercorrendo i fondamenti dell'Intelligenza Artificiale, del Machine Learning e del Deep Learning, necessari per comprenderne il funzionamento, per poi presentare nel dettaglio i quattro modelli oggetto di studio, *ChatGPT*, *DeepSeek*, *Claude* e *Gemini*, descrivendone le principali caratteristiche e modalità di interazione.

Su questa base abbiamo condotto due case study distinti. Nel primo abbiamo sottoposto i modelli a quattro scenari di adattamento di ricette di pizza, imponendo vincoli nutrizionali progressivamente più numerosi e, in alcuni casi, tra loro contraddittori, al fine di valutarne la capacità di produrre contenuti coerenti, completi e realizzabili. Da questa analisi è emerso che le differenze tra i modelli non sono costanti, ma dipendono in larga misura dalla complessità dei vincoli imposti. Con richieste semplici i modelli tendono a convergere su risultati equivalenti, mentre con vincoli multipli o contraddittori adottano strategie di interazione molto diverse tra loro, e mostrano in misura variabile la tendenza a privilegiare l'ottimizzazione numerica a scapito della coerenza gastronomica.

Nel secondo case study abbiamo osservato i modelli nello sviluppo iterativo di un'applicazione software con integrazione dell'API USDA FoodData Central, valutandone non solo la correttezza sintattica del codice prodotto ma, soprattutto la capacità di interpretare correttamente i requisiti e di gestire la complessità crescente del progetto. Anche in questo caso abbiamo riscontrato che la correttezza formale del codice non è sufficiente a discriminare la qualità dei modelli, mentre il fattore realmente distintivo è emerso nella profondità del ragionamento applicato in fase di debugging, nella capacità di individuare la causa reale di un malfunzionamento anziché limitarsi a correggerne il sintomo, e nella trasparenza con cui ciascun modello ha comunicato i propri limiti tecnici.

Complessivamente, il lavoro svolto ha permesso di delineare un quadro comparativo che va oltre la semplice valutazione della qualità delle risposte, evidenziando pattern di comportamento ricorrenti e specifici di ciascun modello, utili sia in ottica di ricerca sia come riferimento pratico per la scelta dello strumento più adatto in base al tipo di compito da svolgere. Un quadro di questo tipo resta, tuttavia, legato allo stato attuale dei modelli analizzati. Trattandosi di un settore in rapidissima evoluzione, sarebbe utile estendere il confronto a un numero maggiore di LLM, incluse le versioni più recenti di quelli già presi in esame, per verificare se i pattern individuati in questa tesi si mantengano stabili nel tempo o vengano superati dall'evoluzione tecnologica.

Bibliografia

AGGARWAL, R., SACHAN, S., VERMA, R. e DHANDA, N. (2025), «Traditional ai vs modern ai», in «The Confluence of Cryptography, Blockchain and Artificial Intelligence», p. 51–75, CRC Press.

AL-AMIN, M., ALI, M. S., SALAM, A., KHAN, A., ALI, A., ULLAH, A., ALAM, M. N. e CHOWDHURY, S. K. (2024), «History of generative Artificial Intelligence (AI) chatbots: past, present, and future development», *arXiv preprint arXiv:2402.05122*.

AMIGONI, F., SCHIAFFONATI, V., SOMALVICO, M. e OTHERS (2008), «Intelligenza artificiale», *Enciclopedia della Scienza e della Tecnica*.

AN, J., DING, W. e LIN, C. (2023), «Chatgpt», *tackle the growing carbon footprint of generative AI*, vol. 615, p. 586.

BANDI, A., ADAPA, P. V. S. R. e KUCHI, Y. E. V. P. K. (2023), «The power of generative ai: A review of requirements, models, input–output formats, evaluation metrics, and challenges», *Future Internet*, vol. 15 (8), p. 260.

BANH, L. e STROBEL, G. (2023), «Generative artificial intelligence», *Electronic markets*, vol. 33 (1), p. 63.

CAO, Y., LI, S., LIU, Y., YAN, Z., DAI, Y., YU, P. S. e SUN, L. (2023), «A comprehensive survey of ai-generated content (aigc): A history of generative ai from gan to chatgpt», *arXiv preprint arXiv:2303.04226*.

FEUERRIEGEL, S., HARTMANN, J., JANIESCH, C. e ZSCHECH, P. (2024), «Generative AI: S. Feuerriegel et al.», *Business & information systems engineering*, vol. 66 (1), p. 111–126.

GOYAL, M. e MAHMOUD, Q. H. (2024), «A systematic review of synthetic data generation techniques using generative AI», *Electronics*, vol. 13 (17), p. 3509.

GU, J., JIANG, X., SHI, Z., TAN, H., ZHAI, X., XU, C., LI, W., SHEN, Y., MA, S., LIU, H. e OTHERS (2026), «A survey on llm-as-a-judge», *The Innovation*, vol. 7 (6).

GUO, X. e CHEN, Y. (2024), «Generative ai for synthetic data generation: Methods, challenges and the future», *arXiv preprint arXiv:2403.04190*.

JANIESCH, C., ZSCHECH, P. e HEINRICH, K. (2021), «Machine learning and deep learning: C. Janiesch et al.», *Electronic markets*, vol. 31 (3), p. 685–695.

KELLEHER, J. D. (2019), *Deep learning*, MIT press.

LIU, A., FENG, B., XUE, B., WANG, B., WU, B., LU, C., ZHAO, C., DENG, C., ZHANG, C., RUAN, C. e OTHERS (2024), «Deepseek-v3 technical report», *arXiv preprint arXiv:2412.19437*.

NAM, D., MACVEAN, A., HELLENDOORN, V., VASILESCU, B. e MYERS, B. (2024), «Using an llm to help with code understanding», in «Proceedings of the IEEE/ACM 46th International Conference on Software Engineering», p. 1–13.

SARKER, I. H. (2021), «Machine learning: Algorithms, real-world applications and research directions», *SN computer science*, vol. 2 (3), p. 1–21.

SEETHA, H., TIWARI, V., ANUGU, K. R., MAKKA, D. e KARNATI, D. (2023), «A GUI based application for PDF processing tools using python & customTkinter», *Int J Res Appl Sci Eng Technol*, vol. 11 (1), p. 1613–1618.

SNELL, C., LEE, J., XU, K. e KUMAR, A. (2024), «Scaling llm test-time compute optimally can be more effective than scaling model parameters», *arXiv preprint arXiv:2408.03314*.

TEAM, G., ANIL, R., BORGEAUD, S., ALAYRAC, J.-B., YU, J., SORICUT, R., SCHALKWYK, J., DAI, A. M., HAUTH, A., MILLICAN, K. e OTHERS (2023), «Gemini: a family of highly capable multimodal models», *arXiv preprint arXiv:2312.11805*.

VALAVANIDIS, A. (2023), «Artificial Intelligence (AI) Applications: The most important technology we ever develop and we must ensure it is safe and beneficial to human civilization», *Website: chem-tox-ecotox. org/ScientificReviews... April*.

Sitografia

- Anthropic – `www.anthropic.com`
- ChatGPT – `chatgpt.com`
- Claude – `claude.ai`
- CREA, Alimenti e Nutrizione – `www.alimentinutrizione.it`
- DeepSeek – `chat.deepseek.com`
- Gazzetta Ufficiale – `www.gazzettaufficiale.it`
- Gemini – `gemini.google.com`
- GialloZafferano – `www.giallozafferano.it`
- Google AI for Developers – `ai.google.dev`
- Google DeepMind – `deepmind.google`
- IBM – `www.ibm.com`
- Intelligenza Artificiale, Il portale dedicato all'Intelligenza Artificiale – `www.intelligenzaartificiale.it`
- GitHub – `github.com`
- OpenAI – `openai.com`
- The Guardian – `www.theguardian.com`
- Wikipedia – `it.wikipedia.org`

Ringraziamenti

A Bolo, colonna portante della mia vita. Grazie per esserci stato sempre e comunque, di avermi capito nonostante le nostre differenze e di avermi sostenuto nelle difficoltà. Questo mio successo è anche tuo fratello, senza di te non ce l'avrei mai fatta. Nessuno mai potrà comprendere quanto tu sia speciale per me.

A mamma e papà. Grazie per aver accettato ogni mia scelta in tutti questi anni, soprattutto quando non la condividevate. Grazie per avermi rassicurato nei momenti di difficoltà e per aver festeggiato con me nei momenti di gioia, nello studio e non solo.

Ai miei super nonni. Neanche immaginate la gioia e la felicità che provo a poter condividere con voi questo traguardo. Grazie per tutti quei momenti che mi avete regalato nel corso di tutti questi anni, a tutte quelle mattine e tutti quei pomeriggi passati assieme a voi nell'orto, a cucinare o ad osservare le auto passare per strada.

Ai miei fratelli. Grazie per tutti quei momenti di leggerezza che mi regalate. A tutte quelle cose che rendono il nostro rapporto colorato, unico e speciale. Dalle risate fatte a tavola, passando per le sciate fatte assieme, fino alle discussioni e altre tantissime altre cose, non modificherei nulla di tutto cio che siamo.

A Bacca e Bernets. Quante risate ci siamo fatti assieme. A tutte le volte che, nello scherzo e nella serietà, non abbiamo mai smesso di dimostrarci a vicenda l'amicizia che ci lega. Grazie per avermi sempre spronato a dare il meglio di me, per essere stati pazienti e per tutti quei sorrisi che mi avete strappato quando mi vedevate giù di morale o affranto per qualcosa.

A Nicolò, per tutte quelle volte che mi hai dato conforto. Per tutte quelle volte che hai sempre cercato una soluzione per risollevarmi il morale quando non facevo altro che vedere nero. Non hai idea di quanto io sia fiero di poter dire di avere un amico così accanto. Sei una delle persone più pure che io conosca, genuina e sempre pronta a farsi in quattro per aiutare qualcuno; non cambiare mai.

A Denise, Sara e Alessio. Grazie per esserci stati dal primo giorno che ci siamo conosciuti. Grazie per i miliardi di ricordi che ci siamo regalati in tutti questi anni, per aver saputo accettare gli uni i cambiamenti degli altri supportandoci sempre a vicenda e per esservi stretti ancor di più a me nel momento del bisogno. L'orgoglio di avervi accanto oggi, così come nel giorno della mia maturità, è un qualcosa di indefinibile.

Ai ragazzi di OS. Grazie per dimostrarmi ogni giorno che non siamo solamente un gruppo di amici, ma una famiglia. Che nessuno viene lasciato indietro e che non si è mai soli, nonostante tutto. Non avete mai smesso di tifare per me, di spronarmi nell'inseguire questo traguardo con la vostra leggerezza e la vostra genuina curiosità. Il voler festeggiare assieme a tutti voi questo traguardo è stata la motivazione più grande che potessi avere in questi ultimi mesi.

A ragazzi dell'uni, per tutti quei momenti passati assieme. Le partite a poker, le serate assieme, le giornate di studio, i caffè e tutti quei momenti di risate che mi porterò per sempre nel cuore. Grazie per avermi sempre aiutato per ogni evenienza, per esservi sempre mostrati disponibili nei miei confronti, pronti a togliermi ogni tipo di dubbio. Vi porto nel cuore.

Vorrei ringraziare anche il Prof. Ursino, che mi ha accompagnato nel lavoro di tirocinio e tesi seguendomi costantemente con estrema disponibilità, rispondendo a tutti i miei quesiti in maniera precisa e chiara.

Grazie a tutti voi, per aver contribuito, ogni giorno, a rendermi la persona che oggi sono.

Davide