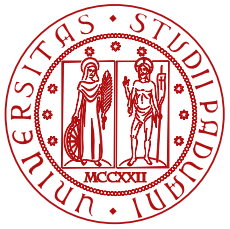


Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Report di Programmazione di Sistemi Embedded

Let an AI Do the Work

Studente

Giovanni Cinel

Matricola 2000147

Studente

Gianluca Nordio

Matricola 2007959

Studente

Federico Pivotto

Matricola 2008810

Anno Accademico
2022/2023

Indice

Elenco delle Figure

Elenco dei Frammenti di Codice

1

Intelligenze artificiali

1.1 Passato e presente delle IA

La comprensione del passato e del presente dell'Intelligenza Artificiale (IA) richiede un'analisi attenta e un'articolazione riflessiva che vada al di là dei dibattiti etici che spesso tendono a dominare la discussione. La storia dell'IA è costellata di momenti significativi che hanno portato a sviluppi cruciali nel campo, aprendo nuovi orizzonti e presentando sfide affascinanti. Il passato dell'IA si radica nel fervore intellettuale degli anni '50 e '60, quando i pionieri della disciplina sognarono di creare macchine intelligenti in grado di emulare le capacità umane.

Questo periodo di fervida innovazione vide l'emergere di approcci diversi, come la logica simbolica e le reti neurali, che segnarono le prime tappe di quello che sarebbe diventato il panorama attuale dell'IA. Tuttavia, l'entusiasmo iniziale venne mitigato da sfide tecniche complesse e dalla scoperta delle limitazioni intrinseche ai modelli di intelligenza artificiale dell'epoca. L'IA stentava a raggiungere la promessa di replicare fedelmente il pensiero umano.

Con l'avanzare del tempo, l'IA ha compiuto progressi significativi, abbracciando nuovi paradigmi e approcci che hanno reso possibile l'ottenimento di risultati più rilevanti. L'avvento del deep learning e dell'apprendimento automatico ha permesso alle macchine di elaborare grandi quantità di dati e di apprendere modelli complessi, portando a risultati sorprendenti in campi come il riconoscimento vocale, la visione artificiale e la traduzione automatica.

Oggi, l'IA permea molteplici aspetti della nostra vita quotidiana, dalla ricerca su Internet agli assistenti vocali e alle auto autonome. L'IA sta rivoluzionando l'industria, la medicina, il settore finanziario e molte altre discipline. Tuttavia, nonostante gli enormi progressi raggiunti, l'IA attuale continua a presentare sfide e limitazioni. L'interpretazione delle decisioni prese dalle macchine rimane un'area complessa da affrontare, poiché la trasparenza e l'etica della decisione algoritmica richiedono attenzione e responsabilità.

In conclusione, il passato e il presente dell'IA si intrecciano in un continuum di scoperte scientifiche e applicazioni concrete. L'IA ha compiuto progressi significativi, trasformando il modo in cui viviamo e lavoriamo.

1.2 Strumenti di IA: Generativi vs Non generativi

L'avvento dell'Intelligenza Artificiale ha catalizzato un dibattito costante riguardo all'utilizzo e all'efficacia degli strumenti generativi rispetto a quelli non generativi. Questo confronto affonda le sue radici nell'obiettivo comune di creare sistemi di IA in grado di produrre risultati di alta qualità e raggiungere livelli di comprensione e creatività simili a quelli umani.

Gli strumenti di IA generativi sono caratterizzati dalla loro capacità di creare nuovi contenuti autonomamente, attraverso l'elaborazione di modelli e dati di input. Essi cercano di simulare i processi di generazione umani, producendo testi, immagini o addirittura suoni originali. Ciò offre un'enorme potenzialità per la creatività, poiché i sistemi generativi possono ideare nuovi concetti e soluzioni a problemi complessi. Tuttavia, questa libertà creativa può anche portare a risultati imprevedibili o poco coerenti, richiedendo un'ulteriore supervisione umana per migliorarne la qualità.

D'altra parte, gli strumenti di IA non generativi si basano su approcci più deterministici e regolati, che utilizzano modelli predefiniti e dati di addestramento per effettuare compiti specifici. Questi strumenti mirano a fornire risposte e risultati più prevedibili e controllati e sono particolarmente efficaci quando si tratta di compiti ripetitivi e ben definiti, come il riconoscimento di immagini o il rilevamento del linguaggio naturale. La loro natura più limitata in termini di creatività li rende più adatti a situazioni in cui l'obiettivo principale è l'accuratezza e la coerenza piuttosto che l'originalità.

Nella scelta tra strumenti generativi e non generativi, è fondamentale considerare l'obiettivo specifico dell'applicazione e il contesto in cui verranno uti-

lizzati. Se l'obiettivo è promuovere la creatività e l'innovazione, gli strumenti generativi potrebbero essere la scelta migliore, anche se richiedono un maggiore sforzo per il controllo della qualità. D'altro canto, se l'obiettivo è ottenere risultati affidabili e prevedibili, gli strumenti non generativi possono rappresentare una soluzione più appropriata, riducendo al minimo il rischio di output indesiderati.

È importante sottolineare che, al di là del dibattito tra generativi e non generativi, i progressi nell'IA stanno portando verso l'adozione di approcci ibridi che combinano entrambe le modalità. Questa sinergia mira a sfruttare i punti di forza dei due approcci, cercando di massimizzare la creatività senza sacrificare la coerenza e la precisione. In definitiva, la scelta degli strumenti di IA dipenderà dalla situazione specifica, dall'obiettivo e dalle esigenze dell'applicazione, nonché dalla capacità di adattamento e supervisione umana richiesta per raggiungere i risultati desiderati.

2

Assistenti IA generativi

Nel contesto dell'era digitale in continua evoluzione, emergono costantemente nuovi strumenti e tecnologie che mirano a semplificare il lavoro degli sviluppatori e a potenziare la produttività nel settore della programmazione. Tra le ultime innovazioni che hanno suscitato un notevole interesse, si annoverano ChatGPT, GitHub Copilot e Tabnine. Questi strumenti si inseriscono nel panorama dell'intelligenza artificiale applicata al processo di sviluppo del software, offrendo assistenza e suggerimenti durante la stesura del codice. ChatGPT, GitHub Copilot e Tabnine rappresentano tre eccellenti esempi di come l'intelligenza artificiale stia rivoluzionando il modo in cui gli sviluppatori scrivono il codice. Questi strumenti forniscono un supporto significativo durante il processo di sviluppo del software, offrendo suggerimenti contestualmente rilevanti e accelerando la produttività. Pur avendo chiaramente i loro limiti, l'evoluzione di queste tecnologie promette di continuare a migliorare, aprendo nuove opportunità e sfide nel campo dello sviluppo software. Presentiamo inoltre Studio Bot, uno strumento generativo di IA realizzato per aiutare lo sviluppatore nella creazione di app Android, che tuttavia non verrà approfondito in seguito poiché non è stato utilizzato nel progetto dato che non è accessibile dall'Italia.

2.1 ChatGPT

ChatGPT di OpenAI, alimentato dalla potente architettura Generative Pretrained Transformer, detta GPT, si presenta come un modello di linguaggio basato su apprendimento automatico che permette di condurre conversazioni

in modo fluido e naturale. ChatGPT, che fornisce accesso alle versioni 3.5 e 4 del modello GPT, è addestrato su una vasta quantità di testo proveniente da diverse fonti, come libri, articoli di notizie, siti web e altro ancora. Questo gli consente di avere una notevole conoscenza su una vasta gamma di argomenti e di rispondere a domande complesse in modo coerente e comprensibile.

L'integrazione di ChatGPT in diverse piattaforme di messaggistica e assistenti virtuali ha aperto nuove possibilità di interazione con le macchine. Gli utenti possono porre domande, chiedere consigli, ottenere spiegazioni su concetti complessi e persino avere conversazioni più informali. ChatGPT cerca di capire l'intento dell'utente e di fornire risposte rilevanti e utili.

Tuttavia, è importante notare che ChatGPT non possiede una vera comprensione del mondo come gli esseri umani. Può generare risposte che sembrano coerenti, ma potrebbe anche produrre informazioni errate o non affidabili. OpenAI ha riconosciuto questo limite e sta lavorando per affinare il modello e migliorare la sua capacità di discernimento.

Una delle sfide che OpenAI affronta con ChatGPT è quella di bilanciare la generazione creativa di risposte con la necessità di fornire informazioni accurate. Sono state introdotte alcune restrizioni per evitare la generazione di contenuti inappropriati o fuorvianti. Inoltre, OpenAI sta esplorando modi per consentire agli utenti di influenzare le risposte di ChatGPT in modo da poter fornire informazioni più precise e personalizzate.

In conclusione, ChatGPT rappresenta un notevole passo avanti nell'interazione con l'intelligenza artificiale. Sfruttando il potenziale di GPT-3.5, consente agli utenti di impegnarsi in conversazioni virtuali, ottenere risposte e consigli utili su vari argomenti. Pur essendo ancora in fase di sviluppo, ChatGPT promette di offrire nuove opportunità di interazione e di ampliare le possibilità di utilizzo dell'intelligenza artificiale.

Tuttavia, il vero valore di ChatGPT nel contesto della programmazione risiede nella sua capacità di comprendere e generare codice. Grazie alla sua vasta base di conoscenze, ChatGPT può fornire suggerimenti, rispondere a domande specifiche e persino proporre soluzioni algoritmiche per affrontare determinati problemi. Ciò rappresenta una grande risorsa per gli sviluppatori, poiché consente loro di risparmiare tempo e sforzi, avendo a disposizione un assistente virtuale in grado di supportarli nella scrittura del codice.

2.2 GitHub Copilot

GitHub Copilot è un'innovativa soluzione di completamento automatico del codice sviluppata da GitHub in collaborazione con OpenAI, l'azienda che ha sviluppato il modello di linguaggio GPT. È stato lanciato ufficialmente nel giugno 2021.

La storia di GitHub Copilot inizia con la creazione di GPT-3, un potente modello di intelligenza artificiale basato su reti neurali trasformative. GPT-3 è stato addestrato su una vasta quantità di dati testuali provenienti da Internet e ha dimostrato una notevole capacità di generare testo coerente e comprensibile. La comunità degli sviluppatori ha iniziato a sperimentare GPT-3 per scopi di programmazione, scoprendo che il modello poteva essere utilizzato per generare codice sorgente.

GitHub, una popolare piattaforma di hosting e collaborazione per il codice sorgente, ha riconosciuto l'enorme potenziale di questa tecnologia per migliorare il flusso di lavoro degli sviluppatori. In collaborazione con OpenAI, hanno sviluppato GitHub Copilot come un'applicazione basata su GPT-3 che aiuta gli sviluppatori a scrivere codice in modo più rapido ed efficiente.

GitHub Copilot funziona come un assistente virtuale per la scrittura del codice. Si integra direttamente negli ambienti di sviluppo integrati (IDE), come Android Studio e Visual Studio Code, e fornisce suggerimenti di completamento del codice in tempo reale. Durante la stesura del codice, Copilot genera automaticamente proposte di completamento basate su modelli di codice provenienti da una vasta gamma di fonti, come repository pubbliche su GitHub, documentazione, librerie e altro ancora.

Il modello GPT-3 alla base di GitHub Copilot è stato addestrato su miliardi di righe di codice sorgente, che gli consentono di offrire suggerimenti pertinenti e completamenti automatici in base al contesto. Copilot può riconoscere il linguaggio di programmazione utilizzato, comprendere i commenti nel codice e suggerire pezzi di codice corretti e pertinenti per accelerare il processo di sviluppo.

Tuttavia, è importante sottolineare che GitHub Copilot non sostituisce gli sviluppatori. È progettato per essere uno strumento di assistenza che può aiutare a risparmiare tempo e migliorare la produttività, ma gli sviluppatori sono ancora responsabili della comprensione del codice generato e della sua correttezza.

Per garantire una migliore esperienza utente e una maggiore precisione, GitHub Copilot può essere addestrato con il feedback degli sviluppatori. Gli utenti possono segnalare la qualità dei suggerimenti e fornire un feedback diretto all'applicazione per migliorarne le capacità nel tempo.

In sintesi, GitHub Copilot è un'estensione basata su intelligenza artificiale che fornisce suggerimenti di completamento del codice in tempo reale, aiutando gli sviluppatori a scrivere codice in modo più rapido ed efficiente.

2.3 Tabnine

Un'altra tecnologia che ha suscitato l'attenzione nella comunità degli sviluppatori è Tabnine. Sfruttando l'intelligenza artificiale e il machine learning, Tabnine si presenta come un'estensione per vari IDE che offre suggerimenti di completamento del codice in tempo reale. L'algoritmo di Tabnine è stato addestrato su una vasta gamma di codice sorgente proveniente da progetti open source, consentendogli di fornire suggerimenti pertinenti in base al contesto e al linguaggio di programmazione utilizzato. Questa funzionalità è particolarmente utile per gli sviluppatori alle prese con la scrittura del codice, in quanto può accelerare il processo e migliorare la precisione complessiva.

2.4 Studio Bot

Nonostante l'impossibilità di utilizzo (poiché non disponibile in Italia) e quindi di poterne fornire un'opinione, è opportuno citare Studio Bot, uno strumento di IA creato direttamente da Google e integrato nelle ultime versioni di Android Studio. Studio Bot è un innovativo strumento progettato per assistere gli sviluppatori Android nel processo di codifica. Grazie alla sua esperienza conversazionale integrata in Android Studio, Studio Bot mira a migliorare la produttività e fornire preziose informazioni sfruttando l'intelligenza artificiale. Grazie alla comprensione del linguaggio naturale da parte di Studio Bot, gli sviluppatori possono interagire in lingua inglese, rendendolo un compagno di codifica user-friendly e accessibile.

Una delle principali funzionalità di Studio Bot è la generazione di frammenti di codice e l'assistenza per le query di sviluppo Android. Che si tratti di aggiungere il supporto della fotocamera a un'app, creare un database Room o

implementare un tema scuro, Studio Bot può generare esempi di codice personalizzati per compiti specifici. Analizzando il contesto della conversazione, Studio Bot assicura che i frammenti di codice forniti siano pertinenti e allineati alle esigenze degli sviluppatori. Questa funzionalità permette di risparmiare tempo e sforzi preziosi consentendo agli sviluppatori di concentrarsi su altri aspetti del processo di sviluppo dell'app.

Oltre alla generazione di codice, Studio Bot agisce come uno strumento per la scoperta delle risorse. Quando gli sviluppatori hanno domande o richiedono ulteriori informazioni su argomenti specifici dello sviluppo Android, Studio Bot può offrire documentazione e riferimenti pertinenti. Suggerendo risorse, Studio Bot consente agli sviluppatori di approfondire ulteriormente la materia e ampliare le loro conoscenze.

Inoltre, la comprensione contestuale di Studio Bot consente agli sviluppatori di avere conversazioni interattive. Essi possono fare domande di approfondimento e richiedere esempi di codice specifici in Kotlin. Studio Bot sfrutta questa comprensione contestuale per fornire risposte personalizzate, garantendo che gli sviluppatori ricevano la guida più rilevante e accurata per le loro esigenze specifiche.

Per iniziare a utilizzare Studio Bot, è necessario scaricare l'ultima versione di Android Studio Hedgehog. Consentendo la condivisione dei dati con Google, gli sviluppatori possono contribuire al miglioramento dell'efficacia di Studio Bot. Una volta installato, Studio Bot può essere accessibile tramite l'interfaccia di Android Studio, offrendo un'esperienza integrata.

3

Obiettivi da raggiungere

Nel presente capitolo verranno presentati gli obiettivi fissati nella fase di progettazione dell'app, sfruttando gli strumenti di intelligenza artificiale generativa come ChatGPT, GitHub Copilot e Tabnine. La valutazione sul raggiungimento degli obiettivi sarà trattata nel capitolo 6.

3.1 Obiettivi prefissati

Prima dell'inizio del progetto, sono stati definiti gli obiettivi chiave da raggiungere utilizzando gli strumenti di intelligenza artificiale generativa. I principali obiettivi sono stati identificati come segue:

1. Incrementare la produttività degli sviluppatori, utilizzando gli strumenti di intelligenza artificiale generativa per accelerare il processo di sviluppo, riducendo il tempo necessario per scrivere codice e suggerendo soluzioni efficienti.
2. Migliorare la qualità del codice verificando se gli strumenti di intelligenza artificiale generativa sono in grado di fornire suggerimenti di codice di alta qualità, riducendo gli errori e migliorando la manutenibilità complessiva dell'applicazione.
3. Espandere le capacità creative del team grazie all'utilizzo degli strumenti di intelligenza artificiale generativa per generare idee innovative volte a migliorare l'esperienza dell'utente nell'utilizzo dell'applicazione e introducendo tutte le funzionalità necessarie per ottenere un'applicazione efficiente.

3.2 Approccio allo sviluppo

Per raggiungere gli obiettivi prefissati, abbiamo adottato un metodologia basata su diverse fasi durante il processo di sviluppo dell'applicazione Android. L'approccio è stato strutturato come segue:

1. Scelta dello strumento di IA per la generazione: in questa fase iniziale, il team ha condotto una ricerca approfondita sugli strumenti di intelligenza artificiale generativa disponibili, concentrandosi particolarmente su ChatGPT, GitHub Copilot e Tabnine, valutando l'alternativa migliore per generare il frammento di codice richiesto.

2. Richiesta di generazione del codice: in questa fase il codice viene generato tramite una richiesta specifica a ChatGPT o mediante l'inserimento di un suggerimento tramite commento nel codice per la generazione da parte di Tabnine e GitHub Copilot. L'obiettivo principale di questa richiesta è ottenere il codice desiderato.

3. Integrazione e personalizzazione: in questa fase il codice generato nella fase precedente viene integrato al codice del progetto e rifinito dove errato o non adatto.

4. Ottimizzazione del codice: gli strumenti di intelligenza artificiale generativa sono stati utilizzati per fornire ulteriori feedback per migliorare la qualità del codice e delle prestazioni dell'applicazione.

3.3 Criteri di valutazione

Nel capitolo 6, si discuterà il raggiungimento o meno degli obiettivi prefissati in base ai risultati ottenuti dall'utilizzo degli strumenti di intelligenza artificiale generativa. Saranno considerati diversi criteri di valutazione, tra cui:

1. Produttività degli sviluppatori: sarà valutato il tempo risparmiato utilizzando gli strumenti di intelligenza artificiale generativa rispetto all'approccio tradizionale di sviluppo.

2. Qualità del codice: sarà analizzata la qualità generale del codice prodotto utilizzando gli strumenti di intelligenza artificiale generativa, concentrandosi sulla presenza di errori e sulla manutenibilità del codice.

3. Capacità creative: sarà valutata la capacità degli strumenti di intelligenza artificiale di generare soluzioni innovative e suggerire funzionalità per l'applicazione.

Sulla base dei risultati ottenuti, saranno discussi i benefici e le limitazioni riscontrate nell'utilizzo degli strumenti di intelligenza artificiale generativa nel contesto del progetto di applicazione Android, evidenziando i contributi apportati e le sfide incontrate durante il processo di sviluppo.

4

Utilizzo delle IA generative nelle fasi dello sviluppo

4.1 Ricerca dell'idea

Nella fase di ricerca dell'idea, abbiamo utilizzato ChatGPT per esplorare diverse possibilità di sviluppo di un'app Android che potesse sfruttare al massimo le potenzialità dell'intelligenza artificiale nel suo sviluppo. L'interazione con ChatGPT ci ha permesso di ottenere rapidamente una serie di idee generiche, risparmiandoci così prezioso tempo nella fase iniziale di brainstorming.

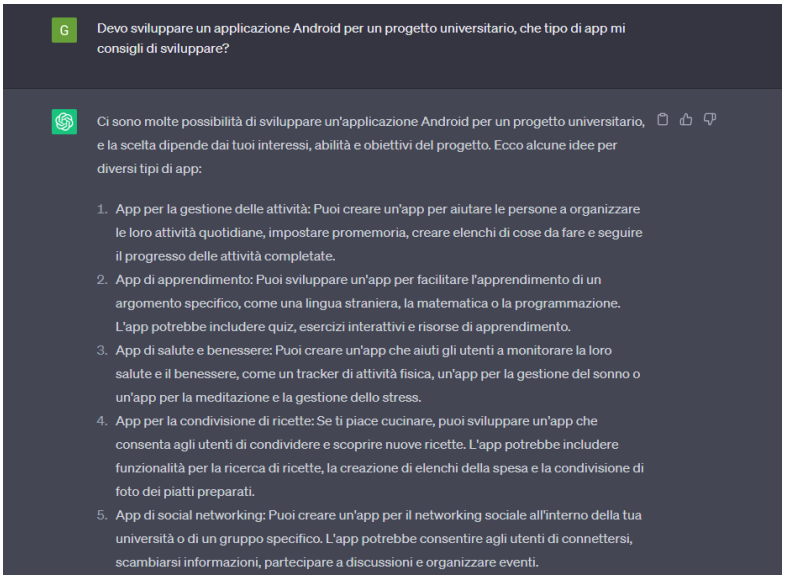


Figura 4.1: Idee fornite da ChatGPT sulle possibili applicazioni da sviluppare.

Dopo aver esplorato diverse opzioni con ChatGPT, ci siamo concentrati sull'idea di sviluppare un'app per la gestione degli eventi e delle attività quotidiane. Questa scelta è stata facilitata dall'aiuto di ChatGPT nel valutare l'effort richiesto per ciascuna idea e nel fornirci suggerimenti pratici ed efficienti.

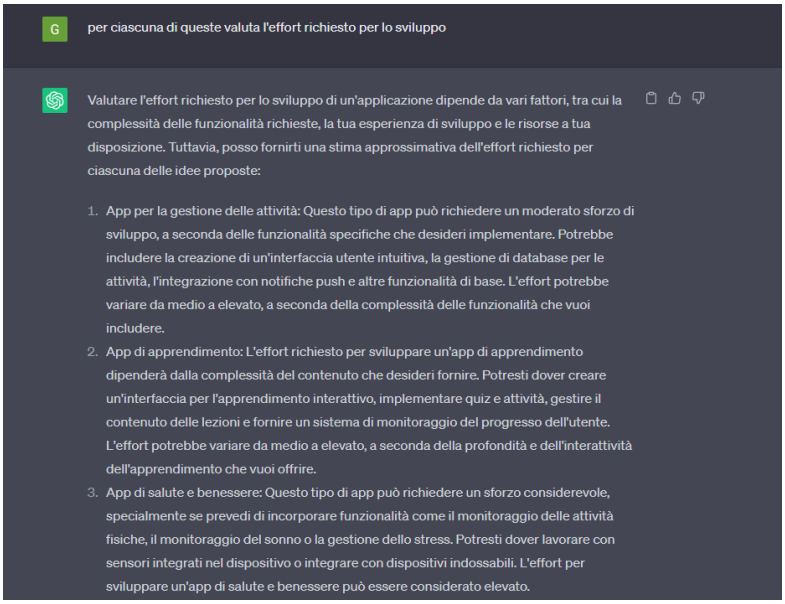


Figura 4.2: Idee fornite da ChatGPT sull'effort necessario per sviluppare ciascuna delle applicazioni proposte.

Una volta selezionata l'idea che avrebbe costituito la base dell'app, abbiamo continuato a sfruttare le capacità di generazione del linguaggio naturale di

ChatGPT per ottenere suggerimenti per il nome dell'applicazione. L'aiuto di ChatGPT in questa fase ci ha consentito di risparmiare ulteriormente tempo nella ricerca di un nome appropriato, fornendoci diverse opzioni valide e stimolanti da considerare.

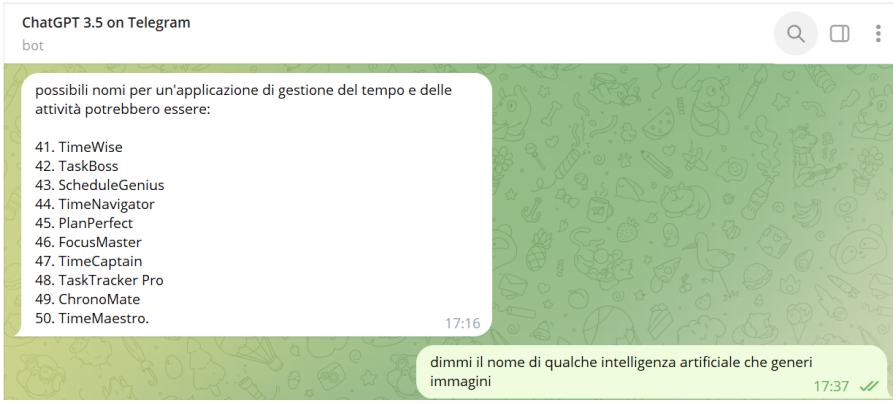


Figura 4.3: Idee fornite da ChatGPT sui possibili nomi per l'applicazione.

Dopo un'attenta valutazione delle proposte generate da ChatGPT, abbiamo scelto il nome "TimeWise". Questa decisione è stata raggiunta in modo rapido ed efficiente grazie all'assistenza di ChatGPT nel fornirci opzioni pertinenti e interessanti. Il tempo che abbiamo risparmiato in questa fase ci ha permesso di concentrarci meglio sullo sviluppo e le altre fasi del progetto.

4.2 Organizzazione del lavoro

Scelta l'idea per l'app, abbiamo successivamente affrontato il compito cruciale di suddividere il lavoro da svolgere per lo sviluppo dell'app e assegnarlo ai membri del gruppo. In questa fase, abbiamo deciso di non utilizzare ChatGPT, poiché abbiamo ritenuto che l'organizzazione del lavoro fosse una parte estremamente delicata del progetto, in cui l'esperienza e le competenze acquisite nel corso degli anni di studio e nelle esperienze pregresse fossero fondamentali per il suo successo.

Per organizzare il lavoro in modo efficace, abbiamo sfruttato esclusivamente la conoscenza accumulata dal nostro team nel campo dello sviluppo di software e delle metodologie di gestione del progetto. Abbiamo considerato attentamente le competenze e le abilità di ciascun membro del gruppo, assegnandoci compiti specifici in base alle nostre esperienze e capacità. Abbiamo anche tenuto conto

dei vincoli di tempo e delle risorse disponibili per garantire un equilibrio ottimale nella distribuzione del lavoro.

L'esperienza pregressa e l'expertise del nostro team sono stati fondamentali nel prendere decisioni valide e nel pianificare il lavoro in modo efficiente. Abbiamo fatto affidamento su pratiche consolidate e metodologie di gestione del progetto per garantire una divisione chiara dei compiti, una comunicazione efficace e un monitoraggio costante del progresso.

Riconosciamo che, nonostante l'ausilio di strumenti di intelligenza artificiale come ChatGPT nelle fasi precedenti, l'organizzazione del lavoro richiede una comprensione approfondita delle dinamiche di progetto e delle competenze necessarie per il suo successo. Pertanto, abbiamo deciso di affidarci esclusivamente all'esperienza e alle competenze del nostro team, garantendo così un approccio più stabile e controllato per questa fase critica dello sviluppo dell'app.

4.3 Creazione dell'icona

Nel contesto dello sviluppo di un'applicazione, l'immagine dell'icona gioca un ruolo fondamentale nell'identificazione e nella distinzione dell'app all'interno dell'ecosistema digitale. Inizialmente, abbiamo considerato l'utilizzo di algoritmi generativi come un'opzione promettente per generare l'icona desiderata. Tuttavia, questa strada si è rivelata più complessa del previsto.

La ricerca di risorse generative adatte ha rivelato che la maggior parte di esse è disponibile a pagamento, il che rende il processo di generazione dell'icona costoso e non conveniente data la limitatezza delle nostre risorse. Inoltre, gli algoritmi generativi gratuiti hanno spesso prodotto risultati privi di senso o inutilizzabili, senza una chiara connessione con la finalità o l'identità dell'applicazione.

Di fronte a queste sfide abbiamo deciso di adottare una soluzione alternativa: la creazione manuale dell'icona. Questo approccio ci ha permesso di avere un controllo diretto sul processo creativo, esprimendo in modo preciso la nostra visione e l'identità dell'app. Coinvolgendo il team, abbiamo esplorato diversi stili e concetti, affrontando iterativamente le proposte e raffinando l'aspetto visivo dell'icona.

L'approccio manuale ci ha offerto vantaggi significativi. Innanzitutto, ci ha dato un controllo creativo completo, consentendoci di plasmare l'icona in base alle specifiche esigenze dell'applicazione. Abbiamo potuto personalizzarla, includendo dettagli che riflettono le funzionalità e lo scopo dell'app. Inoltre, siamo

riusciti a creare un'icona unica e distintiva, che si differenzia dalle altre applicazioni presenti sul mercato. Questa distinzione contribuisce alla riconoscibilità dell'app da parte degli utenti.

Un altro beneficio dell'approccio manuale è stato il coinvolgimento attivo del team. Abbiamo valorizzato le competenze artistiche dei membri del nostro team, stimolando la collaborazione e l'entusiasmo per il progetto. Questa collaborazione ha contribuito alla creazione di un'icona che rispecchia appieno la nostra visione e i valori dell'applicazione.

Infine, l'approccio manuale ci ha permesso di ottenere una soddisfazione estetica completa. Abbiamo raggiunto un risultato finale che ci ha soddisfatto appieno dal punto di vista estetico, rappresentando accuratamente l'immagine che desideravamo proiettare attraverso l'icona.

In conclusione, nonostante i tentativi iniziali di affidarci ad algoritmi generativi per generare l'icona dell'applicazione, ci siamo confrontati con sfide significative e risultati insoddisfacenti. Abbiamo quindi optato per l'approccio manuale, che ci ha offerto vantaggi evidenti.

In Figura 4.4 vengono presentate le icone generate da IA e l'icona generata manualmente.

(a) (b) (c)

Figura 4.4: Le icone (a) e (b) generate da App Icon, l'icona (c) generata manualmente.

4.4 Progettazione del database

Nel contesto della progettazione concettuale del database per la nostra applicazione Android, l'impiego degli strumenti di intelligenza artificiale, in particolare ChatGPT, ha fornito un supporto significativo. ChatGPT ha fornito un'idea preliminare su quali tabelle creare e quali vincoli di chiave esterna implementare. Tale contributo è stato di notevole importanza nel processo di progettazione iniziale del database.

A dimostrazione dell'efficacia di ChatGPT riportiamo la progettazione concettuale generata nel caso particolare della gestione della lista delle attività, che coincide con quella utilizzata nell'applicazione.

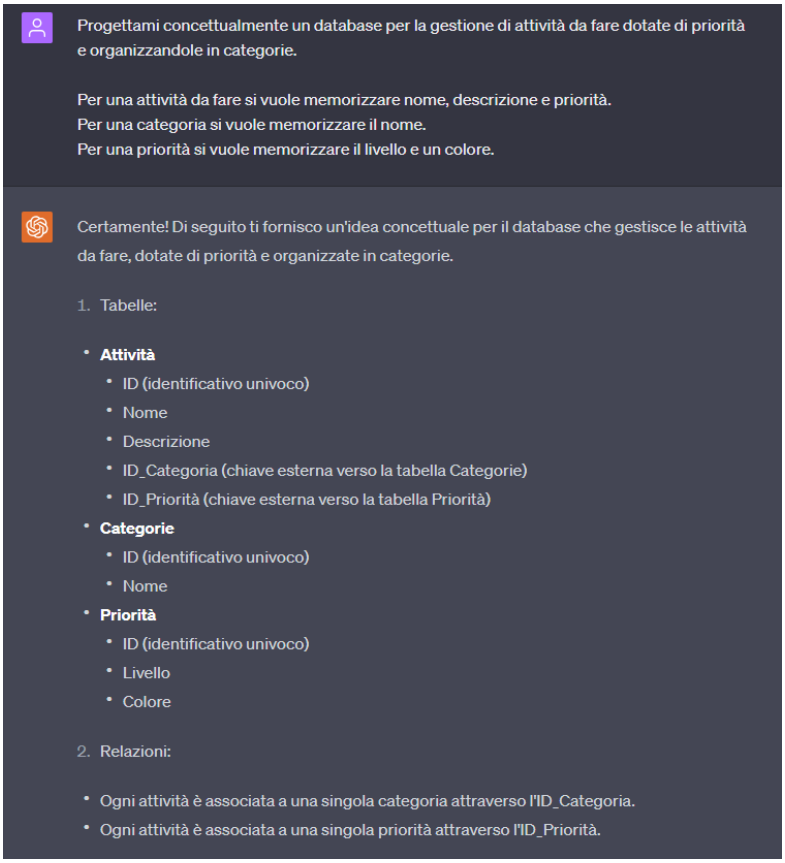


Figura 4.5: Progettazione concettuale del database generata da ChatGPT.

Successivamente, nel corso della progettazione fisica, e dunque dell'implementazione del database mediante l'utilizzo della libreria Room, ChatGPT si è dimostrato altrettanto utile, poiché ha fornito il codice Kotlin di base per la realizzazione delle classi AppDatabase, Dao ed Entity. Tuttavia, queste implementazioni preliminari non includevano la creazione dei vincoli di chiave esterna tra le tabelle né le chiamate a callback.

È stato quindi necessario apportare delle modifiche alle implementazioni fornite da ChatGPT. In particolare, sono stati aggiunti i vincoli di chiave esterna tra le tabelle, al fine di garantire l'integrità referenziale nel database. Sono state inoltre introdotte le opportune chiamate a callback per l'inizializzazione corretta del database dopo l'installazione dell'applicazione.

A dimostrazione, riportiamo un esempio di codice con tali lacune generato da ChatGPT e il codice Kotlin effettivamente utilizzato. Si riporta sia il caso delle chiavi esterne sia il caso delle callback.

Vincoli di chiave esterna

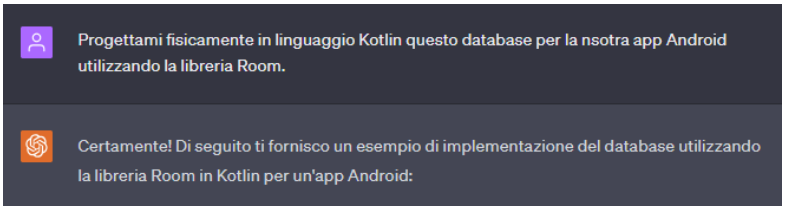


Figura 4.6: Richiesta progettazione fisica a ChatGPT.

```kotlin
// Codice prodotto da ChatGPT

@Entity(tableName = "attivita")
data class Attivita(
    @PrimaryKey(autoGenerate = true)
    val id: Long = 0,
    val nome: String,
    val descrizione: String,
    @ColumnInfo(name = "id_categoria")
    val idCategoria: Long,
    @ColumnInfo(name = "id_priorita")
    val idPriorita: Long
)
```

Codice 4.1: Codice prodotto da ChatGPT per la creazione dell'entità task

```kotlin
// Codice utilizzato nel progetto

@Entity(
    tableName = "task_table",
    foreignKeys = [
        ForeignKey(
            entity = Category::class,
            parentColumns = ["id"],
```

```
            childColumns = ["category"]
        ),
        ForeignKey(
            entity = Priority::class,
            parentColumns = ["id"],
            childColumns = ["priority"]
        )
    ],
    indices = [
        Index(value = ["category"]),
        Index(value = ["priority"])
    ]
)
data class Task(
    // Auto-generated integer task id
    @PrimaryKey(autoGenerate = true) val id: Int = 0,
    // Other attributes of the task
    @ColumnInfo(name = "name") val name: String,
    @ColumnInfo(name = "description") val description:
   String,
    // Foreign keys of the task
    @ColumnInfo(name = "category") val category: Int,
    @ColumnInfo(name = "priority") val priority: Int
)
```

Codice 4.2: Codice utilizzato nel progetto per la creazione dell'entità task

Callback per le priorità

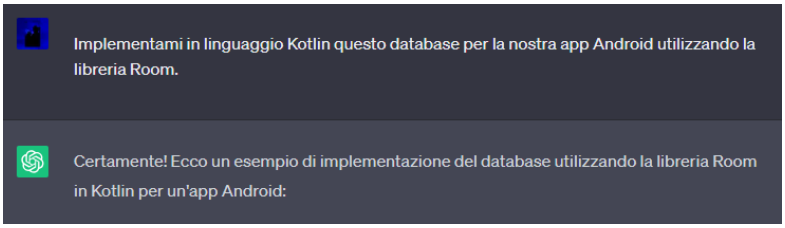


Figura 4.7: Richiesta callback per le priorità a ChatGPT.

```
// Codice prodotto da ChatGPT

@Database(entities = [Attivita::class, Categoria::class
    , Priorita::class], version = 1)
abstract class AppDatabase : RoomDatabase() {
    // ...

    companion object {
        // ...

        private val roomCallback = object :
    RoomDatabase.Callback() {
            override fun onCreate(db:
    SupportSQLiteDatabase) {
                super.onCreate(db)
                // Inizializza la tabella delle
    priorità con i valori predefiniti
                val priorityDao = instance?.prioritaDao
    ()
                priorityDao?.insertPriorita(Priorita(
    nome = "Molto Basso", livello = "Very Low", colore =
    "#CCCCCC"))
                priorityDao?.insertPriorita(Priorita(
    nome = "Basso", livello = "Low", colore = "#FFCCCC"))
                priorityDao?.insertPriorita(Priorita(
    nome = "Medio", livello = "Medium", colore = "#FFFFCC
    "))
                priorityDao?.insertPriorita(Priorita(
    nome = "Alto", livello = "High", colore = "#CCFFCC"))
            }
        }

        private fun buildDatabase(context: Context):
    AppDatabase {
```

```
            return Room.databaseBuilder(context.
    applicationContext, AppDatabase::class.java, "
    app_database")
                .fallbackToDestructiveMigration()
                .addCallback(roomCallback) // Aggiungi
    la callback alla configurazione del database
                .build()
        }
    }
}
```

Codice 4.3: Codice prodotto da ChatGPT per la callback delle priorità

```
// Codice utilizzato nel progetto

@Database(
    entities = [Category::class, Priority::class, Task
    ::class, EventType::class, Event::class],
    version = 1,
    exportSchema = false
)
@TypeConverters(Converters::class)
abstract class AppDatabase : RoomDatabase() {
    // Category Dao for categories manipulation
    abstract fun categoryDao(): CategoryDao

    // Task Dao for tasks manipulation
    abstract fun taskDao(): TaskDao

    // Priority Dao for priorities manipulation
    abstract fun priorityDao(): PriorityDao

    // EventType Dao for event types manipulation
    abstract fun eventTypeDao(): EventTypeDao

    // Event Dao for event manipulation
    abstract fun eventDao(): EventDao
```

```
24
25      // Companion object necessary to work with database
        instances
26      companion object {
27          // Database instance declaration
28          // Singleton prevents multiple instances of
    database opening at the same time
29          @Volatile
30          private var INSTANCE: AppDatabase? = null
31
32          // Get method to return database instance
33          fun getInstance(context: Context): AppDatabase
    {
34              // Check database instance existence
35              if (INSTANCE == null) {
36                  // Database instance creation
37                  // The following code is a coroutine
    that create a database instance
38                  // A coroutine is a thread that
    executes code asynchronously
39                  synchronized(this) {
40                      INSTANCE = Room.databaseBuilder(
41                          context.applicationContext,
42                          AppDatabase::class.java,
43                          "timewise_database"
44                      )
45                          .fallbackToDestructiveMigration
    () // Wipes and rebuilds database instead of
    migrating
46                          .addCallback(
    AppDatabaseCallback(CoroutineScope(Dispatchers.
    Default))) // Add a callback to populate database
47                          .allowMainThreadQueries() //
    Allow query execution in the main activity
48                          .build()
49                  }
```

```
50              }
51
52              // Return database instance
53              return INSTANCE as AppDatabase
54          }
55
56          // Callback to populate database
57          private class AppDatabaseCallback(private val
    scope: CoroutineScope) : Callback() {
58              // Override onCreate method to populate
    database to get data persistence
59              override fun onCreate(db:
    SupportSQLiteDatabase) {
60                  super.onCreate(db)
61                  // Default database population
62                  INSTANCE?.let { database ->
63                      scope.launch(Dispatchers.IO) {
64                          populateDatabase(database.
    priorityDao(), database.eventTypeDao())
65                      }
66                  }
67              }
68          }
69
70          // Populate the database with default
    priorities and event types
71          fun populateDatabase(priorityDao: PriorityDao,
    eventTypeDao: EventTypeDao) {
72              // Default priority table initialization
73              priorityDao.insert(Priority(1, "
    white_priority", "low"))
74              priorityDao.insert(Priority(2, "
    yellow_priority", "medium"))
75              priorityDao.insert(Priority(3, "
    orange_priority", "high"))
76              priorityDao.insert(Priority(4, "
```

```
    red_priority", "very high"))
77
78              // ...
79          }
80      }
81  }
```

Codice 4.4: Codice utilizzato nel progetto per la callback delle priorità

4.5 Sviluppo frontend: layout grafica

Durante la fase di sviluppo del frontend e del layout grafico dell'app Android, abbiamo sfruttato diverse intelligenze artificiali, tra cui ChatGPT, GitHub Copilot e Tabnine, al fine di creare le interfacce utente mediante l'approccio dichiarativo, ossia utilizzando file XML.

Nel processo di creazione del layout, abbiamo constatato che Tabnine non ha fornito un aiuto significativo. Nonostante le sue funzionalità di completamento automatico del codice, Tabnine non ha dimostrato una comprensione sufficiente del contesto specifico delle interfacce utente e non ha fornito suggerimenti adeguati per la creazione dei file XML.

Al contrario, sia ChatGPT che GitHub Copilot si sono rivelati strumenti preziosi in questa fase. Abbiamo utilizzato ChatGPT per ottenere un input generale per la struttura dei file XML, basandoci sulle indicazioni e le specifiche che gli abbiamo fornito. Sebbene il codice generato da ChatGPT fosse piuttosto elementare, ha gettato le basi per il layout generale di gran parte delle schermate, e successivamente lo abbiamo personalizzato e adattato alle nostre esigenze.

GitHub Copilot, d'altra parte, è stato particolarmente utile per aggiungere facilmente personalizzazioni alle diverse componenti dell'interfaccia utente. Ci ha fornito suggerimenti intelligenti e completamenti automatici durante la scrittura del codice XML, consentendoci di accelerare il processo di sviluppo e di mantenere una coerenza nel design delle varie schermate grazie alla sua capacità di riconoscere il contesto. Inoltre, GitHub Copilot ci ha permesso di ripetere facilmente componenti UI già definite, come TextView o Button, per gestire diversi dati o azioni all'interno delle schermate.

A dimostrazione, riportiamo un esempio di codice XML generato da ChatGPT per il layout della schermata di inserimento di un evento compleanno e il codice XML effettivamente utilizzato.

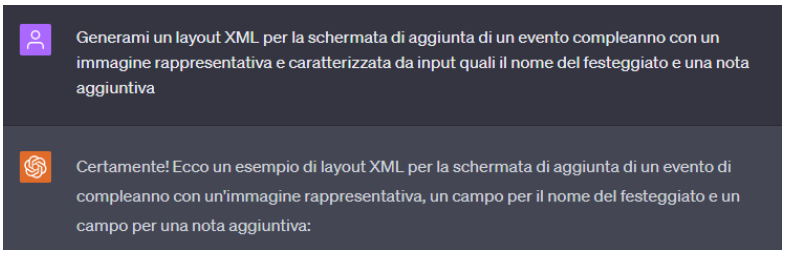


Figura 4.8: Richiesta layout XML per l'aggiunta di un compleanno a ChatGPT.

```xml
// Codice prodotto da ChatGPT

<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/
   android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <!-- Immagine rappresentativa -->
    <ImageView
        android:id="@+id/imageViewBirthday"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:src="@drawable/birthday_image"
        android:scaleType="centerCrop"/>

    <!-- Campo per il nome del festeggiato -->
    <EditText
        android:id="@+id/editTextName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Nome del festeggiato"/>

    <!-- Campo per la nota aggiuntiva -->
```

```
    <EditText
        android:id="@+id/editTextNote"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Nota aggiuntiva"/>

    <!-- Bottone di conferma -->
    <Button
        android:id="@+id/buttonAddEvent"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Aggiungi evento"/>

</LinearLayout>
```

Codice 4.5: Codice prodotto da ChatGPT per il layout dell'aggiunta dell'evento compleanno

```
// Codice utilizzato nel progetto

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com
   /apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="?attr/colorPrimaryVariant"
    android:gravity="center"
    android:orientation="vertical"
    android:padding="16dp">

    <ImageView
        android:id="@+id/imageViewTop"
        android:layout_width="250dp"
        android:layout_height="250dp"
        android:padding="16dp"
```

```
19          android:scaleX="0.7"
20          android:scaleY="0.7"
21          android:src="@drawable/birthday_cake"
22          app:tint="?attr/android:textColor"
23          android:contentDescription="@string/
    event_birthday_img_1" />
24
25      <ScrollView
26          android:layout_width="match_parent"
27          android:layout_height="wrap_content"
28          android:fillViewport="true">
29
30          <LinearLayout
31              android:layout_width="match_parent"
32              android:layout_height="0dp"
33              android:gravity="center"
34              android:orientation="vertical">
35
36              <com.google.android.material.textfield.
    TextInputLayout
37                  android:id="@+id/textInputLayoutName"
38                  android:layout_width="match_parent"
39                  android:layout_height="wrap_content"
40                  android:hint="@string/event_birthday_1"
    >
41
42                  <com.google.android.material.textfield.
    TextInputEditText
43                      android:id="@+id/editTextName"
44                      android:layout_width="match_parent"
45                      android:layout_height="wrap_content
    "
46                      android:inputType="text"
47                      android:maxLines="1"
48                      android:textColor="?attr/android:
    textColor"
```

```
                    tools:ignore="TextContrastCheck,
VisualLintTextFieldSize" />

        </com.google.android.material.textfield.
TextInputLayout>

        <com.google.android.material.textfield.
TextInputLayout
            android:id="@+id/textInputLayoutNotes"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="@string/event_birthday_2"
>

            <com.google.android.material.textfield.
TextInputEditText
                android:id="@+id/editTextNotes"
                android:layout_width="match_parent"
                android:layout_height="wrap_content
"
                android:inputType="text"
                android:maxLines="1"
                android:textColor="?attr/android:
textColor"
                tools:ignore="TextContrastCheck,
VisualLintTextFieldSize" />

        </com.google.android.material.textfield.
TextInputLayout>

    </LinearLayout>
 </ScrollView>

 <Button
    android:id="@+id/buttonSave"
    android:layout_width="match_parent"
```

```
76          android:layout_height="wrap_content"
77          android:text="@string/accept"
78          android:textColor="?attr/android:textColor"
79          tools:ignore="VisualLintButtonSize" />
80
81  </LinearLayout>
```

Codice 4.6: Codice utilizzato nel progetto per il layout dell'aggiunta dell'evento compleanno

In generale abbiamo potuto osservare che il codice XML per i layout generato da ChatGPT presenta delle lacune significative, che non possono essere ignorate. Oltre alla mancanza di personalizzazione dei componenti del layout, come precedentemente menzionato, è evidente l'assenza totale di una gestione adeguata della responsività rispetto alle diverse dimensioni dello schermo, poi ottenuta manualmente mediante una ScrollView. Questa mancanza di adattabilità compromette la fruibilità dell'applicazione su dispositivi con schermi di dimensioni diverse, creando potenziali problemi di visualizzazione e interazione per gli utenti. Un altro dettaglio da non tralasciare riguarda le stringhe utilizzate come literal nel codice XML generato, senza seguire l'approccio standard secondo cui vengono definite in appositi file XML, e cablate con nomi identificativi.

Complessivamente, sebbene vi siano mancanze come ad esempio quelle appena riportate, durante lo sviluppo del frontend e del layout grafico, abbiamo sfruttato significativamente ChatGPT e GitHub Copilot per generare il codice iniziale e facilitare la personalizzazione delle interfacce utente. Mentre ChatGPT ha fornito un input generale sulla struttura dei file XML, GitHub Copilot è stato un valido alleato per aggiungere personalizzazioni e ripetere componenti UI già definite, migliorando l'efficienza e la coerenza del processo di sviluppo.

4.6 Sviluppo backend: manipolazione dati

Durante lo sviluppo del backend per l'app Android, abbiamo utilizzato ChatGPT, GitHub Copilot e Tabnine per creare le varie classi, inclusi Activity, Fragment, Dialog, RecyclerView e altre componenti.

Tuttavia, l'utilizzo di ChatGPT per generare il codice completo ha presentato alcune sfide. Sebbene ChatGPT fosse in grado di fornire un codice completo, spesso il codice generato non funzionava correttamente e conteneva errori. Inol-

tre, il codice prodotto da ChatGPT non rispettava appieno le nostre richieste e non era sempre in linea con il contesto dell'app. Pertanto, abbiamo dovuto dedicare del tempo per correggere gli errori e adattare il codice alle nostre specifiche esigenze. Se ne riporta un esempio riguardante la creazione della classe CategoryDatasource fornita da ChatGPT a confronto con quella utilizzata nel progetto.

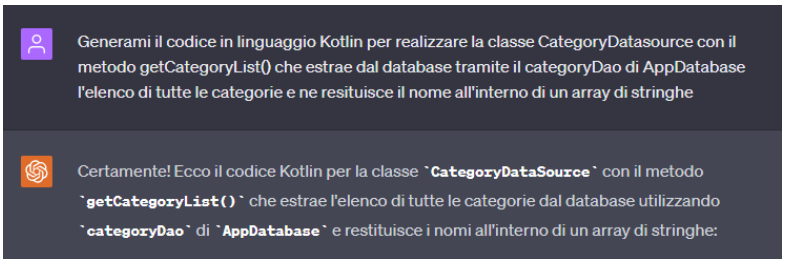


Figura 4.9: Richiesta codice per la creazione della classe CategoryDatasource.

```kotlin
// Codice prodotto da ChatGPT

// Definizione della classe CategoryDataSource
class CategoryDataSource(private val database:
   AppDatabase) {
    fun getCategoryList(): Array<String> {
        val categoryNames = database.categoryDao().
   getAllCategoryNames()
        return categoryNames.toTypedArray()
    }
}
```

Codice 4.7: Codice prodotto da ChatGPT per creazione della classe CategoryDatasource

```kotlin
// Codice utilizzato nel progetto

class CategoryDatasource(private val context: Context)
   {
    // Database instance
    private lateinit var db: AppDatabase

```

```
7       // Category Dao for executing queries
8       private lateinit var categoryDao: CategoryDao
9
10      // Method to return all categories from the
        database
11      fun getCategoryList(): Array<String> {
12          // Get database instance and category Dao
13          db = AppDatabase.getInstance(context)
14          categoryDao = db.categoryDao()
15          // Select all categories from database
16          val categoryList = categoryDao.getAll()
17
18          // String array of all category name
19          var categoryNameArray = emptyArray<String>()
20          categoryList.forEach { category ->
21              // Category name append
22              categoryNameArray += category.name
23          }
24
25          // Return category name array
26          return categoryNameArray
27      }
28  }
```

Codice 4.8: Codice utilizzato nel progetto per creazione della classe CategoryDatasource

Si può notare la stretta somiglianza due codici, infatti con poche e semplici modifiche è stato possibile ottenere il codice finale a partire dal codice generato.

GitHub Copilot è stato un prezioso alleato durante tutto il processo di sviluppo del backend. Copilot era in grado di comprendere il codice che stavamo scrivendo e forniva suggerimenti validi e utili. Abbiamo potuto fare affidamento su Copilot per ottenere soluzioni corrette e adattabili, riducendo significativamente il tempo dedicato a compiti ripetitivi come la manipolazione dei dati. Inoltre, Copilot riconosceva automaticamente dati simili e suggeriva la ripetizione del codice adattato a tali dati, semplificando ulteriormente il nostro processo di sviluppo.

In questa fase dello sviluppo, GitHub Copilot è stato un supporto essenziale, contribuendo a ridurre notevolmente i tempi associati alle operazioni ripetitive e fornendo soluzioni corrette e coerenti. Tuttavia, è importante prestare attenzione nella scelta dei suggerimenti di Copilot per assicurarsi che siano adeguati al contesto specifico del progetto.

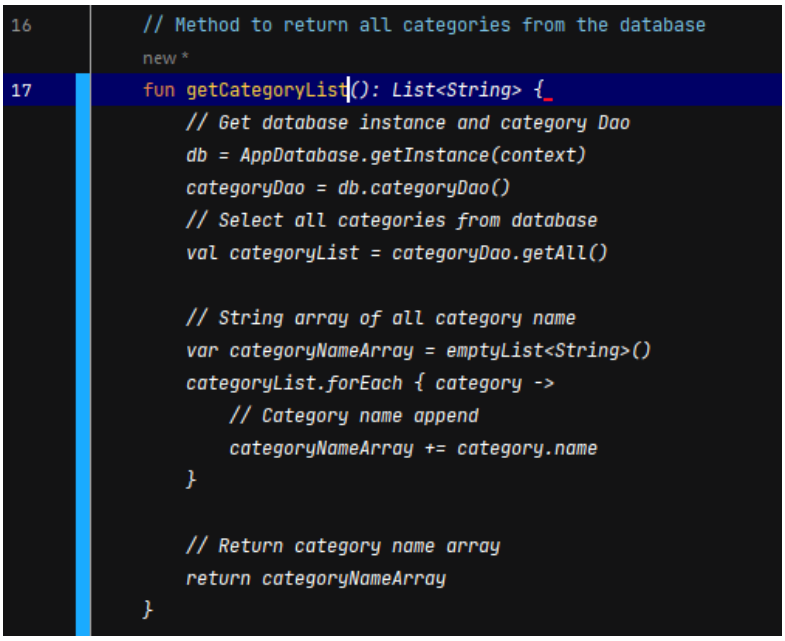


Figura 4.10: Suggerimento per la funzione fornito da GitHub Copilot.

D'altra parte, Tabnine si è rivelato praticamente inutile in questa fase del processo di sviluppo. Tabnine si limita a fornire suggerimenti per il completamento di parole o frasi senza avere una visione ampia del contesto di ciò che viene scritto. Spesso i suggerimenti di codice forniti da Tabnine erano imprecisi o non adatti alle nostre esigenze, rendendolo poco utile per le nostre attività di sviluppo.

Complessivamente, l'utilizzo combinato di ChatGPT, GitHub Copilot e Tabnine ha arricchito il nostro processo di sviluppo del backend. Sebbene ChatGPT presentasse limitazioni e Tabnine fosse poco utile, GitHub Copilot è stato fondamentale per fornire suggerimenti pertinenti, ridurre i tempi associati alle operazioni ripetitive e facilitare lo sviluppo delle varie componenti dell'app.

4.7 Fase di evoluzione estetica e funzionale dell'app

Una volta terminata la fase di sviluppo delle funzionalità dell'applicazione siamo passati alla fase di evoluzione estetica e funzionale dell'app, fase nella quale l'obiettivo è quello di apportare eventuali migliorie al fine di garantire un'esperienza utente sempre più soddisfacente, andando a migliorare l'aspetto estetico e la comodità di fruizione dell'applicazione.

Nell'ambito del miglioramento estetico, l'intervento delle intelligenze artificiali generative ha dimostrato di essere mediamente efficace. Le richieste riguardanti le migliorie estetiche alle IA generative hanno riscontrato da un lato la difficoltà da parte di queste, di generare soluzioni creative, tuttavia su richieste molto precise mirate a modificare l'estetica ChatGPT si è rilevata molto funzionale, aiutando a ottimizzare l'aspetto visivo dell'applicazione.

L'impiego delle IA generative per migliorare l'esperienza funzionale delle app si è dimostrato molto efficace, per discuterne riportiamo un'esempio di aiuto fornito da chatGPT nella generazione di codice necessario a rendere più funzionale l'applicazione nel suo utilizzo con il dispositivo orientato orizzontalmente. L'orientamento orizzontale è diventato un aspetto importante da considerare, visto l'aumento delle dimensioni degli schermi dei dispositivi mobili. Durante l'utilizzo dell'app in questa modalità, la presenza dell'appbar può risultare intrusiva e limitare la visualizzazione del contenuto, specialmente nella schermata del calendario dell'applicazione TimeWise.

L'eliminazione dell'appbar durante l'orientamento orizzontale presenta numerosi vantaggi. Innanzitutto, consente all'utente di concentrarsi completamente sul contenuto dell'app, evitando distrazioni visive superflue. Infine, l'aspetto estetico dell'app viene ulteriormente migliorato, poiché la rimozione dell'appbar durante l'orientamento orizzontale crea un'interfaccia più pulita e uniforme.

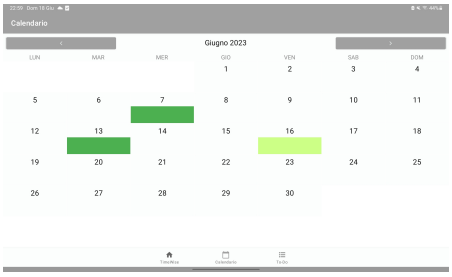
In risposta alla richiesta presentata riguardo la rimozione dell'appbar ChatGPT ha fornito il seguente codice Kotlin:

```
1 if (resources.configuration.orientation ==
    Configuration.ORIENTATION_LANDSCAPE && resources.
    configuration.screenHeightDp < 600) {
2           supportActionBar?.hide();
3 }
```

Codice 4.9: Codice Kotlin per rimuovere appbar nell'orientamento orizzontale

Osserviamo ora i risultati del codice prodotto da ChatGPT:

(a) Layout orizzontale per smartphone



(b) Layout orizzontale per tablet

Figura 4.11: Risultato della rimozione dell'appbar in orientamento orizzontale per schermi inferiori ai 600dp di altezza

E interessante osservare come la soluzione fornita da ChatGPT vada a differenziare il comportamento in base alle dimensioni del dispositivo utilizzato, quindi rimuovendo l'appbar in dispositivi di altezza inferiore a 600db e lasciandola per dispositivi di altezza superiore, quindi andando a rimuovere l'appbar solo dove la presenza di questa rappresenta una limitazione scomoda nell'uso dell'applicazione.

L'eliminazione dell'appbar durante l'orientamento orizzontale rappresenta solo uno dei molti modi in cui l'IA ha contribuito a migliorare l'esperienza utente e le funzionalità dell'app.

4.8 Debug del codice

Durante lo sviluppo dell'app, abbiamo utilizzato ChatGPT per affrontare gli errori di compilazione ed esecuzione che abbiamo incontrato. Quando Android Studio segnalava errori di codice, abbiamo copiato i messaggi di errore e li abbiamo forniti a ChatGPT insieme al codice della classe in cui si verificava l'errore di compilazione. In alcuni casi, ChatGPT ci ha fornito assistenza, ma ha mostrato anche alcune limitazioni.

Nel caso degli errori di compilazione, ChatGPT ha avuto difficoltà a comprendere le cause dell'errore in situazioni in cui il codice era elaborato e richiedeva una conoscenza dell'app nel suo complesso. Non avendo una visione globale

del codice dell'app, ChatGPT non era sempre in grado di fornire una soluzione accurata o completa per risolvere l'errore di compilazione.

Per quanto riguarda gli errori di esecuzione, l'approccio e i risultati ottenuti da ChatGPT erano simili. Tuttavia, anche in questo caso, alcuni errori erano difficili da comprendere e non sempre potevano essere risolti in modo diretto. ChatGPT non aveva la capacità di analizzare l'intero flusso di esecuzione dell'app e spesso mancava di conoscenza specifica per identificare la causa principale degli errori in esecuzione.

In conclusione, sebbene abbiamo sperimentato l'utilizzo di ChatGPT per risolvere gli errori di compilazione ed esecuzione, è importante sottolineare che il suo supporto era limitato in confronto alle sfide che affrontavamo. ChatGPT mostrava difficoltà nel comprendere codici complessi e non aveva la conoscenza completa dell'app per risolvere tutti gli errori di compilazione ed esecuzione. Pertanto, abbiamo dovuto fare affidamento su altre risorse e approcci per affrontare tali problemi, come la consultazione della documentazione di Android e l'analisi manuale del codice.

5

Problematiche riscontrate nell'utilizzo di IA generative

L'utilizzo di IA generative per scrivere codice è una pratica sempre più diffusa nell'industria del software. L'idea di utilizzare l'IA per generare codice è allettante per ridurre i tempi di sviluppo, migliorare la qualità del codice e aumentare l'efficienza del processo di sviluppo. Tuttavia, l'utilizzo di IA generative per scrivere codice può portare a diverse problematiche, sia tecniche che legali. In questo capitolo, analizzeremo le problematiche principali che possono sorgere nell'utilizzo di IA generative per scrivere codice.

5.1 Generazione di codice errato

ChatGPT è stato addestrato su una vasta quantità di testi provenienti da Internet, ma non possiede una conoscenza diretta della realtà come la percepiamo, quindi nonostante possa generare risposte coerenti e pertinenti, non ha una comprensione intrinseca dei concetti reali, delle verità oggettive o delle informazioni verificate.

Poiché ChatGPT manca di una comprensione formale della realtà, può risultare incapace di riconoscere la veridicità dei contenuti. Il modello genera risposte sulla base delle informazioni apprese dai testi di addestramento, che possono includere sia informazioni accurate che informazioni false o inventate. Se gli utenti forniscono dati o richieste basate su informazioni errate, ChatGPT

potrebbe rispondere in modo coerente con quelle informazioni errate, senza rendersene conto.

La mancanza di discernimento di ChatGPT rischia di generare contenuti falsi o inventati. Quando gli utenti formulano domande o richiedono informazioni, il modello elabora la richiesta in base a ciò che ha appreso, indipendentemente dalla veridicità delle informazioni fornite. Sebbene ChatGPT cerchi di fornire risposte coerenti, potrebbe creare classi o concetti che non esistono nella realtà, basandosi su associazioni di parole o modelli di addestramento.

Tra i vari contenuti che possono non essere veritieri ricade anche la generazione di codice, è molto comune imbattersi nell'utilizzo da parte di ChatGPT di librerie o metodi inesistenti rendendo la risposta generata inutilizzabile.

La responsabilità di utilizzare ChatGPT in modo consapevole ricade sugli utenti e sulle piattaforme che lo implementano. Gli utenti devono comprendere le limitazioni del modello e prendere le risposte con cautela, evitando di considerarle come verità definitive senza ulteriori verifiche.

5.2 Generazione di materiale protetto da Copyright

L'avvento delle tecnologie di intelligenza artificiale ha portato a significativi progressi nel campo della generazione di codice e delle risposte linguistiche coerenti. Tuttavia, insieme a questi sviluppi, sorgono importanti questioni legali e di utilizzo riguardanti i diritti di copyright associati al codice generato da modelli di intelligenza artificiale, come il notevole GPT-3.5 sviluppato da OpenAI. Il presente capitolo si propone di esplorare il quadro dei diritti di copyright in relazione al codice prodotto da tali modelli e di discutere le considerazioni riguardanti l'allenamento del modello e la dipendenza del codice generato da esso.

Il codice generato dai modelli di intelligenza artificiale, incluso il GPT-3.5, è soggetto alla protezione dei diritti di copyright. Il diritto d'autore si applica automaticamente a qualsiasi forma di espressione creativa, compreso il codice informatico. Gli autori di tali modelli, come OpenAI, detengono i diritti esclusivi sulla distribuzione, la riproduzione e la modifica del codice generato. Pertanto, il codice prodotto tramite il modello GPT-3.5 è soggetto alle leggi di copyright che proteggono la proprietà intellettuale.

L'uso del codice generato da modelli di intelligenza artificiale è soggetto a determinati limiti. Gli utenti possono utilizzare il codice generato per scopi

personali, come l'apprendimento, la sperimentazione e la comprensione concettuale. Tuttavia, l'uso commerciale del codice generato richiede il consenso appropriato dai detentori dei diritti d'autore, di solito l'organizzazione o l'entità che ha sviluppato il modello. È fondamentale rispettare le politiche di copyright e i diritti di proprietà intellettuale associati al codice generato dai modelli di intelligenza artificiale.

Inoltre, è importante considerare il processo di allenamento del modello GPT-3.5 e la dipendenza del codice generato da esso. I modelli di intelligenza artificiale come GPT-3.5 sono stati addestrati su enormi quantità di dati, inclusi testi protetti da copyright. Ciò solleva domande riguardanti la potenziale violazione dei diritti d'autore durante l'allenamento e l'uso di tali modelli. Gli sviluppatori e gli utenti devono fare attenzione a non utilizzare il codice generato per copiare o riprodurre in modo illecito opere protette da copyright.

Per affrontare queste questioni, le organizzazioni come OpenAI stanno adottando politiche e linee guida specifiche per l'utilizzo responsabile dei modelli di intelligenza artificiale e del codice generato da essi. Queste politiche includono la necessità di ottenere le autorizzazioni appropriate per l'uso commerciale del codice generato e l'implementazione di meccanismi per garantire il rispetto dei diritti di copyright.

In conclusione, i diritti di copyright si applicano al codice generato da modelli di intelligenza artificiale come GPT-3.5, e il suo utilizzo è soggetto a limitazioni specifiche. È fondamentale rispettare le leggi di copyright e ottenere le autorizzazioni necessarie per l'uso commerciale del codice generato. Inoltre, sia durante l'allenamento che nell'uso dei modelli di intelligenza artificiale, è importante evitare la violazione dei diritti d'autore e adottare pratiche responsabili per preservare la proprietà intellettuale e la legalità.

È importante sottolineare che le considerazioni riguardanti i diritti di copyright e le limitazioni d'uso non si applicano solo al codice generato da modelli di intelligenza artificiale come GPT-3.5, ma anche al testo generale prodotto da tali modelli. Il diritto d'autore si estende a tutte le forme di espressione creativa, compresi i testi e le risposte linguistiche generate da questi modelli. Pertanto, le stesse considerazioni legali e di utilizzo riguardanti i diritti di copyright e le autorizzazioni sono applicabili anche al testo generato dai modelli di intelligenza artificiale.

5.3 Generazione di codice proveniente da fonti open-source

L'impiego dell'intelligenza artificiale generativa nella scrittura di codice solleva diverse questioni importanti, tra cui quella relativa alla protezione del codice da violazioni di copyright. L'intelligenza artificiale ha la capacità di generare codice che potrebbe risultare simile o addirittura identico a quello sviluppato da altri programmatori, portando così a una violazione dei diritti di proprietà intellettuale. Un esempio evidente è quando l'IA genera codice identico a quello di un progetto open source, ma soggetto a licenze che richiedono la condivisione del codice in modalità open source. Questa situazione può causare non solo problemi legali, ma anche danni alla reputazione aziendale.

Per mitigare questa problematica, diventa fondamentale utilizzare l'intelligenza artificiale esclusivamente per generare codice che non violi i diritti di proprietà intellettuale altrui. Inoltre, è cruciale che l'IA sia in grado di riconoscere il codice che viola tali diritti e di evitare la sua generazione. In ogni caso, è indispensabile sottoporre il codice generato dall'IA a una scrupolosa revisione umana prima di utilizzarlo.

L'utilizzo dell'IA generativa per la scrittura di codice richiede un'adeguata cautela al fine di garantire il rispetto dei diritti di proprietà intellettuale e di evitare possibili conseguenze legali. È essenziale adottare meccanismi di rilevamento e prevenzione delle violazioni di copyright nell'IA per garantire una protezione adeguata delle opere intellettuali e preservare un ambiente di sviluppo etico ed equo.

5.4 Problemi di sicurezza

Un problema di rilevanza nell'impiego di intelligenza artificiale generativa per la scrittura di codice riguarda l'emergere di questioni di sicurezza connesse alla generazione di codice potenzialmente vulnerabile. L'IA potrebbe non riuscire a comprendere appieno le necessità di sicurezza del software, generando quindi codice soggetto a vulnerabilità.

Al fine di mitigare tale problematica, è imprescindibile che l'intelligenza artificiale sia in grado di apprezzare le esigenze di sicurezza del software e produrre codice che rispetti tali requisiti. Inoltre, è di fondamentale importanza

che il codice generato dall'IA venga attentamente sottoposto a una revisione umana da parte di esperti di sicurezza informatica, prima di essere impiegato.

Questa strategia mira a garantire un livello di sicurezza elevato nella produzione del codice, poiché l'intervento umano esperto può individuare e correggere eventuali vulnerabilità o debolezze che potrebbero essere sfuggite all'intelligenza artificiale. L'analisi critica da parte degli esperti è fondamentale per valutare la robustezza del codice generato, considerando i potenziali rischi per la sicurezza del sistema.

Inoltre, è consigliabile implementare una metodologia di revisione continua, in cui l'IA e gli esperti di sicurezza lavorino in collaborazione per affinare il processo di generazione del codice. Questo permette di acquisire un'appropriata conoscenza delle peculiarità e dei requisiti di sicurezza specifici del software in questione, affiancando l'efficienza dell'IA alla competenza umana.

In aggiunta alla problematica della sicurezza, un ulteriore aspetto di rilievo nell'utilizzo dell'intelligenza artificiale generativa per la scrittura del codice è la tendenza dell'IA a commettere gli stessi errori che gli esseri umani fanno durante il processo di sviluppo. Dato che l'intelligenza artificiale è addestrata su un vasto insieme di dati provenienti da codice sorgente creato da sviluppatori umani, è suscettibile ad apprendere e replicare le imperfezioni e le limitazioni tipiche della programmazione umana.

Poiché l'IA assorbe e modella il comportamento umano nell'elaborazione dei dati, può ricadere negli stessi difetti che spesso affliggono i programmatori umani, come la presenza di bug, codice inefficiente o scarsa manutenibilità. Questo può comportare una minaccia per l'affidabilità e la qualità del software generato dall'IA.

Al fine di mitigare questa problematica, è cruciale adottare una duplice strategia. Innanzitutto, è necessario migliorare la qualità dei dati utilizzati per l'addestramento dell'IA. Ciò implica la cura nella selezione di dataset di alta qualità, corretti e ben strutturati, al fine di fornire all'IA una base di conoscenze solida e priva di errori. Inoltre, è fondamentale perseguire un costante miglioramento della qualità dei codici generati dall'IA, sottoponendoli a un accurato processo di revisione e valutazione umana.

5.5 Problemi di compatibilità

Quando l'IA viene utilizzata per scrivere codice, essa può non essere in grado di comprendere appieno le esigenze dell'utente o il contesto in cui il software deve operare. Pur avendo la capacità di generare codice funzionante, l'IA potrebbe non considerare gli standard di programmazione utilizzati nel progetto o le interazioni con altre componenti software. Questo può portare a problemi di compatibilità e nell'integrazione del software generato nel progetto.

Ad esempio, se l'IA genera una porzione di codice che utilizza una libreria o una sintassi specifica che non è supportata dal resto del software, si verificheranno errori durante la compilazione o l'esecuzione. Anche se il codice generato può sembrare corretto dal punto di vista sintattico, potrebbe non rispettare le regole o le convenzioni stabilite nel progetto, rendendolo difficile da integrare con altre parti del software o da mantenere in futuro.

Inoltre, l'IA potrebbe non essere in grado di comprendere appieno le aspettative dell'utente riguardo al codice generato. L'utente può avere requisiti specifici in termini di funzionalità, prestazioni o sicurezza del software, ma l'IA potrebbe non essere in grado di cogliere appieno tali esigenze. Di conseguenza, il codice generato potrebbe non soddisfare le aspettative dell'utente o potrebbe addirittura presentare errori o vulnerabilità che compromettono la qualità e la sicurezza del software.

Questi problemi possono essere mitigati attraverso un'attenta validazione e revisione umana del codice generato dall'IA. Gli sviluppatori devono essere consapevoli delle limitazioni dell'IA generativa e assumere la responsabilità di garantire che il codice sia compatibile e soddisfi le aspettative del progetto. È fondamentale effettuare un rigoroso controllo di qualità e condurre test approfonditi per individuare eventuali incompatibilità o problemi di integrazione prima dell'implementazione del codice generato.

5.6 Conoscenza temporale limitata

E' importante riconoscere le limitazioni delle IA generative che possiedono una conoscenza limitata ad un determinato istante temporale, soprattutto quando si tratta della generazione di codice. Queste IA, pur essendo straordinarie

nel loro potenziale creativo, tendono ad utilizzare librerie e framework obsoleti, non tenendo conto degli sviluppi più recenti e delle best practice attuali.

Un esempio emblematico di questa sfida è rappresentato da ChatGPT, un modello di intelligenza artificiale conversazionale, il quale ha una conoscenza limitata fino ad un determinato anno. Mentre ChatGPT può essere estremamente utile per una vasta gamma di compiti comunicativi, nel contesto specifico della generazione di codice, il suo limitato know-how può comportare la produzione di risultati non ottimali.

Il codice sorgente è un universo in costante evoluzione, in cui nuove librerie, framework e strumenti emergono costantemente per migliorare l'efficienza, la sicurezza e la manutenibilità del software. Le IA generative che non tengono il passo con queste evoluzioni rischiano di produrre codice che fa affidamento su librerie obsolete o metodi superati, portando a potenziali inefficienze, vulnerabilità e difficoltà di manutenzione.

La generazione di codice affidabile richiede una solida comprensione del panorama tecnologico attuale, dei linguaggi di programmazione, dei framework e delle librerie più aggiornate. Le IA generative che non hanno accesso a questa conoscenza aggiornata possono generare soluzioni che potrebbero funzionare solo su versioni specifiche di librerie o framework, risultando incompatibili con le ultime versioni e quindi poco pratiche per un ambiente di sviluppo moderno.

È fondamentale riconoscere che le IA generative con una conoscenza limitata rappresentano ancora uno strumento prezioso per l'ispirazione e la generazione di bozze di codice. Tuttavia, per produrre soluzioni pratiche e all'avanguardia, è necessario integrare l'expertise umana e l'aggiornamento costante sulle nuove tecnologie e best practice. Solo combinando l'intelligenza artificiale con l'esperienza umana si potranno ottenere risultati di alta qualità e mantenere il passo con i rapidi progressi del settore dello sviluppo software.

Nuovi strumenti, come Bard di Google, che effettuano ricerche istantanee su Internet potrebbero dimostrarsi una soluzione a questo problema. Tuttavia, durante i nostri esperimenti non siamo riusciti a testare quest'ultimo mezzo in quanto uscito nell'ultimo periodo.

6

Obiettivi raggiunti

6.1 Produttività

Le IA generative in generale si propongono principalmente come acceleratori di produttività. Ciò risulta essere ancora più evidente per strumenti di assistenza alla programmazione, come GitHub Copilot e TabNine. Questi forniscono costantemente statistiche sul tempo fatto risparmiare ai programmatori, tanto che Tabnine arriva ad avere una pagina dedicata a queste statistiche per il singolo sviluppatore.

La scrittura di codice boilerplate è una parte inevitabile del processo di sviluppo del software, ma può essere estremamente noiosa e ripetitiva. Gli strumenti IA generativi come GitHub Copilot offrono la possibilità di generare automaticamente porzioni di codice, riducendo notevolmente il tempo necessario per completare questa attività. Ad esempio, Copilot può suggerire in modo intelligente il completamento di un blocco di codice comune, come un ciclo for o una funzione di ordinamento, basandosi sul contesto e sui modelli di codice precedenti. Ciò consente ai programmatori di concentrarsi su compiti più creativi e di valore aggiunto, migliorando l'efficienza complessiva del processo di sviluppo.

La ripetitività delle attività di sviluppo, come la scrittura di codice ridondante, può portare ad una sensazione di noia e frustrazione tra i programmatori. Questo può influire negativamente sulla loro produttività e sulla qualità del lavoro svolto. Gli strumenti IA generativi offrono la possibilità di ridurre la

ripetitività automatizzando alcune delle attività più tediose. Ciò consente ai programmatori di concentrarsi su compiti più stimolanti e creativi, migliorando la loro soddisfazione nel processo di sviluppo.

Inoltre, quando i programmatori vedono ridursi il tempo impiegato per operazioni ripetitive grazie all'uso di strumenti di IA generativi, si può instaurare un senso di gratificazione e motivazione. Questo può contribuire a mantenere alta la loro motivazione e interesse per il lavoro, favorendo un ambiente di sviluppo più produttivo. La possibilità di affrontare problemi più complessi e interessanti può aumentare l'entusiasmo dei programmatori e favorire la nascita di soluzioni innovative.

Alla luce di quanto esposto, concordiamo con l'opinione che gli strumenti di IA generativi, come GitHub Copilot, agiscano effettivamente come acceleratori della produttività per i programmatori e soprattutto aumentino la soddisfazione dello sviluppatore. Ma riteniamo che non sia oro tutto ciò che luccica, infatti, se la fase di sviluppo risulta meno stressante se non si procede con cautela si rischia di utilizzare tutto il tempo risparmiato, se non di più, ad effettuare il debugging del codice scritto. Dietro alle statistiche di tempo guadagnato non si considera il tempo perso a sistemare piccoli errori introdotti dall'IA. Quest'ultima essendo allenata su codice umano tende, infatti, a commettere errori simili. E trattandosi in generale di codice non completamente scritto dallo sviluppatore umano, il rilevamento di problemi risulta particolarmente lungo a causa della necessità di avere una comprensione completa di quanto scritto dall'IA per poter risolvere il bug.

In conclusione, l'IA generativa può diventare un alleato potente se utilizzata correttamente, combinando l'intelligenza artificiale con l'esperienza umana per ottenere risultati ottimali nello sviluppo software.

6.2 Qualità del codice

L'introduzione delle intelligenze artificiali generative nel campo dello sviluppo del software ha sollevato diverse domande e dubbi sulle loro capacità di migliorare la qualità del codice. Una delle sfide principali affrontate dagli sviluppatori è la produzione di codice di alta qualità, che sia privo di errori e facilmente manutenibile. In questo contesto, ci si chiede se gli strumenti di IA generativa siano in grado di fornire suggerimenti di codice che possano effettivamente contribuire a raggiungere questi obiettivi.

Uno dei problemi comuni nel processo di sviluppo del software è la presenza di codice di scarsa qualità. Questo può essere causato da diverse ragioni, come mancanza di esperienza, tempi di consegna stretti o scarsa conoscenza delle migliori pratiche. Gli sviluppatori possono commettere errori nella scrittura del codice, causando problemi di funzionalità, efficienza o sicurezza. Inoltre, la manutenibilità del codice può essere compromessa se non vengono seguiti gli standard di progettazione e sviluppo.

Riteniamo che nella maggior parte dei casi, l'utilizzo di codice generato da IA tende ad abbassare la qualità del codice nei progetti. Ciò è principalmente attribuibile alla tendenza umana ad accettare passivamente il codice generato, senza approfondirne appieno la logica e senza considerare l'importanza delle buone pratiche, quali i design pattern, che spesso vengono trascurati dall'IA.

Questo può essere dovuto a vari fattori, come la mancanza di tempo, la mancanza di conoscenza approfondita del codice generato dall'IA o la fiducia eccessiva nella sua correttezza a causa del suo funzionamento apparentemente corretto. La tendenza a dare per scontato che il codice generato sia corretto può portare a una mancanza di attenzione nella revisione e nel controllo di qualità del codice.

Inoltre, l'IA potrebbe generare codice che risolve il problema specifico per cui è stata addestrata, ma potrebbe non prendere in considerazione altri aspetti importanti come la performance, la sicurezza o la manutenibilità del codice. Questo può portare a un codice che funziona in modo apparentemente corretto ma che presenta vulnerabilità o inefficienze che potrebbero essere evitate con una progettazione più accurata.

È importante sottolineare che l'IA non è intrinsecamente difettosa nella generazione di codice di scarsa qualità. L'IA è uno strumento potente che può essere utilizzato in modo efficace se il programmatore comprende le sue limitazioni e le integra correttamente nel processo di sviluppo del software. Il punto critico è che il programmatore deve assumersi la responsabilità di approfondire il codice generato e di applicare le buone pratiche necessarie per produrre un codice di qualità.

6.3 Capacità creative

Nel corso degli ultimi anni, l'Intelligenza Artificiale generativa ha fatto enormi passi avanti nella generazione di nuove idee e concetti. Questa tecnologia ha

dimostrato di essere straordinariamente efficace nel fornire soluzioni innovative e nello stimolare la creatività umana. Tuttavia, è importante riconoscere che ci sono ambiti in cui l'IA generativa mostra limiti significativi, come nella creazione di elementi grafici piacevoli per le applicazioni o nell'esprimere opinioni.

Innanzitutto, l'IA generativa si basa su algoritmi che analizzano un'enorme quantità di dati esistenti per identificare schemi e tendenze. Questo approccio funziona molto bene quando si tratta di generare idee concettuali o di proporre soluzioni innovative per problemi specifici. Tuttavia, quando si tratta di creare elementi grafici piacevoli, come l'interfaccia utente di un'applicazione o la grafica di un sito web, l'aspetto estetico diventa fondamentale. L'IA generativa, pur potendo generare output coerenti, potrebbe non avere la sensibilità estetica e la comprensione delle preferenze umane necessarie per produrre un design visivamente attraente e accattivante. L'occhio umano e il gusto artistico rimangono indispensabili per tali compiti.

Inoltre, l'IA generativa è intrinsecamente priva di esperienze personali e opinioni soggettive. Poiché l'IA è programmata per analizzare e replicare i dati forniti, non ha una coscienza propria o una prospettiva individuale. Di conseguenza, richiedere all'IA di fornire un'opinione o un punto di vista su una questione specifica risulta molto difficile. L'IA può elaborare dati e fornire informazioni oggettive, ma manca della comprensione e dell'esperienza umana necessarie per esprimere giudizi o opinioni che riflettano una vera consapevolezza individuale.

In conclusione, l'IA generativa ha dimostrato il suo valore nella generazione di nuove idee e concetti, stimolando la creatività umana e offrendo soluzioni innovative. Tuttavia, quando si tratta di creare elementi grafici piacevoli o fornire opinioni, l'IA generativa presenta limitazioni significative. L'estetica e l'esperienza umana rimangono fondamentali per garantire un design visivamente accattivante e per ottenere un punto di vista autentico e personale. Pertanto, l'utilizzo dell'IA generativa dovrebbe essere integrato con l'intervento umano in questi ambiti, in modo da sfruttare al meglio le forze complementari delle macchine e dell'intelligenza umana.

7

Quantificazione del tempo

Nell'era attuale, caratterizzata da un'accelerazione senza precedenti del progresso tecnologico, è fondamentale comprendere l'importanza di ridurre il tempo impiegato per la realizzazione di soluzioni software.

Infatti, l'evoluzione del mercato e le esigenze dei clienti richiedono una risposta rapida e tempestiva. Le aziende che sono in grado di sviluppare software in tempi brevi, senza compromettere la qualità, si trovano in una posizione vantaggiosa nel raggiungimento del successo. La rapidità di consegna permette di ottenere un vantaggio competitivo, di anticipare i bisogni degli utenti e di rispondere prontamente alle mutevoli condizioni di mercato.

Quindi, la quantificazione del tempo svolge un ruolo fondamentale nell'ambito dello sviluppo software. Comprendere come gli strumenti di IA influenzano la quantificazione del tempo è di cruciale importanza per valutare il loro impatto complessivo sull'intero processo di sviluppo.

Il capitolo si articola in diverse sezioni che approfondiscono gli aspetti chiave legati alla quantificazione del tempo nell'utilizzo di ChatGPT, GitHub Copilot e Tabnine. Nello specifico, esaminando la dicotomia tra il tempo guadagnato e il tempo perso utilizzando questi strumenti.

Infine, dedicheremo attenzione al tempo dedicato a ciascuna fase dello sviluppo utilizzando questi strumenti di IA. Esaminando il loro impatto nel processo di sviluppo dell'applicazione Android.

7.1 Tempo guadagnato vs perso

Gli strumenti di IA generativi, in particolare ChatGPT, Tabnine e GitHub Copilot, si sono dimostrati utili nello sviluppo della nostra applicazione Android. Uno degli obiettivi principali tenuti in considerazione è stato quantificare il tempo risparmiato rispetto al tempo perso durante il processo di sviluppo utilizzando questi assistenti. L'IA generativa ci ha fatto guadagnare una quantità di tempo notevole nella fase di generazione del codice, tuttavia l'utilizzo di questi strumenti ha introdotto alcune complicazioni che hanno richiesto ulteriori operazioni di adattamento e correzione del codice prodotto per renderlo funzionante.

7.2 ChatGPT

Tra i molteplici aspetti presi in considerazione, va sottolineato che ChatGPT ha dimostrato di offrire il vantaggio maggiore soprattutto nell'ambito della fornitura di un input che fungesse da base per il codice. Questo approccio ha permesso di ridurre considerevolmente il tempo di ricerca iniziale, garantendo una maggiore efficienza nello sviluppo del progetto.

Tuttavia, è importante sottolineare che ChatGPT ha evidenziato alcune limitazioni significative nelle fasi successive del processo. In particolare, si è riscontrato che spesso i consigli forniti da ChatGPT risultavano incoerenti o contenevano errori. Questa mancanza di coerenza e precisione ha rappresentato una sfida per gli sviluppatori che dovevano continuamente correggere e adattare le indicazioni offerte da ChatGPT.

Di conseguenza, la necessità di apportare modifiche al codice generato da questi strumenti ha contribuito ad annullare parzialmente il tempo risparmiato nelle fasi precedenti. È fondamentale, quindi, valutare attentamente i benefici e gli svantaggi dell'uso di questo strumento.

7.3 GitHub Copilot come assistente

L'utilizzo di GitHub Copilot si è dimostrato l'aiuto più prezioso tra gli strumenti analizzati grazie ai suggerimenti forniti che spesso hanno contribuito a risparmiare tempo per scrivere codice. I suggerimenti forniti da GitHub Co-

pilot hanno agevolato il team di sviluppo, riducendo il tempo necessario per affrontare piccole problematiche tecniche e semplificando le attività di codifica.

Inoltre, va sottolineato che GitHub Copilot non solo ha contribuito a ridurre i tempi di stesura del codice, ma ha anche fornito un significativo supporto nella fase di pulizia e commento del codice. L'automatizzazione dei commenti ha consentito di risparmiare tempo, poiché i commenti stessi si adattavano autonomamente al codice che veniva commentato e spesso mantenevano una struttura simile nei diversi file del progetto.

Abbiamo quindi ritenuto GitHub Copilot il miglior alleato e ne abbiamo potuto sfruttare a pieno le capacità.

7.4 Tabnine a completamento del testo

Tabnine, a differenza di ChatGPT e GitHub Copilot, ha avuto un impatto trascurabile nella riduzione dei tempi di sviluppo del codice e di commento, principalmente a causa delle sue scarse capacità rispetto al linguaggio Kotlin e XML. Tabnine si limitava a suggerire le prossime parole da scrivere o a completare quelle correnti. Tuttavia, tali suggerimenti proposti erano spesso poco correlati al contesto in cui ci si trovava a scrivere.

Nel caso del codice, i suggerimenti di Tabnine erano spesso errati o poco utili. Inoltre, sono stati sempre generici e brevi.

Invece, nel caso dei commenti, i suggerimenti di Tabnine erano parzialmente accettabili, poiché riusciva a fornire completamenti automatici di frasi comuni. Questo ha in parte contribuito a ridurre il tempo necessario per la scrittura dei commenti, ma non ha raggiunto un livello di personalizzazione e di comprensione del contesto adeguati. E in ogni caso, Tabnine ha comunque introdotto un apporto poco rilevante nella fase di commento del codice rispetto a GitHub Copilot.

7.5 Tempo dedicato ad ogni fase dello sviluppo

Durante lo sviluppo dell'app Android, ogni fase ha richiesto un certo tempo, il quale può essere quantificato in percentuale rispetto al monte ore totale. Le fasi principali analizzate nel seguito sono:

1. Progettazione del database

2. Sviluppo frontend: layout grafica
3. Sviluppo backend: manipolazione dati
4. Debug del codice

7.5.1 Progettazione del database

Nella fase di progettazione del database, che ha richiesto approssimativamente il 10% del monte ore, l'utilizzo di ChatGPT non si è rivelato particolarmente utile per la progettazione concettuale. Come già discusso in precedenza, ChatGPT non possiede una comprensione completa della progettazione di un database, portando a possibili problemi di consistenza. Pertanto, non è stato possibile ottenere un risparmio di tempo significativo in questa fase che abbiamo dovuto strutturare manualmente. Una volta definita la progettazione concettuale, sia ChatGPT che GitHub Copilot hanno contribuito in modo efficace all'implementazione in linguaggio Kotlin del database con la libreria Room. Tuttavia, dettagli specifici come chiavi esterne e vincoli sulle operazioni di inserimento, aggiornamento ed eliminazione non venivano forniti, il che ha comportato la necessità di dedicare tempo alla ricerca di soluzioni online e nella documentazione.

7.5.2 Sviluppo frontend: layout grafica

Nella fase di sviluppo del frontend, in particolare della grafica dei layout, che ha richiesto approssimativamente il 20% del monte ore, l'utilizzo di ChatGPT ha consentito di ottenere layout XML di base. Ma non è stato capace di spingersi oltre, principalmente per l'incapacità di comprendere il gusto estetico umano. Quindi anche questa fase è stata fatta prevalentemente in modo manuale.

7.5.3 Sviluppo backend: manipolazione dati

Nella fase di sviluppo del backend, che ha richiesto approssimativamente il 30% del monte ore, sia ChatGPT che GitHub Copilot hanno contribuito a ridurre i tempi di sviluppo. Il compito che ha richiesto il numero maggiore di ore è stata la manipolazione dei dati, nonostante alcune complicazioni introdotte da questi strumenti che a volte richiedevano ulteriore tempo per adattare e correggere il

codice. Essendo questa fase cruciale per il corretto funzionamento dell'app e per garantirne la consistenza, ha richiesto più tempo rispetto alla fase precedente.

7.5.4 Debug del codice

Infine, la maggior parte del tempo è stata impiegata nella fase di debug del codice, che ha richiesto approssimativamente il 40% del monte ore. Durante il processo di debugging, ci si può trovare di fronte a problematiche più complesse che richiedono una comprensione approfondita di un'area specifica o di un argomento specifico. In questi casi, ChatGPT ha dimostrato di non essere sempre in grado di fornire una risposta adeguata, inventando spesso classi o generando risposte incoerenti. Pertanto, per superare tali problematiche, la soluzione migliore consiste nell'approfondire l'argomento coinvolto utilizzando risorse esterne come documentazione, internet o fonti di conoscenza specializzate. Questo approccio permette ai programmatori di acquisire una comprensione più approfondita della sezione di codice problematica e di identificare soluzioni efficaci. La ricerca esterna consente di colmare le lacune di conoscenza di ChatGPT e di fornire al sistema le informazioni necessarie per affrontare la problematica in modo accurato ed efficiente.

D'altra parte, ChatGPT dimostra un notevole vantaggio nell'affrontare problemi di debug di dimensioni minori, riducendo in modo drastico il tempo di ricerca e risultando in alcuni casi più veloce di una ricerca su Internet.

8

Intelligenze generative nello sviluppo di codice: analisi dei pro e dei contro

8.1 Assistenti IA in progetti complessi

L'integrazione dell'Intelligenza Artificiale nello sviluppo delle app Android può portare a risultati straordinari, ma solo se applicata nel modo appropriato e con una piena comprensione dei suoi punti di forza e delle sue limitazioni. L'obiettivo di questo capitolo è la discussione dell'utilizzo di strumenti di Intelligenza Artificiale, come ChatGPT, nello sviluppo di app Android e in progetti complessi in generale. È importante riconoscere che, sebbene questi sistemi possano offrire vantaggi significativi in determinati contesti, vi sono delle limitazioni che potrebbero pregiudicarne l'efficacia nel contesto dello sviluppo di app.

Una delle principali sfide associate all'utilizzo di strumenti di IA come ChatGPT è la mancanza di conoscenza completa del contesto nel quale si deve integrare il codice da generare. Quando si tenta di utilizzare tali sistemi per supportare lo sviluppo di app Android, ci si può imbattere in problematiche legate alla generazione di classi o pacchetti inesistenti. Questo può rendere difficile o addirittura impossibile svolgere un lavoro accurato ed efficace e in questo caso il codice generato risulta totalmente inutile.

Tuttavia, c'è un altro aspetto da considerare. Se invece si chiede a ChatGPT di produrre porzioni di codice che siano indipendenti da un contesto specifico,

allora il suo utilizzo può rivelarsi estremamente utile. In questi casi, ChatGPT genera codice di alta qualità e funzionalità, dato che non necessità di avere una conoscenza dettagliata del contesto nel quale verrà utilizzato il codice da generare.

È fondamentale capire che l'utilizzo di strumenti di IA per lo sviluppo di app Android non è una soluzione universale. È importante valutare attentamente le circostanze e comprendere i limiti e le potenzialità di questi strumenti. L'esperienza e la competenza degli sviluppatori umani al momento sembrano insostituibili, in quanto solo loro sono in grado di comprendere appieno l'architettura del codice e le esigenze specifiche del progetto.

In conclusione, sebbene i sistemi come ChatGPT possano rappresentare una risorsa preziosa nello sviluppo di app Android e in progetti complessi, è importante essere consapevoli delle loro limitazioni. Quando si tratta di affrontare il codice esistente, la mancanza di conoscenza completa potrebbe portare a risultati errati o confusi. Tuttavia, quando si tratta di generare porzioni di codice indipendenti dal contesto, questi strumenti possono dimostrarsi straordinariamente utili.

8.2 Riduzione dello sforzo cognitivo

L'integrazione dell'Intelligenza Artificiale nel processo di sviluppo software offre una serie di vantaggi significativi, tra cui la riduzione della difficoltà cognitiva associata alla programmazione. L'IA può essere utilizzata in diversi modi per semplificare e ottimizzare il lavoro dei programmatori. In questo capitolo, esamineremo alcuni esempi di come l'IA può ridurre la fatica mentale e migliorare l'efficienza nel processo di programmazione.

Uno dei modi principali in cui l'IA può facilitare la programmazione è attraverso l'offerta di suggerimenti intelligenti durante la scrittura del codice. Gli strumenti di sviluppo basati sull'IA possono anticipare e completare parti di codice, suggerire correzioni di sintassi e fornire consigli sulla struttura del programma. Questa funzionalità aiuta i programmatori a superare la difficoltà di ricordare la sintassi corretta o la logica di programmazione complessa. Grazie ai suggerimenti intelligenti è possibile risparmiare tempo prezioso e ridurre lo sforzo mentale richiesto per scrivere codice di alta qualità.

Un altro modo in cui l'IA può ridurre la difficoltà cognitiva è attraverso l'automazione delle attività ripetitive. Spesso, i programmatori si trovano a dover

affrontare compiti noiosi e ripetitivi, come la creazione di interfacce utente o l'accesso ai dati. L'IA può automatizzare tali attività, consentendo ai programmatori di concentrarsi su compiti più creativi e complessi. I generatori di codice basati sull'IA, ad esempio, possono generare automaticamente parti di codice comuni, risparmiando tempo e sforzo mentale. Questa automazione delle attività ripetitive consente ai programmatori di dedicare le proprie energie a problemi più interessanti e stimolanti.

Il processo di debug può rappresentare una grande sfida mentale per i programmatori. Fortunatamente, l'IA può fornire assistenza preziosa in questa fase. L'IA può aiutare a identificare e localizzare i bug nel codice, suggerendo possibili punti di errore o eseguendo analisi statistiche per individuare potenziali problemi. Grazie all'assistenza dell'IA nel debugging, i programmatori possono accelerare il processo di ricerca degli errori e ridurre la fatica mentale associata alla loro individuazione. Ciò consente di risparmiare tempo e garantire una maggiore precisione nella risoluzione dei problemi.

Infine, l'IA può fornire un supporto prezioso nella ricerca di informazioni e nella documentazione di riferimento. Spesso, i programmatori si trovano a dover cercare manualmente informazioni tecniche o consultare documentazione specifica per completare una determinata attività. L'IA può semplificare questo processo fornendo assistenza nella ricerca di informazioni attraverso chatbot intelligenti o motori di ricerca avanzati. I programmatori possono porre domande specifiche agli strumenti basati sull'IA o cercare documentazione in modo più efficiente. Questo riduce la difficoltà cognitiva associata alla necessità di ricordare dettagli specifici o alla ricerca manuale di risposte a domande tecniche.

In conclusione, l'integrazione dell'IA nel processo di sviluppo software può ridurre la difficoltà cognitiva e migliorare l'efficienza dei programmatori, questo attraverso suggerimenti intelligenti, automazione delle attività ripetitive, assistenza nel debugging e supporto nella ricerca di informazioni. Quindi l'IA semplifica il lavoro dei programmatori, consentendo loro di concentrarsi su compiti più complessi e creativi.

8.3 Qualità del codice

La sperimentazione ripetuta nell'ambito della scrittura di codice mediante l'ausilio dell'intelligenza artificiale, senza alcuna pregressa conoscenza dell'argomento trattato, ha chiaramente evidenziato l'esistenza di una correlazione

diretta tra l'assenza di competenza specifica e la qualità scadente del codice prodotto. Tale fenomeno si manifesta attraverso una funzionalità base dell'IA, che permette al codice di eseguire le operazioni richieste senza tuttavia rispettare le best practice che costituiscono i pilastri fondamentali dell'ingegneria del software.

Un aspetto rilevante di tale scenario è la tendenza intrinseca dell'intelligenza artificiale a focalizzarsi sull'obiettivo finale, senza considerare gli approcci e le metodologie consolidati nel campo della programmazione. Ad esempio, l'IA, se non specificamente istruita in merito, tende ad ignorare del tutto le tecniche di progettazione, come i design pattern, che rappresentano soluzioni strutturate e collaudate a problemi ricorrenti nell'ambito dello sviluppo software.

Questo atteggiamento della macchina, seppur guidato dalla logica computazionale e dal raggiungimento dell'obiettivo specificato, si traduce in un approccio superficiale, quindi concentrato sulla scrittura del codice, tuttavia privo della ricchezza e della profondità concettuale che derivano dalla conoscenza specialistica e dall'esperienza umana nel settore. La mancanza di una solida base di conoscenza nell'argomento trattato influisce negativamente sulle scelte progettuali, sulle performance dell'applicazione e sulla manutenibilità del codice stesso.

Questa constatazione dimostra che l'intelligenza artificiale, pur offrendo indubbi vantaggi nel contesto dello sviluppo software, non può sostituire né prescindere dalla figura umana, con la sua capacità di comprendere le sfumature, di adattarsi a contesti complessi e di applicare principi architetturali consolidati. L'expertise umana, maturata attraverso l'apprendimento e l'esperienza, rimane un elemento fondamentale per la scrittura di codice di qualità superiore, capace di seguire le best practice e di garantire la scalabilità, la robustezza e l'efficienza richieste nei progetti software.

In conclusione vogliamo riportare quanto osservato nel progetto riguardo la scrittura di codice tramite intelligenza artificiale senza alcuna conoscenza specialistica in determinate fasi dello sviluppo dell'applicazione ed effettuando una successiva analisi del codice generato dopo aver approfondito l'argomento a lezione. Questo ha messo in luce l'inadeguatezza dell'approccio automatizzato nel rispettare i canoni di qualità che caratterizzano l'ingegneria del software. L'intervento umano, con la sua competenza e consapevolezza, continua ad essere irrinunciabile per garantire un codice che sia non solo funzionante, ma anche strutturato, manutenibile e conforme alle migliori pratiche dell'industria.

9 Riflessioni

9.1 ChatGPT-3 vs ChatGPG-3.5

ChatGPT-3.5 è una versione successiva all'architettura GPT-3, che si è focalizzata sulla precisione e sull'eliminazione di errori o incongruenze presenti nella versione precedente.

9.1.1 Sviluppo di codice

Uno dei miglioramenti più significativi di ChatGPT-3.5 rispetto alla versione 3 è la maggiore precisione nel fornire risposte e informazioni pertinenti. Grazie a un addestramento più avanzato e a una raccolta dati più recente, ChatGPT-3.5 ha una comprensione più accurata delle domande e dei contesti che gli vengono presentati. Ciò significa che le risposte fornite sono generalmente più affidabili e pertinenti alle richieste degli utenti.

Inoltre, l'errore menzionato riguardante l'utilizzo di librerie Android inesistenti nella versione 3.5 è significativamente meno presente rispetto alla versione 3, dove si presentava in modo più ricorrente e fuorviante. Tuttavia, rimane ancora un problema non completamente risolto. Nella versione 3, essendo addestrata su un vasto corpus di testo proveniente da varie fonti, potevano verificarsi risposte errate o inconsistenti in merito all'uso di librerie Android. Tuttavia, con l'ulteriore addestramento e il processo di sviluppo iterativo, questi errori sono stati ridotti in ChatGPT-3.5. Ora, l'algoritmo ha una migliore comprensione del-

le librerie Android esistenti e delle loro funzionalità, attenuando la possibilità di fornire informazioni errate o inesistenti in merito a esse.

La differenza in termini di prestazioni nello sviluppo di codice deve essere comunque uno spunto per ricordare quanto questa tecnologia possa ancora migliorare nel tempo.

9.1.2 Scrittura di testo

Con il passaggio da ChatGPT versione 3 alla versione 3.5, la capacità di generare testo è notevolmente migliorata, in particolare per quanto riguarda la riduzione dei pattern ripetitivi che erano facilmente identificabili. Prima del miglioramento, molti dei ragionamenti prodotti da ChatGPT potevano seguire una struttura prevedibile: tesi, antitesi, conclusioni e sintesi.

Nella versione precedente, il modello tendeva a generare una tesi iniziale seguita da un'antitesi, che poteva essere una considerazione o un punto di vista opposto alla tesi. Successivamente, il modello forniva delle conclusioni basate sul confronto tra la tesi e l'antitesi e infine offriva una sintesi dei punti principali trattati.

Tuttavia, con l'introduzione della versione 3.5 di ChatGPT, OpenAI ha lavorato per mitigare questo problema e rendere i ragionamenti generati dal modello più coerenti e meno prevedibili. Grazie all'addestramento su un dataset più vasto e diversificato, il modello è stato esposto a una maggiore varietà di strutture e stili di testo. Ciò ha permesso di superare in gran parte i pattern ripetitivi che erano comuni nella versione precedente.

Ora, ChatGPT versione 3.5 ha una maggiore capacità di produrre ragionamenti più originali e sorprendenti, senza rimanere vincolato a una rigida struttura di tesi, antitesi, conclusioni e sintesi. Il modello è in grado di generare testo più fluido e naturale, adattandosi meglio al contesto specifico e fornendo risposte più personalizzate e creative.

Tuttavia, è importante notare che, nonostante i miglioramenti, ChatGPT versione 3.5 può ancora occasionalmente produrre risposte che sembrano ripetitive o che seguono una struttura prevedibile. Ciò può dipendere dal contesto specifico della conversazione o dalla natura delle informazioni a cui il modello è stato esposto durante l'addestramento.

9.1.3 ChatGPT-4

Durante il processo di sviluppo per un breve lasso di tempo abbiamo potuto utilizzare ChatGPT-4 gratuitamente, nonostante sia a pagamento, grazie al progetto GPT4free. Tuttavia, la possibilità di utilizzo è durata poco non consentendo un'analisi dettagliata. E gli strumenti presenti online per l'utilizzo gratuito presentano problemi di privacy e un limite di richieste bassisimo. E' nata, infatti, fin da subito una controversia tra OpenAI e Xtekky (il proprietario della repo) che ha sollevato importanti interrogativi riguardo all'accesso gratuito a modelli di intelligenza artificiale, con particolare riferimento a ChatGPT4. Mentre OpenAI ha scelto di offrire ChatGPT4 solo attraverso abbonamenti a pagamento o tramite l'API dell'azienda, Xtekky ha sviluppato un progetto chiamato GPT4free che ha attirato una considerevole attenzione online. Tuttavia, questa iniziativa ha scatenato una controversia legale che solleva importanti questioni etiche e giuridiche sull'utilizzo di tali modelli.

OpenAI ha adottato una strategia commerciale per ChatGPT4, rendendo l'accesso possibile solo tramite abbonamenti a pagamento o tramite l'API dell'azienda. Questa scelta ha suscitato polemiche riguardo all'accessibilità del modello, specialmente per gli utenti comuni e gli appassionati che desiderano sfruttarne il potenziale.

In risposta a ciò, Xtekky ha sviluppato GPT4free come un progetto open source. Questo progetto ha rapidamente guadagnato popolarità online, attirando l'interesse di migliaia di sviluppatori e accumulando oltre 14.000 stelle su GitHub. GPT4free offre la possibilità di utilizzare ChatGPT4 senza dover pagare, sfruttando l'accesso consentito a siti come You.com e Forefront, che hanno pagato per integrare GPT4 nei loro chatbot.

Tuttavia, OpenAI ha inviato una lettera a Xtekky richiedendo la rimozione del repository di GPT4free entro cinque giorni, minacciando di intraprendere azioni legali. La ragione dietro questa richiesta risiede nel fatto che l'utilizzo di GPT4free comporta un consumo di risorse da parte dei siti collegati, potenzialmente causando una perdita di entrate pubblicitarie per tali siti.

Xtekky si è dimostrato disponibile a collaborare con i siti interessati, rimuovendo le script che utilizzano le API specifiche di ciascun sito su loro richiesta. Ha anche suggerito ai siti di implementare le pratiche di sicurezza standard per proteggere le proprie API e limitare l'accesso solo a quelli autorizzati. Tuttavia, al momento nessuno dei siti interessati ha adottato queste misure, lasciando

aperta la possibilità per altri sviluppatori di sfruttare la situazione.

Questa controversia solleva importanti questioni etiche riguardo all'utilizzo dei modelli di intelligenza artificiale senza restrizioni e senza una giusta compensazione per i servizi offerti. Mentre Xtekky sta cercando di risolvere la situazione attraverso un processo di rebranding che consenta un utilizzo legale dei servizi, è evidente che molti altri potrebbero sostituire GPT4free se venisse rimosso.

È importante considerare le implicazioni di queste azioni e riflettere sulla necessità di trovare una soluzione equa che permetta l'accesso ai modelli di intelligenza artificiale, garantendo nel contempo una giusta compensazione per le risorse utilizzate e proteggendo gli interessi dei fornitori di servizi. Sia OpenAI che Xtekky sono attori coinvolti in questa dinamica, e sarà fondamentale trovare un punto di incontro che soddisfi le esigenze di tutte le parti coinvolte.

9.2 ChatGPT e le problematiche con la privacy

Tramite il provvedimento di urgenza del 30 marzo 2023, il Garante italiano per la Protezione dei dati personali ha disposto la limitazione provvisoria del trattamento dei dati personali degli interessati stabiliti nel territorio italiano da parte di OpenAI L.L.C., società statunitense gestrice di ChatGPT. Dopo il provvedimento, che lautorità italiana ha disposto ai sensi dellart. 58, par. 2, lett. f del Regolamento europeo sulla protezione dei dati personali, OpenAI ha deciso di bloccare l'accesso dall'Italia. Il chatbot è stato poi ripristinato in Italia dopo che OpenAI ha annunciato di aver affrontato e chiarito le questioni sollevate dal Garante per la privacy italiano.

Le preoccupazioni sollevate riguardavano principalmente l'uso dei dati personali e l'accesso dei minori al chatbot. OpenAI ha preso misure per affrontare tali questioni, incluso l'inserimento di un sistema di filtro per l'età che esclude gli utenti sotto i 13 anni e richiede l'autorizzazione dei genitori per i minorenni. Inoltre, hanno reso disponibili informazioni aggiuntive sulla raccolta e l'uso dei dati personali e hanno fornito agli utenti la possibilità di escludere i propri dati personali dall'addestramento degli algoritmi compilando un modulo.

Il Garante per la privacy ha espresso soddisfazione per le misure adottate da OpenAI, ma ha sottolineato che l'attività istruttoria nei confronti dell'azienda continuerà e che un'apposita task force delle Autorità per la privacy dell'Unione europea proseguirà il lavoro su questo tema.

In questo periodo abbiamo comunque avuto accesso a ChatGPT tramite canali Telegram, che facevano uso dell'API pubblica, e VPN. Motivo per cui l'immagine 4.3 risulta essere uno screen non dalla piattaforma ufficiale.

9.3 Github Copilot X

Dopo la positiva sorpresa da parte di GitHub Copilot siamo molto amareggiati non aver potuto provare Github Copilot X, un'evoluzione del già noto GitHub Copilot.

Una delle principali caratteristiche di Copilot X è l'adozione del modello di intelligenza artificiale GPT-4 di OpenAI, che permette al sistema di offrire suggerimenti di codice ancora più efficienti ed accurati. La nuova versione supporta inoltre il dialogo sia attraverso chat che tramite comandi vocali, consentendo agli sviluppatori di interagire con l'IA in modo più naturale e intuitivo.

Copilot X offre inoltre un supporto avanzato per le richieste di pull request su GitHub, permettendo di generare descrizioni personalizzate basate sul lavoro svolto e facilitando la comprensione delle modifiche apportate al codice da parte dei revisori. Questa funzionalità aiuta a velocizzare il processo di revisione del codice e a migliorare la collaborazione tra i membri del team di sviluppo.

Un'altra caratteristica interessante di Copilot X è la possibilità di generare documentazione direttamente nell'ambiente di sviluppo integrato dell'utente. Questa documentazione è altamente personalizzata e fornisce risposte specifiche alle domande poste dagli sviluppatori, seguendo gli standard e le linee guida definite dai manutentori del progetto. Ciò consente agli sviluppatori di accedere rapidamente alle informazioni necessarie senza dover effettuare ricerche esterne.

Inoltre, Copilot X introduce la capacità di automatizzare i test unitari. Il sistema è in grado di individuare i test mancanti e di generare nuovi casi di test in modo automatico, contribuendo così a migliorare la qualità del codice prodotto e a garantire una maggiore sicurezza fin dalle prime fasi di sviluppo.

Rispetto alla versione precedente, GitHub Copilot, Copilot X rappresenta un passo avanti significativo nell'evoluzione degli strumenti di assistenza all'intelligenza artificiale per i programmatori. Grazie alle sue nuove funzionalità e alla maggiore efficienza del modello GPT-4, Copilot X offre un supporto ancora più completo e accurato durante il processo di sviluppo del software.

È importante sottolineare che GitHub Copilot X non mira a sostituire i programmatori, ma piuttosto ad aiutarli ad svolgere il proprio lavoro in modo più

efficiente. Come qualsiasi altro strumento di programmazione, Copilot X è uno strumento che può assistere gli sviluppatori nel loro lavoro quotidiano, ma la supervisione umana rimane fondamentale per garantire che il codice prodotto sia pronto per la produzione.

10

Conclusioni

ChatGPT, in alcune occasioni, può agevolare il processo di ricerca, assumendo una sorta di ruolo analogo a quello di Google, ma con maggiore comodità. È importante notare che, tuttavia, è possibile che generi risposte inventate, cosa che invece risulta un'eventualità molto più rara quando si utilizza Google.

All'interno del gruppo si sono tuttavia formate delle opinioni divergenti sotto certi termini. Alcuni componenti hanno ritenuto ChatGPT come migliore prima scelta, mentre altri componenti ritengono Internet una scelta iniziale migliore. Tuttavia, è idea comune che ChatGPT spicchi per ottimizzare processi ripetitivi. Ed è stato decretato da tutti che al momento ChatGPT non possa sostituire gli esseri umani nel processo di sviluppo, in quanto esso è intrinsecamente complesso. È tuttavia opportuno notare che l'evoluzione osservata, passando dalla versione 3 di ChatGPT alla 3.5, non consente di formulare previsioni certe sul futuro.

D'altro canto, strumenti come Github Copilot dimostrano una maggiore utilità, in quanto offrono un supporto concreto ai programmatori nella stesura del codice. Tuttavia, ogni singola riga di codice scritta richiede un'attenta verifica al fine di individuare eventuali errori che potrebbero portare a sessioni di debug prolungate in un secondo momento. Pertanto, anche questo strumento sembra ancora lontano dall'essere in grado di sostituire completamente il programmatore umano. Nonostante ciò, tali strumenti rappresentano un valido supporto, in quanto facilitano la scrittura del codice di base senza richiedere il ricordo o la ricerca online di parti di codice che si ripetono frequentemente.

In generale, è possibile che il codice prodotto da intelligenza artificiale sia

funzionante, ma potrebbero essere presenti errori logici o dettagli trascurati dall'IA. Tali errori possono comportare una considerevole perdita di tempo. Infatti, se da un lato si accelera notevolmente il processo di sviluppo, dall'altro si aumenta significativamente il tempo necessario per il debug. È ben noto che gli esseri umani preferiscono nettamente la prima situazione rispetto alla seconda.

11 TimeWise

TimeWise è un'applicazione mobile sviluppata per dispositivi Android, progettata per aiutare gli utenti a organizzare il proprio tempo e gestire le attività quotidiane in modo efficiente. Con una serie di funzionalità intuitive e personalizzabili, TimeWise offre una soluzione completa per la pianificazione e la gestione del tempo.

11.1 Funzionalità principali

11.1.1 Calendario

Il Calendario è una delle funzionalità centrali di TimeWise. Gli utenti possono facilmente memorizzare e gestire diversi tipi di eventi nel loro calendario personale. L'app supporta una vasta gamma di eventi, tra cui compleanni, viaggi, riunioni, appuntamenti, scadenze e altro ancora. Ogni tipo di evento può essere personalizzato con dati specifici, come la data, l'ora, la descrizione e la posizione. Gli utenti possono visualizzare gli eventi in diverse viste, come anteprima nella vista mensile e con maggiore dettaglio nelle viste giornaliere.

11.1.2 Lista delle attività

La Lista delle attività offre agli utenti un modo semplice per tenere traccia delle attività da svolgere. Gli utenti possono creare elenchi di cose da fare, assegnando a ciascuna attività un titolo, una descrizione e una priorità selezionabile

tra quattro opzioni (alta, media, bassa, molto bassa). Inoltre, gli utenti possono organizzare le attività in categorie personalizzabili, creando un'organizzazione gerarchica per una gestione più efficiente.

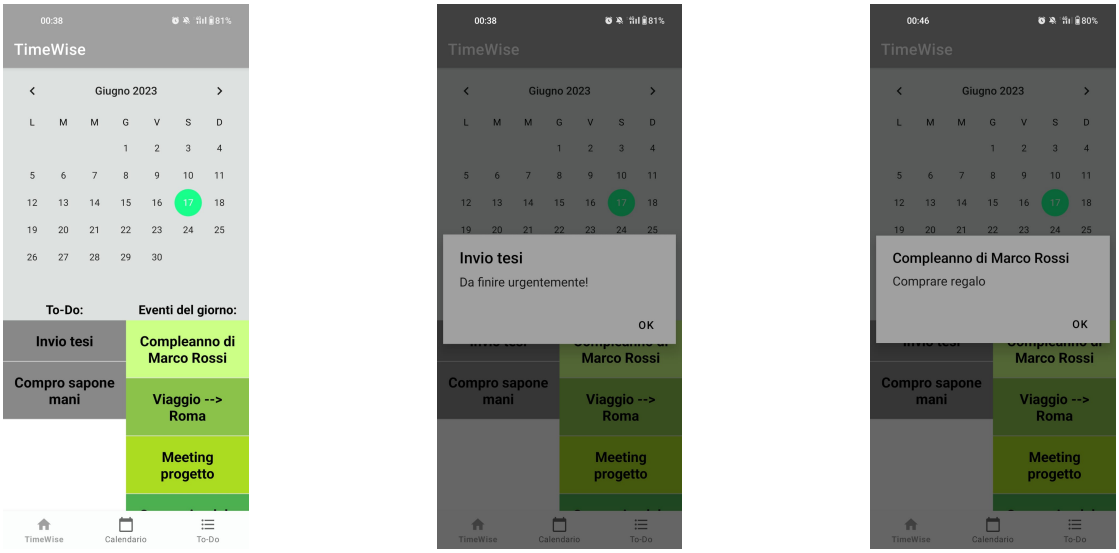
11.2 Interfaccia utente

11.2.1 Schermate principali

L'interfaccia dell'app TimeWise è composta da tre schermate principali:

Home

La schermata Home è la schermata di avvio dell'app. Qui gli utenti possono visualizzare un calendario per la consultazione rapida. Dalla Home è possibile inoltre consultare gli eventi in programma per la giornata corrente e le attività con priorità alta, fornendo un promemoria visivo per le attività urgenti.



(a) Schermata principale

(b) Dettaglio attività

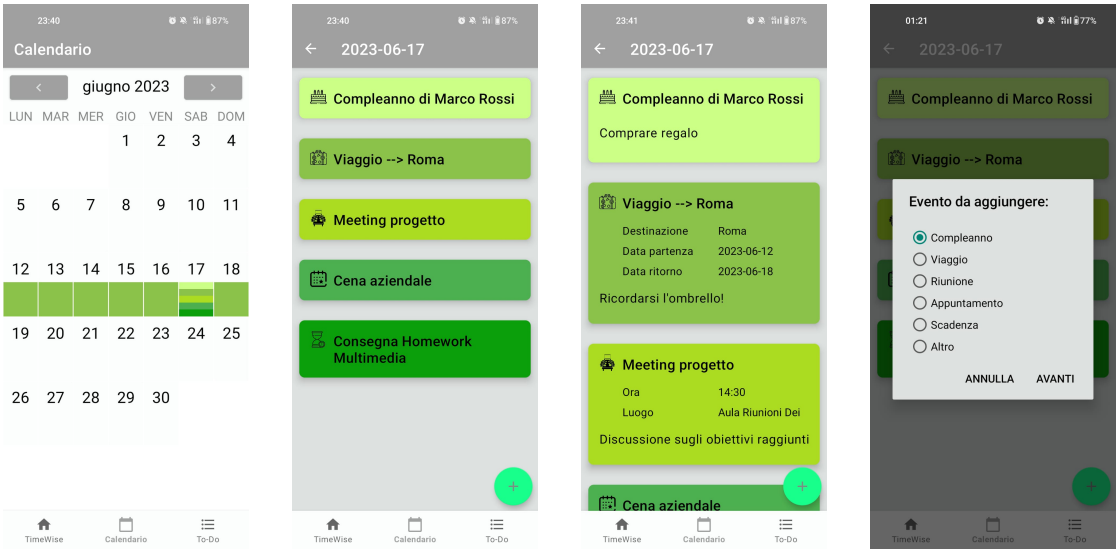
(c) Dettaglio evento

Figura 11.1: Home

Calendario

La schermata del Calendario offre una visualizzazione intuitiva degli eventi programmati. Gli utenti possono passare facilmente tra le diverse viste del

calendario (giornaliera, mensile) per vedere gli eventi pianificati in base alle proprie preferenze. È possibile creare, modificare ed eliminare eventi direttamente nella schermata del Calendario. Inoltre, gli utenti possono visualizzare i dettagli di un evento specifico cliccandolo.

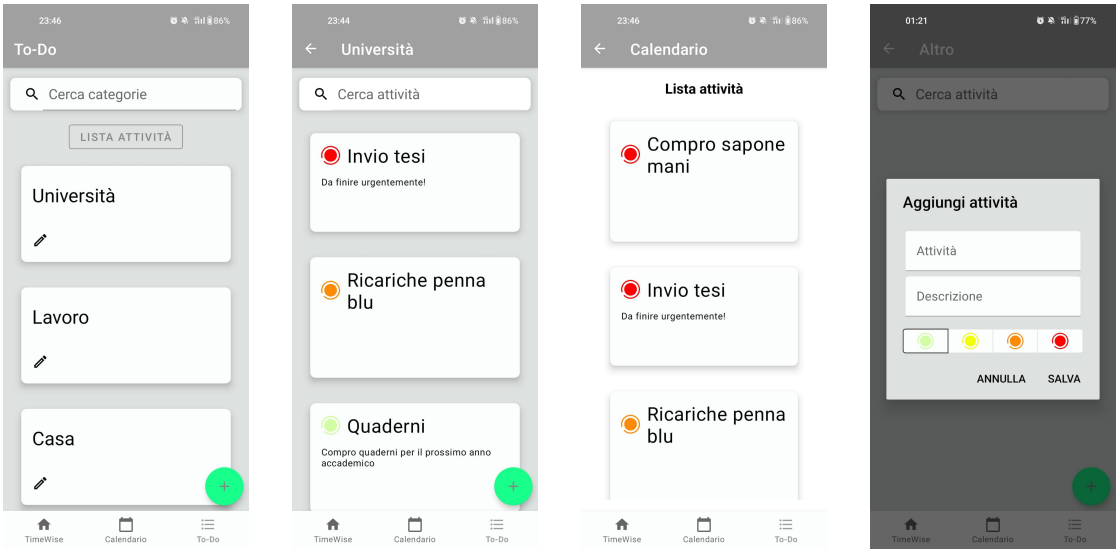


(a) Vista mese (b) Vista giorno (c) Dettagli eventi (d) Nuovo evento

Figura 11.2: Calendario

To-Do

La schermata della lista To-Do permette agli utenti di visualizzare e gestire le attività pianificate. Gli utenti possono creare nuove attività, assegnarle a categorie personalizzate e impostare priorità. L'interfaccia della lista attività mostra un elenco di attività con i relativi titoli, descrizioni e priorità. Gli utenti possono selezionare un elemento per visualizzare ulteriori dettagli e modificare o eliminare l'attività.



(a) Categorie (b) Attività (c) Lista attività (d) Nuova attività

Figura 11.3: To-Do

11.3 Notifiche

TimeWise ti aiuta a ricordare i tuoi eventi impostando automaticamente una notifica di promemoria al salvataggio di un evento. Per gli eventi di tipo compleanno viene impostata una notifica a inizio giornata, altrimenti la notifica viene impostata all'orario dell'evento.

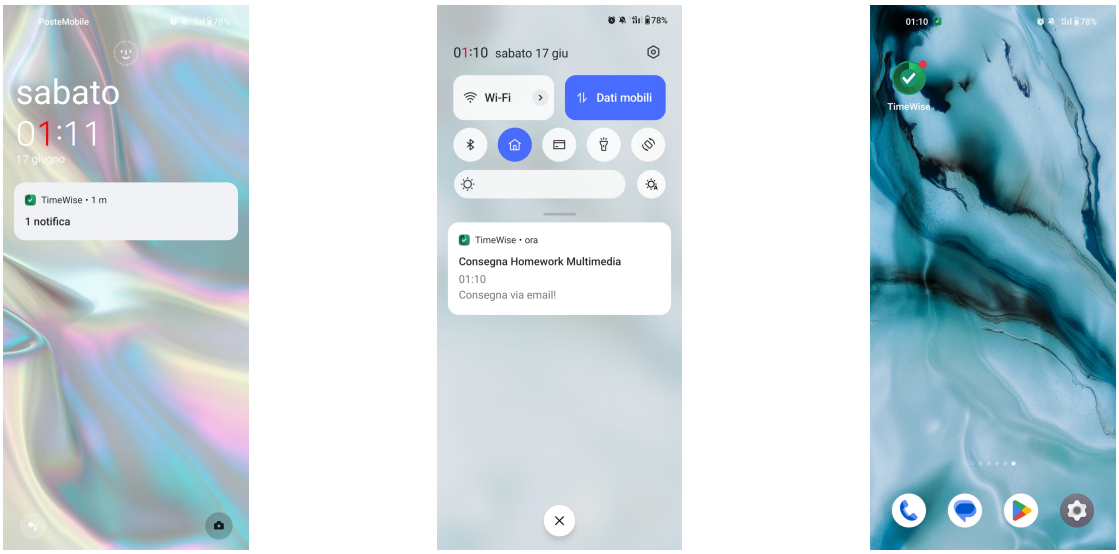


Figura 11.4: Notifica

11.4 Progettazione grafica

L'interfaccia grafica di TimeWise è stata sviluppata con attenzione per offrire un'esperienza utente semplice e intuitiva. L'app presenta una combinazione di colori armoniosi e un layout pulito, consentendo agli utenti di concentrarsi sulle proprie attività senza distrazioni. Gli elementi interattivi, come pulsanti e icone, sono chiaramente visibili e facilmente riconoscibili. Inoltre, la definizione del tema chiaro e tema scuro consente una gradevole visione dell'app sia di giorno che di notte, con colori che si adattano quindi alle condizioni di luce per non stancare la vista dell'utente.

11.5 Conclusioni

TimeWise rappresenta un'efficace soluzione per la gestione del tempo e delle attività quotidiane attraverso l'utilizzo di un calendario eventi completo e una lista delle cose da fare organizzata. L'interfaccia utente intuitiva e la personalizzazione delle funzionalità consentono agli utenti di adattare l'app alle proprie esigenze specifiche. Grazie a TimeWise, gli utenti possono ottimizzare la loro produttività e rimanere organizzati in ogni aspetto della loro vita quotidiana.