Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica

Analisi di sicurezza e compliance OWASP per AI tramite Gandalf Test

Tesi di laurea

Relatore
Prof. Marco Zanella

Laureando
Carmelo Russello
Matricola 2076421

Anno Accademico 2025-2026

"Tu sei la mia speranza. In questo mondo dove il potere è tutto, anche se vengo ferita
e umiliata, voglio ancora lottare, correre e vivere il più liberamente possibile!
Se la penso così è solo grazie a te"

— Ragazzina senza nome.

Dedicato a

Sommario

Il presente documento descrive il lavoro svolto durante il periodo di stage, della durata di circa trecentoventi ore, dal laureando Carmelo Russello presso l'azienda Var Group S.p.A. Gli obbiettivi da raggiungere erano molteplici.

- In primo luogo era richiesto lo sviluppo di un sistema di testing automatizzato per l'analisi di sicurezza e compliance OWASP dei modelli di intelligenza artificiale, utilizzando lo strumento Gandalf Test.
- In secondo luogo era richiesta l'implementazione di una dashboard web per la visualizzazione dei risultati dei test eseguiti.

"No one in this world deserves to be laughed at as they die!"

— Monkey D. Luffy, *One Piece*

Ringraziamenti

Innanzitutto, vorrei esprimere la mia gratitudine al Prof. Marco Zanella, relatore della mia tesi, per l'aiuto e il sostegno fornitomi durante la stesura del lavoro.

TBA

TBA

Padova, Aprile 2026 Carmelo Russello

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

Il presente capitolo introduce il contesto applicativo oggetto dello stage: automazione di testing su applicazioni che sfruttano l'intelligenza artificiale generativa, come chatbot o assistenti virtuali. Queste applicazioni trattano input testuali e informazioni sensibili, esponendo a rischi generali quali manipolazioni degli input, fuoriuscite di dati e comportamenti imprevisti dei modelli.
Il progetto mira a definire e implementare un approccio integrato per valutare e migliorare la sicurezza, l'affidabilità e la conformità delle applicazioni, attraverso analisi del codice, test e pratiche operative dedicate.

1.1 L'azienda

L'azienda con cui ho deciso di intraprendere il mio percorso di Stage è Var Group S.p.A., una realtà italiana specializzata in soluzioni IT e servizi di consulenza per le imprese.
La scelta è stata influenzata dalla mia precedente esperienza con il progetto di Software Engineering svolto presso Var Group, durante il quale ho avuto modo di apprezzare la professionalità e la competenza del team.

1.2 Introduzione al caso di studio

Negli ultimi anni, l'intelligenza artificiale generativa ha rivoluzionato il modo in cui le applicazioni interagiscono con gli utenti, offrendo esperienze più naturali e coinvolgenti. Questa rivoluzione ha cambiato e continua a cambiare il modo in cui gli utenti e le aziende si interfacciano alle applicazioni software.
Una considerevole porzione delle aziende di tutto il mondo e di tutti gli ambiti hanno integrato o stanno integrando modelli di intelligenza artificiale nei propri prodotti.

1.2.1 Necessità del testing per AI generativa

Tuttavia l'adozione di questa innovativa tecnologia viene con nuove sfide da affrontare per quanto riguarda la sicurezza e la privacy.
Le applicazioni basate su AI generativa sono esposte a rischi specifici inesistenti prima della loro introduzione, come attacchi di *prompt injection*, *data leakage* e *allucinazioni*

dei modelli. Questi rischi possono compromettere la sicurezza delle applicazioni, la privacy degli utenti e l'integrità dei dati trattati.
Per riuscire a mitigare i rischi associati all'utilizzo di modelli di intelligenza artificiale generativa, è fondamentale seguire delle linea guida specifiche come quelle fornite da OWASP, che ha stilato una lista delle top 10 vulnerabilità legate all'utilizzo dell'intelligenza artificiale generativa.
Il progetto di stage si concentra proprio su questo aspetto, il testing delle applicazioni che integrano una LLM per verificarne la conformità con le linee guida OWASP in modo tale da scovare le vulnerabilità e prevenirne l'utilizzo da parte di utenti malintenzionati.

1.3 Software da sviluppare

Riguardo la stesura del testo, relativamente al documento sono state adottate le seguenti convenzioni tipografiche:

- gli acronimi, le abbreviazioni e i termini ambigui o di uso non comune menzionati vengono definiti nel glossario, situato alla fine del presente documento;
- per la prima occorrenza dei termini riportati nel glossario viene utilizzata la seguente nomenclatura: *parola*[g];
- i termini in lingua straniera o facenti parti del gergo tecnico sono evidenziati con il carattere *corsivo*.

Capitolo 2

Metodologie di sviluppo del software

In questo capitolo viene descritto l'approccio organizzativo adottato per lo stage e come viene pianificato e verificato il lavoro su base settimanale.

2.1 Metodologia adottata

Per lo svolgimento del progetto è stata adottata una metodologia **iterativo-incrementale** con **timebox settimanale**. Il tracciamento avviene tramite un elenco di *obiettivi settimanali*, derivati dalla pianificazione riportata nel Capitolo 3 (sezione Pianificazione).

2.2 Introduzione al modello di sviluppo iterativo-incrementale

2.2.1 Introduzione

Lo sviluppo iterativo-incrementale consiste nel miglioramento continuo del prodotto attraverso cicli ripetuti aggiungendo gradualmente nuove funzionalità a ogni ciclo. Questo approccio consente di adattarsi rapidamente ai cambiamenti dei requisiti e di integrare il feedback degli *stakeholder* in modo tempestivo, migliorando la qualità complessiva del prodotto finale.

2.2.2 Origine

Molti esempi di impiego precoce sono riportati nell'articolo di Craig Larman e Victor Basili "Iterative and Incremental Development: A Brief History" **LarmanBasili2003**, tra i quali uno dei più antichi è il Project Mercury della NASA negli anni '60. Alcuni ingegneri coinvolti in Mercury formarono successivamente una divisione dentro IBM; un esempio notevole di successo Iterative and Incremental Development (IID) in quel contesto fu lo sviluppo del software avionico principale dello Space Shuttle, realizzato tra il 1977 e il 1980 tramite 17 iterazioni in 31 mesi.

Organizzazioni come il Dipartimento della Difesa degli Stati Uniti hanno mostrato una preferenza per metodologie iterative e la Department of Defense (DoD) del

2000 esponeva chiaramente una preferenza per un approccio evolutivo/iterativo alla realizzazione delle capacità software: "An evolutionary approach is preferred...". Le revisioni successive di DoDI 5000.02 non menzionano più esplicitamente lo "spiral development", ma sostengono comunque l'approccio iterativo come baseline per programmi software-intensive.

Anche agenzie di sviluppo internazionale, come la United States Agency for International Development (USAID), adottano un approccio iterativo e incrementale nel ciclo di programmazione per progettare, monitorare, valutare, apprendere e adattare i progetti, privilegiando collaborazione, apprendimento e adattamento continuo.

2.2.3 Fasi del modello di sviluppo iterativo-incrementale

Il modello iterativo-incrementale si articola in diverse fasi principali:

- **Inception**: identifica l'ambito del progetto, i requisiti (funzionali e non funzionali) e i rischi a un livello elevato, ma con dettagli sufficienti affinché il lavoro possa essere stimato. In questa fase si definiscono il business case, gli stakeholder principali, gli obiettivi di alto livello e i vincoli di progetto; si produce un backlog iniziale e si effettuano stime di massima e una prima analisi dei rischi per decidere il go/no go. I deliverable tipici includono la vision, la lista dei requisiti prioritari, il piano di progetto e il registro dei rischi.

- **Elaboration**: fornisce un'architettura funzionante che mitiga i principali rischi e soddisfa i requisiti non funzionali. Comprende la realizzazione di prototipi o proof of concept per validare scelte architetturali, la definizione della baseline tecnica e delle specifiche non funzionali (performance, sicurezza, scalabilità). In questa fase si aggiornano stime e piani, si dettagliano i criteri di accettazione e si prepara il piano di test per garantire la fattibilità tecnica.

- **Construction**: riempie progressivamente l'architettura con codice pronto per la produzione, prodotto dall'analisi, progettazione, implementazione e test dei requisiti funzionali. È caratterizzata da iterazioni multiple con sviluppo incrementale, integrazione continua, test automatici e revisioni del codice; ogni iterazione produce incrementi rilasciabili, documentazione tecnica aggiornata e attività di refactoring per mantenere la qualità del codice.

- **Transition**: consegna il sistema nell'ambiente operativo di produzione. Include le attività di deploy e migrazione dati, le operazioni di training per gli utenti finali e il supporto iniziale post rilascio, il monitoraggio delle prestazioni e la raccolta dei feedback per eventuali correzioni. I deliverable finali comprendono la documentazione utente, la documentazione di deployment, il registro delle issue risolte e il verbale di acceptance/sign off.

2.3 Applicazione del modello iterativo-incrementale nel progetto

2.3.1 Motivazioni della scelta

La scelta del modello iterativo-incrementale per lo svolgimento del progetto di stage è stata immediata e naturale dato che i tutor aziendali adottano tale metodologia con la quale è iniziato lo sviluppo sin dalla prima settimana. IID è particolarmente adatto a

contesti in cui i requisiti possono evolvere rapidamente e dove è fondamentale integrare il feedback degli utenti in modo continuo. Poiché il caso di studio del progetto è in continua evoluzione e sviluppo, l'approccio iterativo-incrementale consente di adattarsi rapidamente ai cambiamenti e di migliorare costantemente il prodotto finale.

2.3.2 Metodologia operativa adottata

La cadenza operativa prevede:

Allineamento settimanale un unico incontro (30/45 minuti) che combina *review* breve dei risultati ottenuti e *planning* della settimana successiva; durante l'incontro si definisce esplicitamente il **Goal della settimana**. L'agenda tipica comprende stato degli obiettivi, rischi/impedimenti incontrati, decisioni da prendere e pianificazione dei prossimi passi. L'esito atteso è un goal chiaro, con criteri di accettazione condivisi e una stima realistica in base alla capacità disponibile; se emergono dipendenze o ostacoli, si pianifica subito come rimuoverli o si riduce il perimetro mantenendo il timebox.

Supporto durante la settimana, il tutor aziendale fornisce supporto per sblocchi tecnici e decisioni operative quando necessario. Il confronto avviene tramite brevi sincronizzazioni giornaliere tramite la piattaforma Slack dove ogni giorno si discute su quello che è stato fatto in modo tale da mantenere un allineamento costante e reindirizzare il lavoro se necessario.

Incremento ogni timebox produce un risultato verificabile (es. script/report, esiti di test su codice reale, componente di dashboard, documentazione), tutto ciò viene mostrato al tutor durante l'allineamento settimanale successivo per raccogliere feedback e pianificare i passi successivi.

La misura dell'avanzamento è basata sul raggiungimento dei *goal settimanali*. Questo approccio consente di ridurre il rischio, incorporare rapidamente i feedback e mantenere la tracciabilità rispetto alla conformità OWASP. In caso di scostamenti si adatta il perimetro mantenendo fisso il timebox, privilegiando il soddisfacimento dei requisiti obbligatori.

Capitolo 3

Descrizione dello stage

In questo capitolo viene fornita una panoramica del progetto di stage incentrato sulla sicurezza di AI generativa (OWASP, Gandalf Test), dei rischi e delle mitigazioni, dei requisiti e degli obiettivi, e della pianificazione fino alla produzione di report e dashboard.

3.1 Introduzione al progetto

3.2 Analisi preventiva dei rischi

Durante la fase iniziale di analisi sono stati identificati i principali rischi potenziali connessi al progetto, classificati per ambito (tecnico, di progetto e infrastrutturale) e priorizzati in base all'impatto e alla probabilità. Per ciascun rischio è stato predisposto un piano di mitigazione che definisce azioni concrete, tempistiche e responsabilità precise.

Le contromisure prevedono attività di sperimentazione controllata degli strumenti, revisioni manuali dei risultati, integrazione e test in ambienti rappresentativi, oltre a piani di escalation per le criticità più gravi. È inoltre prevista una procedura di monitoraggio continuo e revisione periodica delle valutazioni e delle soluzioni adottate, in modo da aggiornare rapidamente le contromisure alla luce di nuovi dati o evoluzioni tecnologiche.

3.2.1 Rischi tecnici

1. Complessità nell'applicare strumenti di security testing ad AI generativa (tool immaturi o non sempre affidabili).
Descrizione: Le difficoltà nell'adattare i tool di security testing all'AI generativa possono derivare dalla loro immaturità o dalla mancanza di affidabilità.
Soluzione: Una lunga fase di sperimentazione e testing dei tool ha mitigato i rischi, garantendo risultati affidabili.

2. Possibili falsi positivi o negativi nei test di vulnerabilità.
Descrizione: I test di vulnerabilità potrebbero generare risultati inaccurati, con falsi positivi (segnalazioni errate di vulnerabilità) o falsi negativi (mancata rilevazione di vulnerabilità reali).

Soluzione: Implementare una fase di revisione manuale dei risultati dei test per convalidare le segnalazioni e ridurre il rischio di falsi positivi e negativi.

3. Difficoltà di integrazione dei tool con codice reale e pipeline di sviluppo.
Descrizione: Le difficoltà di integrazione possono derivare da incompatibilità tra i tool di testing e l'infrastruttura esistente, nonché dalla complessità del codice reale su cui si stanno eseguendo i test.
Soluzione: Collaborare con gli sviluppatori del codice reale per garantire che i tool di testing siano compatibili con l'infrastruttura esistente e fornire supporto durante l'integrazione.

3.2.2 Rischi di progetto

4. Mancanza di esperienza pregressa su OWASP o sicurezza AI.
Descrizione: La poca familiarità con le best practices di OWASP o con le specificità della sicurezza nell'AI generativa potrebbe rallentare l'avanzamento del progetto.
Soluzione: Studio e formazione con risorse adeguate per aumentare la familiarità con OWASP e la sicurezza dell'AI generativa.

5. Possibile difficoltà a rispettare la pianificazione a causa della curva di apprendimento iniziale.
Descrizione: La curva di apprendimento iniziale per l'utilizzo di nuovi strumenti e tecnologie potrebbe richiedere più tempo del previsto, influenzando la pianificazione del progetto.
Soluzione: Pianificazione di margini di tempo aggiuntivi per la formazione e l'adattamento agli strumenti, nonché monitoraggio attento dei progressi.

3.2.3 Rischi infrastrutturali

6. Limitazioni di risorse computazionali nei test di AI.
Descrizione: Le risorse computazionali disponibili per l'esecuzione dei test di AI potrebbero non essere sufficienti, causando rallentamenti o interruzioni nei test.
Soluzione: Ottimizzazione dell'uso delle risorse disponibili e richiesta di accesso a risorse computazionali aggiuntive.

7. Problemi di compatibilità con ambienti cloud o di deployment.
Descrizione: Le differenze tra gli ambienti di sviluppo e produzione potrebbero causare problemi di compatibilità, rendendo difficile l'esecuzione dei test in modo uniforme.
Soluzione: Testare i tool di testing in ambienti simili a quelli di produzione e documentazione di eventuali problemi di compatibilità.

3.3 Requisiti e obiettivi

3.3.1 Obiettivi obbligatori

- Valutazione comparativa degli strumenti di analisi.
- Applicazione pratica dei test su codice reale.
- Prototipo in grado di generare report sulle vulnerabilità AI rispetto a OWASP.
- Documentazione tecnica e presentazione finale.

3.3.2 Obiettivi desiderabili

- Dashboard interattiva con visualizzazioni avanzate
- Integrazione del prototipo in pipeline CI/CD esistente.
- Estensione dei test ad altri framework oltre Gandalf Test.
- Raccomandazioni per un framework interno di AI Security by Design.

3.4 Pianificazione

La pianificazione del lavoro di progetto è stata suddivisa in fasi settimanali, con obiettivi specifici per ciascuna fase. Di seguito è riportata una panoramica della pianificazione prevista:

Settimana	**Attività**
Settimana 1	Studio preliminare su OWASP e rischi AI, overview di Gandalf Test, setup ambiente di lavoro.
Settimana 2	Analisi comparativa di tool di analisi statica e dinamica (open-source e commerciali). Creazione matrice di valutazione.
Settimana 3	Applicazione degli strumenti a piccoli progetti demo, valutazione dei risultati e raccolta criticità.
Settimana 4	Esecuzione dei primi test su componenti reali del team, documentazione dei risultati, identificazione vulnerabilità.
Settimana 5	Realizzazione di script/report per aggregare risultati, definizione dei KPI di compliance OWASP.
Settimana 6	Sviluppo di dashboard interattiva per monitorare vulnerabilità e andamento dei test.
Settimana 7	Test end-to-end sul prototipo, miglioramento dei tool e dei report.
Settimana 8	Redazione di documentazione tecnica, manuale utente e materiale per la presentazione della tesi.

Tabella 3.1: Pianificazione delle attività di progetto

3.4.1 Settimana 1

Durante la prima settimana di lavoro il focus è stato posto sullo studio di OWASP e alla comprensione del ambito di studio del progetto. In questo periodo è stata fatta una estensiva ricerca sulla top 10 delle vulnerabilità delle LLM secondo OWASP e dei metodi di testing, attaco e red teaming più comuni ed efficaci in modo tale da avere una visione completa delle problematiche di sicurezza legate all'AI. Essendo l'ambito di studio in continua evoluzione è stato fondamentale raccogliere informazioni sulle tecnologie più recenti e le metodologie attuali per il testing delle vulnerabilità delle LLM. Nei primi giorni della settimana ho avuto modo di provare di persona il gandalf test in modo tale da comprendere a fondo come le LLM possono essere ingannate a rivelare informazioni sensibili (role play, uso di lingua differente, richieste

implicite, ecc.). Nell'ultima parte della settimana ho approfondito sul concetto di red teaming e le sue applicazioni pratiche nel contesto delle LLM poichè ho avuto modo di vedere che molti tool di security testing per AI generativa si basano su questa metodologia di attacco. A valle della ricognizione iniziale ho mappato le categorie OWASP più rilevanti ai casi d'uso previsti (prompt injection, disclosure di informazioni sensibili, hallucination e output non sicuri, uso di tool esterni eccessivamente permissivi, data poisoning, differenze tra test in black-box e scenari più informati), cercando di capire come tradurre ciascun rischio in casi di test ripetibili. Ho inoltre analizzato la letteratura più recente (whitepaper, linee guida e report tecnici) per identificare pattern ricorrenti di attacco e difesa e per definire un insieme minimo di metriche di valutazione (riproducibilità del test, tasso di successo del jailbreak, severità dell'impatto, copertura delle categorie OWASP) utile a confrontare approcci manuali e automatizzati.

Sul fronte sperimentale, con il gandalf test ho eseguito più iterazioni variando strategia e contesto per osservare come cambiano le risposte del modello al variare dell'intento e della formulazione (cambio di persona nel role play, ricorso a lingue miste, parafrasi progressive, codifiche/decodifiche semplici, richieste spezzate su più turni, evocazione di autorità fittizie o regole alternative). Ho annotato quali tattiche risultano più efficaci e in quali condizioni falliscono (rate limit, filtri di sicurezza, memoria contestuale), in modo da derivare linee guida utili alla fase di automazione. Ho iniziato anche a delineare il perimetro etico e di compliance, chiarendo i confini del red teaming responsabile e le cautele nella gestione di output potenzialmente sensibili. Questo lavoro preliminare ha permesso di costruire una base metodologica solida, utile per selezionare in modo informato gli strumenti da valutare nelle settimane successive e per impostare una prima matrice di tracciamento tra rischi OWASP, scenari di test e criteri di accettazione.

3.4.2 Settimana 2

Nel corso della seconda settimana di lavoro l'interesse si è concentrato sull'analisi approfondita delle varie tecnologie e tool esistenti per il testing delle LLM. Ho condotto una ricerca esaustiva per identificare sia soluzioni open-source che commerciali, valutando ciascuna in base a criteri quali facilità d'uso, capacità di integrazione, copertura delle vulnerabilità OWASP, scalabilità e costi associati. Ho creato una matrice di valutazione comparativa (osservabile nel capitolo 5, sezione 5.1) per sintetizzare i punti di forza e le limitazioni di ogni strumento, facilitando così la selezione dei candidati più promettenti per le fasi successive del progetto. Durante l'analisi, ho esaminato tool come PromptFoo, PyRIT, LangFuse, DeepEval/DeepTeam, Garak, Giskard, Galileo e LakeraGuard, approfondendo le loro funzionalità specifiche per il security testing delle LLM. Ho valutato come ciascuno di questi strumenti affronta le principali categorie di vulnerabilità identificate nella settimana precedente, e ho visionato numerosi talk e conferenze per comprendere al meglio ogni tool sottoposto ad analisi e le loro applicazioni pratiche. Ho creato piccoli script per testare alcune delle funzionalità offerte dai tool, in modo tale da farmi un'idea più precisa delle loro capacità e limitazioni. Al termine della settimana, ho redatto la matrice di valutazione prima citata la quale servirà come base per la selezione degli strumenti da utilizzare nelle fasi successive del progetto, garantendo che le scelte siano informate e allineate agli obiettivi di sicurezza definiti in precedenza.

3.4.3 Settimana 3

TBA

3.4.4 Settimana 4

TBA

3.4.5 Settimana 5

TBA

3.4.6 Settimana 6

TBA

3.4.7 Settimana 7

TBA

3.4.8 Settimana 8

TBA

Capitolo 4

Analisi dei requisiti

Breve introduzione al capitolo

4.1 Casi d'uso

Per lo studio dei casi di utilizzo del prodotto sono stati creati dei diagrammi. I diagrammi dei casi d'uso (in inglese *Use Case Diagram*) sono diagrammi di tipo Unified Modeling Language (UML) dedicati alla descrizione delle funzioni o servizi offerti da un sistema, così come sono percepiti e utilizzati dagli attori che interagiscono col sistema stesso. Essendo il progetto finalizzato alla creazione di un tool per l'automazione di un processo, le interazioni da parte dell'utilizzatore devono essere ovviamente ridotte allo stretto necessario. Per questo motivo i diagrammi d'uso risultano semplici e in numero ridotto.

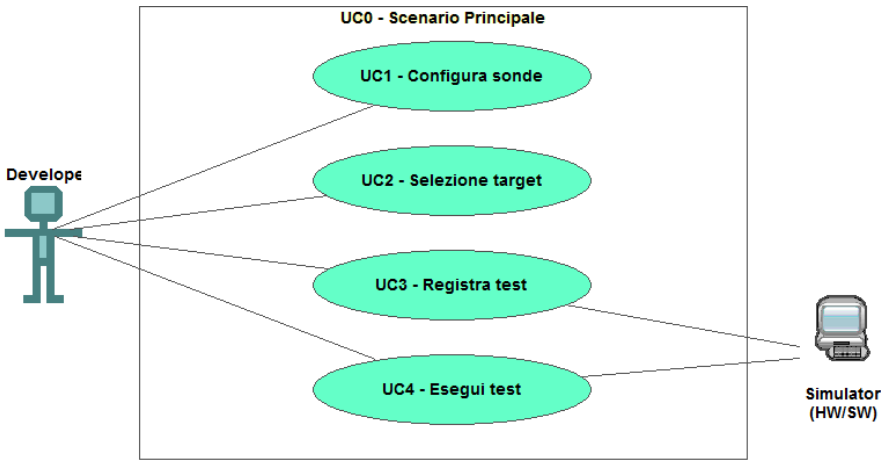

Figura 4.1: Use Case - UC0: Scenario principale

UC0: Scenario principale

Attori Principali: Sviluppatore applicativi.

Precondizioni: Lo sviluppatore è entrato nel plug-in di simulazione all'interno dell'IDE.

Descrizione: La finestra di simulazione mette a disposizione i comandi per configurare, registrare o eseguire un test.

Postcondizioni: Il sistema è pronto per permettere una nuova interazione.

4.2 Tracciamento dei requisiti

Da un'attenta analisi dei requisiti e degli use case effettuata sul progetto è stata stilata la tabella che traccia i requisiti in rapporto agli use case.
Sono stati individuati diversi tipi di requisiti e si è quindi fatto utilizzo di un codice identificativo per distinguerli.
Il codice dei requisiti è così strutturato R(F/Q/V)(N/D/O) dove:

R = requisito

F = funzionale

Q = qualitativo

V = di vincolo

N = obbligatorio (necessario)

D = desiderabile

O = opzionale

Nelle tabelle 4.1, 4.2 e 4.3 sono riassunti i requisiti e il loro tracciamento con gli use case delineati in fase di analisi.

Tabella 4.1: Tabella del tracciamento dei requisti funzionali

Requisito	Descrizione	Use Case
RFN-1	L'interfaccia permette di configurare il tipo di sonde del test	UC1

Tabella 4.2: Tabella del tracciamento dei requisiti qualitativi

Requisito	Descrizione	Use Case
RQD-1	Le prestazioni del simulatore hardware deve garantire la giusta esecuzione dei test e non la generazione di falsi negativi	-

Tabella 4.3: Tabella del tracciamento dei requisiti di vincolo

Requisito	Descrizione	Use Case
RVO-1	La libreria per l'esecuzione dei test automatici deve essere riutilizzabile	-

Capitolo 5

Progettazione e tecnologie

In questo capitolo

5.1 Tecnologie e strumenti

Di seguito viene data una panoramica delle tecnologie e strumenti presi in considerazione per lo sviluppo del prototipo. L'analisi ha valutato compatibilità, licenza, facilità d'integrazione, costi e capacità di security testing specifiche per modelli generativi. Gli strumenti e le tecnologie esaminate includono:

- **PromptFoo** – piattaforma di testing per LLM con supporto a test automatizzati e integrazione CI/CD.
- **PyRIT** – tool open-source per l'identificazione e mitigazione dei rischi sui modelli generativi.
- **LangFuse** – piattaforma di osservabilità e valutazione di LLM.
- **DeepEval / DeepTeam** – framework di valutazione e tool di red teaming per benchmark e safety testing.
- **Garak** – scanner di vulnerabilità per modelli generativi focalizzato su attacchi pratici (prompt injection, jailbreak, leakage).
- **Giskard** – piattaforma di red teaming automatizzato con opzioni SaaS e libreria Python.
- **Galileo** – piattaforma di osservabilità con SDK, utile per monitoraggio ma meno orientata al security testing.
- **LakeraGuard** – suite di sicurezza commerciale per modelli, include test come il Gandalf test.
- **Tecnologie di base** – Python, Docker, Git/GitHub, CI/CD (GitHub Actions), librerie ML (PyTorch, Hugging Face), e strumenti di logging/monitoring.

Questa panoramica ha guidato la selezione degli strumenti adottati per il prototipo, privilegiando soluzioni open-source facilmente integrabili nell'ambiente di sviluppo e con funzionalità mirate al security testing dei modelli generativi.

5.1.1 Strumenti analizzati

Durante le prime due settimane di Stage è stato condotto uno studio preliminare su diversi strumenti di security testing per AI generativa. Di seguito viene fornita la matrice di valutazione dei tools e una breve descrizione di ciascuno strumento analizzato.

Criterio di valutazione	Categoria	Peso (%)	PromptFoo	PyRIT	LangFuse	DeepEval	Garak	Giskard	Galileo	LakeraGuard
Copertura OWASP AI Top 10	Efficacia Tecnica	20	5	4	2	2	3	4	3	4
Velocità di Scansione	Performance	10	4	4	4	4	4	4	4	4
Costo Totale (TCO)	Costi	10	4	5	5	5	5	3	1	1
Supporto Tecnico & Community	Supporto	10	4	3	4	3	3	3	2	4
Adversarial Attack Detection	AI-Specific	10	5	5	2	3	3	4	4	5
Efficacia Gandalf Test	Efficacia Tecnica	8	3	5	1	1	3	3	3	4
Profondità di Analisi	Efficacia Tecnica	8	4	5	2	2	4	4	3	5
Integrazione CI/CD	Usabilità	8	4	4	3	3	2	4	2	5
Scalabilità	Performance	8	4	4	3	3	4	4	3	4
Model Explainability	AI-Specific	8	4	4	2	2	3	3	4	4
TOTALE			4,22	4,26	2,78	2,78	3,38	3,64	2,9	3,96

Figura 5.1: Matrice di Valutazione degli Strumenti Analizzati

PromptFoo

PromptFoo è una piattaforma di testing per modelli di linguaggio che consente agli sviluppatori di creare, eseguire e gestire test automatizzati per valutare le prestazioni, l'affidabilità e la sicurezza dei loro modelli di linguaggio naturale. Offre funzionalità come la creazione di casi di test personalizzati, l'integrazione con pipeline CI/CD e report dettagliati sui risultati dei test. Questo tool è stato tenuto in molta considerazione in quanto offre funzionalità specifiche per il testing delle vulnerabilità OWASP top 10 per AI generativa, obbiettivo principale del progetto di stage.

PyRIT

PyRIT è uno tool open-source progettato da Azure Microsoft per l'identificazione e la mitigazione dei rischi associati all'uso di modelli di intelligenza artificiale generativa. PyRIT è stato creato per valutare modelli di AI per potenziali vulnerabilità di sicurezza, bias e problemi di conformità, fornendo raccomandazioni su come migliorare la sicurezza e l'affidabilità dei modelli.

LangFuse

LangFuse è una piattaforma open-source per la valutazione e l'osservazione delle applicazioni che utilizzano intelligenza artificiale generativa. La piattaforma è integrata con diversi modelli di linguaggio naturale e permette l'utilizzo tramite cloud o in self host. Langfuse offre inoltre un LLM playground per testatre i modelli che aveva catturato la mia attenzione all'inizio del progetto. Tuttavia il tool non offre una vera e propria funzionalità di security testing, focus del progetto di stage.

DeepEval / DeepTeam

DeepEval è un framework open-source per la valutazione e il benchmarking dei modelli di intelligenza artificiale generativa. DeepEval non fornisce funzionalità specifiche per

il testing dei modelli. Per ovviare a questa mancanza è stato creato un tool di testing basato su DeepEval chiamato DeepTeam, un tool di red teaming. Tuttavia, DeepTeam non ha un focus sulle vulnerabilità OWASP in quanto si concentra sul testing delle safety guidelines.

Garak

Garak è uno scanner di vulnerabilità per modelli di intelligenza artificiale generativa open-source creato da NVIDIA per facilitare il testing dei modelli. Il focus di Garak sta nei metodi di attacco specifici per far fallire in modo imprevisto una LLM o un sistema di dialogo. Garak testa diverse vulnerabilità tra cui allucinazioni, data leakage, prompt injection, jailbreaks ecc.

Giskard

Giskard è una piattaforma di red teaming automatizzato per testare, valutare ed analizzare modelli di intelligenza artificiale generativa. Esistono due metodi di utilizzo di Giskard: Come servizio HUB a pagamento o come libreria Python open-source da installare localmente. La libreria fornita è però fortemente limitata nelle funzionalità rispetto al servizio HUB in quanto incentrata sulla ricerca.

Galileo

Galileo è una piattaforma che permette la valutazione e osservazione di applicazioni basate su AI generativa. Galileo offre SDK in Python e TypeScript per integrarlo direttamente nei propri progetti. Galileo è molto flessibile per quanto riguarda il deploy e viene utilizzato da molte aziende di rilievo. Nonostante ciò la piattaforma non offre funzionalità specifiche per il security testing, focus di questo progetto, ed è a pagamento.

LakeraGuard

LakeraGuard è una piattaforma di sicurezza per modelli di intelligenza artificiale che aiuta gli sviluppatori a proteggere i loro modelli da minacce e vulnerabilità. Offre funzionalità come la scansione delle vulnerabilità, la gestione delle patch e il monitoraggio delle minacce in tempo reale.
Essendo l'azienda svizzera Lakera la creatrice del gandalf test, focus principale delle prime settimane del progetto di stage, il tool da loro creato è stato uno dei primi ad essere analizzato. Tuttavia il tool out of the box fa già quello che lo stage chiede di implementare quindi non ha suscitato uno studio approfondito in quanto avrebbe reso superfluo lo sviluppo del prototipo.
Inoltre è un tool a pagamento che offre un tier gratuito il quale è però limitato a 10000 richieste Application Program Interface (API) al mese, limite che avrebbe reso difficile l'utilizzo futuro del tool.

5.1.2 Strumenti utilizzati

Di seguito viene data una panoramica delle tecnologie e strumenti utilizzati per lo sviluppo del prototipo.

Python + PyRIT

Python è un linguaggio di programmazione versatile e ampiamente utilizzato, particolarmente adatto per lo sviluppo di applicazioni backend. PyRIT è uno strumento open-source progettato da Azure Microsoft per l'identificazione e la mitigazione dei rischi associati all'uso di modelli di intelligenza artificiale generativa. Insieme, Python e PyRIT forniscono un ambiente potente per lo sviluppo e la sicurezza delle applicazioni AI.DA AGGIUNGERE PERCHÉ USATO

React

React è una libreria JavaScript per la creazione di interfacce utente, sviluppata da Facebook. È ampiamente utilizzata per costruire applicazioni web dinamiche e reattive. La sua architettura basata su componenti consente agli sviluppatori di creare interfacce utente modulari e riutilizzabili, semplificando il processo di sviluppo.DA AGGIUNGERE PERCHÉ USATO

MongoDB

MongoDB è un database NoSQL orientato ai documenti, progettato per gestire grandi volumi di dati non strutturati. Utilizza un modello di dati flessibile basato su BSON, che consente agli sviluppatori di archiviare e recuperare informazioni in modo efficiente. MongoDB è particolarmente adatto per applicazioni che richiedono scalabilità e prestazioni elevate.DA AGGIUNGERE PERCHÉ USATO

5.2 Progettazione

5.2.1 Progettazione Backend

Descrizione struttura backend & spiegazione infrastruttura di persistenza dei dati

5.2.2 Progettazione Frontend

Descrizione pagine frontend

5.3 Design Pattern utilizzati

STRATEGY PER CAMBIARE AL VOLO IL TIPO DI TEST (DATASET E SCORER PROMPT)

Capitolo 6

Conclusioni

6.1 Consuntivo finale

6.2 Raggiungimento degli obiettivi

6.3 Conoscenze acquisite

6.4 Valutazione personale

Appendice A

Appendice A

Citazione

Autore della citazione

Acronimi e abbreviazioni

API Application Program Interface. 12

UML Unified Modeling Language. 7

Bibliografia