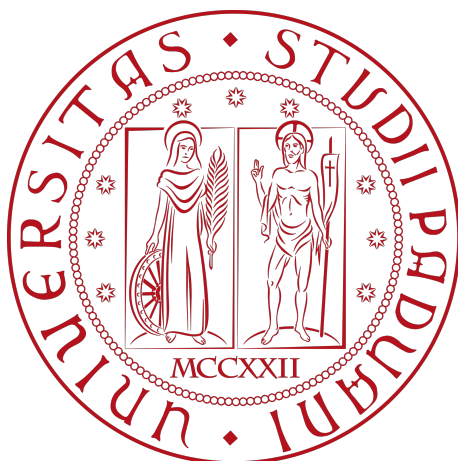


Università degli Studi di Padova

DIPARTIMENTO DI MATEMATICA “TULLIO LEVI-CIVITA”

CORSO DI LAUREA IN INFORMATICA



**Analisi di sicurezza e compliance OWASP
per modelli di intelligenza artificiale
generativa**

Tesi di laurea

Relatore

Prof. Marco Zanella

Laureando

Carmelo Russello

Matricola 2076421

ANNO ACCADEMICO 2025-2026

"TBA"
— TBA.

Dedicato a TBA

Sommario

Il presente documento descrive il lavoro svolto durante il periodo di stage, della durata di circa trecentoventi ore, presso l'azienda Var Group S.p.A. L'azienda necessita di un sistema automatizzato per il *security testing* di modelli di intelligenza artificiale generativa, in particolare per l'analisi delle vulnerabilità OWASP *top 10* per AI generativa in modo tale da poter testare i modelli utilizzati nei propri progetti.

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](#)^[g]*web* per la visualizzazione dei risultati dei *test* eseguiti.
- Infine, era richiesta la stesura della documentazione tecnica relativa al sistema sviluppato e la redazione di un manuale utente per l'utilizzo del sistema.

“TBA”

— TBA TBA

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

1	Introduzione	1
1.1	L'azienda	1
1.1.1	Var Group S.p.A.	1
1.1.2	Opzioni tecnologiche	2
1.1.3	Ruolo dello <i>stage</i> nell'azienda	2
1.2	Introduzione al caso di studio	2
1.3	Software da sviluppare	3
1.3.1	Necessità del testing per AI generativa	3
1.3.2	L'idea	3
1.4	Struttura del documento	4
2	Metodologie di sviluppo del software	5
2.1	Metodologia adottata	5
2.2	Introduzione al modello di sviluppo iterativo-incrementale	6
2.2.1	Introduzione	6
2.2.2	Origine	6
2.2.3	Fasi del modello di sviluppo iterativo-incrementale	6
2.3	Applicazione del modello iterativo-incrementale nel progetto	7
2.3.1	Motivazioni della scelta	7
2.3.2	Confronto con altre metodologie	7
2.3.3	Metodologia operativa adottata	8
2.3.4	Esempio di iterazione settimanale	8
3	Descrizione dello stage	10
3.1	Requisiti e obiettivi	11
3.1.1	Obiettivi obbligatori	11
3.1.2	Obiettivi desiderabili	11
3.2	Pianificazione	12
3.2.1	Settimana 1	13
3.2.2	Settimana 2	14
3.2.3	Settimana 3	14
3.2.4	Settimana 4	15
3.2.5	Settimana 5	15
3.2.6	Settimana 6	16
3.2.7	Settimana 7	16
3.2.8	Settimana 8	17
3.3	Pianificazione in retrospettiva	17

4	Analisi dei requisiti	18
4.1	Casi d'uso	18
4.2	Tracciamento dei requisiti	32
5	Progettazione e tecnologie	34
5.1	Tecnologie e strumenti	34
5.1.1	Strumenti analizzati	35
5.1.2	Strumenti utilizzati	37
5.2	Progettazione	37
5.2.1	Progettazione <i>Backend</i>	37
5.2.2	Progettazione <i>Frontend</i>	37
5.3	<i>Design Pattern</i> utilizzati	38
6	Conclusioni	39
6.1	Consuntivo finale	39
6.2	Raggiungimento degli obiettivi	39
6.3	Conoscenze acquisite	39
6.4	Valutazione personale	39
A	Appendice A	40
	Acronimi e abbreviazioni	42
	Glossario	43
	Bibliografia	48

Elenco delle figure

1.1	Logo di Var Group S.p.A.	2
2.1	Modello di sviluppo iterativo-incrementale	5
4.1	Use Case - UC1 to UC4	19
4.2	Use Case - UC4 to UC15	22
4.3	Use Case - UC16 to UC19	23
5.1	Matrice di Valutazione degli Strumenti Analizzati	35

Elenco delle tabelle

2.1	Esempio di iterazione settimanale	9
3.1	Pianificazione delle attività di progetto	12
4.1	Tabella del tracciamento dei requisiti funzionali	33
4.2	Tabella del tracciamento dei requisiti qualitativi	33
4.3	Tabella del tracciamento dei requisiti di vincolo	33

Capitolo 1

Introduzione

Il presente capitolo mira a fornire una panoramica generale del contesto in cui si inserisce il progetto di stage, illustrando l'azienda ospitante, le tecnologie utilizzate e le motivazioni alla base dello sviluppo del software proposto. Verranno inoltre delineati gli obiettivi principali del progetto e le sfide affrontate durante il suo svolgimento.

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.

1.1.1 Var Group S.p.A.

Var Group S.p.A. è una delle principali aziende italiane nel settore dell'Information Technology, con una vasta gamma di servizi che spaziano dalla consulenza IT alla fornitura di soluzioni software personalizzate. Fondata nel 1993, l'azienda ha sede a Milano e opera a livello nazionale e internazionale, servendo clienti in diversi settori industriali.

Con oltre 30 anni di esperienza, Var Group si è affermata come un partner affidabile per le imprese che cercano di innovare e ottimizzare i propri processi attraverso l'adozione di tecnologie avanzate. L'azienda si distingue per la sua capacità di integrare soluzioni tecnologiche all'avanguardia con una profonda comprensione delle esigenze di business dei clienti.

Var Group offre una vasta gamma di servizi, tra cui consulenza strategica, sviluppo di *software* su misura, implementazione di sistemi [Enterprise Resource Planning \(ERP\)](#)^[8], gestione dell'infrastruttura IT e servizi di *cloud computing*. Inoltre, l'azienda si impegna a fornire soluzioni che promuovono la trasformazione digitale, aiutando le imprese a migliorare l'efficienza operativa e a sfruttare le opportunità

offerte dalle nuove tecnologie.



Figura 1.1: Logo di Var Group S.p.A.

1.1.2 Opzioni tecnologiche

Grazie alla vasta gamma di collaborazioni che Var Group intrattiene con diverse aziende tecnologiche leader del settore, come AWS e Azure, che forniscono soluzioni comunemente utilizzate in ambienti di lavoro reali, ho avuto l'opportunità di lavorare con tecnologie all'avanguardia e di acquisire competenze pratiche preziose per il mio futuro professionale.

L'azienda mi ha permesso di scegliere quali tecnologie utilizzare tra quelle offerte e supportate, garantendo la disponibilità per l'acquisto delle licenze *software* necessarie allo sviluppo del progetto di *stage*, il che ha reso il lavoro più fluido e stimolante, poiché ho potuto impiegare tecnologie che non avrei facilmente testato in altri contesti.

Var Group, inoltre, lavora attivamente nell'ambito dell'intelligenza artificiale e nello sviluppo di [chatbot](#)^[g], il che mi ha consentito di approfondire le mie conoscenze grazie alla collaborazione con colleghi esperti del settore, tra cui laureati magistrali in intelligenza artificiale, e di partecipare a progetti reali in cui l'AI è integrata in modo professionale.

1.1.3 Ruolo dello *stage* nell'azienda

Il progetto di *stage* si inserisce nel contesto generale del *workflow* aziendale, in particolare quello dello sviluppo di *software* con l'integrazione di modelli di intelligenza artificiale generativa. Il progetto di *stage* infatti permetterà di testare la sicurezza e l'affidabilità dei modelli di intelligenza artificiale generativa che Var Group integra nei propri prodotti *software*, garantendo che questi rispettino le linee guida di sicurezza stabilite da [Open Web Application Security Project \(OWASP\)](#)^[g].

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. Esempi lampanti sono Amazon con il suo *chatbot* basato su AI Rufus per aiutare gli utenti

con gli acquisti sulla propria piattaforma e l'integrazione di *chatbot* in numerosi *browser* come Brave o Opera per aiutare gli utenti nella navigazione *web* e avere una esperienza più personalizzata di ricerca.

1.3 Software da sviluppare

Il software da sviluppare durante il periodo di *stage* consiste in un sistema automatizzato per il *security testing* di modelli di intelligenza artificiale generativa, con particolare attenzione all'analisi delle vulnerabilità OWASP *top 10* per AI generativa.

1.3.1 Necessità del testing per AI generativa

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](#)^[g], [Data leakage](#)^[g] e [Allucinazioni](#)^[g] dei modelli. Questi rischi possono compromettere la sicurezza delle applicazioni, la *privacy* degli utenti e l'integrità dei dati trattati.

Un esempio reale di rischio è l'integrazione da parte di una azienda di un modello di intelligenza artificiale privo di [guardrail](#)^[g] nel proprio applicativo che permette ad utenti inesperti o malintenzionati di utilizzare il modello per scopi non previsti come ottenere informazioni su come compiere atti illegali o dannosi. Per riuscire a mitigare i rischi associati all'utilizzo di modelli di intelligenza artificiale generativa, è fondamentale seguire delle linee 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 un [LLM](#)^[g] 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.

1.3.2 L'idea

L'idea iniziale per affrontare il problema della sicurezza nelle applicazioni che integrano modelli di intelligenza artificiale generativa consisteva nell'impiegare strumenti già disponibili sul mercato, per poi catalogare e analizzare i risultati al fine di identificare le vulnerabilità dei vari modelli. Dopo un'attenta riflessione, ho deciso di sviluppare un applicativo completo e la relativa logica di *testing*, così da avere maggiore controllo sul processo di verifica e sulle opzioni di personalizzazione. In particolare il mio interesse si è concentrato sulla personalizzazione e l'intercambiabilità dei [Dataset](#)^[g] utilizzabili per l'analisi e il *testing* delle LLM. Un altro fattore che mi ha portato alla creazione di un applicativo personalizzato è quello della possibilità di *testing* filtrando soltanto le categorie interessate senza dover effettuare test completi in casi in cui non sia necessario o richiesto. Un

altro scenario per cui ho scelto questo approccio per la risoluzione del problema è quello in cui l'azienda necessita di particolari modifiche nei casi di *testing*, cosa impossibile con *tool* acquistabili poiché non facilmente personalizzabili; in questo modo, nel caso l'azienda necessiti di un caso di *testing* specifico per un particolare modello, questo può essere implementato e testato in modo più semplice ed efficiente. Questo permette anche di intercambiare i modelli di valutazione delle risposte del modello testato in modo tale da avere più risultati da analizzare per un *testing* più comprensivo e accurato.

Questa scelta permette di adattare le procedure di valutazione ai diversi contesti applicativi, ridurre la dipendenza da *tool* proprietari e favorire l'evoluzione del prodotto in risposta a nuove minacce e requisiti normativi. Inoltre, l'approccio garantisce una migliore interoperabilità con i sistemi esistenti e facilita la manutenzione, la scalabilità e il *versioning* dei casi di test.

1.4 Struttura del documento

Il documento si divide in sei capitoli principali, inclusa questa introduzione.

Il [secondo capitolo](#) fornisce una panoramica delle tecnologie e degli strumenti utilizzati per lo sviluppo del prototipo, nonché una descrizione dettagliata della progettazione e dell'architettura del sistema.

Il [terzo capitolo](#) descrive il processo di implementazione del prototipo, illustrando le sfide affrontate e le soluzioni adottate.

Il [quarto capitolo](#) presenta i risultati ottenuti durante la fase di testing del prototipo, analizzando le prestazioni e l'efficacia del sistema sviluppato.

Il [quinto capitolo](#) descrive la progettazione e le tecnologie utilizzate.

Il [sesto e ultimo capitolo](#) riassume le conclusioni del progetto di *stage*, evidenziando i principali risultati raggiunti e proponendo possibili sviluppi futuri per migliorare ulteriormente il prototipo.

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).

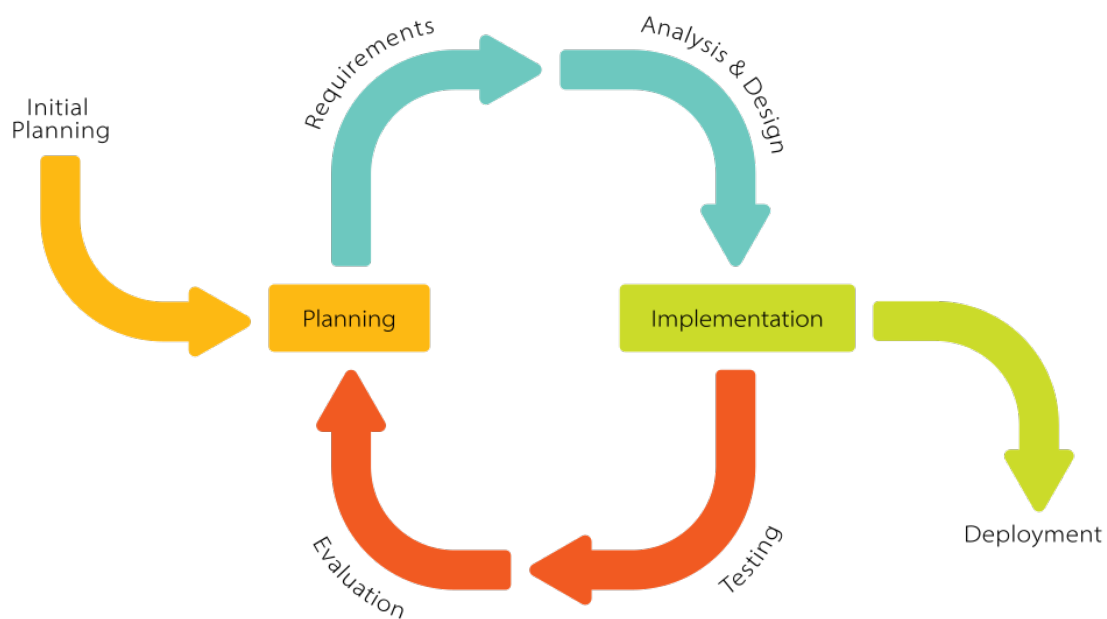


Figura 2.1: Modello di sviluppo iterativo-incrementale

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 nell'uso di [Iterative and Incremental Development \(IID\)](#)^[g] 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 \(DoD\)](#)^[g] 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 architettureali, 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 Confronto con altre metodologie

La metodologia iterativo-incrementale è stata preferita rispetto ad altre alternative per le seguenti ragioni:

- **Waterfall:** il modello a cascata prevede fasi sequenziali rigide (analisi, progettazione, implementazione, test, rilascio) senza possibilità di tornare indietro. Questo approccio non era adatto al progetto di *stage* poiché i requisiti relativi al *testing* di LLM sono in continua evoluzione e richiedono frequenti adattamenti basati sui risultati ottenuti;
- **Scrum puro:** sebbene *Scrum* sia una metodologia agile molto diffusa, la sua implementazione completa (con *daily standup*, *Sprint* di durata fissa, ruoli definiti come *Scrum Master* e *Product Owner*) risultava sovradimensionata

per un progetto individuale di *stage*. Si è quindi optato per un approccio più snello che mantiene i principi iterativi senza l'*overhead* organizzativo di *Scrum*;

- **Kanban**: pur essendo un metodo flessibile, *Kanban* si concentra sul flusso continuo senza iterazioni definite, rendendo più difficile la pianificazione e la verifica periodica degli obiettivi con il tutor aziendale.

2.3.3 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](#)^[g] 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.

2.3.4 Esempio di iterazione settimanale

Di seguito viene riportato un esempio rappresentativo di come si è svolta una tipica iterazione settimanale durante il progetto di *stage*.

Questo *template* è stato utilizzato come traccia per documentare ogni iterazione, garantendo coerenza nella reportistica e facilitando il confronto tra settimane successive.

Tabella 2.1: Esempio di iterazione settimanale

Elemento	Descrizione
Settimana	[Numero settimana, es. Settimana 3]
Goal	[Obiettivo principale della settimana, es. “Implementare il modulo di <i>testing</i> per <i>prompt injection</i> ”]
Attività pianificate	1) [Attività 1, es. Analisi dei <i>dataset</i> esistenti] 2) [Attività 2, es. Sviluppo dello <i>script</i> di <i>testing</i>] 3) [Attività 3, es. Esecuzione test su modello target]
Deliverable	[Output atteso, es. <i>Report</i> con risultati dei test e analisi delle vulnerabilità rilevate]
Impedimenti	[Eventuali ostacoli incontrati, es. Limite di <i>rate</i> delle API del modello]
Esito	[Risultato finale, es. <i>Goal</i> raggiunto / Parzialmente raggiunto / Non raggiunto]
Feedback tutor	[Note e suggerimenti ricevuti durante la <i>review</i>]

Capitolo 3

Descrizione dello stage

In questo capitolo viene fornita una panoramica del progetto di stage incentrato sulla sicurezza dell'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.

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 prioritizzati 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.

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*^[g] 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..

4. Mancanza di esperienza pregressa su *OWASP* o sicurezza *AI*.

Descrizione: La poca familiarità con le *best practice* 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..

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..

3.1 Requisiti e obiettivi

3.1.1 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.

3.1.2 Obiettivi desiderabili

- *Dashboard* interattiva con visualizzazioni avanzate.
- Integrazione del prototipo in *pipeline* [Continuous Integration/Continuous Deployment \(CI/CD\)](#)^[8] esistente.

- Estensione dei *test* ad altri *framework* oltre [Gandalf Test](#)^[g].
- Raccomandazioni per un *framework* interno di *AI Security by Design*.

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

Tabella 3.1: Pianificazione delle attività di progetto

3.2.1 Settimana 1

Durante la prima settimana di lavoro il *focus* è stato posto sullo studio di *OWASP* e sulla comprensione dell'ambito di studio del progetto. In questo periodo è stata fatta un'estensiva ricerca sulla *top 10* delle vulnerabilità delle LLM secondo *OWASP* e dei metodi di *testing*, attacco 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 il concetto di *red teaming* e le sue applicazioni pratiche nel contesto delle LLM, poiché 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](#)^[8], 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.

3.2.2 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.

3.2.3 Settimana 3

Durante lo svolgimento della terza settimana ho avuto l'opportunità di mettere in pratica le conoscenze acquisite nelle settimane precedenti, applicando i *tool* di *security testing* selezionati a dei modelli di AWS Bedrock forniti dall'azienda. Questa fase sperimentale è stata fondamentale per valutare l'efficacia degli strumenti in scenari reali e per identificare eventuali criticità o limitazioni nell'uso pratico. Ho testato diversi modelli forniti da Bedrock come Nova micro e Nova pro, eseguendo una serie di *test* mirati a rilevare vulnerabilità specifiche secondo le categorie *OWASP* identificate in precedenza anche se è importante prendere in considerazione il fatto che i modelli di bedrock non dispongono di dati di training specifici o dati degli utenti come i chatbot e quindi non è stato possibile effettuare il testing delle categorie *OWASP* che fanno leva su queste caratteristiche. Durante questa fase, ho documentato attentamente i risultati ottenuti, annotando sia i successi che le difficoltà incontrate nell'uso dei *tool*.

Ho analizzato i dati raccolti per valutare l'accuratezza e l'affidabilità dei *test*, confrontando i risultati con le aspettative basate sulla letteratura e sulle best practice del settore. Questa esperienza pratica mi ha permesso di comprendere meglio le dinamiche del *security testing* delle LLM e di identificare aree di miglioramento sia nei *tool* utilizzati che nelle strategie di *testing* adottate.

3.2.4 Settimana 4

La quarta settimana di lavoro è stata una delle settimane più complicate dell'intero percorso di stage. Il problema principale è stato l'impossibilità di effettuare dei *test* su componenti reali del *team* a causa di problemi di accesso e permessi. Nonostante i numerosi tentativi di risolvere la situazione, non sono riuscito ad ottenere l'accesso necessario per eseguire i *test* sul *chatbot* aziendale, il quale era il componente principale su cui avrei dovuto effettuare i *test*. A causa di questi problemi, ho dovuto adattare il mio piano di lavoro e effettuare il testing su altre piattaforme, in particolare sono riuscito ad interfacciarmi tramite API al chatbot del gandalf test fornito da Lakera per effettuare dei test sul corretto funzionamento dell'applicativo. Inoltre ho dedicato del tempo a migliorare il sistema di *testing* sviluppato nelle settimane precedenti, ottimizzando gli *script* e aggiungendo nuove funzionalità per renderlo più robusto e flessibile. Nonostante le difficoltà incontrate, sono riuscito a fare progressi significativi nel miglioramento del sistema di *testing*, preparandomi al meglio per quando finalmente avrò accesso ai componenti reali del *team*. Al termine della settimana, ho stilato un elenco di azioni da intraprendere non appena avrò risolto i problemi di accesso, in modo da poter riprendere rapidamente il lavoro sui *test* non appena possibile.

3.2.5 Settimana 5

Durante la quinta settimana di lavoro, l'attenzione si è concentrata sulla creazione di un sistema automatizzato per l'esecuzione dei *test* di sicurezza sulle LLM e sulla generazione di *report* dettagliati sui risultati ottenuti. Ho iniziato sviluppando una serie di *script* in Python che consentono di eseguire i *test* in modo sistematico e ripetibile, integrando le funzionalità dei *tool* di *security testing* selezionati nelle settimane precedenti. Questi *script* sono stati progettati per essere modulari e facilmente estendibili, in modo da poter aggiungere nuovi scenari di *test* o adattare quelli esistenti in base alle esigenze specifiche del progetto. I risultati dei test giudicati dal modello di scoring utilizzato sono stati raccolti e organizzati in un formato strutturato, facilitando l'analisi e la visualizzazione dei dati. Ho implementato un sistema di generazione automatica di *report* che sintetizza i risultati dei *test*, evidenziando le vulnerabilità identificate, il loro impatto potenziale e le raccomandazioni per la mitigazione. Questi *report* sono stati progettati per essere chiari e comprensibili, in modo da poter essere utilizzati sia da tecnici esperti che da stakeholder non tecnici. Inoltre, ho definito una serie di [KPI](#)^[g] per misurare l'efficacia del sistema di *testing* e la conformità alle linee guida *OWASP*, consentendo così di monitorare i progressi nel tempo e identificare aree di miglioramento. Al termine della settimana, il sistema automatizzato era funzionale e pronto per essere utilizzato nelle fasi successive del progetto, rappresentando un passo significativo verso il raggiungimento degli obiettivi prefissati.

3.2.6 Settimana 6

Nel corso della sesta settimana di lavoro, l'attenzione si è concentrata sulla creazione della *dashboard* interattiva richiesta dal progetto di *stage*. Il *framework* scelto per lo sviluppo della *dashboard* è stato *React*, scelto poiché è uno strumento che viene utilizzato regolarmente dai membri dell'azienda per lo sviluppo dei loro prodotti garantendo supporto e aiuto in caso di problemi nell'utilizzo dello strumento, inoltre permette una facile integrazione del *prototipo* nel contesto aziendale esistente. La *dashboard* è stata progettata per fornire una visualizzazione chiara e intuitiva dei risultati dei *test* di sicurezza eseguiti sulle *LLM*, con particolare attenzione alla rappresentazione delle vulnerabilità identificate in relazione alle categorie *OWASP*. Ho implementato grafici interattivi e tabelle che consentono agli utenti di esplorare i dati in modo dinamico, filtrando e ordinando i *test* in base a vari criteri come data di esecuzione, tipo di *scorer* utilizzato e *coverage* dei *test*. Ho inoltre integrato una visualizzazione personalizzata per ogni singolo *test* in modo tale da poter analizzare l'andamento di ogni singolo *test* per ogni categoria *OWASP*, permettendo così agli utenti di identificare rapidamente le aree di maggiore rischio, da qui è anche possibile visualizzare ogni singolo *prompt* inviato con relativa risposta e *rationale* dello *scorer* che spiega l'andamento del *test* e controlla se il *test* è stato passato con successo o ha fallito. La *dashboard* è stata sviluppata con un'attenzione particolare all'usabilità, garantendo che anche utenti non tecnici possano navigare facilmente tra i dati e comprendere le implicazioni delle vulnerabilità identificate. Al termine della settimana, la *dashboard* era funzionale e pronta per essere integrata con il *prototipo* di *test* sviluppato nelle settimane precedenti, rappresentando un passo significativo verso il completamento degli obiettivi del progetto di *stage*.

3.2.7 Settimana 7

Durante la settima settimana di lavoro si è posta l'attenzione alla risoluzione dei problemi della quarta settimana relativi al *testing* del *prototipo* su componenti reali del *team*. Dopo aver risolto i problemi e ottenuto l'accesso al *chatbot* aziendale sono riuscito ad effettuare dei *test* su un *prodotto* effettivo in modo tale da valutare l'accuratezza e l'efficienza del mio *prototipo*, purtroppo per il poco tempo il *testing* non è potuto essere approfondito quanto avremmo voluto. In questa settimana ho inoltre migliorato il sistema di *testing* dei *modelli* grazie ai risultati ottenuti dal *testing* sul *chatbot* aziendale e ho formattato il *codice* in modo tale da renderlo più leggibile e manutenibile. Ho inoltre migliorato il sistema di *reportistica* automatica in modo tale da poter generare *report* più dettagliati e completi sui risultati dei *test* eseguiti. Infine, ho integrato il *prototipo* con la *dashboard* sviluppata nella settimana precedente, permettendo così una visualizzazione immediata e interattiva dei risultati dei *test* direttamente all'interno della *dashboard*. Questa integrazione ha migliorato notevolmente l'usabilità del *prototipo*, consentendo agli utenti di accedere facilmente ai dati e alle analisi dei *test* di sicurezza delle *LLM*. Sotto consiglio del tutor aziendale ho aggiunto alla *dashboard* un *mock button* che verrà poi implementato e servirà per l'aggiunta di nuovi *dataset* per il *testing* in modo tale da poter ampliare il numero di *test* possibili oltre al *testing* *OWASP*. Al termine

della settimana, il *prototipo* era completamente integrato con la *dashboard* e pronto per essere utilizzato per ulteriori *test* e analisi.

3.2.8 Settimana 8

L'ottava e ultima settimana è stata dedicata alla redazione della documentazione tecnica e alla creazione del manuale utente per il *prototipo* sviluppato. È stato scelto di integrare il manuale utente direttamente nell'*applicazione web* del *prototipo* in modo tale da renderlo facilmente accessibile agli utenti finali. La redazione della documentazione è stata fondamentale per lo sviluppo della tesi, in particolare la stesura della documentazione si è concentrata sull'analisi delle tecnologie utilizzate e le ragioni dietro la scelta di tali tecnologie.

Durante questo periodo è stata inoltre preparata la *presentazione finale* della tesi, che includeva una panoramica del progetto, i risultati ottenuti e le conclusioni tratte. La *presentazione* è stata strutturata in modo da evidenziare i punti chiave del lavoro svolto e per comunicare efficacemente i risultati raggiunti durante lo *stage*. L'azienda ospitante ha fornito uno spazio in cui ho potuto presentare il lavoro svolto davanti a colleghi e esperti nel settore per emulare una situazione reale di *presentazione professionale* e prepararmi alla *presentazione* della tesi di laurea.

3.3 Pianificazione in retrospettiva

La pianificazione del progetto di stage si è rivelata in gran parte efficace, con la maggior parte delle attività previste completate nei tempi stabiliti. Tuttavia, alcune sfide impreviste, come i problemi di accesso ai componenti reali del *team*, hanno richiesto un adattamento del piano di lavoro.

Nonostante queste difficoltà, sono riuscito a mantenere il focus sugli obiettivi principali del progetto e a fare progressi significativi in tutte le fasi pianificate e il tutor aziendale si è detto soddisfatto del lavoro svolto e dei risultati ottenuti, sottolineando l'importanza del progetto per l'azienda e il valore aggiunto apportato dal *prototipo* sviluppato. La flessibilità nella gestione del tempo e delle risorse si è dimostrata cruciale per affrontare gli imprevisti e garantire il successo del progetto complessivo.

Capitolo 4

Analisi dei requisiti

In questo capitolo vengono analizzati i requisiti funzionali e non funzionali del sistema, con particolare attenzione ai casi d'uso principali che descrivono le interazioni tra gli utenti e il sistema stesso.

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 \(UML\)](#)^[8] dedicati alla descrizione delle funzioni o servizi offerti da un sistema, così come sono percepiti e utilizzati dagli attori che interagiscono col sistema stesso.

UC1: Visualizzazione dashboard

Attori Principali: Utente.

Precondizioni: L'utente ha effettuato l'accesso al sistema.

Descrizione: L'utente visualizza la dashboard principale con le statistiche dei test eseguiti sui modelli di intelligenza artificiale generativa..

Postcondizioni: L'utente ha visualizzato una panoramica dei risultati dei test, inclusi grafici e tabelle riassuntive..

UC2: Visualizzazione statistiche

Attori Principali: Utente.

Precondizioni: L'utente sta visualizzando la dashboard.

Descrizione: L'utente visualizza le card con le statistiche aggregate dei test eseguiti sui modelli di intelligenza artificiale generativa..

Postcondizioni: L'utente ha visualizzato le statistiche aggregate dei test eseguiti, come il numero di test superati, il numero di test effettuati e altre metriche rilevanti..

UC3: Visualizzazione grafico temporale

Attori Principali: Utente.

Precondizioni: L'utente sta visualizzando la dashboard.

Descrizione: L'utente visualizza un grafico temporale con l'andamento dei test eseguiti sui modelli di intelligenza artificiale generativa..

Postcondizioni: L'utente ha visualizzato il grafico temporale con l'andamento dei test..

UC4: Visualizzazione tabella dei test

Attori Principali: Utente.

Precondizioni: L'utente sta visualizzando la dashboard.

Descrizione: L'utente visualizza una tabella con i test eseguiti sui target scelti durante la fase di testing.

Postcondizioni: L'utente ha visualizzato la tabella con i test eseguiti sui target scelti durante la fase di testing..

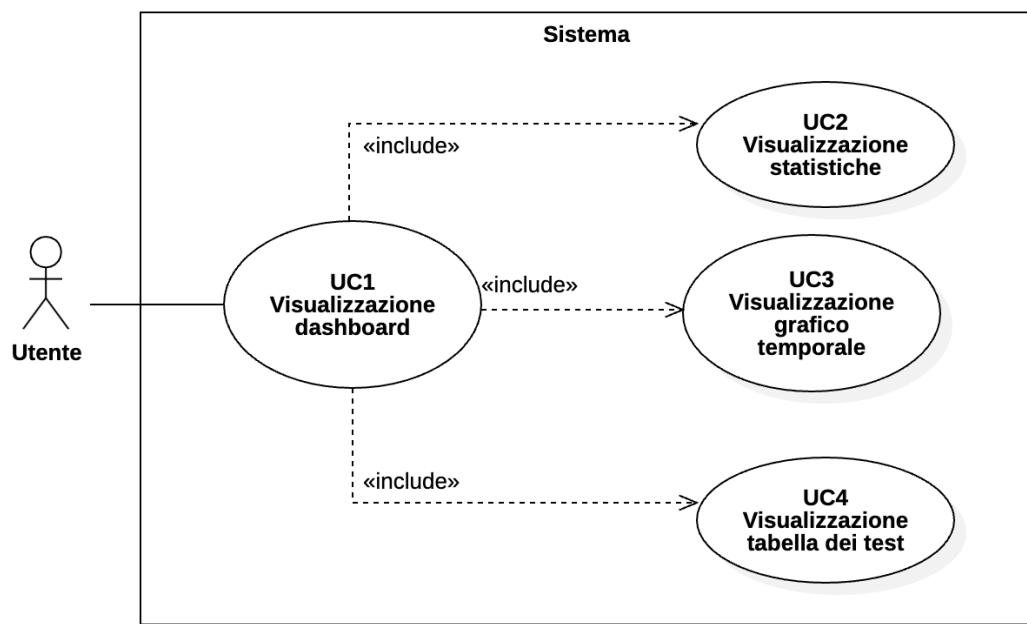


Figura 4.1: Use Case - UC1 to UC4

UC5: Visualizzazione singolo test

Attori Principali: Utente.

Precondizioni: L'utente ha selezionato un test dalla tabella dei test.

Descrizione: L'utente visualizza le informazioni dettagliate di un singolo test eseguito sul target di riferimento che comprende nome, scorer, dataset, target, status, coverage ed età del test..

Postcondizioni: L'utente ha visualizzato le informazioni dettagliate del singolo test selezionato..

UC6: Visualizzazione nome test

Attori Principali: Utente.

Precondizioni: L'utente sta visualizzando un singolo test.

Descrizione: L'utente visualizza il nome assegnato allo specifico test in esame.

Postcondizioni: L'utente ha visualizzato il nome del test selezionato..

UC7: Visualizzazione scorer test

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

UC8: Visualizzazione dataset test

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

UC9: Visualizzazione target test

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

UC10: Visualizzazione status test

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

UC11: Visualizzazione coverage test

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

UC12: Visualizzazione età test

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

UC13: Visualizzazione Status "Done"

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

UC14: Visualizzazione Status "Cancelled"

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

UC15: Visualizzazione Status "Failed"

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

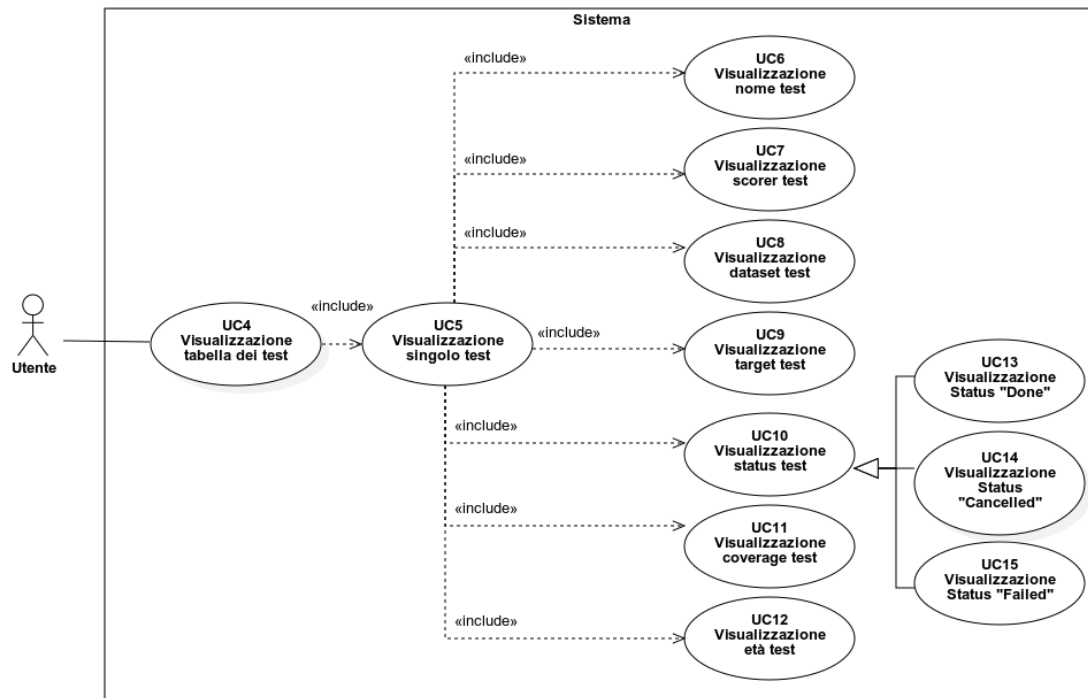


Figura 4.2: Use Case - UC4 to UC15

UC16: Visualizzazione dettagli singolo test

Attori Principali: Utente.

Precondizioni: L'utente ha effettuato l'accesso alla dashboard e ha selezionato un test dalla lista dei test.

Descrizione: L'utente visualizza i dettagli completi di un singolo test.

Postcondizioni: L'utente ha visualizzato i dettagli completi del singolo test selezionato.

UC17: Visualizzazione attacchi riusciti

Attori Principali: Utente.

Precondizioni: L'utente si trova nella pagina dei dettagli di un singolo test.

Descrizione: L'utente visualizza gli attacchi riusciti durante il test.

Postcondizioni: L'utente ha visualizzato gli attacchi riusciti durante il test.

UC18: Visualizzazione attacchi falliti

Attori Principali: Utente.

Precondizioni: L'utente si trova nella pagina dei dettagli di un singolo test.

Descrizione: L'utente visualizza gli attacchi falliti durante il test.

Postcondizioni: L'utente ha visualizzato gli attacchi falliti durante il test.

UC19: Filtrare attacchi per categoria OWASP

Attori Principali: Utente.

Precondizioni: L'utente si trova nella pagina dei dettagli di un singolo test.

Descrizione: L'utente filtra gli attacchi visualizzati in base alla categoria OWASP di appartenenza.

Postcondizioni: L'utente ha filtrato gli attacchi visualizzati in base alla categoria OWASP di appartenenza .

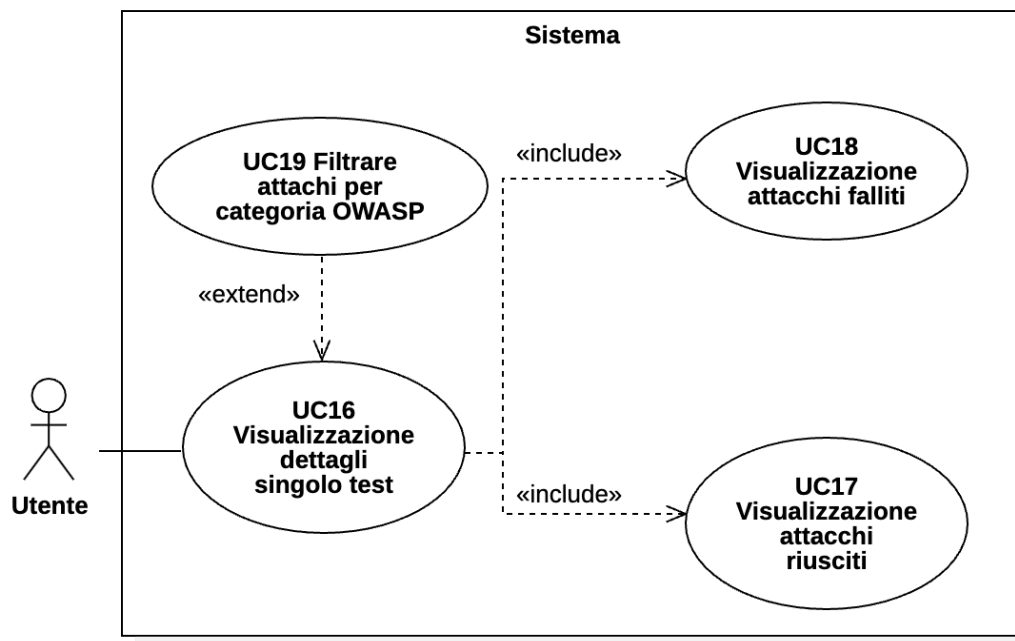


Figura 4.3: Use Case - UC16 to UC19

UC20:

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

UC21:

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

UC22:

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

UC23:

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

UC24:

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

UC25:

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

UC26:

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

UC27:

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

UC28:

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

UC29:

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

UC30:

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

UC31:

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

UC32:

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

UC33:

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

UC34:

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

UC35:

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

UC36:

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

UC37:

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

UC38:

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

UC39:

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

UC40:

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

UC41:

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

UC42:

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

UC43:

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

UC44:

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

UC45:

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

UC46:

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

UC47:

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

UC48:

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

UC49:

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

UC50:

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

UC51:

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

UC52:

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

UC53:

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

UC54:

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

UC55:

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

UC56:

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

UC57:

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

UC58:

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

UC59:

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

UC60:

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

UC61:

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

UC62:

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

UC63:

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

UC64:

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

UC65:

Attori Principali: Utente.

Precondizioni: Precondizione da definire.

Descrizione: Descrizione da definire.

Postcondizioni: Postcondizione da definire.

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 requisiti funzionali

Requisito	Descrizione	Use Case
RFN-1	L'interfaccia permette di visualizzare la dashboard principale con le statistiche dei test eseguiti sui modelli di intelligenza artificiale generativa	UC1
RFN-2	L'interfaccia permette di visualizzare le card con le statistiche aggregate dei test eseguiti sui modelli di intelligenza artificiale generativa	UC2
RFN-3	L'interfaccia permette di visualizzare un grafico temporale con l'andamento dei test eseguiti sui modelli di intelligenza artificiale generativa	UC3
RFN-4	L'interfaccia permette di visualizzare una tabella con i test eseguiti sui target scelti durante la fase di testing	UC4
RFN-5	L'interfaccia permette di visualizzare le informazioni dettagliate di un singolo test eseguito sul target di riferimento, inclusi nome, scorer, dataset, target, status, coverage ed età del test	UC5
RFN-6	L'interfaccia permette di visualizzare il nome assegnato a uno specifico test	UC6

Tabella 4.2: Tabella del tracciamento dei requisiti qualitativi

Requisito	Descrizione	Use Case
RQD-1	TBA	-

Tabella 4.3: Tabella del tracciamento dei requisiti di vincolo

Requisito	Descrizione	Use Case
RVO-1	TBA	-

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](#)^[8], 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 *tool* 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, obiettivo 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 *compliance*, 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 testare 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*, *jailbreak* 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 \(API\)](#)^[8] 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](#). [23](#), [27](#), [29](#)

CI/CD [Continuous Integration/Continuous Deployment](#). [11](#), [21](#), [22](#), [27](#), [30](#)

DoD [United States Department of Defense](#). [6](#), [27](#), [30](#)

ERP [Enterprise Resource Planning](#). [1](#), [27](#), [30](#)

IID [Iterative and Incremental Development](#). [6](#), [27](#), [31](#)

OWASP [Open Web Application Security Project](#). [2](#), [27](#), [32](#)

UML [Unified Modeling Language](#). [14](#), [27](#), [32](#)

Glossario

AI generativa il termine *AI generativa* si riferisce a una categoria di modelli di intelligenza artificiale progettati per generare nuovi contenuti, come testo, immagini o video. Questi modelli sono addestrati su grandi quantità di dati prodotti dall'uomo e utilizzano tecniche avanzate di apprendimento automatico per creare output che possono essere simili a quelli creati dagli esseri umani.. [27](#)

Allucinazioni la *allucinazione* si riferisce a un fenomeno in cui un modello di intelligenza artificiale genera output che sono errati, fuorvianti o privi di fondamento nella realtà. Questo può accadere quando il modello interpreta in modo errato l'input o quando i dati di addestramento contengono informazioni imprecise.. [3](#), [27](#)

Analisi dinamica la *analisi dinamica* è una tecnica di verifica del software che analizza il comportamento del codice durante l'esecuzione, al fine di identificare vulnerabilità, errori o violazioni delle best practice.. [27](#)

Analisi statica la *analisi statica* è una tecnica di verifica del software che analizza il codice sorgente senza eseguirlo, al fine di identificare potenziali vulnerabilità, errori o violazioni delle best practice.. [27](#)

API in informatica con il termine *Application Programming Interface API* (ing. interfaccia di programmazione di un'applicazione) si indica ogni insieme di procedure disponibili al programmatore, di solito raggruppate a formare un set di strumenti specifici per l'espletamento di un determinato compito all'interno di un certo programma. La finalità è ottenere un'astrazione, di solito tra l'hardware e il programmatore o tra software a basso e quello ad alto livello semplificando così il lavoro di programmazione. [27](#), [28](#)

Backend Il *backend* è la parte di un'applicazione software che gestisce la logica di business, l'elaborazione dei dati e la comunicazione con il database. Opera lato server e non è direttamente visibile all'utente finale.. [27](#)

Black-box Il *testing black-box* è una metodologia di test in cui il tester non ha accesso al codice sorgente o alla struttura interna del sistema. I test vengono eseguiti basandosi esclusivamente sugli input e output del sistema.. [27](#)

Chatbot un *chatbot* è un programma informatico progettato per simulare una conversazione con utenti umani, tipicamente attraverso interfacce testuali o

vocali. I chatbot moderni spesso integrano modelli di intelligenza artificiale generativa per fornire risposte più naturali e contestuali.. [2](#), [27](#)

CI/CD La *CI/CD* (Continuous Integration/Continuous Deployment) è una pratica di sviluppo che automatizza l'integrazione del codice e il rilascio, migliorando la qualità del software e riducendo i tempi di consegna.. [27](#), [28](#)

Compliance La *compliance* indica la conformità a normative, standard, regolamenti e best practice applicabili a un determinato settore o ambito tecnologico.. [27](#)

Dashboard Una *dashboard* è un'interfaccia grafica che presenta in modo visuale e sintetico informazioni, metriche e indicatori chiave di performance (KPI), permettendo agli utenti di monitorare lo stato di un sistema o processo.. [iii](#), [27](#)

Data leakage il *data leakage* si riferisce alla situazione in cui informazioni sensibili o riservate vengono inavvertitamente esposte o divulgate a persone non autorizzate, spesso a causa di vulnerabilità nei sistemi di intelligenza artificiale generativa.. [3](#), [27](#)

Data poisoning Il *data poisoning* è un tipo di attacco informatico in cui dati malevoli vengono introdotti nel dataset di addestramento di un modello di machine learning per comprometterne il comportamento e le prestazioni.. [27](#)

Dataset Un *dataset* è una raccolta di dati utilizzata per addestrare, testare e valutare modelli di intelligenza artificiale. I dataset possono variare in dimensione, formato e contenuto, e la loro qualità è fondamentale per il successo dei modelli di machine learning.. [3](#), [27](#)

DeepEval / DeepTeam *DeepEval* è un framework open source per la valutazione e il benchmarking di modelli di AI generativa. *DeepTeam* è il relativo strumento di red teaming per testare la sicurezza dei modelli.. [27](#)

DoD il *Department of Defense (DoD)* è il Dipartimento della Difesa degli Stati Uniti d'America, responsabile della sicurezza nazionale e delle forze armate statunitensi. Nel contesto dello sviluppo software, il DoD ha storicamente promosso e standardizzato metodologie di sviluppo iterativo-incrementale per i propri progetti.. [27](#), [28](#)

ERP l'*Enterprise Resource Planning (ERP)* è un sistema software integrato che consente alle aziende di gestire e automatizzare i principali processi di business, come contabilità, gestione delle risorse umane, produzione, supply chain e vendite, attraverso un'unica piattaforma centralizzata.. [27](#), [28](#)

Frontend Il *frontend* è la parte di un'applicazione software con cui l'utente interagisce direttamente. Include l'interfaccia utente, il design grafico e tutti gli elementi visuali dell'applicazione.. [27](#)

Galileo *Galileo* è una piattaforma commerciale per la valutazione e l'osservabilità di applicazioni basate su AI generativa, con SDK disponibili in Python e TypeScript.. [27](#)

Gandalf Test il *Gandalf Test* è un gioco creato da Lakera che permette all'utente di entrare nel mondo del testing delle LLM fornendo vari livelli di difficoltà incrementale in cui l'obiettivo è estrapolare una password da un LLM vulnerabile.. [11](#), [12](#), [21](#), [23](#), [27](#)

Garak *Garak* è uno scanner di vulnerabilità open source sviluppato da NVIDIA per testare modelli di AI generativa, focalizzato su attacchi come prompt injection, jailbreak e data leakage.. [27](#)

Giskard *Giskard* è una piattaforma di red teaming automatizzato per testare e analizzare modelli di AI, disponibile sia come servizio cloud a pagamento che come libreria Python open source.. [27](#)

Guardrail nel contesto dell'intelligenza artificiale, i *guardrail* sono meccanismi di sicurezza e controllo implementati per limitare o prevenire comportamenti indesiderati dei modelli di AI. Includono filtri sui contenuti, limiti sugli argomenti trattabili e sistemi di moderazione automatica delle risposte.. [3](#), [27](#)

IID lo sviluppo *Iterative and Incremental Development (IID)* è una metodologia di sviluppo software che combina la progettazione iterativa con un modello incrementale di costruzione. Il prodotto viene sviluppato attraverso cicli ripetuti (iterazioni) e in porzioni più piccole (incrementi), permettendo agli sviluppatori di sfruttare ciò che è stato appreso durante lo sviluppo delle versioni precedenti e di adattarsi ai cambiamenti dei requisiti.. [27](#), [28](#)

Jailbreak Nel contesto dell'intelligenza artificiale, il *jailbreak* è una tecnica utilizzata per aggirare le restrizioni e i filtri di sicurezza implementati in un modello di linguaggio, inducendolo a generare contenuti che normalmente sarebbero bloccati o censurati.. [12](#), [27](#)

KPI I *Key Performance Indicator (KPI)* sono metriche quantificabili utilizzate per valutare il successo di un'organizzazione, progetto o attività nel raggiungimento degli obiettivi prefissati.. [11](#), [27](#)

LakeraGuard *LakeraGuard* è una piattaforma commerciale di sicurezza per modelli di AI sviluppata da Lakera, che offre scansione delle vulnerabilità, monitoraggio delle minacce e protezione in tempo reale.. [27](#)

LangFuse *LangFuse* è una piattaforma open source per l'osservabilità e la valutazione di applicazioni basate su modelli di linguaggio, con funzionalità di tracciamento, analisi e debugging.. [27](#)

LLM in informatica, un *Large Language Model (LLM)* è un modello di intelligenza artificiale addestrato su un'enorme quantità di testo e codice con la capacità di comprendere, generare e tradurre il linguaggio umano. [3](#), [27](#)

Matrice di valutazione Una *matrice di valutazione* è uno strumento analitico che permette di confrontare diverse opzioni o alternative sulla base di criteri predefiniti, assegnando punteggi per facilitare la scelta della soluzione migliore.. [27](#)

Open source Modello di sviluppo e distribuzione del software in cui il codice sorgente è reso disponibile al pubblico e può essere utilizzato, modificato e ridistribuito secondo i termini della relativa licenza.. [27](#)

OWASP OWASP è una fondazione no profit che si occupa di migliorare la sicurezza del software. Offre risorse e linee guida per la sicurezza delle applicazioni web e di sistemi che integrano componenti di intelligenza artificiale (tra cui liste di vulnerabilità e best practice).. [27](#), [28](#)

Pipeline Una *pipeline* è una sequenza automatizzata di processi o fasi che trasformano un input in un output finale. Nel contesto dello sviluppo software, indica tipicamente l'insieme di operazioni automatiche di build, test e deploy.. [10](#), [27](#)

Prompt injection la *prompt injection* è una tecnica di attacco in cui un attaccante manipola l'input fornito a un modello di linguaggio per indurlo a generare output dannosi o indesiderati. Questo tipo di attacco può compromettere la sicurezza e l'affidabilità dei modelli di intelligenza artificiale generativa.. [3](#), [27](#)

PromptFoo *PromptFoo* è una piattaforma open source per il testing e la valutazione di modelli di linguaggio, con supporto per test automatizzati, integrazione CI/CD e analisi delle vulnerabilità OWASP per AI generativa.. [27](#)

Prototipo Un *prototipo* è una versione preliminare di un prodotto, utilizzata per testare e convalidare idee, concetti o funzionalità prima dello sviluppo finale. I prototipi possono variare in fedeltà e complessità, da schizzi su carta a modelli interattivi ad alta fedeltà.. [27](#)

PyRIT *PyRIT (Python Risk Identification Tool)* è uno strumento open source sviluppato da Microsoft Azure per l'identificazione e la mitigazione dei rischi di sicurezza nei modelli di intelligenza artificiale generativa.. [27](#)

Red teaming il *red teaming* è una pratica di sicurezza informatica in cui un team di esperti / uno strumento simula attacchi informatici per identificare vulnerabilità e punti deboli nei sistemi di difesa di un'organizzazione.. [27](#)

SaaS *Software as a Service (SaaS)* è un modello di distribuzione del software in cui le applicazioni sono ospitate da un provider di servizi e rese disponibili agli utenti tramite Internet, senza necessità di installazione locale.. [27](#)

SDK Un *Software Development Kit (SDK)* è un insieme di strumenti, librerie, documentazione e codice di esempio che consente agli sviluppatori di creare applicazioni per una specifica piattaforma o servizio.. [21](#), [27](#)

Slack *Slack* è una piattaforma di comunicazione aziendale che offre funzionalità di messaggistica istantanea, canali tematici, condivisione di file e integrazione con altri strumenti di produttività. È ampiamente utilizzata per la collaborazione in team di sviluppo software.. [8](#), [27](#)

Token Nel contesto dei modelli di linguaggio, un *token* è l'unità minima di testo elaborata dal modello. Può corrispondere a una parola, parte di una parola, un carattere o un simbolo di punteggiatura.. [27](#)

UML in ingegneria del software *UML*, *Unified Modeling Language* (ing. linguaggio di modellazione unificato) è un linguaggio di modellazione e specifica basato sul paradigma object-oriented. L'*UML* svolge un'importantissima funzione di "lingua franca" nella comunità della progettazione e programmazione a oggetti. Gran parte della letteratura di settore usa tale linguaggio per descrivere soluzioni analitiche e progettuali in modo sintetico e comprensibile a un vasto pubblico. [27](#), [28](#)

Caso d'uso Un *caso d'uso* (o *use case*) è una descrizione di come un utente interagisce con un sistema per raggiungere un obiettivo specifico. Nell'ingegneria del software viene utilizzato per definire i requisiti funzionali.. [27](#)

Bibliografia