UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Laurea Triennale in Informatica

TESI DI LAUREA

Classificazione di Requisiti di Sicurezza tramite tecniche di Quantum NLP

RELATORE

Prof. **Fabio Palomba**

Dott. **Francesco Casillo**

Università degli Studi di Salerno

CANDIDATO

David Coccorullo

Matricola: 0512110452

Anno Accademico 2022-2023

Questa tesi è stata realizzata nel

おめでとう！

Abstract

L'identificazione di requisiti non funzionali è una fase fondamentale e molto delicata dell'ingegneria dei requisiti, che viene spesso svolta con il supporto di modelli di Machine Learning i quali sfruttano le tecnologie di Natural Language Processing al fine di permettere alla macchina di interpretare il linguaggio umano.
Nel prossimo futuro, le innovazioni tecnologiche fornite dal quantum computing promettono di rivoluzionare completamente ciò che sappiamo sulla computazione; i processori quantistici sono infatti in grado di eseguire calcoli che sarebbero impossibili o estremamente difficili da eseguire su computer classici.
In questo lavoro si propone uno studio per l'identificazione di requisiti non funzionali di sicurezza di un sistema svolto tramite apprendimento quantistico, i cui risultati verranno confrontati con quelli ottenuti mediante apprendimento tradizionale, così da trarre conclusioni e considerazioni sulle differenze tra i due approcci in termini di performance, tempistiche e metriche valutative, con l'obiettivo di fornire una buona base per futuri sviluppi su nuove tecnologie che non hanno ancora espresso del tutto il loro potenziale.

Indice

Elenco delle figure

Elenco delle tabelle

CAPITOLO 1

Introduzione

1.1 Contesto Applicativo

L'elicitazione [1] è una fase fondamentale dell'Ingegneria del Software il cui scopo è quello di acquisire informazioni sul sistema da sviluppare e ricavarne dei **requisiti**, la cui identificazione, comprensione e analisi è fondamentale per una buona riuscita di un progetto.
Tra i requisiti non funzionali più importanti di un software si annoverano sicuramente quelli di **sicurezza**, che non concernono solo possibili problematiche legate ad accessi indesiderati al sistema o *data leaks* [2], ma anche, ad esempio, quelle che sorgono per una cattiva gestione delle autenticazioni o delle autorizzazioni del personale a cui è destinato il prodotto software, o ancora, i requisiti di privacy, il cui focus è quello di assicurare che i dati sensibili dell'utenza vengano trattati seguendo le normative del Regolamento Generale sulla Protezione dei Dati (GDRP) [3].
È dunque chiaro che errori commessi in questa frase potrebbero compremettere l'andamento dell'intera progettazione del software [4], a volte in maniera irreversibile.

1.2 Motivazioni e Obiettivi

Il **Natural Language Processing** è sempre più presente nella nostra quotidianità [5]: basti pensare al vasto numero di *chatbot* con cui interagiamo ogni giorno, che sono tra gli esempi più lampanti del vantaggio di questa branca del Machine Learning, riuscendo a permettere infatti all'utente finale di interagire con la macchina proprio come farebbe con un altro essere umano.
Per quanto apparentemente rivoluzionarie ed innovative, le attuali tecnologie di NLP si basano su standard di computazione classica ormai ampiamente diffusi e conosciuti dagli esperti del settore, mentre le recenti e sempre più promettenti conoscenze che si stanno acquisendo in merito alla **computazione quantistica** stanno aprendo nuove porte a diversi approcci alle pratiche di NLP [6]; non è ancora chiaro quanto la computazione quantistica applicata al Machine Learning e in particolare all'NLP sia effettivamente superiore o vantaggiosa rispetto a quella classica, siccome non si conosce ancora pienamente come sfruttare al meglio algoritmi quantistici, o semplicemente si è ancora bloccati da limitazioni tecniche e prestazionali.
A queste necessità emerse da un punto di vista principalmente di ricerca, si unisce una sempre crescente necessità di migliorare l'identificazione dei **requisiti di sicurezza** in un sistema; questa mansione è particolarmente importante e delicata per il buon funzionamento di un organismo aziendale: l'umanità, infatti, è sempre più "dipendente" dai sistemi informatici, e pericoli di ogni natura e specie sono sempre dietro l'angolo. Vale davvero la pena, dunque, orientare la ricerca sulla sicurezza, al fine di preservare i sistemi da eventuali danni, che se proprio non possono essere evitati vanno almeno contenuti.
Si propone quindi uno studio che vuole fruire da punto di partenza per un valido confronto tra la computazione classica e quella quantistica nei processi di identificazione di requisiti non funzionali di sicurezza, evidenziando le differenze principali in termini di prestazioni e metriche di valutazione, così da costruire non solo una buona base per sperimentazioni future finalizzate a una migliore comprensione del vasto mondo del quantum computing, ma anche per capire quanto il QNLP possa essere efficace nell'identificazione di requisiti di sicurezza, un compito la cui particolare importanza è stata precedentemente motivata.

1.3 Risultati Ottenuti

Per condurre la sperimentazione è stato necessario il supporto fornito da librerie e toolkit che permettono di rendere molto più accessibili e semplici gli esperimenti quantistici, che coinvolgono spesso algoritmi non banali da implementare, e simulatori che hanno aiutato a rendere più accessibili tali pratiche che sarebbero altrimenti risultate particolarmente costose da realizzare sull'*hardware ad hoc* che non è ancora ampiamente diffuso.
I modelli realizzati sono stati poi testati su un dataset relativo alla sanità, le cui frasi che lo compongono sono state rese compatibili con il processo da svolgere tramite strumenti di parsing forniti dalla libreria scelta; i risultati sono stati infine validati mediante tecniche di convalida incrociata. È risultato che l'approccio classico abbia ottenuto risultati migliori per le metriche di Recall e F1-Score rispetto al quantistico che è risultato più efficace in Precision e Accuracy.

1.4 Struttura della Tesi

Il capitolo 2 propone, dopo una necessaria introduzione all'**Ingegneria dei Requisiti**, al **Machine Learning**, al **Natural Language Processing** e alle loro varianti quantistiche, uno sguardo all'attuale **stato dell'arte** per quanto concerne lavori affini alle tematiche presentate.
Nel capitolo 3 si è descritto nel dettaglio l'**obiettivo della sperimentazione**, soffermandosi sulle **tecnologie** e il dataset utilizzati, per poi esplicitare come sia stato svolto lo studio e come si sia svolta la fase di **validazione**.
Nel capitolo 4 ci si è concentrati nel commentare i **risultati** emersi dalla validazione delle pipeline implementate, mostrando **differenze** tra i due approcci e le **problematiche** che si sono palesate.
Nel capitolo 5 si sono infine tratte le **conclusioni**, con un *recap* sui risultati ottenuti e su come questo studio possa contribuire ad eventuali **sviluppi futuri**.

CAPITOLO 2

Background e Stato dell'Arte

2.1 Background

2.1.1 Ingegneria dei Requisiti

L'**ingegneria dei requisiti** [4] è una fase fondamentale nello sviluppo del software, che si basa su raccolta, analisi, documentazione e gestione di un sistema in maniera sistematica e ben strutturata.
I requisiti software possono essere suddivisi principalmente in due categorie:

- **Requisiti funzionali:** definiscono le funzionalità specifiche che il sistema deve eseguire e le azioni da intraprendere in risposta a input dati dall'utente o scatenati da altre componenti del sistema;
- **Requisiti non funzionali**: definiscono i vincoli del sistema, influenzandone la qualità complessiva senza specificarne le funzionalità.

La vastità dei diversi requisiti non funzionali (da qui *NFR*) che può comprendere un sistema software, oltre all'esigenza di rispettarli pienamente al fine di evitare errori - potenzialmente anche fatali - in un sistema, ha portato alla necessità di una loro

categorizzazione il più precisa possibile, attualmente realizzata mediante il modello *FURPS* [Grady, 1992] che fornisce le seguenti categorie di requisiti non funzionali:

- **Usability**: la facilità con cui l'utente può operare e interagire col sistema.
- **Reliability**: la capacità di un sistema di eseguire le sue funzioni in un determinato periodo di tempo. Questa categoria include, a sua volta, i requisiti di **robustezza** e **sicurezza**.
- **Performance**: attributi quantificabili del sistema, ad esempio il *throughput* o il tempo di risposta.
- **Supportability**: la possibilità di mantenere il sistema aggiornato e supportato anche dopo il rilascio.

Il modello appena descritto è stato poi espanso nel modello *FURPS+* [Jacobson et al., 1999], che etichetta, tra gli altri, anche requisiti legati agli aspetti di implementazione, di interfaccia e legali del sistema.

Requisiti di Sicurezza

È doveroso soffermarsi sui **requisiti di sicurezza** [7], che consistono in un vasto numero di mansioni volte a garantire valide condizioni di sicurezza per il sistema e per i suoi utenti, ad esempio:

- assicurare che gli utenti siano **identificati** in maniera opportuna;
- assicurare che gli utenti abbiano **accesso** solo ai dati per cui è loro concesso;
- rilevare **intrusioni esterne non autorizzate**;
- preservare il sistema da **applicazioni malevole**, come *virus* informatici;
- garantire che i dati non vengano **corrotti**, **sottratti**, **clonati** o **smarriti**, proteggendoli da danni accidentali e non;
- fornire copie di **backup** per rendere possibile il ripristino del sistema in caso di danni.

2.1.2 Natural Language Processing

Il Natural Language Processing (*NLP*) [8] è un ramo dell'Intelligenza Artificiale che mira a permettere ai computer di comprendere affermazioni, parole, domande e concetti espressi in lingua umana [9].
L'interfacciamento della lingua umana con il computer si divide sostanzialmente in due fasi, ossia il *Natural Language Understanding*, che consiste nella comprensione, da parte della macchina, dei concetti espressi da un umano, e la *Natural Language Generation*, che prevede la generazione di testo comprensibile, chiaro ed utile.
Il *preprocessing* è dunque la fase iniziale del processo di NLP, e ha lo scopo di preparare il testo per l'analisi. Questa fase è fondamentale perché può migliorare significativamente l'accuratezza e l'efficienza dei metodi di analisi, e si divide nelle seguenti pratiche:

- **Tokenizzazione** - si tratta dell'attività di suddivisione del testo in unità di base, come parole o frasi.
- **Identificazione delle stopwords** - le stopwords sono le parole che non contengono informazioni rilevanti, come articoli, preposizioni e congiunzioni. La loro rimozione può migliorare particolarmente l'accuratezza dei metodi di analisi, poiché queste parole spesso non sono significative per il contesto.
- **Stemming** - lo stemming è l'attività di riduzione delle parole alla loro radice.
- **Normalizzazione** - la normalizzazione è l'attività di trasformazione delle parole in un formato standard, ad esempio correggendo errori ortografici o cambiando tutte le lettere in minuscole al fine di facilitare la comprensione.

Recentemente, l'NLP ha mostrato come non mai le sue capacità nel rappresentare il linguaggio umano e analizzarlo da un punto di vista computazionale: l'esempio più palese si è visto, negli ultimi tempi, per permettere agli utenti comuni di interfacciarsi con agenti conversazionali e chatbot in maniera agevole [10].
Il connubio tra Natural Language Processing e Quantum Computing ha inoltre mostrato un potenziale davvero interessante, promettendo di essere sempre più presente nel futuro: l'idea alla base è che il modo migliore per permettere a una

macchina di unire, comprendere e interpretare strutture grammaticali sia fornito proprio dalle metodologie quantistiche [6], sulle quali ci soffermeremo in seguito.

2.1.3 Machine Learning

Il **machine learning** [11], o apprendimento automatico, è una branca dell'intelligenza artificiale che consente ai computer di imparare a svolgere compiti senza essere esplicitamente programmati. I computer possono imparare dalle esperienze pregresse e migliorare nel tempo, senza bisogno di intervento umano.

L'idea alla base del machine learning è semplice: se forniamo ai computer un set di dati sufficiente, essi possono imparare a identificare schemi, pattern e relazioni nei dati; quanto appreso potrà poi fungere da base nel prendere decisioni o eseguire azioni.

Tra le forme del machine learning si annoverano:

- **Apprendimento supervisionato** - si basa su dati etichettati, generalmente forniti dall'umano.
  Fanno parte di questa categoria la **classificazione** e la **regressione**.

- **Apprendimento non supervisionato** - questo tipo di apprendimento non si basa su dati etichettati, bensì il computer viene addestrato a identificare pattern tra i dati in input.
  Un esempio di questo tipo di apprendimento è il **clustering**.

- **Apprendimento per rinforzo** - questo tipo di machine learning viene utilizzato quando si desidera che il computer impari a prendere decisioni in un ambiente dinamico, addestrandolo mediante un sistema di *rewards* e *penalties*, portandolo a esplorare l'ambiente e a imparare dalle proprie azioni.

Classificazione

La classificazione in machine learning è un'attività di apprendimento supervisionato utile a categorizzare i dati in classi distinte, predicendo le etichette per le nuove istanze di input sulla base delle precedenti osservazioni.
Esistono tre tipi principali di classificazione [12]:

- **Classificazione binaria**: La classificazione binaria è un tipo di classificazione in cui i dati vengono suddivisi in due classi. Ad esempio, un algoritmo di classificazione binaria potrebbe essere utilizzato per identificare se un'immagine contiene una persona o meno, o se un'e-mail è spam o non spam.

- **Classificazione multiclasse**: La classificazione multiclasse è un tipo di classificazione in cui i dati vengono suddivisi in più di due classi. Ad esempio, un algoritmo di classificazione multiclasse potrebbe essere utilizzato per identificare il tipo di fiore in un'immagine, o determinare la razza di un gatto da una sua fotografia.

- **Classificazione multilabel**: La classificazione multilabel è un tipo di classificazione in cui un'istanza di dati può appartenere a più di una classe. Ad esempio, un algoritmo di classificazione multietichetta potrebbe essere utilizzato per identificare i tag rilevanti per un articolo di notizie, o le malattie che potrebbero affliggere un paziente.

Alcuni algoritmi di machine learning comunemente utilizzati per la classificazione includono:

- Support Vector Machines (SVM);
- K-Nearest Neighbors (KNN);
- Alberi Decisionali;
- Random Forest;
- Naive Bayes

2.1.4 Quantum Computing

Negli ultimi anni la computazione quantistica [13] è stata osservata con grande interesse dagli addetti al settore, e sembra avere tutto il potenziale per superare i limiti posti dalla computazione classica, rivoluzionando il mondo della programmazione e non solo; buona parte della comunità scientifica ritiene che il ventunesimo secolo verrà ricordato come l'**era del quantum** [14].

Con *quantum supremacy* si intende proprio la capacità della computazione quantistica nel risolvere tutti i problemi già risolvibili mediante la computazione tradizionale, ma in tempo significativamente minore [15].
La differenza sostanziale tra la computazione classica e quella quantistica si trova già nell'unità alla base della computazione quantistica, ossia i *qubit*, rappresentati da stati quantistici di particelle elementari, come fotoni, elettroni o neutroni. Questi stati possono essere sovrapposti, ovvero possono essere in più stati contemporaneamente; tale proprietà, nota come *superposizione*, è alla base di molti degli aspetti innovativi della computazione quantistica, che offre diversi vantaggi rispetto alla computazione classica, tra cui:

- **Velocità**: i computer quantistici possono eseguire calcoli in modo esponenzialmente più veloce dei computer classici.
- **Precisione**: potendo sfruttare le proprietà della fisica quantistica, i computer quantistici possono eseguire calcoli con una precisione molto maggiore dei computer classici.
- **Potenza di calcolo**: i computer quantistici possono eseguire calcoli che sono impossibili o molto costosi con i computer classici. Possono, ad esempio, essere utilizzati per risolvere problemi di ottimizzazione, simulazione e crittografia che sono attualmente fuori dalla portata dei computer classici.

Apprendimento Quantistico

Il Quantum Machine Learning (**QML**) è un campo emergente che sta mettendo insieme la potenza della computazione quantistica con le tecniche di Machine Learning tradizionali al fine di risolvere problemi del mondo reale in maniera molto più efficiente [16], sfruttando l'unità del *qbit* e i *Parametrized Quantum Circuits* [17], parallelo delle reti neurali nella computazione classica.

2.2 Stato dell'Arte

Tradizionalmente, l'identificazione e la classificazione di requisiti non funzionali è un lavoro svolto a mano da analisti; è chiaro dunque quanto questo processo possa

talvolta risultare laborioso e soggetto a errori. Gli analisti di requisiti potrebbero infatti non essere in grado di identificare tutti gli NFR rilevanti, o magari potrebbero classificarli in modo errato, portando a errori anche gravi nello sviluppo di un progetto.
Se è vero che il Machine Learning possieda il potenziale per migliorare l'accuratezza e l'efficienza dell'identificazione e la classificazione di requisiti non funzionali, è chiaro che la computazione quantistica potrebbe ulteriormente migliorare tali task, consentendo di identificare e classificare gli NFR in modo più rapido e accurato rispetto ai modelli di machine learning classici.

2.2.1 Identificazione di requisiti non funzionali

Quba *et al.* (2021) [18] hanno effettuato uno studio sul dataset *PROMISEexp* sfruttando i modelli **K-NearestNeighbors** (KNN) [19] e **Support Vector Machines** (SVM) [20] sfruttando la tecnica del **Bag of Words** [21], che viene utilizzata per rappresentare il testo come un insieme di parole, facendo sì che ogni parola del testo venga considerata come un'unità indipendente; ne è emerso che l'utilizzo di BoW con algoritmi SVM restituisca le migliori misure di performance per la classificazione dei requisiti rispetto al KNN con BoW.

Identificazione di requisiti di sicurezza

Riaz *et al.* (2014) [22] hanno sviluppato un tool, Security Discoverer (SD) per l'identificazione dei requisiti di sicurezza, prendendo in input dei requisiti formulati in linguaggio naturale e un classificatore KNN tenendo a mente il dominio del problema, estraendo da ognuna di esse le categorie di sicurezza, restituendo buone metriche valutative.
Wang *et al.* (2019) [23] hanno invece constatato come la maggior parte dei metodi d'identificazione di requisiti di sicurezza presenti in letteratura si basino su informazioni già presenti in file di testo, finendo dunque per non essere compatibili con un'idea d'identificazione di requisiti di sicurezza espressi *just-in-time*, forma in cui appaiono frequentemente tali requisiti per progetti *open source*, siccome vengono espressi sottoforma di commenti in forum e siti *ad hoc*; è stato proposto allora un mo-

dello basato su un classificatore lineare, ottenendo ottimi risultati in tutte le metriche analizzate su dataset di progetti *open source* come *Apache*, *Drools* e *GeoServer*.

2.2.2 Approccio quantistico

Per quanto riguarda studi di classificazione su pipeline quantistica, Terashi *et al.* (2021) [24] confrontano la performance di tecniche standard di classificazione basata su alberi decisionali con degli algoritmi quantistici applicati su eventi di fisica delle particelle, mostrando dei risultati abbastanza simili tra i due tipi di computazione, ricavandone dunque che in futuro sarà sicuramente possibile un maggior impiego del quantum per questo tipo di task, soprattutto avendo maggiore cura degli errori di misurazione.
In letteratura, il numero di studi che coinvolgono applicazioni di Quantum Natural Language Processing è continuamente crescente.
Ganguly *et al.* (2022) [25] hanno effettuato delle sperimentazioni di NLP su pipeline quantistica con il toolkit **lambeq**, per la pratica della Sentiment Analysis su un dataset non eccessivamente vasto riguardo generi letterari, ottenendo ottimi risultati e soprattutto uno studio che possa fruire da buona base per sviluppi futuri.
Il toolkit lambeq risulta attualmente il più diffuso negli approcci quantistici al NLP; tra gli altri, Vogel *et al.* (2022) [26] ne propongono un impiego per l'identificazione di possibili *fake news* contenute nei post pubblicati da utenti del social network *X*, ottenendo un buon F1-score e dei risultati davvero promettenti.

CAPITOLO 3

Metodologia e sviluppo

3.1 Obiettivi della sperimentazione

Il fine di questo lavoro di tesi è lo studio del già citato toolkit **lambeq** su pipeline **classica** e **quantistica**, sperimentandone metodi e funzionalità su un dataset di frasi dalle quali estrarre dei requisiti non funzionali di sicurezza, praticando poi delle tecniche di classificazione multiclasse per assegnare ogni requisito alla rispettiva sotto-categoria.
La sperimentazione seguirà e soddisferà dunque le seguenti domande di ricerca:

RQ$_1$. *In che misura possiamo identificare NFRs di sicurezza tramite la pipeline classica?*

Per rispondere a questa domanda, verrà sfruttata la libreria *lambeq* per la classificazione in maniera tradizionale, per poi validare i risultati e trarre conclusioni basandosi su metriche di valutazione, risorse e tempistiche impiegate.

RQ$_2$. *In che misura possiamo identificare NFRs di sicurezza tramite la pipeline quantistica?*

La metodologia da seguire per rispondere a questa domanda sarà la stessa che avrà aiutato a rispondere alla precedente, ma sfruttando, questa volta, le potenzialità

di un approccio quantistico.

RQ$_3$. *In che misura si differenziano i risultati delle identificazioni di NFRs di sicurezza tra le due pipeline?*

A questa domanda saremo in grado di rispondere proprio confrontando i risultati ottenuti alla fine delle due fasi, tentando di individuare le differenze sostanziali tra l'uno e l'altro approccio in termini di metriche di valutazione e risorse impiegate.

3.1.1 Tecnologie utilizzate

Per condurre una buona fase di sperimentazione sul dataset scelto, è stato necessario uno studio preliminare per entrare in confidenza con il toolkit *lambeq*, una libreria open-source e modulare per l'implementazione di pipeline quantistiche [27]: in questo stadio del lavoro, sono stati quindi condotti dei semplici test delle funzioni principali fornite dalla libreria seguendo i *tutorial* proposti dalla documentazione [28].
In questa fase è stato più che sufficiente impiegare una macchina con sistema operativo *Windows 10*, processore *Intel* i7-97500H, scheda grafica *NVidia GeForce* RTX-2060 e 16GB di memoria RAM DDR4.
Per le successive fasi, sicuramente più esigenti in termini di prestazioni, l'hardware non si è dimostrato in grado di sostenere un tale carico di lavoro, soprattutto dovendo, negli stadi più avanzati dello studio, emulare una macchina quantistica; la simulazione delle pipeline è stata quindi eseguita su macchine fornite dalla piattaforma **Google Colab Pro** [29], sebbene anche qui siano state riscontrate delle limitazioni che hanno portato a dover praticare degli accorgimenti.

3.2 Analisi del Dataset e Preparazione dei Dati

3.2.1 Analisi del Dataset

Il dataset impiegato per la sperimentazione riguarda il settore della sanità ed è un agglomerato di sei file che si compongono di frasi espresse in linguaggio naturale inglese estratte dai ricercatori, che hanno poi selezionato quelle contenenti requisiti di sicurezza, categorizzandoli nelle seguenti categorie:

1. **Accountability** - la capacità di un sistema di identificare un singolo utente e il suo comportamento;
2. **Access Control Identity** - la disciplina dell'assegnare alle entità le rispettive risorse senza alcun impedimento;
3. **Availability** - requisito riguardante la disponibilità di un servizio del sistema;
4. **Confidentiality** - protezione di informazioni sensibili da parti indesiderate;
5. **Identity** - la possibilità di identificare "chi fa cosa" all'interno del sistema
6. **Integrity** - la protezione dei dati e delle informazioni da possibili modifiche indesiderate;
7. **Operational** - caratteristiche funzionali del sistema;
8. **Privacy** - requisito riguardante la riservatezza delle informazioni private dei soggetti coinvolti.

Ogni riga del dataset è composta dalle seguenti colonne:

- **Sentence** - la frase da cui estrarre il requisito;
- **Entities** - la tokenizzazione della rispettiva frase, dunque le singole parole che la compongono;
- **Dependencies** - le relazioni tra i token del requisito;
- **Parts of Speech** - cosa costituisce ogni token dal punto di vista sintattico;

- **File** - il file da cui proviene la frase in questione;
- **Categories** - le categorie a cui sono assegnati i requisiti;
- **Security Words** - le parole chiave di una frase che si riferiscano a requisiti di sicurezza;
- **Security** - colonna il cui valore sarà 1 in caso di requisiti di sicurezza, 0 altrimenti.

Delle 5976 righe del dataset, 4028 contengono dei requisiti di sicurezza e sono dunque interessanti per la presente sperimentazione, mentre le altre non saranno coinvolte.

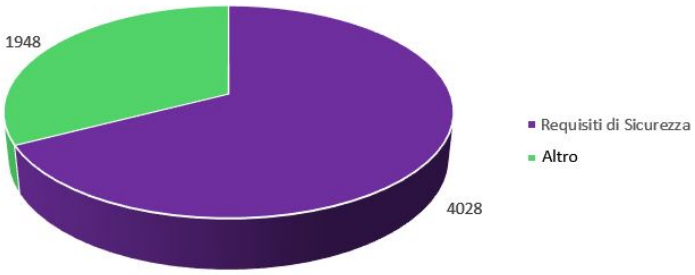

Figura 3.1: Proporzione tra frasi riguardanti requisiti di sicurezza e altre.

3.2.2 Setup e Preparazione dei Dati

Il dataset oggetto dello studio era colmo di informazioni non essenziali allo svolgimento dello stesso, dunque sono stati praticati degli accorgimenti per condurre l'insieme di dati a una forma più essenziale e vicina a quella utile ai nostri scopi grazie all'utilizzo della libreria **pandas** [30] di Python.
Sono state rimosse le colonne non necessarie, mantenendo solo frasi e categorie, utilizzando un carattere "**.**" come separatore.
Non tutte le righe, tuttavia, sono risultate idonee e compatibili con i tool utilizzati nelle fase successive: le istanze con meno di due parole, con segni di punteggiatura e caratteri speciali irrilevanti al fine di una basilare comprensione del testo sono stati rimossi, mentre è risultata molto utile anche la riformulazione e semplificazione di alcuni costrutti (ad esempio, "*e.g.*" o "*i.e.*" sono stati rimpiazzati con "*for example*" o

"*for instance*").
È necessario soffermarsi anche su alcune problematiche sorte durante la stesura del codice: innanzitutto, allo stato attuale *lambeq* non supporta ancora pienamente la classificazione multiclasse; il tentativo è stato dunque realizzato creando un classificatore binario per ognuna delle categorie di NFR elencate in precedenza, pertanto si è rivelato necessario adattare anche i dati a questo tipo di approccio.
Per ogni classe, quindi, si avrà un file di testo formattato come nell'esempio che segue, dove in ogni riga è presente la frase coinvolta e un valore numerico pari a 1 se essa è un requisito della categoria che si sta analizzando, 0 altrimenti.
Un'altra problematica è derivata dalla grande quantità dei dati, che ha portato più volte all'esaurimento della memoria RAM a disposizione in fase di addestramento quantistico: è stato quindi necessario ridurre vistosamente la mole di dati fornita al modello, così da completare la sperimentazione con l'hardware definito.

```
Receives information about the patient's preferences and displays information according to the patient's preferences.   1
Allows nurses to view patients care directives available in the EHR upon request.       0
Allows nurses to view the patients preferences available in the EHR upon request such as those related to diversity.    0
```

Figura 3.2: Esempio di formattazione del dataset per la classe *privacy*.

3.3 Sviluppo della pipeline

Una volta terminata la fase di preparazione dei dati, si è finalmente iniziata la sperimentazione con l'ausilio del toolkit *lambeq*.
Sostanzialmente, *lambeq* permette di trasformare una qualsiasi frase in un circuito quantistico passando per i seguenti step:

Figura 3.3: *Pipeline* generale per la trasformazione di una frase in circuito quantistico

In breve, a partire dalla frase si ottiene un **albero sintattico**, ossia una rappresentazione gerarchica di una frase che riflette le relazioni sintattiche tra le parole che la

compongono. Quest'albero verrà poi convertito in un **diagramma di stringa**, ovvero una forma di rete di tensori con delle proprietà aggiuntive, ad esempio viene mantenuto e rispettato l'ordine delle parole. Questi diagrammi possono essere semplificati o trasformati con delle **regole di *rewriting***: si possono rimuovere parti della frase ridondanti o poco utili per il compito in questione. A questo punto, il diagramma viene convertito finalmente in una rete di tensori o un circuito quantistico sulla base di uno **schema di parametrizzazione** e della scelta degli ***ansätze***, che determinano diverse proprietà del circuito, come il numero di qubits da assegnare ogni collegamento del diagramma di stringa. Finalmente, l'output sarà pronto per la fase di **training**.
Sebbene questo procedimento possa sembrare davvero complesso, la libreria *lambeq* ci viene incontro facilitando di molto il compito.

3.3.1 Tokenization

La **tokenizzazione** consiste nel dividere una data frase in unità più piccole, appunto i *token*.
Per questa fase, *lambeq* adopera di default la classe **SpacyTokenizer**, che si basa sul pacchetto **SpaCy** [31], molto popolare nel campo dell'NLP.

```
from lambeq import SpacyTokeniser

tokeniser = SpacyTokeniser()
sentence = "This sentence isn't worth £100 (or is it?)."
tokens = tokeniser.tokenise_sentence(sentence)
tokens
```

Figura 3.4: Esempio di codice che esegue la tokenizzazione di una frase.

In output, come previsto, si avrà la separazione degli elementi della frase:

```
['This',
 'sentence',
 'is',
 "n't",
 'worth',
 '£',
 '100',
 '(',
 'or',
 'is',
 'it',
 '?',
 ')',
 '.']
```

Figura 3.5: Output dell'esempio di codice in **Figura 3.4**.

Le frasi, una volta suddivise in token, sono pronte per la conversione in diagrammi di stringhe.

3.3.2 Parsing in diagrammi di stringhe

Diagrammi di stringhe

Al fine di semplificare la comprensione del linguaggio naturale su macchine quantistiche, *lambeq* rappresenta le frasi come **diagrammi di stringa**, che esprimono la computazione in una **categoria monoidale** [32], un'astrazione particolarmente vicina a come un computer quantistico processa i dati.
I diagrammi di stringa sono caratterizzate dalla presenza dei **tipi**, che mostrano come le parole in una frase interagiscono secondo delle regole grammaticali.

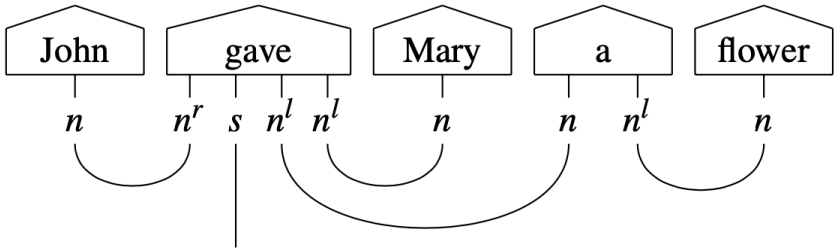

Figura 3.6: Esempio di un diagramma di stringa

Nell'esempio, si noti che ogni collegamento all'interno della frase è etichettato con una tra le seguenti lettere:

- ***n*** - *noun*, ossia un sostantivo nella frase;
- ***s*** - *sentence*, indica che, fino a quel punto, la frase può considerarsi effettivamente completa;
- ***nr*** - indica che ci si aspetta che alla sinistra di questo elemento ci sia un sostantivo;
- ***nl*** - indica che ci si aspetta che alla destra di questo elemento ci sia un sostantivo;

Parser in *lambeq*

lambeq fornisce, oltre al già citato modello basato su Bag-of-Words e ai *tree readers*, due tipi diversi di *parser*, ossia **syntax-based** e **word-sequence** model.
I **syntax-based model** si concentrano sulle relazioni tra le parole nella frase; in *lambeq*, il principale modello syntax-based è il **BobcatParser** che, sebbene sia stato provato per questa sperimentazione, è risultato troppo pesante e oneroso da un punto di vista prestazionale, in quanto l'ambiguità derivante dalla naturalezza delle frasi del dataset ha portato a generare diagrammi complessi e talvolta inutilizzabili per l'addestramento del modello.
Proprio per ovviare a questo problema si è optato per i **word-sequence models**, che trattano le frasi come una sequenza di token, ignorandone le relazioni sintattiche; *lambeq* propone, tra gli altri, lo **StairsReader** e il **CupsReader**: entrambi hanno prodotto output semplici, sebbene con strutture logiche dei diagrammi differenti. Tra i due, si è preferito utilizzare i CupsReader.

Da questo punto della sperimentazione in poi, gli approcci per la pipeline classica e quello per la pipeline quantistica risultano radicalmente diversi sia in termini di scelte implementative che di codice.

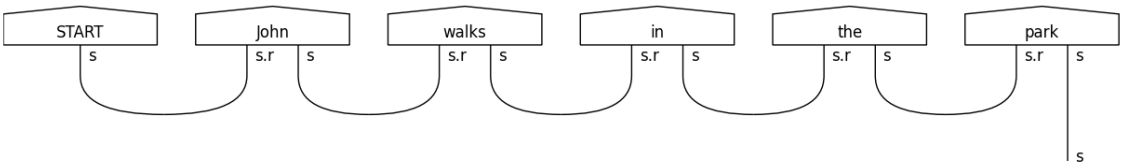

Figura 3.7: Esempio di un diagramma di stringa generato mediante CupsReader

```
from lambeq import cups_reader

# Create string diagrams based on cups reader
cups_diagram = cups_reader.sentence2diagram(sentence)

pregroups.draw(cups_diagram, figsize=(12,3), fontsize=12)
```

Figura 3.8: Codice per la generazione di un diagramma di stringa mediante CupsReader

3.3.3 Approccio tradizionale

Parsing in circuiti

Al momento, le frasi sono ancora rappresentate come diagrammi di stringhe; nella pipeline classica, si desidera interpretare la frase come una **rete di tensori** [33], che fornisce un'efficiente rappresentazione grafica e più intuitiva del circuito.
Come spiegato precedentemente, la trasformazione da diagramma di stringa a rete di tensori avviene tramite l'applicazione di un **ansatz**. Tra i vari tipi di ansatz esistenti, si è scelto di condurre la sperimentazione classica con uno **SpiderAnsatz**, che restituisce dei circuiti strutturalmente più semplici e lineari rispetto alle altre varianti. Questo tipo di ansatz divide i tensori più di due dimensioni in sequenze di tensori bidimensionali, in altre parole, **matrici**.

Nell'esempio, tratto dalla documentazione di *lambeq*, si noti che la preposizione "*in*" è stata ora divisa in una matrice prodotto di quattro matrici interconnesse, il ché aiuta ad alleggerire notevolmente il carico computazionale.

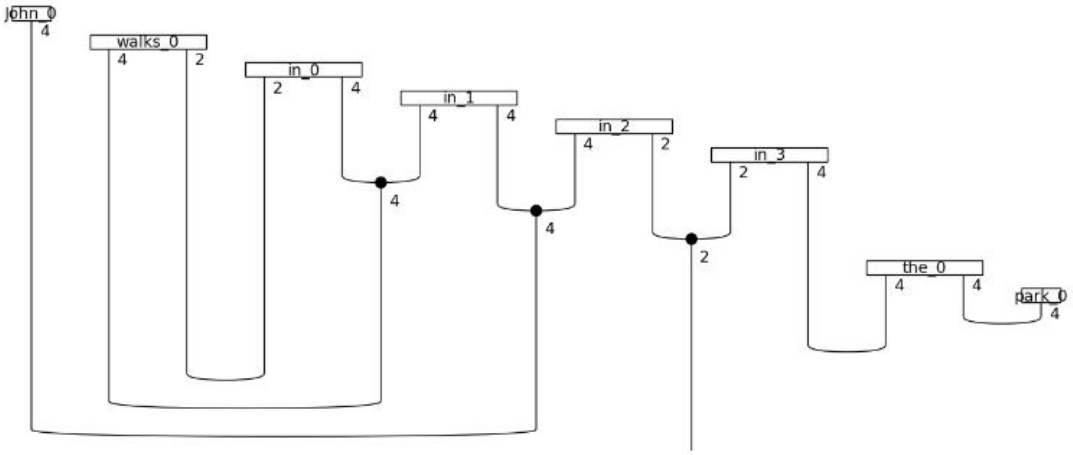

Figura 3.9: Esempio di una frase trasformata mediante SpiderAnsatz

```
from lambeq import SpiderAnsatz
from discopy.tensor import Dim

spider_ansatz = SpiderAnsatz({N: Dim(4), S: Dim(2)})
spider_diagram = spider_ansatz(diagram)
spider_diagram.draw(figsize=(13,6), fontsize=13)
```

Figura 3.10: Codice per la trasformazione in rete di tensori tramite SpiderAnsatz

Training

In quest'ultima fase ci si è concentrati sulla scelta e sull'impostazione del modello allenato per la classificazione in questione. *lambeq* fornisce diversi modelli, tra i quali si ricordano:

- **PytorchModel**: Questo modello risulta essere il più adatto alla sperimentazione classica, proprio perché i diagrammi di stringhe vengono trattati come reti di tensori. Si tratta del modello scelto per la presente sperimentazione.
- **NumpyModel**: Questo modello sfrutta i simulatori di matrici di densità di *DisCoPy*, che converte i circuiti quantistici in reti di tensori. Si tratta dunque di un altro modello particolarmente versatile e utile per sperimentazioni semplici. Come vedremo più avanti, si tratta del modello scelto per la fase di sperimentazione quantistica di questo studio.
- **TketModel**: Questo modello sfrutta le funzionalità del pacchetto *pytket*, quindi è particolarmente adatto per le sperimentazioni quantistiche, e lo si utilizza in

combinazione con il *QuantumTrainer*. Lo si predilige quando si ha a disposizione un hardware quantistico a tutti gli effetti, dunque non lo si è adoperato avendo svolto la simulazione della pipeline quantistica su una macchina virtuale.

Come anticipato, la scelta del modello impiegato è ricaduta proprio sul **PytorchModel**.

```
from lambeq import PytorchModel

all_circuits = train_circuits + val_circuits
model = PytorchModel.from_diagrams(all_circuits)
```

Figura 3.11: Implementazione del PytorchModel

Il modello, al quale sono stati forniti i circuiti creati nella fase precedente grazie allo SpiderAnsatz, è stato poi allenato mediante il **PytorchTrainer**, fornito anch'esso da *lambeq*, che permette di personalizzare i parametri del trainer in maniera intuitiva, essenziale ma comunque dettagliata.

```
trainer = PytorchTrainer(
model=model,
loss_function = torch.nn.BCEWithLogitsLoss(),
optimizer=torch.optim.AdamW,
learning_rate=LEARNING_RATE,
epochs=EPOCHS,
evaluate_functions=eval_metrics,
evaluate_on_train=True,
verbose='text',
seed=SEED)
```

Figura 3.12: Creazione e personalizzazione del PytorchTrainer

Come mostra lo screen, il modello è stato addestrato con un trainer inizializzato coi seguenti parametri:

- **model**: il modello da addestrare, in questo caso, quello creato precedentemente.

- **loss_function**: una funzione che stima quanto la predizione del modello risulta lontana dal valore effettivo, distanza che si desidera chiaramente minimizzare. La loss function scelta, ossia **BCEWithLogitsLoss**, è stata suggerita dalla documentazione di *lambeq* come la più versatile.
- **optimizer**: l'ottimizzatore scelto, in questo caso *AdamW* dal package *Torch*.
- **learning_rate**: il tasso d'apprendimento, ossia "quanto velocemente" un modello apprende. In questo caso, è stato lasciato il valore di default di *1e-3*.
- **epochs**: il numero di epoche d'addestramento, impostato a *5*, una quantità decisamente bassa ma che ha garantito di velocizzare il più possibile la computazione dei risultati, soprattutto in seguito con la pipeline quantistica.
- **evaluate_functions**: la lista delle metriche da calcolare, dunque *accuracy*, *precision*, *recall* e *F1-score*.
- **evaluate_on_train**: permette di decidere se valutare le metriche anche sul set d'addestramento.
- **verbose**: la quantità di output testuale da fornire in merito all'andamento delle operazioni, in questo caso si è scelto di avere output particolarmente dettagliati.
- **seed**: il seed scelto per inizializzare la componente pseudo-casuale.

3.3.4 Approccio quantistico

Parsing in circuiti

Anche qui, come per l'approccio classico, è stato necessario trasformare le frasi da diagrammi di stringhe a una forma con cui un processore quantistico possa lavorare al meglio, ossia dei **circuiti quantistici**; questo *parsing* è stato svolto tramite l'**IQPAnsatz**, come anticipato in precedenza.

Nel codice in figura, viene esplicata la differenza tra **noun**, indicati nel diagramma con la lettera **n**, che vengono mantenuti e rappresentati da un sistema basato su un singolo qubit, mentre le **sentence**, indicate con la **s**, vengono scartate.

```
from lambeq import AtomicType, IQPAnsatz, remove_cups

ansatz = IQPAnsatz({AtomicType.NOUN: 1, AtomicType.SENTENCE: 0},
                   n_layers=1, n_single_qubit_params=3)

train_circuits = [ansatz(remove_cups(diagram)) for diagram in train_diagrams]
val_circuits =  [ansatz(remove_cups(diagram))  for diagram in val_diagrams]

train_circuits[0].draw(figsize=(9, 10))
```

Figura 3.13: Codice per la trasformazione in circuito quantistico tramite IQPAnsatz

Viene anche utilizzata la funzione **remove_cups**, al fine di rendere il modello più efficiente andando a ridurre le **post-selezioni** possibili.
Il risultato del codice è un circuito simile a questo:

Training

Per quanto riguarda la scelta del modello, si è deciso di impiegare, come detto in precedenza, il **NumpyModel**, alla cui creazione sono stati forniti in input i circuiti appena convertiti.
Si noti come, all'istanzazione del modello, è stato scelto di porre la variabile *use_jit* a *false*. Se si fosse scelto di impostarla a *true*, il modello avrebbe utilizzato la compilazione Just-In-Time di **JAX**, particolarmente efficiente ma onerosa in termini di prestazioni.

Proprio come nella fase classica dell'addestramento, anche per il training quantistico è stato necessario istanziare un trainer, in questo caso il **QuantumTrainer** fornito da *lambeq*.

I parametri su cui è stato impostato il modello sono i seguenti:

- **model**: il modello da addestrare, ossia quello appena creato.
- **loss_function**: la loss function scelta, cioè **BinaryCrossEntropyLoss**, è stata suggerita dalla documentazione di *lambeq*.
- **epochs**: il numero di epoche d'addestramento, impostato a *5*, una quantità decisamente bassa ma che ha garantito di velocizzare il più possibile la computazione dei risultati.

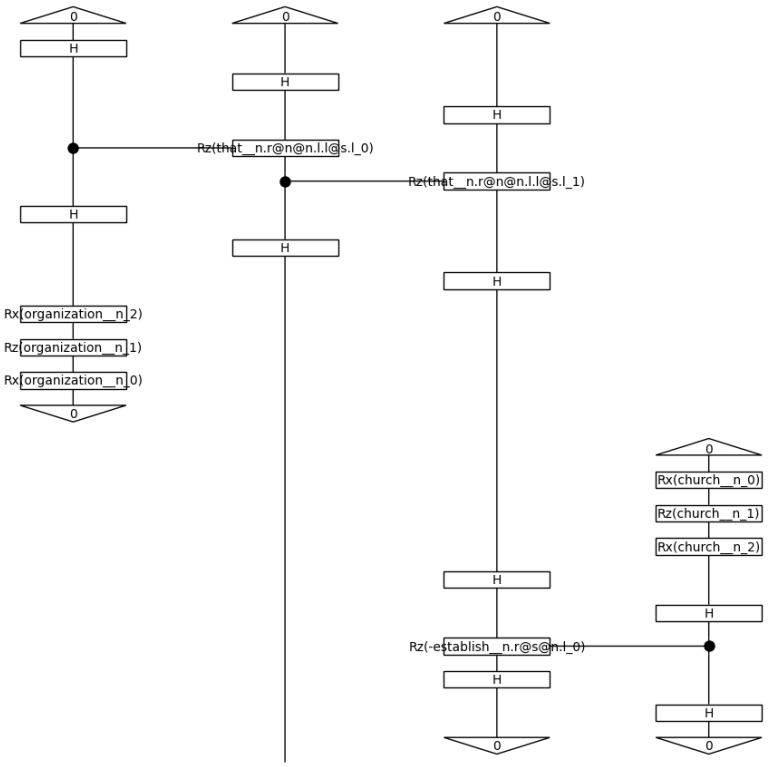

Figura 3.14: Circuito quantistico generato da una frase tramite IQPAnsatz

```
from lambeq import NumpyModel

all_circuits = train_circuits + val_circuits
model = NumpyModel.from_diagrams(all_circuits, use_jit=False)
```

Figura 3.15: Esempio di creazione del NumpyModel

- **optimizer**: l'ottimizzatore scelto, in questo caso *SPSAOptimizer* [34].
- **optim_hyperparams**: gli iperparametri in input all'ottimizzatore scelto, che influiscono sul suo comportamento.
- **evaluate_functions**: la lista delle metriche da calcolare, dunque *accuracy*, *precision*, *recall* e *F1-score*.
- **evaluate_on_train**: permette di decidere se valutare le metriche anche sul set d'addestramento.

```
trainer = QuantumTrainer(
model = model,
loss_function = BinaryCrossEntropyLoss(use_jax = True),
epochs = EPOCHS,
optimizer = SPSAOptimizer,
optim_hyperparams = {'a': 0.2, 'c': 0.06, 'A' : 0.01 * EPOCHS},
evaluate_functions = {"acc": acc, "rec": rec, "pre": prec, "f1": f1},
evaluate_on_train = True,
verbose = 'text',
seed = SEED
)
```

Figura 3.16: Esempio di creazione del NumpyModel

- **verbose**: la quantità di output testuale da fornire in merito all'andamento delle operazioni, in questo caso si è scelto di avere output particolarmente dettagliati.
- **seed**: il seed scelto per inizializzare la componente pseudo-casuale.

3.4 Validazione del modello

3.4.1 Tecniche di validazione del modello

Nel creare un modello di machine learning è fondamentale dedicare attenzione alla fase di **validazione** dello stesso, così da poter comprenderne al meglio le potenzialità e soprattutto le pecche. Per la validazione del modello creato, si è scelto di sfruttare la tecnica della **cross validation**, che consiste nel costruire il modello solo impiegando una parte dei dati a disposizione, utilizzando la parte restante per la valutazione delle predizioni: nel momento in cui il modello funziona effettivamente bene su entrambe le porzioni, allora ci si può ritenere soddisfatti.
Per entrambe le pipeline, si è scelto di sfruttare un'implementazione molto diffusa della cross-validation, vale a dire la **k-Fold validation** che, come suggerisce il nome, prevede che la validazione del modello venga ripetuta k volte, cambiando ogni volta le parti da valutare e quelle da predire, facendo sì che ogni sottoinsieme venga utilizzato per la validazione almeno una volta nelle ripetizioni del processo.

Si è scelto di impostare il valore di k pari a **5**, in altre parole di validare il modello per 5 volte; i risultati ottenuti sono stati poi elaborati graficamente con l'ausilio della

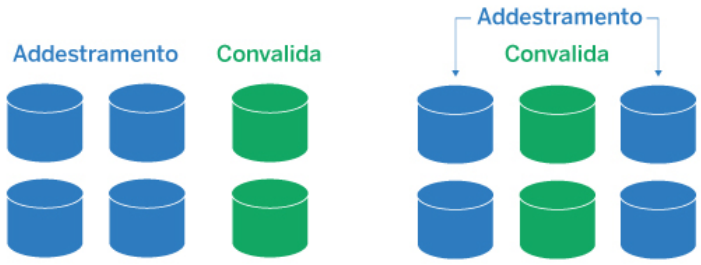

Figura 3.17: Esempio di rappresentazione grafica della k-Fold validation

libreria *MatPlotLib* [35], al fine di poter interpretare al meglio i risultati per trarne delle conclusioni adeguate.

3.4.2 Metriche di validazione del modello

Le metriche di validazione vengono utilizzate per valutare le prestazioni di un modello di machine learning su dati che non sono stati utilizzati per l'addestramento del modello, il che è fondamentale al fine di garantire che il modello sia generalizzabile per impieghi futuri e che possa funzionare bene su nuovi dati.
Si suddividono le predizioni in quattro principali classi:

- **True Positive**: classe di esempi classificati correttamente come positivi.
- **False Positive**: classe di esempi classificati erroneamente come positivi.
- **True Negative**: classe di esempi classificati correttamente come negativi.
- **False Negative**: classe di esempi classificati erroneamente come negativi.

L'accuratezza è una metrica di validazione che misura la percentuale di esempi che sono stati classificati correttamente. Può essere calcolata come segue:

$$Accuracy = \frac{(TP + TN)}{(TP + FP + TN + FN)}$$

La precisione è una metrica di validazione che misura la percentuale di soli esempi positivi che sono stati classificati correttamente. Può essere calcolata come segue:

$$Precision = \frac{(TP)}{(TP + FP)}$$

Il recall è una metrica di validazione che misura la percentuale di soli esempi positivi che sono stati identificati come tali. Può essere calcolata come segue:

$$Recall = \frac{(TP)}{(TP + FN)}$$

L'F1-score è una metrica di validazione che combina la precisione e il recall in una singola misura. Può essere calcolata come segue:

$$F1Score = 2 * \frac{(Precision * Recall)}{(Precision + Recall)}$$

CAPITOLO 4

Risultati

4.1 Confronto dei risultati ottenuti

Come anticipato, i risultati ottenuti nella sperimentazione sono stati validati tramite 5-fold cross validation per ogni classe.
Per brevità dell'elaborato finale si è deciso di non inserire in questo testo i grafici relativi all'andamento dei punteggi delle epoche di addestramento per tutte le classi: seguono dunque i risultati ottenuti per ogni classe riassunti in forma tabellare, dove per ogni fold, rappresentati dalle righe, verrà selezionato il valore più alto raggiunto in un'epoca di apprendimento in ogni metrica studiata, ricavando infine la media per ognuna di esse.
Per facilitare ulteriormente la comprensione, le tabelle riguardanti l'addestramento classico e quello quantistico saranno distinte tra loro da colori di sfondo diversi.
Nell'osservazione dei seguenti risultati, si tenga a mente che i tempi e le risorse necessarie per il completamento degli apprendimenti sono stati significativamente diversi per le pipeline classiche e quantistiche: se la pipeline *classica* ha impiegato in media circa **10 minuti** per ogni fold, non diventando mai eccessivamente impegnativa in termini di memoria RAM, non si può dire lo stesso per la pipeline *quantistica*, che ha richiesto in media **1 ora e 40 minuti** per ogni fold oltre ad impiegare più di **24GB** dei

32 forniti dalle macchine di Google Colab.
Si ricorda inoltre che anche la scelta del numero di epoche e la dimensione del dataset sono state inevitabilmente influenzate dall'alta richiesta di risorse dell'apprendimento quantistico.

Accountability

Fold	Accuracy	Precision	Recall	F1-Score
1	0.510	0.515	0.534	0.513
2	0.513	0.512	0.547	0.526
3	0.516	0.515	0.526	0.520
4	0.502	0.502	0.532	0.516
5	0.519	0.519	0.535	0.526
Media	**0.512**	**0.512**	**0.534**	**0.520**

Tabella 4.1: Apprendimento **classico** - Accountability

Fold	Accuracy	Precision	Recall	F1-Score
1	0.507	0.506	0.322	0.395
2	0.528	0.544	0.346	0.423
3	0.541	0.565	0.354	0.436
4	0.480	0.465	0.263	0.336
5	0.513	0.520	0.327	0.402
Media	**0.513**	**0.520**	**0.322**	**0.398**

Tabella 4.2: Apprendimento **quantistico** - Accountability

Per questa classe, tra le due pipeline l'**accuracy** è quasi identica, mentre la **precision** è leggermente più alta nell'approccio quantistico, a discapito di **recall** e **F1-Score** davvero bassi.

Access Control Identity

Fold	Accuracy	Precision	Recall	F1-Score
1	0.523	0.523	0.538	0.527
2	0.539	0.539	0.541	0.540
3	0.502	0.502	0.531	0.514
4	0.526	0.525	0.561	0.542
5	0.506	0.506	0.517	0.508
Media	**0.519**	**0.519**	**0.537**	**0.526**

Tabella 4.3: Apprendimento **classico** - Access Control Identity

Fold	Accuracy	Precision	Recall	F1-Score
1	0.515	0.524	0.332	0.406
2	0.507	0.512	0.298	0.373
3	0.523	0.540	0.316	0.398
4	0.534	0.556	0.334	0.417
5	0.505	0.509	0.311	0.386
Media	**0.516**	**0.528**	**0.318**	**0.395**

Tabella 4.4: Apprendimento **quantistico** - Access Control Identity

Nella classe di *Access Control Identity* l'unica metrica in cui l'apprendimento quantistico è risultato leggermente migliore è quella di **precision**, mentre l'**accuracy** è lievemente più bassa rispetto alla controparte tradizionale. **Recall** ed **F1-Score** continuano ad essere particolarmente bassi per la pipeline quantistica, mentre in quella classica raggiungono rispettivamente il **53.7%** e il **52.6%**.

Availability

Fold	Accuracy	Precision	Recall	F1-Score
1	0.500	0.500	0.517	0.508
2	0.508	0.508	0.523	0.516
3	0.508	0.508	0.534	0.521
4	0.516	0.514	0.573	0.541
5	0.508	0.509	0.496	0.496
Media	**0.508**	**0.507**	**0.528**	**0.516**

Tabella 4.5: Apprendimento **classico** - Availability

Fold	Accuracy	Precision	Recall	F1-Score
1	0.550	0.577	0.372	0.453
2	0.521	0.536	0.327	0.404
3	0.517	0.527	0.327	0.402
4	0.517	0.526	0.337	0.411
5	0.511	0.520	0.289	0.367
Media	**0.523**	**0.537**	**0.330**	**0.407**

Tabella 4.6: Apprendimento **quantistico** - Availability

Per quanto riguarda l'*availability*, risultano particolarmente alte **accuracy** e **precision** per la pipeline quantistica con punteggi pari a **52.3%** e **53.7%**, con un divario abbastanza ampio rispetto a quella classica, dove raggiungono **50.8%** e **50.7%**. Si riconferma l'ampio *gap* tra le due pipeline per le metriche di **recall** e **F1-Score**.

Confidentiality

Fold	Accuracy	Precision	Recall	F1-Score
1	0.513	0.512	0.517	0.525
2	0.510	0.510	0.523	0.514
3	0.526	0.525	0.534	0.535
4	0.490	0.490	0.573	0.499
5	0.495	0.495	0.496	0.517
Media	**0.506**	**0.506**	**0.530**	**0.518**

Tabella 4.7: Apprendimento **classico** - Confidentiality

Fold	Accuracy	Precision	Recall	F1-Score
1	0.511	0.520	0.306	0.382
2	0.534	0.556	0.345	0.426
3	0.517	0.527	0.324	0.401
4	0.521	0.537	0.315	0.397
5	0.511	0.518	0.321	0.396
Media	**0.518**	**0.531**	**0.322**	**0.380**

Tabella 4.8: Apprendimento **quantistico** - Confidentiality

In questa classe, spicca sicuramente la **precision** della pipeline quantistica, che raggiunge il **53.1%** contro il **50.6%** di quella classica. Leggermente migliore anche l'**accuracy**, mentre restano particolarmente basse **recall** e **F1-Score**.

Identity

Fold	Accuracy	Precision	Recall	F1-Score
1	0.497	0.497	0.523	0.510
2	0.521	0.520	0.542	0.530
3	0.513	0.512	0.564	0.537
4	0.517	0.517	0.527	0.522
5	0.518	0.519	0.508	0.513
Media	**0.513**	**0.513**	**0.532**	**0.522**

Tabella 4.9: Apprendimento **classico** - Identity

Fold	Accuracy	Precision	Recall	F1-Score
1	0.522	0.520	0.306	0.405
2	0.492	0.556	0.345	0.353
3	0.542	0.527	0.324	0.422
4	0.552	0.537	0.315	0.448
5	0.520	0.518	0.321	0.400
Media	**0.525**	**0.542**	**0.363**	**0.405**

Tabella 4.10: Apprendimento **quantistico** - Identity

Anche nella classe dei requisiti di *identity* la **precision** è più alta con approccio quantistico, toccando il **54.2%** rispetto al **51.3%** dell'approccio classico. Anche l'accuracy è lievemente più alta.

Integrity

Fold	Accuracy	Precision	Recall	F1-Score
1	0.491	0.491	0.505	0.498
2	0.480	0.480	0.489	0.485
3	0.530	0.528	0.567	0.546
4	0.521	0.520	0.550	0.532
5	0.515	0.515	0.532	0.519
Media	**0.507**	**0.507**	**0.528**	**0.516**

Tabella 4.11: Apprendimento **classico** - Integrity

Fold	Accuracy	Precision	Recall	F1-Score
1	0.489	0.483	0.301	0.371
2	0.525	0.539	0.342	0.418
3	0.518	0.530	0.324	0.402
4	0.504	0.506	0.303	0.377
5	0.517	0.552	0.334	0.409
Media	**0.510**	**0.522**	**0.320**	**0.395**

Tabella 4.12: Apprendimento **quantistico** - Integrity

Per questa classe si assottiglia il divario per l'**accuracy** e la **precision** delle due pipeline con punteggi abbastanza simili, ma non si può dire lo stesso per **recall** e **F1-Score**, che si dimostrano davvero basse per l'approccio quantistico, non superando il **32**% e il **39.5%** rispettivamente.

Operational

Fold	Accuracy	Precision	Recall	F1-Score
1	0.509	0.509	0.531	0.520
2	0.511	0.511	0.507	0.506
3	0.519	0.519	0.521	0.520
4	0.519	0.519	0.542	0.530
5	0.521	0.520	0.547	0.533
Media	**0.515**	**0.515**	**0.529**	**0.521**

Tabella 4.13: Apprendimento **classico** - Operational

Fold	Accuracy	Precision	Recall	F1-Score
1	0.504	0.508	0.303	0.379
2	0.515	0.525	0.317	0.396
3	0.520	0.532	0.330	0.408
4	0.486	0.479	0.314	0.380
5	0.516	0.528	0.308	0.388
Media	**0.508**	**0.522**	**0.314**	**0.390**

Tabella 4.14: Apprendimento **quantistico** - Operational

Per la classe *operational*, solo la **precision** risulta leggermente migliore con l'approccio quantistico, mentre tutte le altre metriche sono più alte con quello classico, con un divario sicuramente più marcato in **recall** ed **F1-Score**.

Privacy

Fold	Accuracy	Precision	Recall	F1-Score
1	0.508	0.508	0.549	0.528
2	0.519	0.519	0.531	0.525
3	0.531	0.530	0.552	0.541
4	0.504	0.504	0.513	0.505
5	0.482	0.482	0.504	0.493
Media	**0.508**	**0.508**	**0.529**	**0.518**

Tabella 4.15: Apprendimento **classico** - Privacy

Fold	Accuracy	Precision	Recall	F1-Score
1	0.505	0.509	0.312	0.387
2	0.509	0.516	0.309	0.387
3	0.531	0.554	0.343	0.409
4	0.526	0.542	0.342	0.419
5	0.504	0.506	0.324	0.395
Media	**0.515**	**0.525**	**0.326**	**0.399**

Tabella 4.16: Apprendimento **quantistico** - Privacy

In quest'ultima classe le metriche si riconfermano simili alle precedenti, con un'ampia differenza tra **recall** ed **F1-Score** per i due approcci ma non tra **accuracy** e **precision**, dove la pipeline quantistica è appena più alta rispetto alla controparte classica.

4.2 Considerazioni sui risultati ottenuti

Dalle tabelle appena mostrate risulta evidente come le performance siano tendenzialmente molto simili per tutte le sottocategorie di requisiti di sicurezza che si desidera identificare. Osservando questi risultati, siamo in grado di rispondere alle domande di ricerca poste precedentemente.

👉 **Answer to RQ_1:** I risultati migliori con la pipeline **classica** si sono avuti sulla metrica di **Recall**, con un punteggio medio superiore al 53%.

In particolare, per la pipeline classica il punteggio più alto di Recall è stato raggiunto nella categoria di Access Control Identity, con il **53.7%**.

👉 **Answer to RQ_2:** I risultati migliori con la pipeline **quantistica** si sono avuti sulla metrica di **Precision**, con un punteggio medio superiore al 52%.

Per la pipeline quantistica, la miglior prestazione media per la Precision è stata raggiunta nella sottocategoria di Identity, con il **54.2%**. Particolarmente basse invece le metriche di Recall e F1-Score, che in media non superano nemmeno il 40%, fermandosi rispettivamente a **32.6%** e **39.6%**.

👉 **Answer to RQ_3:** Dal confronto delle metriche di valutazione, si evince che l'**apprendimento quantistico sia risultato più efficace su Accuracy e Precision**, con risultati particolarmente bassi per Recall e F1-Score, dove l'apprendimento classico si è dimostrato decisamente migliore.

Come già precisato in apertura del Capitolo 4, si ribadisce che nell'analizzare dei risultati bisogna tenere conto anche delle limitazioni tecniche che hanno condizionato la strategia implementativa delle pipeline e del preprocessing dei dati. L'apprendimento quantistico ha inoltre richiesto una quantità di tempo notevolmente superiore rispetto a quello tradizionale, dunque bisogna tenere conto anche di questa differenza poiché, piuttosto che concentrarsi sul cercare l'approccio "vincente" tra i due, risulta molto più sensato comprendere quale sia più idoneo per la sperimentazione che si sta facendo e per le risorse su cui si può contare.

CAPITOLO 5

Conclusioni e sviluppi futuri

La tecnologia sta progredendo a una velocità impressionante: tecnologie di cui fino a pochi anni fa era raro perfino sentir parlare, come il Machine Learning o il Natural Language Processing, sono ora predominanti e ci condizionano nella vita di tutti i giorni. Ancora più interessante è che tali soluzioni, che ad oggi possono sembrare già particolarmente efficienti e rapide, hanno già delle possibili alternative pronte a sostituirle, come quelle basate sui sistemi quantistici.
Tali sistemi, sebbene siano davvero promettenti, sono ancora in uno stadio embrionale in cui la ricerca punta a esplorarne possibilità e limitazioni, per riuscire a sfruttare al meglio queste tecnologie.
Questo lavoro di tesi ha dunque mirato allo sviluppo di un caso d'uso per il test di un classificatore multiclasse per requisiti non funzionali di privacy, cercando di confrontare l'approccio classico e quello quantistico basandosi su un dataset relativo a servizi sanitari, le cui frasi sono state processate con strumenti di parsing forniti dal toolkit *lambeq*.
La validazione dei classificatori è stata realizzata mediante convalida incrociata a 5 fold per ogni sottocategoria di NFR di sicurezza, e da tale validazione è emerso che sebbene l'approccio quantistico abbia ottenuto punteggi migliori rispetto alla controparte classica per Accuracy e Precision, non si può dire lo stesso per Recall e

F1-Score.
Nonostante questa sperimentazione abbia sofferto di diverse limitazioni tecniche, le conclusioni che si sono tratte possono comunque fungere da solida base per ricerche future in questa direzione.

Sviluppi Futuri

Alcune possibili migliorie al lavoro svolto potrebbero essere le seguenti:

- Eseguire le pipeline su hardware migliore, così da non soffrire di limitazioni tecniche e poter avere più epoche e su un set di dati più ampio.
- In merito al dataset, si potrebbe pensare di testare le pipeline su un dataset meno verticale e più generico.
- Eseguire la pipeline quantistica su processori adeguati; se ciò non fosse possibile, provare ad eseguirla su altri simulatori non impiegati in questa sperimentazione, i.e. *IBM Quantum*.
- Valutare un approccio diverso per il parsing delle frasi, sfruttando dunque altri parser e ansätze forniti dai toolkit coinvolti, confrontando i risultati ottenuti anche in termini di tempo e risorse impiegate.
- Modificare i parametri dei trainer, in particolare testare gli addestramenti basandosi su altre *loss functions* e con altri ottimizzatori, così da comparare le performance.

Bibliografia

[1] B. Bruegge and A. H. Dutoit, *Object–oriented software engineering. using UML, Patterns, and Java*, 2009. (Citato a pagina 1)

[2] I. Nigmatullin, A. Sadovykh, S. Ebersold, and N. Messe, "Rqcode: Security requirements formalization with testing," in *Testing Software and Systems*, S. Bonfanti, A. Gargantini, and P. Salvaneschi, Eds., 2023. (Citato a pagina 1)

[3] "General Data Protection Regulation," https://gdpr-info.eu. (Citato a pagina 1)

[4] B. Bruegge and A. H. Dutoit, *Object–oriented software engineering. using UML, Patterns, and Java*, 2009. (Citato alle pagine 1 e 4)

[5] P. Limna, T. Kraiwanit, K. Jangjarat, P. Klayklung, and P. Chocksathaporn, "The use of chatgpt in the digital era: Perspectives on chatbot implementation," vol. 6, 06 2023. (Citato a pagina 2)

[6] R. Guarasci, G. De Pietro, and M. Esposito, "Quantum natural language processing: Challenges and opportunities," *Applied Sciences*, vol. 12, p. 5651, 06 2022. (Citato alle pagine 2 e 7)

[7] Donald Firesmith, "Engineering security requirements," *Journal of Object Technology*, vol. 2, no. 1, pp. 53–68, Jan. 2003, (column). [Online]. Available: http://www.jot.fm/contents/issue_2003_01/column6.html (Citato a pagina 5)

[8] B. Lutkevich, "What is natural language processing?" 2023. (Citato a pagina 6)

[9] D. Khurana, A. Koli, K. Khatter, and S. Singh, "Natural language processing: State of the art, current trends and challenges," *Multimedia Tools and Applications*, vol. 82, 07 2022. (Citato a pagina 6)

[10] C.-C. Lin, A. Huang, and S. Yang, "A review of ai-driven conversational chatbots implementation methodologies and challenges (1999–2022)," *Sustainability*, vol. 15, p. 4012, 02 2023. (Citato a pagina 6)

[11] R. Lao, "A beginner's guide to machine learning," 2018. (Citato a pagina 7)

[12] A. Kumar, "Difference: Binary, multiclass multi-label classification," 2022. (Citato a pagina 7)

[13] A. De Lucia, F. Palomba, M. De Stefano, F. Pecorelli, and D. Di Nucci, "Software engineering for quantum programming: How far are we?" *The Journal of Systems Software*, 2022. (Citato a pagina 8)

[14] M. Piattini, M. Serrano, R. Perez-Castillo, G. Petersen, and J. L. Hevia, "Toward a quantum software engineering," *IT Professional*, vol. 23, no. 1, pp. 62–66, 2021. (Citato a pagina 8)

[15] Z. Yongli, M. Surya Teja, and P. Hemanth Sai Kumar, "Introduction to quantum computing," *Quantum Computing and Communication*, 2022. (Citato a pagina 9)

[16] S. Raubitzek and K. Mallinger, "On the applicability of quantum machine learning," *Entropy*, vol. 25, p. 992, 06 2023. (Citato a pagina 9)

[17] A. Manzano, D. Dechant, J. Tura Brugués, and V. Dunjko, "Parametrized quantum circuits and their approximation capacities in the context of quantum machine learning," 07 2023. (Citato a pagina 9)

[18] G. Quba, H. Qaisi, A. Thunibat, and S. AlZu'bi, "Software requirements classification using machine learning algorithm's," pp. 685–690, 07 2021. (Citato a pagina 10)

[19] S. Zhang, X. Li, M. Zong, X. Zhu, and D. Cheng, "Learning k for knn classification," *ACM Transactions on Intelligent Systems and Technology*, vol. 8, pp. 1–19, 01 2017. (Citato a pagina 10)

[20] D. Valero-Carreras, J. Alcaraz, and M. Landete, "Comparing two svm models through different metrics based on the confusion matrix," *Computers Operations Research*, vol. 152, p. 106131, 12 2022. (Citato a pagina 10)

[21] Y. Zhang, R. Jin, and Z.-H. Zhou, "Understanding bag-of-words model: A statistical framework," *International Journal of Machine Learning and Cybernetics*, vol. 1, pp. 43–52, 12 2010. (Citato a pagina 10)

[22] M. Riaz, J. King, J. Slankas, and L. Williams, "Hidden in plain sight: Automatically identifying security requirements from natural language artifacts," pp. 183–192, 2014. (Citato a pagina 10)

[23] W. Wang, K. R. Mahakala, A. Gupta, N. Hussein, and Y. Wang, "A linear classifier based approach for identifying security requirements in open source software development," *Journal of Industrial Information Integration*, vol. 14, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2452414X17301000 (Citato a pagina 10)

[24] K. Terashi, M. Kaneda, T. Kishimoto, M. Saito, R. Sawada, and J. Tanaka, "Event classification with quantum machine learning in high-energy physics," *Computing and Software for Big Science*, vol. 5, 12 2021. (Citato a pagina 11)

[25] S. Ganguly, S. Morapakula, and L. Pozo Coronado, "Quantum natural language processing based sentiment analysis using lambeq toolkit," 05 2023. (Citato a pagina 11)

[26] I. Vogel and F. Antonius, "Fraunhofer sit at checkthat! 2022: Semi-supervised ensemble classification for detecting check-worthy tweets," 09 2022. (Citato a pagina 11)

[27] D. Kartsaklis, I. Fan, R. Yeung, A. Pearson, R. Lorenz, A. Toumi, G. Felice, K. Meichanetzidis, S. Clark, and B. Coecke, "lambeq: An efficient high-level python library for quantum nlp," 10 2021. (Citato a pagina 13)

[28] "lambeq 0.3.3 documentation," https://cqcl.github.io/lambeq/. (Citato a pagina 13)

[29] "Google Colab," https://colab.research.google.com. (Citato a pagina 13)

[30] "pandas - Python Data Analisys Library," https://pandas.pydata.org. (Citato a pagina 15)

[31] "SpaCy," https://spacy.io. (Citato a pagina 17)

[32] "lambeq - Monoidal Categories," https://cqcl.github.io/lambeq/tutorials/monoidal.html#Monoidal-categories. (Citato a pagina 18)

[33] J. Biamonte and V. Bergholm, "Tensor networks in a nutshell," 2017. (Citato a pagina 20)

[34] J. Spall, "Implementation of the simultaneous perturbation algorithm for stochastic optimization," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 34, no. 3, pp. 817–823, 1998. (Citato a pagina 25)

[35] "MatPlotLib," https://matplotlib.org/. (Citato a pagina 27)

Questa tesi ha contribuito a piantare un albero di caffè in Ecuador tramite Treedom.

Dale Cooper approves!

`https://www.treedom.net/it/user/sesalab/event/sesa-random-forest`