

Università degli Studi di Salerno

Dipartimento di Informatica



Tesi di Laurea Magistrale in
Informatica

Technical Debt in
Sistemi di Intelligenza Artificiale:
uno Studio Empirico sullo Stato
dell'Arte e della Pratica

Relatore

Prof. Fabio Palomba

Candidato

Gilberto Recupito

Matricola:0522500842

Anno Accademico 2021-2022

A mio padre e mio fratello.

Abstract

La diffusione dell'intelligenza artificiale (AI) nei prodotti industriali ha condotto alla creazione di sistemi in continua crescita, portando alla trasformazione di modelli sperimentali in prodotti commerciali. L'esigenza di favorire l'evoluzione di questi sistemi ha portato all'introduzione delle pratiche di ingegneria del software mirate ai sistemi di intelligenza artificiale. In particolare, la necessità di preservare la qualità dei modelli AI ha incrementato l'interesse della ricerca e dell'industria nel trovare metodologie utili a affrontare le più diffuse minacce alla qualità di questi sistemi. In questo contesto, lo studio e le analisi di prevenzione delle tipologie di Technical Debt specifiche per AI (AI Technical Debt) rappresenta attualmente una delle più grandi sfide, in quanto la gestione di esse consente di identificare e mitigare severe problematiche che influenzano diversi aspetti di qualità dei sistemi. Mentre l'importanza di gestire AI Technical Debt è emergente, l'attuale sforzo della ricerca e dell'industria non è sufficiente a fornire una ben definita tassonomia e catalogazione delle istanze che possono causare AI Technical Debt nei sistemi AI. Il lavoro di tesi definito, quindi, fornisce un cospicuo contributo alla definizione delle istanze di AI Technical Debt, offrendo alla comunità accademica e industriale di poter analizzare la frequenza, la severità e l'impatto sugli aspetti di qualità delle possibili minacce che impediscono l'evoluzione dei moduli AI all'interno dei sistemi. Di conseguenza, è stato condotto uno studio preliminare della letteratura presente nello stato dell'arte, al fine di estrarre le tipologie e le istanze di AI Technical Debt più diffuse. Successivamente, il lavoro di tesi presenta l'investigazione del punto di vista di 54 professionisti di sistemi AI sulla

frequenza, severità e impatto di istanze che possono causare AI Technical Debt sulla struttura del codice e sull'architettura del sistema. I risultati mostrano un'alta diffusione e discussione in letteratura delle istanze di AI Technical Debt relative ai dati (*data debt*), in particolare per le istanze che identificano dipendenze instabili tra i dati e le istanze che identificano informazioni e dati poco utilizzati dal modello. I risultati estratti sia dallo stato dell'arte che dalla pratica evidenziano, inoltre, un'alta influenza delle istanze di AI Technical Debt relative al codice (*code debt*) e all'architettura (*architectural debt*). In particolare, le istanze di *Undeclared Consumers*, *Pipeline Jungle* e *Jumbled Model Architecture* sono state identificate essere tra le più severe e conseguire in un alto impatto nella qualità dei sistemi AI.

Indice

1	Introduzione	2
1.1	Motivazioni e Obiettivi	2
1.2	Risultati	4
1.3	Struttura della tesi	4
2	Background	6
2.1	Qualità del Software	6
2.1.1	ISO/IEC 9126	8
2.2	Technical Debt	9
2.2.1	Code Debt	12
2.3	Intelligenza Artificiale	14
2.3.1	Machine Learning	15
2.4	Ingegneria del Software per Intelligenza Artificiale (SE4AI) . .	18
2.4.1	Ciclo di vita di un applicazione di machine learning . .	21
3	Stato dell'Arte	24
3.1	ML-Specific Technical Debt	24
3.1.1	Data Dependency Debt	25
3.1.2	Analysis Debt	27
3.1.3	System Level Antipattern	28
3.1.4	Configuration Debt	29
3.1.5	Common Smells	30
3.2	Data Technical Debt	31
3.3	Code Technical Debt	32

4 Metodologia	35
4.1 Obiettivo di Ricerca	35
4.2 Systematic Literature Review	36
4.2.1 Processo di Ricerca	36
4.2.2 Identificazione della Ricerca	38
4.2.3 Validazione della qualità degli studi selezionati	41
4.3 Survey	43
4.3.1 Motivazione del survey	43
4.3.2 Obiettivo di Ricerca	43
4.3.3 Research Questions	44
4.3.4 Design del Survey	45
4.3.5 Design e configurazione dell'esperimento	45
4.3.6 Validazione del Survey	52
4.3.7 Reclutamento dei partecipanti	53
5 Analisi e Risultati	56
5.1 Tassonomia definita in letteratura	56
5.1.1 Risultati Preliminari	56
5.1.2 RQ1.1: Quali tipologie di AI Technical Debt sono presenti in letteratura?	58
5.1.3 RQ1.2: Quali sono gli approcci di identificazione e mitigazione di AI Technical Debt?	60
5.2 Revisione del punto di vista degli sviluppatori	62
5.2.1 Risultati Preliminari	62
5.2.2 RQ2.1: Quali sono le tipologie di code debt e architectural debt più frequenti secondo la percezione degli sviluppatori AI?	64
5.2.3 RQ2.2: Quali sono le tipologie di code debt e architectural debt più problematiche secondo la percezione degli sviluppatori AI?	64

<i>INDICE</i>	1
5.2.4 RQ2.3: Qual è l'impatto causato dai code debt e architectural debt secondo la percezione degli sviluppatori AI?	68
5.2.5 RQ 2.4: Quali sono le strategie utilizzate dagli sviluppatori AI per l'identificazione e la mitigazione di code debt e architectural debt?	71
6 Conclusioni ed Implicazioni	74

Capitolo 1

Introduzione

1.1 Motivazioni e Obiettivi

L'emergere dell'intelligenza artificiale (AI) ha colpito in modo pervasivo le industrie software con la proliferazione di sistemi AI-based, integrando, quindi, le componenti AI nei sistemi software che operano nell'industria [1]. Costruire, operare ed effettuare manutenzione dei sistemi AI però è differente dalle tradizionali metodologie di sviluppo e manutenzione di sistemi software. Infatti, in questi ultimi, il comportamento è dettato dal flusso di controllo del programma mentre, nei sistemi AI-based, le decisioni sono dettate dai dati. Questo nuovo punto di vista fa affiorare l'esigenza di esplorare le metodologie e le pratiche di Software Engineering per sviluppare, manutenere ed evolvere i sistemi AI. Questo è ciò che la comunità di ricerca di Software Engineering ha nominato ***Software Engineering for Artificial Intelligence*** (SE4AI).

Le industrie devono affrontare molte sfide al fine di creare soluzioni AI a larga scala per il mercato. Fischer et al. [2] hanno condotto uno studio per scoprire le maggiori sfide nello sviluppo di applicazioni AI. In particolar modo, tra le più grandi sfide che un professionista deve affrontare figurano il garantire la qualità dei dati e del software. La gestione della qualità, sia dei dati che dell'applicazione, è una procedura fondamentale per le applicazioni AI, in quanto una scarsa qualità può portare a gravi conseguenze. Si consideri,

tra i numerosi incidenti che coinvolgono l'AI, il caso di Elaine Herzberg, morta a causa di un errore di un'autovettura autonoma di Uber. Il modello integrato nell'autovettura ha effettuato un errore di classificazione sul pedone che stava attraversando la strada, portando quindi l'autovettura a proseguire [3]. E questo, purtroppo, non è l'unico caso di sistema AI che ha causato ingenti danni (sia a persone che economici) a causa di negligenza nel controllo di qualità del modello. È necessario quindi porre particolare attenzione agli aspetti di qualità dei sistemi AI e in particolare, ai problemi che possono causare un degrado della qualità. La definizione di tecniche di validazione abilitate a prevenire il crollo della qualità rappresenta una delle più grandi sfide nel campo SE4AI. In questo contesto sono cruciali l'identificazione, la diagnosi e la gestione del debito tecnico, noto come technical debt. Il technical debt (TD) viene definito come un insieme di scelte di design subottimali o soluzioni implementative che possono influenzare negativamente i dati e la qualità del codice [4]. I sistemi di Machine Learning hanno una notevole predisposizione nell'essere affetti da technical debt, poiché sono esposti sia alle tradizionali tipologie di technical debt, sia a ***Artificial Intelligence Technical Debt (AITD)***. Questi ultimi rappresentano le tipologie di technical debt che pregiudicano la qualità dei sistemi AI-based, e risultano difficili da rilevare perché presentano diverse granularità, arrivando quindi a definire la rilevazione anche a livello di sistema. È fondamentale, quindi, rilevare e mitigare tempestivamente le problematiche derivanti dal TD, attraverso ciò che viene denominato *technical debt management*. Sebbene sia urgente e necessario gestire il TD dei sistemi AI, allo stato attuale non esistono soluzioni sufficientemente efficaci a rilevare e mitigare queste criticità.

L'obiettivo di questo lavoro di tesi è quello di fornire un contributo cospicuo sia al settore accademico sia al settore industriale per la identificazione delle minacce alla qualità del software AI. Il lavoro di tesi prevede quindi un'analisi preliminare sistematica della letteratura al fine di revisionare lo stato attuale dell'arte e ottenere un quadro generale sulle istanze che possono provocare AITD. Successivamente, questo lavoro si pone di contribuire alla definizione

delle istanze di AITD, attraverso l'investigazione tramite survey su larga scala di esperti e professionisti che hanno esperienza nella realizzazione e la manutenzione di sistemi e la gestione di una pipeline AI. Infine le risposte sono analizzate al fine di ottenere per ogni specifico problema che può causare AITD la sua frequenza, la sua severità e il suo impatto, ed estrarre infine soluzioni volte all'identificazione, alla diagnosi e alla mitigazione di AITD e che, quindi, migliorino la qualità del software AI.

1.2 Risultati

I risultati hanno mostrato un'alta diffusione di diverse istanze di AI Technical Debt, dove le più diffuse sono relative ai dati, alla configurazione del sistema, al codice e all'architettura. Concentrando lo studio successivamente sulle ultime due tipologie, è stata riscontrata un'alta severità e un alto impatto secondo la percezione degli sviluppatori AI. Infine, i professionisti AI attualmente non dispongono dei mezzi necessari al fine di effettuare processi di identificazione e mitigazione delle minacce definite tramite tecniche automatizzate, ma conducono ispezioni e revisioni manuali.

1.3 Struttura della tesi

Il lavoro di tesi è organizzato nel modo seguente: nel capitolo due si forniscono i principi teorici su cui il lavoro è basato, al fine di facilitare la comprensione del contesto e gli l'importanza della gestione del Technical Debt nei sistemi AI. Nel terzo capitolo vengono illustrati i lavori presenti nello stato dell'arte che hanno contribuito alla definizione di una tassonomia preliminare. Nel quarto capitolo è introdotta la metodologia adoperata. Nella prima parte del capitolo è descritto il processo di revisione della letteratura sistematico e il processo di estrazione delle informazioni da essa. Nella seconda parte, invece, viene illustrata la metodologia utilizzata all'investigazione della percezione degli sviluppatori AI. In particolare, viene illustrato il processo di conduzione

del questionario, della selezione dei partecipanti, e della progettazione dell'esperimento. Infine, nel quinto capitolo di questo lavoro sono presentati i risultati delle analisi condotte sulla frequenza, la severità e il loro impatto. Nel sesto capitolo sono fornite le conclusioni del lavoro svolto e delineati gli sviluppi futuri.

Capitolo 2

Background

In questo capitolo sono illustrate le nozioni preliminari che forniscono una base solida al presente lavoro di tesi.

2.1 Qualità del Software

Le tecniche di Ingegneria del Software sono fondamentali alla gestione della qualità del software AI. Lo standard IEEE definisce l'ingegneria del software come l'applicazione di un approccio sistematico, disciplinato e quantificabile volto allo sviluppo e la manutenzione del software [5]. L'insieme delle metodologie, i principi e le tecniche di Ingegneria del software hanno l'obiettivo di ottenere un prodotto software affidabile ed efficiente.

L'insieme delle attività che definiscono le fasi di realizzazione del software, a partire dalla definizione della sua specifica fino alla manutenzione viene definito processo di ciclo di vita del software. Esistono diversi processi di cicli di vita del software altamente adoperati all'interno del settore industriale, e tutti i processi esistenti includono le seguenti fasi [6]:

- **Specifiche del software:** insieme delle attività di comprensione e definizione delle specifiche richieste dal sistema e l'identificazione dei vincoli sulle operazioni e lo sviluppo del sistema.

- **Progettazione e implementazione del software:** insieme delle attività volte alla progettazione dell’architettura, delle interfacce e delle componenti.
- **Validazione del software:** insieme delle attività che effettuano la validazione e la verifica del software, includendo sia attività di ispezione e di revisione del software, sia attività di testing.
- **Evoluzione del software:** insieme delle attività che consentono il continuo aggiornamento e favorire la crescita dinamica del software.

Ognuna di queste fasi prevede una serie di attività utili al supporto della realizzazione del progetto. In particolare, è necessario notare che ogni fase viene supportata da attività di pianificazione del progetto, di monitoraggio e di tracciabilità degli artefatti. Una delle più importanti attività che viene svolta durante la fase di monitoraggio del sistema software è la valutazione della qualità.

Una definizione importante e precisa da considerare quando si interagisce con la qualità del software è quella definita dallo Standard ISO/IEC 25010:2011 [7]. Essa viene definita come *"la capacità del software di soddisfare le esigenze dichiarate e implicite sotto specifiche condizioni quando utilizzato"*. Queste particolari esigenze da soddisfare sono state definite dai modelli di qualità, i quali hanno lo scopo di effettuare una categorizzazione della qualità del software tramite un’insieme di caratteristiche. Successivamente queste caratteristiche sono ulteriormente scomposte ciclicamente in sottocaratteristiche e proprietà. Ogni definizione di sottoinsieme di caratteristiche ha lo scopo di poter ottenere una rappresentazione dettagliata di un aspetto della qualità del software, a tal punto da renderlo misurabile. In figura 2.1 è possibile osservare come dalla definizione generica di un aspetto della qualità del software è possibile quindi andare a definire delle proprietà che sono direttamente misurabili.

Tra i modelli maggiormente utilizzati per la gestione della qualità del software è presente il modello ISO/IEC 9126.

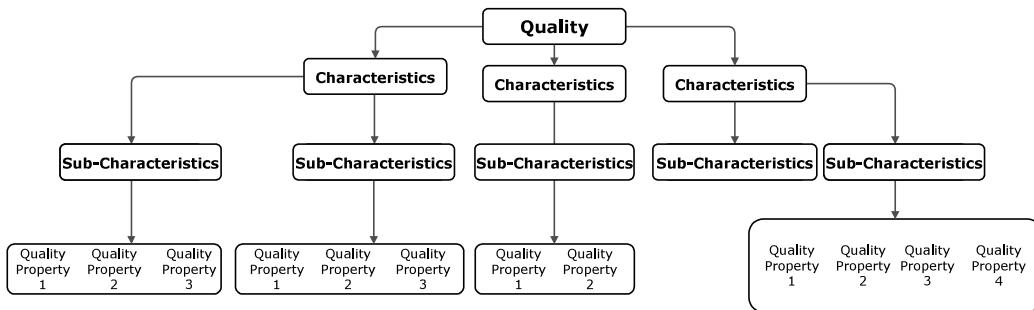


Figura 2.1: Struttura utilizzata per la definizione di modelli di qualità

2.1.1 ISO/IEC 9126

Le norme proposte all'interno del modello *ISO/IEC 9126* sono emesse dall'ISO, l'organismo internazionale di standardizzazione (International Organization for Standardization) e, in particolare, dall'organo interno del settore delle tecnologie e della comunicazione (IEC, International Electrotechnical Commission). Le norme emesse quindi definiscono la qualità del software in quattro parti:

1. Modello di qualità del software.
2. Metriche per la qualità esterna.
3. Metriche per la qualità interna.
4. Metriche per la qualità in uso.

La qualità del software è particolarmente legata alla maturità del processo definito al fine dello sviluppo. E' importante quindi definire una relazione tra le diverse tipologie di qualità presenti nel modello di qualità. Dall'insieme di relazioni che collegano la qualità del processo alla qualità del prodotto, come raffigurato in figura 2.2, è possibile evincere che migliorare aspetti di qualità del processo possono influenzare positivamente aspetti della qualità del prodotto. A sua volta il miglioramento della qualità interna, e quindi indirettamente includendo la qualità esterna del prodotto, porta benefici alla qualità in uso del prodotto. Inoltre, seguendo direttamente le relazioni in

senso inverso, è possibile valutare la qualità in uso al fine di avere un sistema di feedback che fornisce informazioni sull'effetto dei miglioramenti apportati, a sua volta utili per il miglioramento del processo.

Gli attributi interni del software, quindi, sono un prerequisito fondamentale per analizzare anche altri aspetti della qualità del software, come le caratteristiche esterne del prodotto, rappresentano un prerequisito fondamentale per analizzare la qualità in uso desiderata.

Il modello di qualità del software quindi è definito da un insieme di caratteristiche che riguardano diversi aspetti del software, dove in ciascuna caratteristica sono a sua volta definite sottocaratteristiche. In particolare, analizzando il modello *ISO/IEC 9126*, la qualità è modellata dalla definizione di 6 caratteristiche, strutturate come in Tabella 2.1.

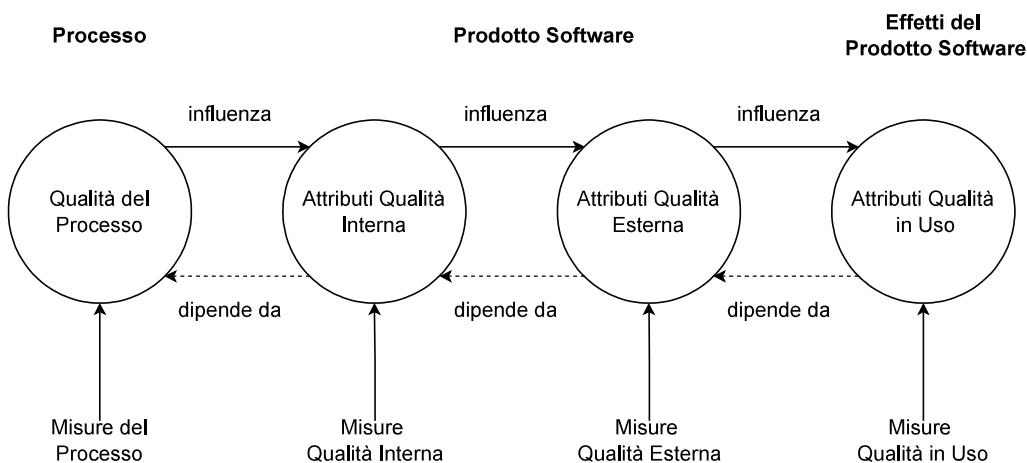


Figura 2.2: Ciclo di sviluppo della qualità del prodotto software

2.2 Technical Debt

Come è fondamentale effettuare analisi e monitorare la qualità del proprio prodotto, è altrettanto fondamentale percepire e identificare problematiche che possano danneggiare la qualità. Molte industrie e diversi professionisti del software, al fine di poter andare incontro al *time-to-market* (i.e., l'ammontare

Caratteristiche	Attributi
Funzionalità (Functionality)	Completezza, Accuratezza, Interoperabilità, Sicurezza, Aderenza alla funzionalità.
Affidabilità (Reliability)	Maturità, Tolleranza ai guasti, Recuperabilità, Aderenza all'affidabilità.
Usabilità (Usability)	Comprensibilità, Apprendibilità, Operabilità, Attrattività, Aderenza all'usabilità.
Efficienza (Efficiency)	Comportamento sul tempo di esecuzione, Utilizzo delle risorse, Aderenza all'efficienza.
Manutenibilità (Maintainability)	Analizzabilità, Modificabilità, Stabilità, Provabilità, Aderenza alla manutenibilità.
Portabilità (Portability)	Adattabilità, Installabilità, Coesistenza, Sostituibilità, Aderenza alla portabilità.
Qualità in uso (Quality in use)	Efficacia, Produttività, Sicurezza, Soddisfazione.

Tabella 2.1: Modello di qualità ISO/IEC 9126

di tempo richiesto per cui un prodotto, a partire dalla concezione dell'idea, raggiunge la fase di distribuzione nel mercato) e altri fattori di business, si ritrovano ad affrontare quello che viene definito come *technical debt*. La prima definizione di Technical Debt è stata coniata da Ward Cunningham [4], dove viene definito come un insieme di scelte di design subottimali o soluzioni implementative che possono influenzare negativamente i dati e la qualità del codice. Questa iniziale definizione porta al collegamento metaforico del

debito finanziario, dove un professionista del software che sceglie di adoperare scelte di design non ottimali può essere associato a una persona che incorre in un debito di pagamento. Quando una persona non salda in tempi regolari e ammissibili il debito assegnato riscontra una forma di interesse. Questa forma di interesse continua a crescere in maniera proporzionale al tempo necessario per poter saldare il debito definito. Allo stesso modo, quando uno sviluppatore decide di non ripagare il debito creato all'interno del sistema (*i.e.*, migliorare la soluzione subottimale introdotta), l'interesse si accumula e l'ammontare del technical debt aumenta. Quindi il costo di mitigazione del technical debt del sistema cresce in modo proporzionale al tempo che intercorre tra l'introduzione del technical debt e la sua mitigazione. In casi estremi, l'interesse viene accumulato a tal punto da essere impossibile da mitigare, causando danni ingenti al sistema e l'abbandono del prodotto. Questa particolare casistica viene definita come *technical bankruptcy*. [8]

La metafora finanziaria del technical debt inoltre ha portato all'adozione della sua terminologia dalla parte dei professionisti di ingegneria del software. Ampatzoglou et al. [9] hanno analizzato la terminologia maggiormente utilizzata per la rappresentazione e l'analisi del Technical Debt nei sistemi e in particolare hanno estratto le definizioni di technical debt *principal* e *interest*:

- **Principal:** ha lo scopo di valutare l'effort richiesto per indirizzare il sistema attuale in un sistema che abbia un livello ottimale di qualità di progettazione.
- **Interest:** ha lo scopo di valutare l'effort addizionale necessario per applicare tecniche di manutenzione e recuperare la qualità del software.

Esistono diverse tipologie di technical debt, il quale ognuno è relativo a una specifica granularità del sistema:

- **Code Debt:** conseguenza dell'introduzione di scelte implementative subottimali, come violazioni rilevate da un tool di analisi statica, o uno stile del codice inconsistente.

- **Design Debt:** conseguenza dell'introduzione di design smells e violazione di regole di progettazione.
- **Test Debt:** conseguenza della mancanza gestione delle operazioni di testing o dalla inadeguatezza del raggiungimento dei criteri di testing.
- **Documentation Debt:** conseguenza della mancanza di documentazione o di una scarsa definizione dei dettagli del sistema.

In questo lavoro di tesi saranno discusse le istanze di technical debt inerenti all'investigazione effettuata. Nella prossima sezione quindi sono introdotte le istanze di Technical debt relative al codice del sistema.

2.2.1 Code Debt

L'implementazione del codice sorgente prevede molte decisioni riguardanti lo stile e il design della soluzione. Il codice sorgente può essere soggetto a revisione, ispezione e analisi statica per rilevare i problemi di piccola granularità. Tipici esempi di problemi di implementazione del codice sono la violazioni di standard del codice, una errata nomenclatura, codice duplicato e fuorvianti o incorretti commenti del codice. Molti di questi sintomi sono definiti come *code smells*. Quando il sistema incorre in technical debt con una granularità al livello del codice, il debito tende a diminuire attributi di qualità come la maintainability a tal punto da rendere difficile di fare le correzioni al sistema quando ne ha bisogno. I code smell sono definiti come scelte implementative di bassa qualità che possono nascondere problemi più gravi alla qualità del sistema [10]. Ogni code smell può influenzare diversi aspetti di qualità del codice e possono avere una diversa severità al variare dei possibili danni che possono causare al sistema. Di seguito saranno elencati e illustrati i code smell più identificati nei sistemi software presenti nello stato della pratica, come riportati da uno studio sulla percezione della presenza e la frequenza dei professionisti del software condotto da Palomba et al [11].

God Class

Un’istanza di *God Class* rappresenta una classe che risponde all’esigenza di multiple responsabilità e contiene quindi un numero alto di funzionalità. Le problematiche di questa classe non sono causate direttamente dalla sua dimensione, ma da fattori che vanno implicitamente a incidere sulla manutenibilità del software. In particolare, quando viene identificata una istanza di questo genere all’interno del proprio sistema, i seguenti attributi di qualità del sistema hanno gravi conseguenze:

- **Coesione:** Grado di correlazione tra le responsabilità e le funzionalità a cui la classe fornisce soluzione. Se la classe ha un’alta coesione, allora le funzionalità all’interno di essa convergono a fornire una soluzione alla stessa responsabilità. Nel caso in cui viene identificata una *God Class*, questa metrica di qualità tende a decrementare data la tua propensione a includere diverse responsabilità.
- **Accoppiamento:** Grado di misurazione della dipendenza della attuale classe con altre componenti del sistema. Se la classe ha un alto accoppiamento, allora la classe presenta un alto numero di dipendenze. Questo comporta alla riduzione di attributi di qualità come la modificabilità del software (appartenente alla caratteristica di manutenibilità del codice), dove ogni attività di manutenzione del software che include questa componente, comporta ad avere un alto impatto su altre componenti.

Complex Class

Un’istanza di *Complex Class* rappresenta una classe che contiene metodi che riportano un’alta complessità. Anche se, apparentemente un’istanza di *God Class* e un’istanza di *Complex Class* possono essere simili, diversi sono gli effetti e i danni in termini di qualità che causano al sistema. In questo caso, quando viene identificata una istanza di questo genere, le gravi conseguenze sono riscontrate in termini di complessità del codice. In particolare, analizzando

le metriche di complessità maggiormente utilizzate, è possibile riscontrare i danni che può causare l'introduzione di una *Complex class*.

- **Cyclomatic Complexity:** Metrica del software utilizzata per indicare la complessità di un programma. È una misura quantitativa ricavata dal numero dei cammini linearmente indipendenti che attraversa il codice sorgente del programma [12]. La complessità di ogni metodo presente all'interno di una *Complex Class* incide su questa metrica aumentando vertiginosamente il suo valore, la quale incide su attributi di qualità come la manutenibilità e la comprensibilità del software.

Long Method

Allo stesso modo in cui una *God Class* tende a centralizzare le responsabilità di un sistema, un'istanza di *Long Method* rappresenta un metodo che contiene diverse responsabilità di una classe, incrementando la difficoltà di manutenzione e di comprensibilità. Per poter analizzare una istanza di questo genere, è necessario fare affidamento anche in questo caso alla misurazione di attributi di qualità come coesione e accoppiamento.

2.3 Intelligenza Artificiale

Il termine intelligenza artificiale ha avuto un processo di evoluzione considerevole negli ultimi anni. Molte definizioni descrivono l'intelligenza artificiale come la capacità di elaboratori o sistemi robotici di poter eseguire operazioni come l'apprendimento o il prendere decisioni in modo analogo alla capacità degli umani [13], [14] Quello che rende l'intelligenza artificiale estremamente diffusa nel mondo accademico e industriale è la sua capacità evolutiva, in particolare nell'ultimo decennio, trasformando questa definizione a tal punto da renderla incompleta. Una definizione che evidenzia maggiormente lo scopo di questa disciplina è stata fornita da John McCarthy [15], la quale viene definita come:

La scienza e l'ingegneria di rendere le macchine intelligenti e in particolare i programmi intelligenti. E' correlata a task simili all'utilizzo del computer per la comprensione dell'intelligenza umana, ma l'intelligenza artificiale non deve limitarsi ai metodi che sono osservabili biologicamente.

Sebbene lo scopo di questa disciplina era inizialmente quello di poter permettere alle macchine di imitare la capacità intellettuale degli umani, ora si ritiene che essa possa andare oltre. L'attuale definizione porta alla definizione di questa disciplina su più livelli [16]:

- **Artificial Narrow Intelligence:** applicazione dell'intelligenza artificiale al fine di avere la capacità di applicare intelligenza in un determinato problema.
- **Artificial General Intelligence:** applicazione dell'intelligenza artificiale al fine di avere la capacità di applicare intelligenza su tutti i problemi su cui un essere umano può applicare.
- **Artificial Super Intelligence:** applicazione dell'intelligenza artificiale al fine di superare le capacità di applicare intelligenza su tutti i problemi rispetto a un essere umano.

2.3.1 Machine Learning

Il machine learning è uno dei rami dell'intelligenza artificiale utilizzato dalle industrie. Esso rappresenta l'insieme di tecniche, metodologie e teorie utili a consentire agli elaboratori di poter apprendere utilizzando metodi matematico-computazionali [17]. Quest'area di ricerca ha l'intento di permettere alle macchine di riconoscere pattern e creare un sistema di decisioni basandosi sui dati. In particolare può essere definita come l'applicazione di migliorare sull'esecuzione di un task T rispetto alla misura di performance P basandosi sull'esperienza E . Un tipico esempio per la rappresentazione di un sistema di

Task (T)	Identificare le e-mail che gli utenti non desiderano ricevere.
Performance (P)	Percentuale di e-mail spam che sono filtrate e percentuale di e-mail che sono incorrettamente filtrate.
Esperienza (E)	Database di e-mail etichettate come e-mail spam e o non-spam.

Tabella 2.2: Esempio di rappresentazione di un sistema di Machine Learning

machine learning è quello del filtraggio delle e-mail spam, come raffigurato in Tabella 2.2.

A seconda della rappresentazione del task T e dell'esperienza E che si ha a disposizione, è possibile utilizzare diverse tecniche di machine learning, classificabili nelle seguenti tipologie [18]:

- Apprendimento supervisionato (Supervised Learning): Questa tipologia di tecnica ha l'obiettivo di creare un modello di predizione utilizzando come input i dati costituiti sia da un insieme di attributi, sia da un attributo target. Attraverso la definizione di questi dati, il modello cerca di comprendere il valore dell'attributo obiettivo (target) su nuove osservazioni, non elaborate prima. Per questo tipo di apprendimento esistono due tipologie che si differenziano in base al tipo di predizione che si vuole effettuare sull'attributo target: classificazione e regressione. La prima tipologia si basa sul predire la classe di appartenenza tra le diverse istanze di oggetti, la seconda, anche se concettualmente simile alla classificazione, si basa sul predire il valore dell'attributo target su insieme di valori continui.
- Apprendimento non supervisionato (Unsupervised Learning): A differenza delle tecniche di apprendimento supervisionato, questa tipologia prevede l'utilizzo di dati non classificati e che appartengono quindi a strutture sconosciute. In particolare, il termine *non supervisionato*

deriva dalla caratteristica che i dati non sono etichettati secondo un determinato attributo target. Una tipica applicazione per le tecniche di questa tipologia sono i task di clustering: lo scopo è quello di effettuare un'analisi al fine di identificare nell'insieme di dati dei sottinsieme (detti *cluster*), la quale suddivisione fornisce informazione sul sott'insieme dei dati.

- Apprendimento per rinforzo (Reinforcement Learning): Le tecniche di apprendimento per rinforzo prevedono l'estrazione dei dati tramite l'interazione del modello con l'ambiente. Il modello esegue un'azione e riceve un feedback dall'ambiente il quale sarà fonte di informazioni utile alla decisione di quale sarà la prossima azione da eseguire. In questo modo, il modello comprende quali sono le migliori azioni al fine di massimizzare il risultato. In particolare, attraverso la definizione di politiche di premialità e di penalità, il modello indirizza la serie di scelte al fine di ottenere il più alto punteggio.

Per la definizione di un modello di machine learning è necessario andare a pianificare il processo di apprendimento, come illustrato in Figura 2.3. Il processo tipicamente utilizzato prevede quindi di partire dall'estrazione dei dati. La scelta dei dati da estrarre dipende dal problema che si affronta e dalla tipologia di tecnica che si utilizza. La qualità dell'insieme dei dati viene poi analizzata. In questa fase saranno identificati tutte le problematiche dei dati che possono comportare al degrado della qualità del modello, come la presenza di *missing value* o *outlier* della distribuzione. Successivamente i dati vengono trasformati e, in particolare, dalla forma grezza iniziale dei dati sarà avviato un processo di *feature engineering*, il quale ci consente di poter estrarre la serie di attributi che aumentano il livello di rappresentazione e dettaglio dei dati. Alla fine di questa fase, i dati assumeranno una forma strutturata in base al task da elaborare. Nel caso di utilizzo di una tecnica di apprendimento supervisionato, dai dati sarà estratto un insieme di osservazioni dove ognuna è rappresentata dalla descrizione di un insieme di attributi (definite in letteratura come *features*) e da un'attributo target su cui andare

a basare il nostro task di classificazione. Una volta che i dati sono stati strutturati, è possibile andare a selezionare la tecnica di machine learning di cui si ha bisogno, i parametri di configurazione del modello finale, e infine effettuare l'addestramento attraverso l'analisi di una parte dell'insieme dei dati definita come *training set*. Il risultato della fase precedente quindi sarà un modello capace di poter effettuare elaborazione del task. Quindi, questo modello sarà soggetto a un processo di validazione al fine di poter misurare le sue performance e verificare la qualità del modello.

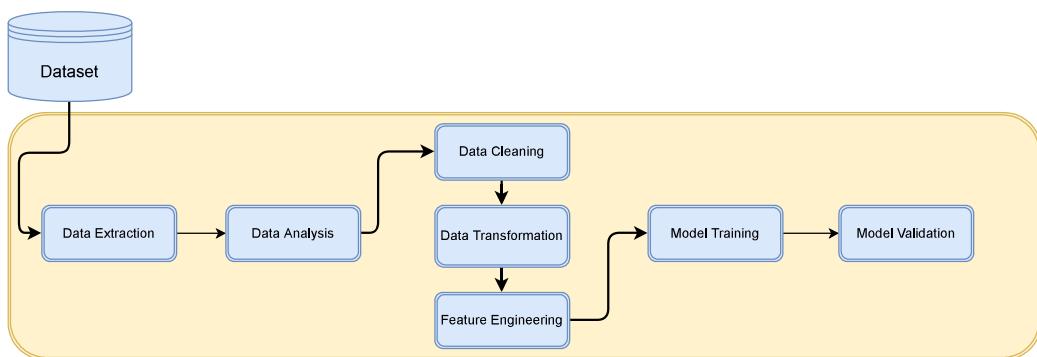


Figura 2.3: Typical Machine Learning process.

2.4 Ingegneria del Software per Intelligenza Artificiale (SE4AI)

L'introduzione di componenti di intelligenza artificiale all'interno dei contesti industriali e del software ha dimostrato un estremo interesse e un'estrema necessità di applicare queste soluzioni al fine di migliorare le pratiche di ingegneria del software. Tuttavia, al crescere della complessità delle componenti AI nei sistemi software, cresce l'esigenza di poter applicare pratiche di ingegneria del software al fine di poter permettere che queste componenti possano essere manutenute e evolvere nell'ambiente in cui operano.

Tim Menzies enuncia quindi le cinque leggi di ingegneria del software per l'intelligenza artificiale al fine di poter divulgare quanto è importante poter creare un'interazione tra queste due discipline [19].

1. I software AI non comprendono per la maggior parte componenti AI:

Un software AI comprende tantissime componenti che sono da supporto al modello utili a effettuare operazioni come la gestione e l'elaborazione dei dati. Il software può anche includere componenti di business utili a fornire le funzionalità desiderate del sistema. Sculley et al. hanno pubblicato la Figura 2.4, rappresentando per numero di linee di codice la dimensione delle diverse componenti di un software Google. Evidenziano inoltre che in generale solo una piccola porzione dei sistemi di machine learning è composta da codice machine di machine learning [20]. E' necessario quindi analizzare e supportare anche la serie di componenti che inglobano il codice AI in un sistema.

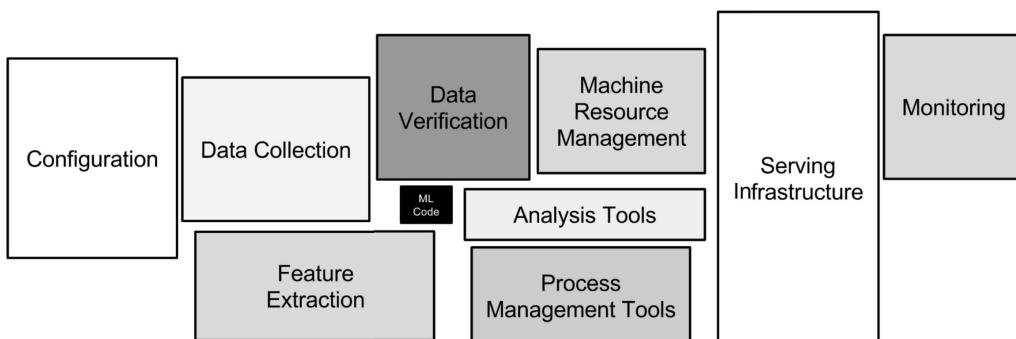


Figura 2.4: Grandezza dei software Google AI in linee di codice, il riquadro nero rappresenta codice core AI, i riquadri chiari invece illustrano il codice software di supporto.

2. I software AI hanno bisogno di ingegneri del software:

Tutti i software (AI e non) hanno bisogno di installazione, configurazione, manutenzione e di pratiche che possano trasformare il codice in prodotto. Tim Menzies quindi definisce che il futuro dei sistemi software

non può essere descritto come l'utilizzo esclusivo di una disciplina. Non può esistere solo ingegneria del software o solo intelligenza artificiale, ma, al contrario, nasceranno promettenti idee dalla combinazione di essi.

3. **Una cattiva pratica di ingegneria del software implica una cattiva applicazione di AI:** Dal momento che anche l'utilizzo di AI rientra nella creazione di un software, le cattive pratiche di ingegneria del software inficiano anche sul software AI. Sculley riporta in una conferenza che i professionisti di machine learning tendono a utilizzare tutti gli attributi che sono a disposizione dai database aziendali per creare modelli predittivi. Questo comporta all'incremento di technical debt nel modello, in quanto i modelli predittivi costruiti soffrono di una stretta dipendenza con i dati aziendali e, quando sarà effettuato un cambiamento all'interno dei dati aziendali, i modelli predittivi dovranno subire aggiornamenti e manutenzione [21]. La violazione di principi di qualità del software, come il principio di accoppiamento e coesione, porta conseguenze anche al modello di intelligenza artificiale.
4. **Una migliore pratica di ingegneria del software implica una migliore applicazione di AI:** Anche se non è necessariamente vero che applicare metodi di ingegneria del software può migliorare la qualità dell'applicazione AI, diversi professionisti di data science all'interno delle industrie hanno trovato estremi vantaggi nel loro utilizzo. Molti framework, come *scikit-learn* [22], forniscono diverse funzionalità orientate al riuso per avere a disposizione modelli di machine learning e metodi di ingegneria del software (ossia continuos integration, cloud-based testing).
5. **Gli ingegneri del software hanno bisogno di particolari tipologie di AI:** Spesso i software di analisi dei dati utilizzano gli algoritmi di AI senza entrare nel dettaglio delle scelte di design del determinato tool. Novielli et al. [23] evidenziano che molti modelli utilizzati nel campo

dell’ingegneria del software volti a effettuare *sentiment-analysis* sono costruiti su dati che non sono relativi al dominio d’interesse. Questo comporta gravi problemi in termini di performance del modello. Quindi, quello che sono considerati tool AI ”generali”, sono in realtà ristretti all’utilizzo nel dominio in cui sono stati addestrati. Gli ingegneri del software devono utilizzare quindi particolari tipologie di tool AI che siano il più possibile adatte al problema da affrontare.

2.4.1 Ciclo di vita di un applicazione di machine learning

Come già definito, le applicazioni di intelligenza artificiale, dato la sua estrema importanza nel campo industriale, si sono evolute in sistemi di larga scala. Questo ha portato un’aumento della complessità di gestione dei determinati sistemi. Uno dei contributi principali che forniscono le pratiche di ingegneria del software alle applicazioni AI è la gestione del ciclo di vita. Uno dei modelli maggiormente utilizzati è il CRISP-DM (Cross Industry Standard Process for Data Mining) [24], il quale scopo è di fornire alla comunità scientifica un modello affidabile e efficiente che supporti per ogni fase le applicazioni data-intensive. Il ciclo di vita è suddiviso quindi in sei fasi, come mostrato in Figura 2.5. I collegamenti interni del processo identificano le interazioni tra le diverse fasi del progetto e, a seconda dell’output di ogni fase, si decide la successiva fase da eseguire. I collegamenti esterni esplicitano come il modello ha una natura ciclica. Alla fine dell’esecuzione delle 6 fasi, ovvero successivamente alla fase di deployment del modello, è possibile ricevere dall’ambiente nuovi eventi che scaturiscono la necessità di implementare ulteriori requisiti di business.

Le sei fasi del processo del ciclo di vita CRISP-DM sono definite in:

- Business Understanding: Questa fase iniziale ha lo scopo di estrarre gli obiettivi e i requisiti del progetto da un punto di vista commerciale. La conoscenza risultante sarà poi convertita nella definizione di un

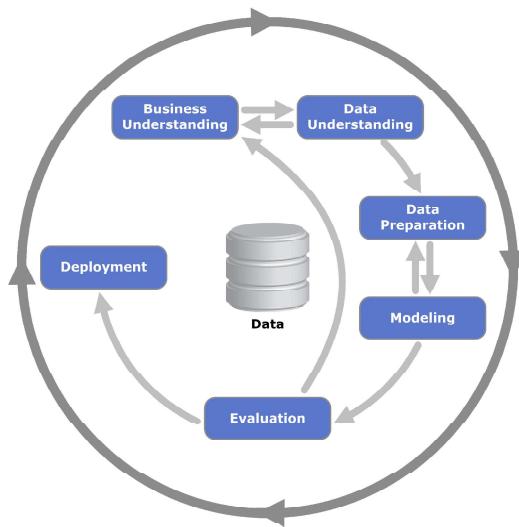


Figura 2.5: Processo di gestione del ciclo di vita CRISP-DM

problema specifico sui dati e infine sarà definita una pianificazione del progetto per raggiungere gli obiettivi.

- Data Understanding: Questa fase prevede la collezione dei dati e procede con le attività allo scopo di poter comprendere il dominio su cui si sta procedendo. Saranno quindi identificati problemi di qualità e saranno estratte le informazioni utili al raggiungimento degli obiettivi.
- Data Preparation: Questa fase prevede tutte le attività utili alla trasformazioni di un insieme di dati grezzi alla costruzione del dataset finale. Tipiche attività riguardanti la fase di data preparation sono la pulizia dei dati, la selezione degli attributi e la ristrutturazione dei dati.
- Modeling: In questa fase diverse tecniche per la creazione di modelli sono selezionate e applicate. Inoltre viene effettuata una configurazione dei parametri del modello al fine di ritrovare i loro valori ottimali e massimizzare le performance del modello.
- Evaluation: Una volta che il modello è stato costruito, è importante andare a valutare l'output della fase precedente e revisionare ogni fase

del modello, al fine di poter validare che gli obiettivi definiti sono stati raggiunti. Lo scopo è quindi quello di identificare che gli obiettivi di business e i requisiti di qualità siano stati rispettati.

- Deployment: L'ultima fase ha come scopo quello di rendere il modello utilizzabile dagli utenti. In questa fase quindi saranno definite la serie di azioni utili a effettuare la distribuzione, l'organizzazione e la presentazione del modello.

Capitolo 3

Stato dell'Arte

3.1 ML-Specific Technical Debt

I sistemi di Machine Learning hanno una caratteristica predisposizione nell’imbattersi in technical debt, poiché sono presenti ad affrontare sia i tradizionali technical debt di una applicazione, sia gli *ML-Specific technical debt*. Questa tipologia di technical debt è difficile da rilevare poichè presentano diverse granularità, arrivando quindi a definire la rilevazione anche a livello di sistema.

Sculley et al. [20] esplicitano il primo approccio alla definizione del *ML-Specific technical debt* a livello di sistema, identificando diverse tipologie di debt che ricoprissero ogni passo della Machine Learning Pipeline:

- **Data dependency debt:** Tipologia di technical debt relativa alla sezione di data management della machine learning pipeline, ovvero considerando le fasi di data ingestion, data collection e data preparation.
- **Analysis debt:** Tipologia di technical debt relativa alla sezione di model management, considerando fasi di feature selection, model selection, model execution e model updating.
- **Configuration debt:** Tipologia di technical debt relativa alla configurazione dei sistemi di machine learning.

3.1.1 Data Dependency Debt

Considerando che la data preparation è il primo step per iniziare a costruire un modello di Machine learning, è necessario considerare che il data dependency debt richiede una particolare attenzione per il pericolo che può causare, poichè è possibile che il technical debt impiegato durante la data preparation possa influenzare tutte le successive fasi della machine learning pipeline. Si definisce quindi il *Data Dependency Debt* quando i dati che si utilizzano all'interno del nostro modello provocano una catena di dipendenze che è difficile da manutenere. In questa particolare tipologia sono definite tre categorie:

Unstable Data Dependency

Il seguente data dependency debt è relativo ai dati o agli *input signals* che sono instabili, ovvero che cambiano il loro comportamento nel tempo. Tipico esempio di *Unstable Data Dependency* è quando i dati provengono da un altro modello oppure vengono analizzati tramite tecnica di Term Frequency/Inverse Document Frequency (tf/idf) [25] il cui score di ogni documento cambia nel tempo. Questi casi rientrano nel principio *CACE: Changing Anything Changes Everything*, ovvero ogni cambiamento, miglioramento o adattamento all'insieme di dati può portare a un qualsiasi effetto di cambiamento all'interno del modello data la dipendenza dei dati.

Underutilized Data Dependencies

Questa tipologia di data dependency debt è relativa alle *feature* degli input o ai *signal* che non portano a incrementi significativi delle performance del modello, avendo come effetto l'aumento dei dati senza significativi vantaggi e la possibilità di introdurre vulnerabilità all'interno del sistema. Possibili cause che comportano la presenza di *Underutilized Data Dependencies* all'interno del sistema sono:

- **Legacy Features:** Si consideri una feature f nell'insieme delle feature S del modello. Se viene aggiunta una successiva feature f' a S , è possibile

che f possa essere ridondante e poco utilizzata dal modello, poiché f' contiene la stessa informazione di f .

- **Bundled Features:** Questa è la situazione in cui al momento dell'aggiunta delle features all'interno del modello, esse vengono valutate nell'insieme, senza considerare una possibile analisi di ogni singola feature. Questa situazione porta che alcune delle features aggiunte non sono causa di un miglioramento delle performance del modello, ma possono risultare in uno scarso guadagno di accuracy.
- **Epsilon Features:** Una comune situazione è quella di aggiungere una feature f all'insieme delle feature che permette di avere leggeri miglioramenti alle performance del modello al costo di aumentarne la complessità.
- **Correlated Features:** Si considerino due feature f e f' inserite all'interno dell'insieme delle feature S del modello. Se le due feature hanno un grado di correlazione alto, significa che entrambe le feature trasmettono la stessa informazione, dove solo una delle due è direttamente causale all'altra. I modelli di machine learning hanno problemi a identificare quale delle due feature è da prendere in considerazione.

Correction Cascades

Questa situazione è ricorrente durante l'aggiornamento e il riutilizzo di un modello. Si consideri un modello a per il problema A . Questo modello viene riutilizzato per la costruzione di nuovo modello a' corrispondente alla soluzione del problema A' . Questa correzione, per quanto inizialmente possa apportare vantaggi e utilità alla risoluzione del problema, porta alla costruzione di un modello che dipende per intero da un altro modello. Questa situazione può rincorrere in continui peggioramenti se le correzioni avvengono in cascata, continuando quindi a generare dipendenze tra i modelli.

3.1.2 Analysis Debt

I sistemi di Machine Learning tendono a influenzare il loro stesso comportamento nel tempo. Questo aumenta la difficoltà di predirre il comportamento del sistema prima della sua esecuzione. Il fenomeno dove un modello tende a influenzare il suo stesso comportamento durante l'aggiornamento del modello nel tempo viene definito come *Feedback Loop*. I *Feedback Loops* possono assumere diverse forme, le quali ognuna presenta un alto grado di difficoltà nella identificazione data dovuta dalla loro natura irregolarità della frequenza con cui i modelli si aggiornano. In particolare sono definite due tipologie di *Feedback Loop*:

Direct Feedback Loop

Un modello può direttamente influenzare la selezione dell'ambiente da cui provengono i dati per il training. Se un modello ha un contatto interattivo con l'ambiente da cui percepisce i dati, il proprio output potrebbe essere causa della definizione dei dati. Questo comporta che i dati che il modello successivamente riceve in input per il training del modello in realtà sono dati influenzati.

Hidden Feedback Loop

Sebbene le istanze di *Direct Feedback Loop* dispongono di interazioni con l'ambiente visibili e identificabili, diverso è per la presente categoria di Technical Debt. Un caso più particolare, ovvero la presenza di *Hidden Feedback Loop* nel sistema, riguarda l'interazione indiretta tra due sistemi, i quali si influenzano indirettamente nell'ambiente.

Si consideri un algoritmo di Machine Learning (\mathbf{M}_a) che ha lo scopo di predirre i luoghi in cui si presenteranno crimini e un altro modello (\mathbf{M}_b), il quale determina dove distribuire in modo ottimale gli agenti di polizia. Il modello \mathbf{M}_b distribuisce quindi più agenti di polizia nell'area identificata dal modello \mathbf{M}_a . Grazie all'incremento del numero di agenti, vengono scoperti

più crimini. Il modello \mathbf{M}_b riceve informazioni al modello \mathbf{M}_a , ma influenza anche l’ambiente di input su cui \mathbf{M}_a opera per effettuare la predizione. Questa influenza indiretta tra i due modelli risulta in un ciclo continuo di aggiornamento automatico. Miglioramenti o modifiche a uno dei due modelli di AI porta come effetto un’alterazione del comportamento nell’altro risultando in un possibile degrado delle performance o, in casi peggiori, causando problematiche come l’overfitting.

3.1.3 System Level Antipattern

L’articolo proposto da Sculley et al. [20] definisce anche due particolari antipattern che sono presenti all’interno delle applicazioni di machine learning, a livello di sistema.

Glue Code

I professionisti che contribuiscono alla costruzione di sistemi di Machine Learning e, in particolare, gli esperti di data science tendono a definire soluzioni *general purpose* come singoli *package* indipendenti e autonomi. Quindi, al momento in cui si necessita utilizzare il package in un altro modello o sistema, si tende a inserire ingenti parti di codice che non sono necessarie allo scopo dell’applicazione, ma sono utili ad adattare queste componenti all’interno del sistema. Un’istanza di *glue code* può portare a lungo termine a un incremento eccessivo dei costi di manutenzione. La continua aggiunta di componenti all’interno di un sistema ha come conseguenza il continuo aumento di risorse necessarie per la gestione dell’applicazione, finché a giungere al blocco totale del sistema. Sculley et al. [20] hanno scoperto che nella comunità accademica solo il 5% del codice delle applicazioni comprende codice relativo alle operazioni di machine learning, mentre il restante 95% è rappresentativo di *glue code*.

Pipeline Jungles

Questa tipologia di antipattern all'interno del sistema può evolvere ad ogni aggiunta di un dato o una nuova informazione all'interno del sistema. Questa nuova informazione può richiedere di aggiungere al processo standard definito dalla pipeline ulteriori operazioni per la elaborazione e la preparazione dei dati. Tali aggiunte continue comportano a un incremento della pipeline di machine learning, ritrovandosi in un problema che richiede alti costi di identificazione e di manutenzione.

3.1.4 Configuration Debt

Questa tipologia di technical debt è relativa alla configurazione di sistemi di machine learning. La gestione e la configurazione dei cambiamenti di un modello, come ad esempio la modifica o l'aggiunta di una *feature*, possono creare situazioni in cui la configurazione diviene difficile da aggiornare. Tipici esempi che comportano al configuration debt sono:

- Una *feature* che, quando utilizzata, richiede per le attività di training il bisogno di utilizzare memoria aggiuntiva al fine di effettuare le operazioni di training del modello in modo efficiente.
- Una *feature* che non è più disponibile quindi deve essere sostituita da altre *features* del modello.
- Una *feature* che preclude l'utilizzo di altre feature a causa della definizione di vincoli semanticici.

Gli autori identificano tutte queste caratteristiche come sintomi di una configurazione difficile da modificare e che può causare svariate problematiche al sistema. Essi propongono quindi una serie di principi di buona pratica per validare il sistema di configurazione utilizzato. I principi esplicitati dagli autori sono i seguenti:

- Deve essere semplice specificare una determinata configurazione come un piccolo cambiamento della sua versione precedente.

- Deve essere difficile commettere errori manuali, omissioni o imprecisioni nella specifica.
- Deve essere facile visualizzare la differenza di configurazione tra due modelli,
- Devono essere facilmente verificabili e valutabili i parametri del modello, come il numero di *feature* utilizzate e la chiusura transitiva delle dipendenze dei dati.
- Deve essere possibile identificare impostazioni ridondanti o inutilizzate.
- Tutte le configurazioni devono sottostare a un processo di *code review* ed essere validate.

3.1.5 Common Smells

Gli autori hanno anche definito una serie di istanze di smells:

- Plain-Old-Data Type Smell: I dati sono trasformati in un tipo primitivo come *float* o *integer*. In questa situazione i dati perdono le informazioni relative alla natura del suo formato (detti anche metadati), come l'intervallo di valori ammissibili.
- Multiple Language Smell: L'utilizzo di più linguaggi può essere apparentemente una soluzione ottimale che permette l'utilizzo di diverse librerie, ma comporta come effetto collaterale l'aumento dei costi di testing e di comprensibilità delle componenti. La severità di questa particolare istanza di smell aumenta in situazioni in cui la responsabilità della componente viene trasferita ad altri esperti.
- Prototype Smell: L'utilizzo di prototipi può essere conveniente per la rappresentazione di idee, ma affidarsi a un ambiente prototipato può portare alla creazione di un sistema instabile e le attività di manutenzione possono conseguire in altissimi costi.

3.2 Data Technical Debt

I sistemi AI sono attualmente considerati data-centric, il quale a differenza dei tradizionali sistemi software, i risultati e la qualità del loro modello dipendono principalmente dai dati piuttosto che il codice [26]. E' necessario quindi comprendere le possibili minacce che possono andare a inficiare la qualità dei dati. In questo contesto, in uno studio condotto da Foidl et al [27], è stata scoperta la presenza di 36 smells che sono presenti nello stato dell'arte. I data smells sono poi categorizzati in base alla loro semantica, sintassi e livello di comprensibilità come segue (Figura 3.1):

- Believability Smells: relativo alla semantica applicata sui valori dei dati. Questa tipologia di smell indica una bassa "credibilità", dovuta spesso all'utilizzo di valori implausibili. Un esempio di istanza di questa categoria di smell è il *dummy value*, dove viene utilizzato un valore ambiguo per la rappresentazione di un valore mancante. Si consideri ad esempio l'utilizzo del valore "999" per la rappresentazione di un dato mancante. Questo valore può essere facilmente interpretato male a seconda del contesto in cui viene applicato, come nel caso in cui il valore è associato al numero telefonico, esso potrebbe essere interpretato come un numero di emergenza.
- Understandability Smells: relativo alla sintassi inappropriata o ambigua dei dati. Questa tipologia di smell può creare problemi di lettura e di elaborazione al software. Un tipo esempio è l'*encoding smell*, in cui un inappropriato tipo assegnato al valore dei dati impedisce di poter fare operazioni su di esso.
- Consistency Smells: relativo all'uso inconsistente del formato, del tipo e della rappresentazione dei dati. Un tipico esempio è l'*abbreviation inconsistency* in cui la rappresentazione del valore tramite abbreviazioni o acronimi non è utilizzato in modo consistente in tutti i valori dei dati.

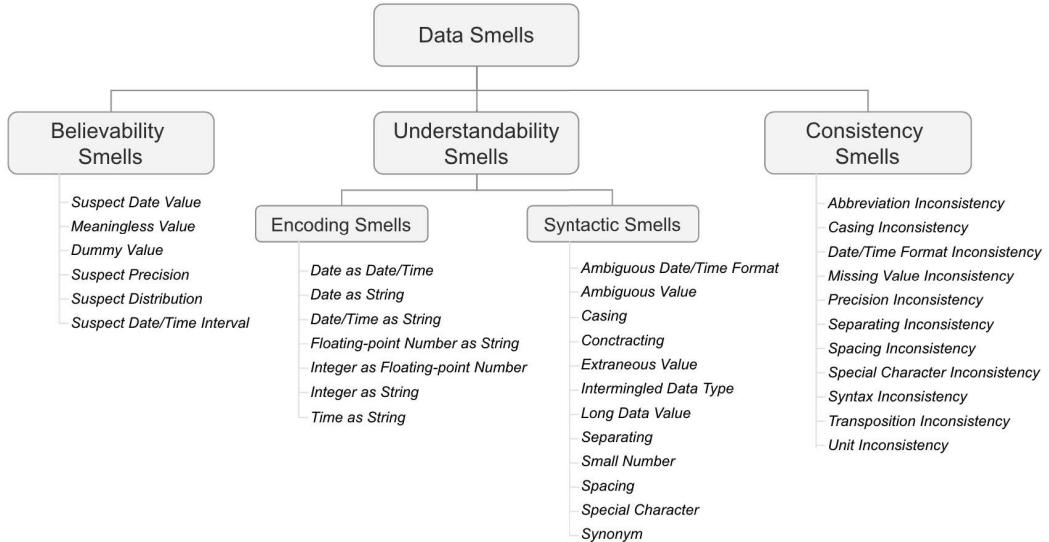


Figura 3.1: Categorie di Data smells

Inoltre, la rappresentazione inappropriate dei dati non è l'unica minaccia alla qualità del software AI. Secondo uno studio condotto da Sharma et al. [28], anche la violazione di best practice relative alla progettazione di uno schema dei dati può essere causa di introduzione di technical debt nei sistemi. Il debito accumulato da una cattiva progettazione dello schema dei dati, come accordato da Al-Barak et al. [29], può impattare negativamente non solo sulla qualità, la manutenibilità e l'evoluzione dello schema, ma può inficiare negativamente l'intero sistema.

3.3 Code Technical Debt

I sistemi di intelligenza artificiale sono composti per la maggior parte da codice di supporto alla trasformazione di un modello in un prodotto, come già definito in 2.4. Come nei sistemi tradizionali, il technical debt quindi può emergere in questi sistemi per causa dell'introduzione di cattive pratiche del software. Bogner et al. [30] hanno condotto uno studio per analizzare la diffusione dei code smell presenti nei sistemi di intelligenza artificiale. I risultati di questo studio hanno fatto emergere la presenza di *dead experimental code paths*

all'interno del codice di produzione. Questa particolare tipologia di smell prevede l'inserimento di sezioni di codice come rami condizionali al fine di poter sperimentare le nuove funzionalità. L'accumulo di codice sperimentale all'interno del sistema di produzione può creare un crescente technical debt che affligge la complessità del sistema e la sua manutenibilità.

Jebnoun et al. [31], attraverso un'analisi comparativa tra progetti di deep learning e progetti tradizionali, hanno riscontrato un'alta frequenza di code smell relativi alla complessità e la lunghezza delle espressioni, arrivando alla conclusione che il codice relativo ai modelli di deep learning includono espressioni più lunghe e più complesse.

Gesi et al. [32] hanno inoltre analizzato la diffusione e la severità di code smell all'interno dei sistemi di deep learning in Python, investigando sulla percezione degli sviluppatori. I risultati mostrano un alta frequenza dei seguenti smell:

- Scattered Use of ML Libraries: Indica l'utilizzo non sistematico delle librerie di machine learning. All'occorrenza di una modifica sulla determinata libreria, riportare gli aggiornamenti e le modifiche diventa un'operazione onerosa da dover effettuare in diversi punti del sistema.
- Unwanted Debugging Code: Indica la presenza di frammenti di codice non più utilizzati all'interno del sistema, aumentando senza alcun beneficio la dimensione e la complessità del sistema.
- Deep God File: E' la rappresentazione della God Class nei sistemi di deep learning (come descritta in sezione 2.2.1). Questa componente ingloba diverse funzionalità relative alle fasi del ciclo di vita di un modello di deep learning.
- Jumbled Model Architecture: Le parti dell'architettura del sistema di deep learning sono inglobate aumentando la difficoltà di comprensione e di manutenzione.

Queste diverse definizioni della tassonomia di technical debt per sistemi AI permettono di poter analizzare la definizione da diversi punti di vista e

con diverse granularità. Tuttavia, la tassonomia definita dai precedenti autori ha bisogno di maggiori approfondimenti, che permettono di identificare la natura e la causa del technical debt nei sistemi AI. Naturalmente, essendo le categorie stesse di techincal debt ancora soggette a definizione e raffinamento, è evidente la mancanza di tool automatici a supporto della loro identificazione all'interno di progetti AI-based. Attualmente, le analisi condotte e presenti nello stato dell'arte prevedono l'applicazione nei sistemi AI della tassonomia dei technical debt presenti nei sistemi tradizionali, senza andare nel dettaglio in possibili istanze che causano technical debt specifiche per l'intelligenza artificiale. Inoltre molti studi risultano essere complementari al lavoro di tesi, in quanto nello stato dell'arte è presente un maggiore livello di dettaglio su aspetti del sistema di intelligenza artificiale riguardante i dati e il modello. Il presente lavoro di tesi, quindi, contribuisce invece ad aumentare il livello di dettaglio di smell specifici per AI sull'architettura e il codice del sistema, con lo scopo di aggiungere alla tassonomia attuale istanze che sono specifiche e presenti esclusivamente nei sistemi intelligenza artificiale.

Capitolo 4

Metodologia

Il lavoro di tesi è stato strutturato in modo da poter condurre l'analisi della diffusione e la severità di istanze che possono causare technical debt sia nello stato dell'arte che nello stato della pratica. Per questo motivo, il seguente lavoro inizia un'analisi preliminare della lettetura attraverso la conduzione di una *Systematic Literature Review*. Successivamente, al fine di ritrovare riscontro nello stato della pratica, è stata condotta un'investigazione al fine di estrarre il punto di vista degli sviluppatori sulla percezione, diffusione e severità di istanze di AI-Technical Debt, focalizzandosi dalla definizione dello stato dell'arte sulle istanze di AI Technical Debt che interagiscono con il codice e con l'architettura.

4.1 Obiettivo di Ricerca

Per ottenere più informazioni possibili sull'identificazione di AI-Technical Debt, è necessario estendere il lavoro di ricerca il più possibile.

Lo scopo di questa tesi è quindi quello di definire una tassonomia utile ai professionisti che operano sui sistemi di Intelligenza Artificiale. Il risultato di questa tesi può essere quindi una guida utilizzabile per la identificazione di minacce che possono danneggiare la qualità del sistema basato su intelligenza artificiale.

Per raggiungere quest'obiettivo, sono state definite le seguenti research questions:

RQ1: Come è definita in letteratura la tassonomia attuale degli AI Technical debt?

RQ2: Qual è la percezione degli sviluppatori rispetto alla pericolosità dei technical debt in sistemi AI?

4.2 Systematic Literature Review

Il processo di conduzione dell'analisi della letteratura è stato condotto seguendo le linee guida definite da Kitchenham [33]:

1. Definizione del processo di ricerca
2. Identificazione della ricerca
3. Validazione della qualità degli studi selezionati
4. Estrazione e sintesi dei dati
5. Reporting della revisione

4.2.1 Processo di Ricerca

Il processo di ricerca per la conduzione dell'analisi della letteratura include la pianificazione di cinque passaggi utili alla collezione della letteratura.

Definizione dell'obiettivo di ricerca

Al fine di poter estrarre una tassonomia che possa essere di riferimento per i professionisti del software AI, l'obiettivo di ricerca dell'analisi della letteratura

include lo scopo di identificare la presenza, l'impatto e la severità di istanze che possono causare AI-Technical Debt. Inoltre, prevede di collezionare informazioni riguardanti possibili strategie di identificazione e di mitigazione della minaccia tramite tecniche di refactoring.

Definizione delle Research Questions

Dall'obiettivo definito, quindi sono state formulate le seguenti research question(RQs):

RQ1.1: Quali tipologie di Technical Debt sono presenti nei sistemi AI?

Molte tipologie di technical debt presenti in letteratura non hanno una definizione chiara e precisa che ne permetta l'identificazione diretta nella pratica. È necessario, quindi, seguire una strada analoga al lavoro introdotto da Ward Cunningham per la definizione del Technical Debt [4].

RQ1.2: Quali sono gli approcci o i tool per identificare e mitigare TD nei sistemi AI?

Al fine di fornire un metodo di riconoscimento ai professionisti del software AI delle istanze che possono causare technical debt, è necessario ricerca approcci che possano favorire l'identificazione. Questa parte del lavoro ha maggior successo se questi approcci sono stati implementati e distribuiti tramite la realizzazione di possibili tool, permettendo agli utenti della tassonomia di avere una tecnica automatizzata per l'identificazione. Allo stesso modo, al fine di poter creare una linea guida utile ai professionisti del software AI, il lavoro di tesi si pone di ricercare possibili tecniche di refactoring utili a mitigare o ridurre il technical debt nei sistemi AI.

Definizione del Coding Schema

Al fine di pianificare lo schema di informazioni utile a indirizzare l'estrazione dei dati è necessario definire un processo di coding. Il coding è il processo che comprende l'organizzazione di un grande ammontare di dati in piccoli frammenti, che favorisce l'estrazione delle informazioni. Il processo prevede quindi l'assegnazione di codici, parole chiavi o frasi che identificano a quale problema o a quale argomento si riferiscono [34].

La pianificazione del coding per il presente lavoro di tesi prevede innanzitutto la definizione di un coding schema preliminare, ossia un insieme di codici estratti dall'obiettivo di ricerca e dalle research questions formulate al fine di indirizzare la ricerca. Lo schema risultante deve poi essere verificato e raffinato attraverso l'applicazione dello schema su un sottoinsieme di articoli, così da ritrovare eventuali codici ancora non inclusi. Il coding schema pianificato è definito in Tabella 4.1.

Definizione del protocollo di revisione

La revisione della letteratura prevede quindi una serie di attività che, a partire dalle sorgenti di dati, sarà possibile estrarre la serie di articoli che sono strettamente relativi all'obiettivo di ricerca definito. Il protocollo di revisione quindi è utile alla conduzione sistematica della revisione per fornire una metodologia di selezione degli studi primari. La Figura 4.1 definisce il protocollo di revisione adoperato per condurre lo studio, basato sulle linee guida definite da Kitchenham [33].

4.2.2 Identificazione della Ricerca

In questa sezione sono definiti le tecniche e le strategie utili alla selezione e la revisione degli articoli, seguendo il protocollo di revisione.

Coding schema

Scopo o topic	Informazioni relative allo scopo su cui opera l'articolo e il suo contributo di ricerca.
Tipologia di risorsa	Conferenza, Workshop o Journal e la tipologia dell'articolo.
Lista di istanze di technical Debt	Elenco dei nomi utilizzati per rappresentare le istanze di technical debt.
Definizioni di technical debt	Per ogni istanza riscontrata, definizione di essa.
Strategia di identificazione	Informazioni sulle strategie di identificazione delle istanze di technical debt.
Strategia di refactoring	Informazioni sulle strategie di refactoring delle istanze di technical debt.
Strumenti di identificazione	Informazioni sugli strumenti di identificazione delle istanze di technical debt.
Strumenti di refactoring	Informazioni sugli strumenti di refactoring delle istanze di technical debt.
Note	Informazioni aggiuntive non raggiungibili dal coding schema.

Tabella 4.1: Coding Schema pianificato per la Systematic Literature Review

Strategia di ricerca

La strategia di ricerca include lo schema che definisce l'insieme delle sorgenti da analizzare, i termini di ricerca rilevanti, la definizione dei criteri di inclusione

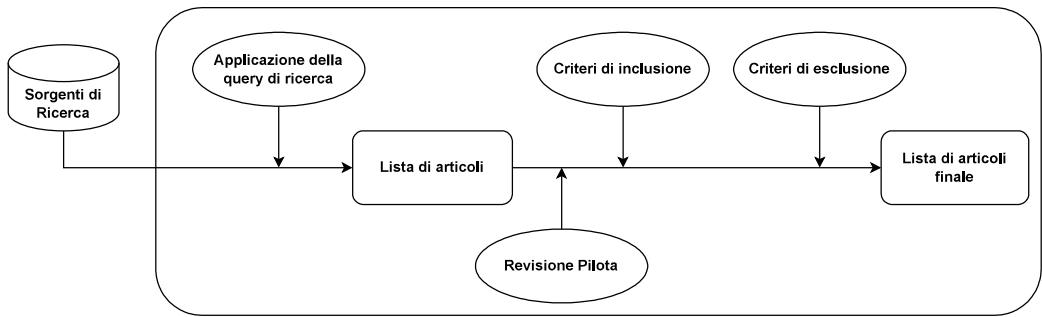


Figura 4.1: Protocollo di revisione della letteratura

e di esclusione e infine la query di ricerca utilizzata.

Per questo lavoro sono state selezionate la lista di sorgenti bibliografiche rilevanti come suggerito da Kitchenham and Charters, definite anche come le più rappresentative del dominio di ingegneria del software [35]. La lista include quindi: ACM Digital Library, IEEEExplore Digital Library, Web of Science e Scopus.

I termini di ricerca sono stati collezionati per poter rappresentare due sottogruppi di terminologie: una rappresentante le diverse tipologie di technical debt, l'altra rappresentante le diverse terminologie utilizzate per indicare l'intelligenza artificiale. La combinazione di questi due sottogruppi ha portato poi alla formulazione della seguente *search query*:

$$\begin{aligned}
 & ("technical debt" \vee "tech debt" \vee "requirements debt" \vee "data \\
 & debt" \vee "data smell*" \vee "code debt" \vee "build debt" \vee "model \\
 & debt" \vee "documentation debt" \vee "versioning debt" \vee "defect debt" \\
 & \vee "configuration debt" \vee "infrastructure debt" \vee "design debt" \vee \\
 & "architecture debt" \vee "architectural debt" \vee "test debt" \vee "defect \\
 & debt" \vee "abstraction debt" \vee "ethics debt" \vee "analysis debt" \vee \\
 & "") \vee antipattern* \vee anti-pattern* \vee smell*) \wedge ("machine learning" \\
 & \vee "deep learning" \vee ai \vee ml \vee dl \vee "artificial intelligence")
 \end{aligned}$$

E' stato utilizzato il carattere " * " per indicare la possibile variazione dei termini (*e.g.*, variazione del termine da singolare a plurale). Per incrementare

la rilevanza degli articoli riscontrati, la search query è stata utilizzata sia sull'analisi del titolo sia sull'analisi dell'abstract dell'articolo.

Definizione degli criteri di inclusione e esclusione

Sono stati definiti i criteri di inclusione da essere applicati sia durante la fase di revisione iniziale in cui viene analizzato titolo e abstract dell'articolo (T&A) sia durante la fase di revisione completa (F), come illustrato in Tabella 4.2.

4.2.3 Validazione della qualità degli studi selezionati

Al fine di migliorare la qualità della metodologia di revisione adoperata, sono state utilizzate delle tecniche di validazione. Per ridurre il fattore di soggettività all'interno del processo, è stato incluso un processo di *Inter-rater assessment*. In dettaglio, per la revisione è stata condotta da tre ricercatori, esperti di Technical Debt per i sistemi AI, al fine di valutare la search query e i criteri di inclusione e esclusione. Il processo di revisione ha previsto un'analisi iniziale condotta da ogni ricercatore analizzando, tramite l'utilizzo dei criteri di inclusione ed esclusione, il titolo e l'abstract di ogni articolo. A ogni articolo, quindi, viene assegnata la revisione da parte di due ricercatori, i quali dovranno esprimere il loro giudizio per l'inclusione (assegnato tramite il valore 1) o l'esclusione (assegnato tramite il valore 0). In caso di disaccordo tra le valutazioni, è stato coinvolto il terzo ricercatore al fine di esprimere il voto finale. Successivamente, è stato condotto un test pilota su un sottoinsieme di 6 articoli, revisionato da coppie di ricercatori. Questo ultimo step ha permesso quindi di poter valutare la bontà del processo definito e di poter ottimizzare e raffinare i dettagli.

Criteria	Criterio di valutazione	Fase
Inclusion	(I1) Articoli che definiscono le tipologie di technical Debt nei sistemi AI	Entrambi
	(I2) Articoli che presentano approcci alla identificazione di technical Debt nei sistemi AI	Entrambi
	(I3) Articoli che presentano approcci alla mitigazione di technical debt nei sistemi AI	Entrambi
	(I4) Articoli che presentano tool per la identificazione di technical debt nei sistemi AI	Entrambi
	(I5) Articoli che presentano tool per la mitigazione di technical debt nei sistemi AI	Entrambi
Exclusion	(E1) Articoli non scritti interamente in lingua inglese	T&A
	(E2) Articoli che non sono stati soggetti a una peer review (i.e., blog, forum, book, book chapter, ...)	T&A
	(E3) Articoli duplicati (da considerare solo la versione più recente)	T&A
	(E4) Position papers e work plans (i.e., articoli che non riportano risultati)	T&A
	(E5) Pubblicazioni non accessibili	T&A

Tabella 4.2: Criteri di inclusione e esclusione degli articoli in letteratura

4.3 Survey

4.3.1 Motivazione del survey

Lo stato dell'arte attualmente non presenta un numero alto di risultati tale da poter estrarre una tassonomia completa dei possibili technical debt nei sistemi AI. E' necessario quindi estendere i confini delle informazioni estraibili allo stato della pratica. L'utilizzo di un survey consente di poter estrarre il punto di vista e l'esperienza di esperti del settore di sistemi di intelligenza artificiale. La loro esperienza sarà quindi utile per due principali motivazioni: ogni informazione estratta dal punto di vista degli esperti può fornire supporto a dettagliare la catalogazione e la tassonomia di AI Technical Debt; inoltre è possibile ricavare ulteriori informazioni riguardanti le necessità dell'esperto di sistemi AI al fine di poter migliorare la qualità dei propri sistemi. Le necessità evidenziate possono diventare quindi un punto di partenza per indirizzare la ricerca accademica verso soluzioni e strategie che rispondono alle esigenze dello stato della pratica.

4.3.2 Obiettivo di Ricerca

Al fine di poter contribuire a definire una tassonomia e una catalogazione ben dettagliata delle possibili istanze che possono causare AI Technical Debt, il lavoro di tesi si focalizza alla investigazione di due settori dettagliati di AI-Technical Debt: code debt e architectural debt.

L'obiettivo principale di ricerca per questo lavoro quindi è il seguente:

Analizzare la diffusione, la severità e l'impatto di AI Technical Debt relativi al codice e all'architettura dei sistemi AI.

Per l'analisi di Ai Technical Debt relativi al codice e all'architettura sono stati estratti dalla letteratura le seguenti 9 istanze che possono causare Technical Debt, come raffigurato in Tabella 4.3.

Debt type	Debt Instance
Architectural debt	Jumbled Model Architecture Pipeline Jungles Multiple Language Smells Undeclared Consumers Correction Cascades
Code debt	Glue Code Deep God File Scattered Use of ML Libraries Unwanted Debugging Code

Tabella 4.3: List of Code and Architectural Smell analyzed in this study

4.3.3 Research Questions

Le domande di ricerca volte al raggiungimento dell’obiettivo sono definite seguendo la metodologia GQM [36], utilizzato particolarmente nel campo di misurazione della qualità del software. Questo è un metodo orientato agli obiettivi che utilizza un approccio top-down. La raffinazione dell’obiettivo viene quindi effettuata attraverso la formulazione delle domande per poi essere trasformate in metriche. L’applicazione di questa metodologia nella ricerca corrisponde ad associare quindi le domande alle research questions formulate e le metriche associate al questionario [37].

Quindi, le research questions definite sono le seguenti:

RQ2.1: Quali sono le tipologie di code debt e architectural debt più frequenti secondo la percezione degli sviluppatori AI?

RQ2.2: Quali sono le tipologie di code debt e architectural debt più problematiche secondo la percezione degli sviluppatori AI?

RQ2.3: Qual è l'impatto causato dai code debt e architectural debt secondo la percezione degli sviluppatori AI?

RQ2.4: Quali sono le strategie utilizzate dagli sviluppatori AI per l'identificazione e la mitigazione di code debt e architectural debt?

4.3.4 Design del Survey

La conduzione del presente studio ha previsto lo svolgimento di differenti fasi, al fine di massimizzare l'affidabilità delle risposte ricevute.

4.3.5 Design e configurazione dell'esperimento

Lo studio prevede di analizzare in dettaglio il punto di vista degli sviluppatori sulla presenza, la severità e l'impatto di 9 istanze che possono causare AI Technical Debt. Al fine di poter aumentare il livello di comprensibilità da parte degli sviluppatori, il tipo di esperimento condotto è basato sulla metodologia di *Experimental Vignette Study* introdotta da Atzmuller et al. [38]. Questa tipologia di esperimento prevede la rappresentazione degli oggetti sperimentali tramite vignetta. Una vignetta è una breve e dettagliata descrizione di una persona, oggetto o situazione che rappresenti una combinazione di caratteristiche. La scelta di quest'approccio deriva dalla sua capacità tramite descrizione di rappresentare scenari realistici, dando la possibilità ai partecipanti di comprendere i dettagli. Le definizioni fornite dalla letteratura per la rappresentazione delle 9 istanze di AI Technical Debt sono state quindi utilizzate per la creazione delle vignette, come raffigurato in Tabella 4.4 e in Tabella 4.5.

Per la progettazione dell'esperimento, al fine di ottenere un numero sufficiente di risposte da ogni partecipante senza oltrepassare i limiti del numero di domande ammissibili, è stato deciso di adoperare una strategia di tipo

AiTechnicalDebt	Vignetta
Glue Code	Suppose that you are working on a machine learning application code. During the development, you notice inside the code a popular general-purpose library that contains a lot of different models. You have seen that the whole library is imported inside the system and the application contains a big part of code that allows you to fit data from the library to the specific model that you have.
Deep God File	Suppose that you are working on a machine learning application. You try to identify and highlight the code of specific functionality of the application. You notice that the entire application is encapsulated in a single file that contains all the operations of the application. So, you attempt to highlight the part of the code that you need but it's very hard to perceive it.
Scattered Use of ML Libraries	Suppose that you are working on a machine learning application. You see in your application that a part of the team introduced many libraries for the steps of the pipeline, without a cohesive manner.
Unwanted Debugging Code	Suppose that you are working on the maintenance of a machine learning application. You see in your application that a part of the team introduced some debugging code, for example, an intermediate sequence of loggers.

Tabella 4.4: Rappresentazione delle istanze di AI Technical Debt di codice attraverso la metodologia della vignetta

mixed-design [38]. Le diverse istanze di Ai Technical Debt sono state suddivise in tre gruppi: A,B e C.

- Gruppo A: Glue Code, Multiple Language Smell, Undeclared Consumers.
- Gruppo B: Pipeline Jungles, Correction Cascades, Scattered Use of ML Libraries.

AI TechnicalDebt	Vignetta
Jumbled Model Architecture	Suppose that you are working on a machine learning application. After seeing the machine learning pipeline, you try to identify and highlight the code that reflects the step of the machine learning pipeline. You notice that all the stages of the machine learning pipeline are all together and it's difficult to recognize each stage.
Pipeline Jungles	Suppose that you are working on a machine learning pipeline application. During the pipeline definition of the model, you see that each part of the team adds several steps for each phase of the pipeline, and you notice that the entire pipeline starts growing more.
Multiple Language Smells	Suppose that you are working on the machine learning application code. During the development, you notice that part of your team introduced a library that allows you to resolve an important task of the Machine Learning Pipeline. On the other side, you see that this component is written in a different language from the rest of the application.
Undeclared Consumers	Suppose that you are working on the maintenance and updating of a machine learning model. In order to modify the model, you want to know what applications use the output data of your model. You discover through the access control that there are some unknown applications that collect the data from the model.
Correction Cascades	Suppose that you are working on a new machine learning application and your team wants to use a built model X to resolve a slightly different problem. So, your team created a model Y that takes X's output in input and makes a hotfix in order to fit the solution of the new problem.

Tabella 4.5: Rappresentazione degli AI-Technical Debt architetturali attraverso la metodologia della vignetta

- Gruppo C: Jumbled Model Architecture, Unwanted Debugging Code, Deep God File.

Prescreening

E' stato elaborato un questionario iniziale per raccogliere le informazioni sulle competenze dei partecipanti, al fine di garantire che il partecipante possa essere considerato un esperto del contesto e le sue risposte possano avere una validità. In particolare, il questionario preliminare di *prescreening* è suddiviso in due sezioni: una sezione che contiene domande sulle informazioni professionali e sulla sua esperienza negli ambiti relativi al contesto; la sezione successiva relativa all'esperienza del partecipante sui sistemi di intelligenza artificiale, cercando di estrarre su quale parte del sistema il partecipante ha maggiore focus.

La prima sezione quindi, richiede l'inserimento dal partecipante di informazioni relative alla dimensione dell'azienda e il suo ruolo nell'azienda, per poi approfondire le conoscenze nei campi della programmazione, ingegneria del software e intelligenza artificiale. Le domande relative alle conoscenze saranno poi analizzate attraverso l'utilizzo di una scala likert per poter rappresentare il livello di competenza, come raffigurato in Tabella 4.6. Nella seconda

ID	Domanda	Tipologia di Domanda
PD1	Qual'è il tuo ruolo nella compagnia?	Caselle di controllo
PD2	Quanti anni di esperienza hai nel tuo ruolo?	Caselle di controllo
PD3	Definisci il tuo livello di conoscenza di programmazione?	Scala Likert
PD4	Definisci il tuo livello di conoscenza di ingegneria del software?	Scala Likert
PD5	Definisci il tuo livello di conoscenza di intelligenza artificiale?	Scala Likert

Tabella 4.6: Domande sulle informazioni professionali del partecipante

sezione, dopo aver fornito un'illustrazione raffigurante una tipica pipeline di un'applicazione AI, è stato chiesto al partecipante una serie di domande

con lo scopo di comprendere quale fosse la parte del sistema dove egli ha maggiore interazione. Inoltre, in questa sezione è stata aggiunta una domanda di controllo dell'attenzione al fine di controllare la consistenza delle risposte e verificare che il partecipante abbia risposto con interesse, come raffigurato in Tabella 4.7. In particolare, l'ultima domanda controlla la conoscenza del partecipante dei sistemi AI, richiedendo di indicare in quale fase si applicano le tecniche di bilanciamento.

ID	Domanda	Tipo Domanda
In quali di queste attività hai esperienza?		
PD6	Data ingestion, data aggregation o altre pratiche di data preparation	Scala Likert
PD7	Model selection e model training	Scala Likert
PD8	Model validation, review process e improving model	Scala Likert
PD9	Monitoring and maintenance and deployment plan	Scala Likert
PD10	Utilizzare librerie di Machine Learning e esecuzione del modello	Scala Likert
PD11	Quale di queste attività raffigura meglio le tue attività di lavoro?	Scelta Multipla
PD12	In quale parte del dataset applicheresti la tecniche di bilanciamento dei dati?	Caselle di Controllo*

* = Domanda per il controllo dell'attenzione

Tabella 4.7: Domande sulle conoscenze dei sistemi di Intelligenza Artificiale del partecipante

La formulazione di queste domande ha lo scopo di poter andare a selezionare i partecipanti che realmente hanno esperienza sul campo dei sistemi AI. Quindi, la selezione di questi partecipanti viene effettuata in base a una serie di criteri:

- Il partecipante deve aver risposto correttamente alla domanda PD12.
- Il partecipante deve avere una conoscenza di programmazione (PD3) almeno pari a Basic (2/5).
- Il partecipante deve avere una conoscenza di ingegneria del software (PD4) o intelligenza artificiale (PD5) almeno pari a Intermediate (3/5).
- Se il partecipante rispetta il requisito precedente, il partecipante deve avere nell'altra disciplina una conoscenza almeno pari a Basic (2/5).
- Il partecipante deve avere conoscenza di monitoraggio, manutenzione e sviluppo del modello (PD9) almeno pari a Intermediate (3/5) oppure una conoscenza di utilizzo di librerie di machine learning (PD10) almeno pari a Intermediate (3/5).

Domande del questionario

Il risultato della fase di progettazione ha previsto di inserire sezioni su diversi aspetti per completare le informazioni sulla conoscenza e l'esperienza del partecipante. In particolare, sono state definite due tipologie di sezioni: la prima intende completare l'estrazione delle informazioni riguardanti le attività professionali del partecipante già definite nel questionario di prescreening. Per questo motivo è stato deciso di richiedere al partecipante informazioni riguardanti la dimensione del team e la dimensione dell'organizzazione in cui esibisce la professione, come raffigurato in Tabella 4.8.

Successivamente, sono state progettate la serie di domande in base agli obiettivi di ricerca definiti. In particolare, per ogni istanza di AI-Technical Debt definito, è stato richiesto di fornire informazioni riguardanti ogni domanda di ricerca definita, ovvero attraverso la seguente categorizzazione:

- Domande riguardanti la frequenza e la severità di AI-Technical Debt.
- Domande riguardanti l'impatto di AI-Technical Debt.

ID	Domanda	Tipologia di Domanda
S1D1	Da quanti membri è composto il tuo team?	Caselle di controllo
S1D2	Da quanti membri è composta la tua organizzazione?	Caselle di controllo

Tabella 4.8: Domande aggiuntive sulle informazioni professionali del partecipante

- Domande riguardanti le possibili strategie e tool di identificazione e refactoring utilizzati per rilevare AI-Technical Debt.

Per poter rispondere alla RQ_1 e alla RQ_2 , le domande progettate relative alla frequenza di AI Technical Debt e la loro severità sono 5, come raffigurato in Tabella 4.9. Le domande chiedono esplicitamente al partecipante se la possibile situazione analizzata tramite vignetta, può risultare problematica all'interno di un sistema AI e se, nella loro esperienza, si sono ritrovati in una situazione simile. Successivamente, al fine di avere informazioni aggiuntive sulla severità della situazione descritta, viene chiesto ai partecipanti l'ammontare di sforzo necessario al fine di poter identificare la situazione e effettuare operazioni di refactoring.

Per rispondere alla RQ_3 , le domande relative all'impatto sono basate su tre importanti aspetti di qualità:

- Impatto sulle altre componenti(I)
- Comprensibilità (C)
- Evoluzione (E)
- Performance (P)
- Accoppiamento (A)
- Manutenibilità (M)

ID	Domanda	Tipologia di Domanda
S2D6	Quanto spesso ritrovi questa situazione all'interno del progetto?	Scala Likert
S2D1	Quanto trovi problematica questa situazione?	Scala Likert
S2D2	Perchè può essere (o può non essere) problematica?	Risposta testuale
S2D5	Quanto sforzo pensi sia necessario al fine di identificare questa situazione?	Scala Likert
S2D7	Quanto sforzo pensi sia necessario al fine di effettuare refactoring, mitigazione o miglioramenti per questa situazione?	Scala Likert

Tabella 4.9: Domande aggiuntive sulle informazioni professionali del partecipante

Dalla definizione di questi 3 aspetti di qualità, sono state formulate le seguenti domande:

Per poter rispondere alla RQ_4 infine, ovvero trovare le possibili strategie e tool utilizzati nello stato della pratica al fine di effettuare identificazione e refactoring, il questionario prevede l'inclusione di una domanda sulla modalità di approccio all'identificazione per indirizzare il partecipante a rispondere, e poi maggiori approfondimenti, come descritto nella Tabella 4.11.

4.3.6 Validazione del Survey

Al fine di poter analizzare e evitare la presenza di potenziali minacce alla validità dell'esperimento, è stato deciso di realizzare una simulazione pilota di partecipanti esperti del settore. E' stata quindi inclusa la partecipazione di 6 ricercatori e esperti dell'Università degli studi di Salerno al fine di valutare la bontà del questionario. I risultati hanno migliorato la qualità descrittiva

ID	Domanda	Tipologia di Domanda
Penso che questa situazione possa ...		
S2D9	... danneggiare le altre componenti dell'applicazione. (I)	Scala Likert
S2D10	... rendere il livello di interpretabilità e comprensione più complesso per il modello. (C)	Scala Likert
S2D11	... rendere difficile l'evoluzione dell'applicazione. (E)	Scala Likert
S2D12	... impattare e decrementare le performance del modello. (P)	Scala Likert
S2D13	... incrementare il numero di dipendenze e aumentare l'accoppiamento. (A)	Scala Likert
S2D14	... incrementare i costi e lo sforzo di manutenzione dell'applicazione. (M)	Scala Likert

Tabella 4.10: Domande aggiuntive sulle informazioni professionali del partecipante

delle domande e, in particolare, la qualità descrittiva delle vignette utilizzate per la rappresentazione delle istanze di AI-Technical Debt.

4.3.7 Reclutamento dei partecipanti

Lo studio è mirato all'acquisizione del punto di vista di figure professionali che hanno esperienza con i sistemi di intelligenza artificiale. Per questa motivazione, il questionario è stato indirizzato verso i professionisti che hanno esperienza di ingegneria del software e di intelligenza artificiale nel campo aziendale.

ID	Domanda	Tipologia di Domanda
S2D3	Quali sono le tue pratiche utili a identificare la situazione descritta?	Caselle di Controllo
S2D4	Puoi specificare con maggiore dettaglio le tue pratiche di identificazione?	Risposta testuale
S2D8	Come effettui le pratiche di refactoring per la situazione descritta?	Risposta testuale

Tabella 4.11: Domande sulle strategie di identificazione e refactoring utilizzate dal partecipante

Il possibile rischio affrontato durante la fase di reclutamento dei partecipanti è stata la possibilità di non ottenere un numero sufficienti di risposte utili a raggiungere i nostri obiettivi di ricerca. Il reclutamento dei partecipanti è stato quindi effettuato tramite la piattaforma *Prolific*, la quale permette di indirizzare il questionario proposto verso i professionisti del settore desiderato, attraverso l'utilizzo di filtri sul profilo. In particolare, nel nostro esperimento sono stati reclutati per rispondere alle domande di prescreening 500 partecipanti, i quali rispettassero i seguenti criteri iniziali:

- Esperienza in programmazione.
- Conoscenza delle tecniche di sviluppo software (*e.g.*, Monitoraggio o testing).

Inoltre, il processo di prescreening descritto precedentemente aiuterà ad avere una ulteriore selezione sui partecipanti che hanno l'esperienza più consona al raggiungimento dell'obiettivo di ricerca.

Dall'insieme dei partecipanti risultanti idonei dopo il processo di prescreening, lo studio prevede di condurre l'investigazione tramite survey suddividendo il nuovo insieme di partecipanti nei tre gruppi definiti per le istanze di AI Technical Debt.

In dettaglio, a ogni partecipante è stato assegnato una combinazione di due gruppi di istanze al fine di aumentare il numero di risposte, come raffigurato in Tabella 4.12.

Partecipanti	Istanze di Ai Technical Debt
Gruppo 1	Gruppo A+B
Gruppo 2	Gruppo B+C
Gruppo 3	Gruppo A+C

Tabella 4.12: Suddivisione dei partecipanti per gruppi di Ai-Technical Debt

I partecipanti saranno ricontattati quindi tramite la piattaforma *Prolific* al fine di rispondere e sottomettere il nuovo questionario.

Capitolo 5

Analisi e Risultati

5.1 Tassonomia definita in letteratura

5.1.1 Risultati Preliminari

Lo studio ha riportato dalla collezione per ogni sorgente un’insieme iniziale di 156 articoli, di cui:

- 89 articoli provenienti da *Scopus*,
- 1 articolo proveniente da *IEEEExplore Digital Library*,
- 21 articoli provenienti da *ACM Digital Library*,
- 45 articoli provenienti da *Web of Science*.

Dalla collezione poi è stata effettuata una rimozione degli articoli duplicati, per giungere a un totale di 80 articoli. In seguito al processo di revisione manuale di titolo e abstract per ogni articolo tramite l’utilizzo dei criteri di inclusione e di esclusione, 66 articoli sono stati esclusi dai revisori. In particolare, durante l’analisi preliminare condotta su titolo e abstract dell’articolo, 9 articoli su 80 sono stati accettati da entrambi i revisori. Il numero di articoli che hanno ricevuto una valutazione in conflitto sono 8. I revisori che hanno svolto il ruolo di terzo revisore hanno accettato 5 articoli su 8, avendo come

risultato dell’analisi preliminare un totale di 14 articoli. Inoltre, i risultati del processo di revisione manuale completo hanno portato all’esclusione di altri 8 articoli di ricerca. Quindi per il primo ciclo di analisi sono stati ritrovati infine 6 articoli di ricerca. Per estendere i risultati è stato deciso di applicare la tecnica di *backward snowballing* [39]. Questa tecnica permettere di utilizzare la lista di referenze di ogni articolo per identificare nuovi articoli da includere. Il primo passaggio prevede quindi, attraverso l’analisi della lista delle referenze, l’esclusione degli articoli tramite l’utilizzo dei criteri di esclusione. Successivamente, si escludono i paper che sono stati già esaminati durante il processo di review. Infine, si riattiva il processo di review per gli articoli che non sono stati esclusi. Attraverso l’utilizzo dello snowballing, altri 2 articoli sono stati accettati dal processo di review e inclusi. Il processo di revisione della letteratura, quindi, ha come risultato l’estrazione di 8 articoli.

La distribuzione dei paper selezionati sugli anni di pubblicazione è mostrata in Figura 5.1. Sulla asse delle X è rappresentata la serie degli anni di pubblicazione, mentre sulla asse delle Y è rappresentato il numero di articoli per ogni anno. Dal grafico è possibile notare che a partire da una pubblicazione iniziale risalente al 2015, l’interesse da parte dei ricercatori è aumentato negli ultimi anni, ritrovando 7 articoli di ricerca pubblicati negli ultimi 4 anni.

Tra gli articoli selezionati in questa ricerca ritroviamo 7 articoli che sono che sono stati pubblicati a conferenze internazionali e 1 articolo pubblicato a un workshop. In particolare, tra gli articoli pubblicati alle conferenze ritroviamo 1 articolo pubblicato alla ”International Conference on Product-Focused Software Process Improvement” (PROFES), 1 articolo pubblicato alla ”International Conference on Technical Debt” (TechDebt) e 2 articoli pubblicati alla ”International Conference of Software Engineering” (ICSE). Questi articoli sono stati pubblicati tutti in conferenze relative all’ingegneria del software, dimostrando quindi che la ricerca relativa all’investigazione di AI Technical Debt ha generato significativo interesse nella comunità di ricerca di ingegneria del software. La lista degli articoli analizzati per questa Systematic Literature review è disponibile nel replication package in [40].

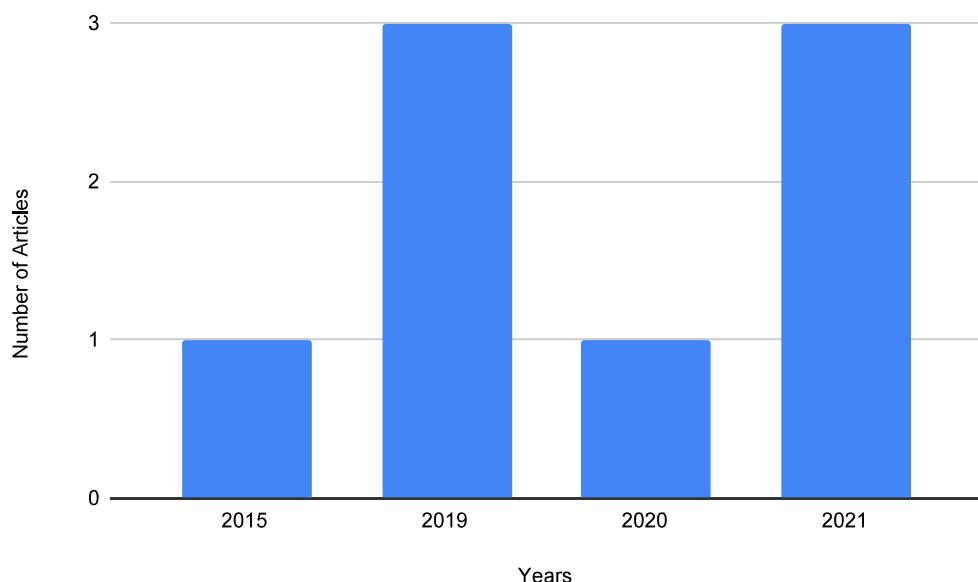


Figura 5.1: Distribuzione degli articoli per anno di pubblicazione

5.1.2 RQ1.1: Quali tipologie di AI Technical Debt sono presenti in letteratura?

La Figura 5.2 mostra la distribuzione di referenze delle tipologie di AI Technical Debt nella letteratura. E' possibile notare come *Data Debt* sia estremamente di riferimento per gli articoli che trattano pratiche di ingegneria del software per i sistemi di intelligenza artificiale, in quanto 5 articoli su 8 trattano questa tipologia di AI Technical Debt. Successivamente, con un alto tasso di riferimento, ritroviamo *Configuration Debt*, *Code Debt* e *Architectural Debt*.

In particolare, la Figura 5.3 mostra la distribuzione di referenze nella letteratura per ogni articolo di ricerca. L'istanza di AI Technical Debt ritrovata maggiormente in letteratura è l'istanza di *Pipeline Jungle*, ritrovando 4 articoli su 8 in letteratura che trattano dei danni che può apportare al sistema AI un incremento smisurato e incontrollato della pipeline AI. Successivamente, molte istanze di AI Technical Debt sono presenti. Unstable Data

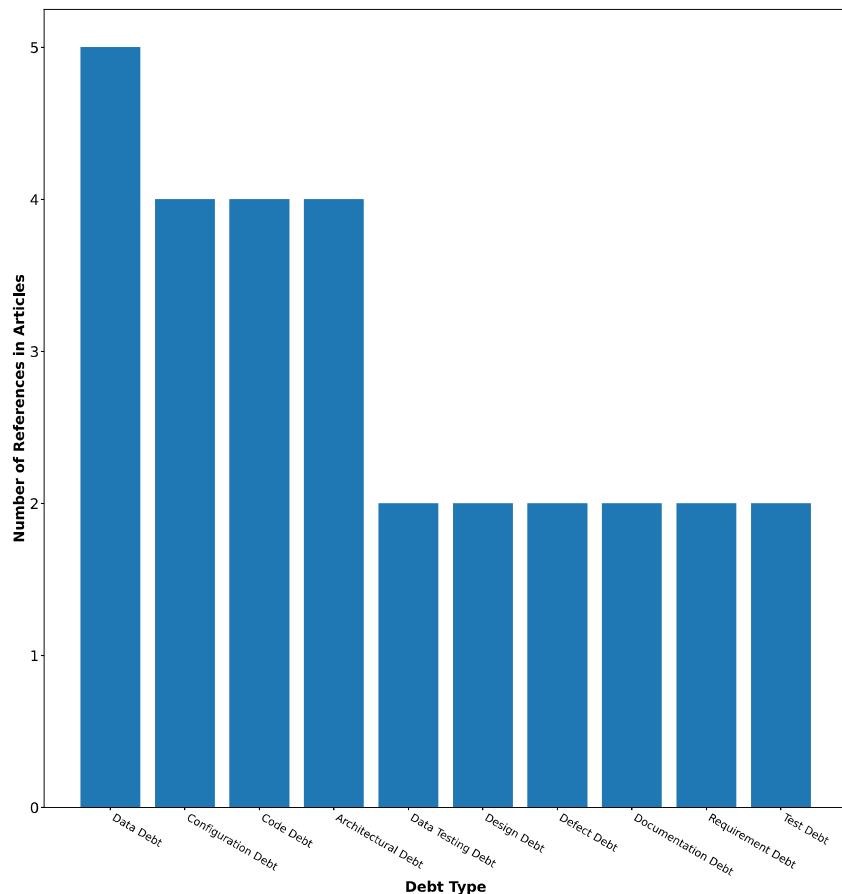


Figura 5.2: Tipologie di AI Technical Debt riscontrate in letteratura

Dependencies, Bundled Features, Correlated Features, Epsilon Features e Feature Entanglement sono le istanze di AI Data Debt che sono state ritrovate, dimostrando l'interesse attuale della ricerca nel trovare metodologie e tecniche utili a preservare la qualità dei dati.

Analizzando le istanze e le tipologie ritrovate, è possibile conseguire che c'è un'alto interesse della ricerca nell'investire lo sforzo al fine di migliorare le metodologie attuali per preservare la qualità e l'elaborazione dei dati. Tuttavia, la ricerca ritiene di alto interesse anche gli aspetti architetturali e del codice

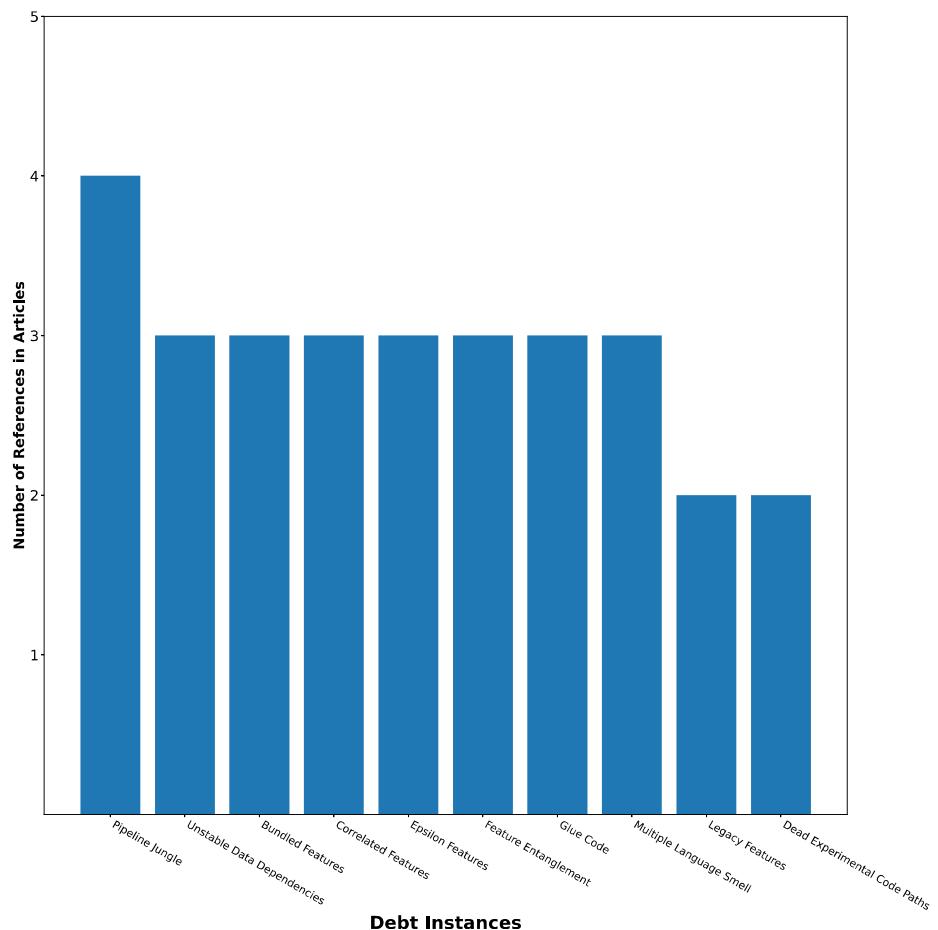


Figura 5.3: Istanze di AI Technical Debt riscontrate in letteratura

del sistema AI, riscontrando un alto tasso di interesse rispettivamente per *Pipeline Jungles*, *Glue Code*, e *Multiple Language Smell*.

5.1.3 RQ1.2: Quali sono gli approcci di identificazione e mitigazione di AI Technical Debt?

Sebbene esiste una definizione per le diverse tipologie e istanze di AI Technical Debt, diverso è per le loro tecniche di identificazione e mitigazione. Possibili

spiegazioni estratte dagli articoli in letteratura sono :

- Il codice relativo al modello contiene la minor quantità di codice di tutto il sistema [S01].
- I professionisti di data science potenzialmente concentrano i loro sforzi al fine di massimizzare le performance del modello, piuttosto che la manutenzione e l'evoluzione del codice AI [S02].

Il nostro studio ha riportato come output una singola tecnica di identificazione per la presenza di *Epsilon Feature* all'interno del sistema AI. La tecnica estratta dalla letteratura è nominata *Leave-One-out Feature Evaluation*. Si consideri un insieme di feature $F = (f_1, f_2, \dots, f_n)$ dove con f_i si rappresenta la feature nella struttura dei dati in posizione i . Si estrae e si calcola l'insieme $(F - f_i)$ dove consideriamo l'insieme delle feature rappresentate sottratto da una particolare feature. Vengono costruiti quindi due modelli M e M_i sui due insieme definiti. Se la differenza delle performance risultanti dai due modelli non è significativa, significa che il modello M conteneva una particolare istanza di *Epsilon Feature* rappresentata da f_i . Il processo viene poi eseguito ciclicamente per ogni f_i appartenente a F .

 **Stato dell'arte** L'interesse della ricerca è focalizzato sullo studio delle istanze di AI Technical Debt relative ai data (data debt), in particolare sulla diffusione e sulla severità di istanze relative alla presenza di dipendenze instabili tra i dati. Inoltre, un consistente sforzo della ricerca è impunitato verso le istanze di AI Technical Debt incentrate sul codice (code debt), sulla configurazione (configuration debt) e sull'architettura (architectural debt).

5.2 Revisione del punto di vista degli sviluppatori

5.2.1 Risultati Preliminari

Lo studio preliminare è stato condotto su 500 partecipanti.

Il numero di partecipanti che ha risposto correttamente alla domanda filtro è 188, rappresentando il 37,8% della popolazione, come raffigurato in Figura 5.4.

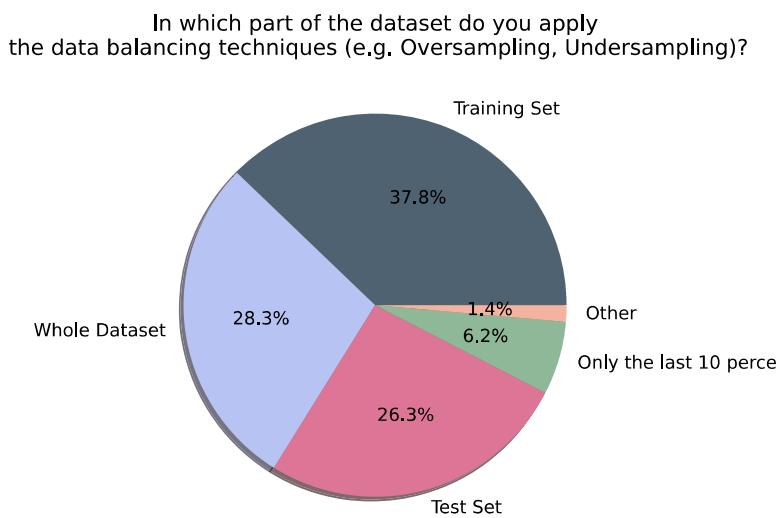


Figura 5.4: Risposte alla domanda filtro del questionario di preselezione

Infine, applicando i filtri sulle competenze generali del settore e sulle competenze specifiche di intelligenza artificiale, sono stati esclusi 75 partecipanti. Il numero di partecipanti che è stato accettato per condurre l'esperimento è 113. La suddivisione dei partecipanti nei tre gruppi sperimentali è stata riportata in Tabella 5.1.

Tuttavia non tutti i partecipanti hanno deciso di continuare l'esperimento, abbandonando lo studio durante la conduzione dell'esperimento.

Partecipanti	Istanze di Ai Technical Debt	Numero
Gruppo 1	Gruppo A+B	37
Gruppo 2	Gruppo B+C	38
Gruppo 3	Gruppo A+C	38

Tabella 5.1: Suddivisione dei partecipanti per gruppi di Ai Technical Debt

Il numero di partecipanti invece a cui è stato sottoposto il questionario finale con successo è 54.

Da ogni risposta sottomessa da ogni partecipante sono state estratte poi i due gruppi sperimentali di riferimento ed è stato possibile quindi ottenere, per ogni gruppo di istanza di AI Technical Debt ottenere il numero di risposte riportato in Tabella 5.2.

Gruppi	Istanze AITD	Numero di Risposte
A	Glue Code, Multiple Language Smell, Undeclared Consumers	38
B	Pipeline Jungles, Correction Cascades, Scattered Use of ML Libraries	36
C	Jumbled Model Architecture, Unwanted Debugging Code, Deep God File	34

Tabella 5.2: Numero di risposte per ogni gruppo sperimentale di AI Technical Debt

La Figura 5.5 e la Figura 5.6 mostrano la distribuzione del livello di competenza dei partecipanti su aspetti generali della programmazione, software engineering e intelligenza artificiale, per poi approfondire sulle pratiche di intelligenza artificiale. In particolare, è possibile notare come i partecipanti in media hanno una discreta conoscenza sulle tre discipline generali definite, riportando nella distribuzione una leggera variazione della mediana, dove le mediane con il valore più alto sono rispettivamente sulla distribuzione delle competenze di programmazione e sulla distribuzione delle competenze di

intelligenza artificiale. Questo dato riporta come informazione che il livello di competenza dei partecipanti, anche se leggermente, è maggiormente orientato sulle conoscenze di intelligenza artificiale che quelle di ingegneria del software. Situazione analoga per le competenze relative alle pratiche di intelligenza artificiale, dove ritroviamo un leggero aumento della mediana a 4/5 per le attività di data preparation e utilizzo del codice delle librerie di Machine learning.

5.2.2 RQ2.1: Quali sono le tipologie di code debt e architectural debt più frequenti secondo la percezione degli sviluppatori AI?

Secondo le analisi effettuate, i risultati si trovano in accordo con quanto ritrovato nella letteratura. Considerando gli AI-Technical Debt che sono relativi all’architettura e al codice del sistema, *Pipeline Jungle* e *Glue Code* sono i più diffusi secondo la percezione degli sviluppatori. La Tabella 5.3 rappresenta la distribuzione dei valori del livello di frequenza che gli sviluppatori hanno dato per ogni istanza di AI Technical Debt. In totale, solo 4 particolari istanze di AI Technical Debt vengono percepite da parte degli sviluppatori nel poter avere un’alta frequenza: *Pipeline Jungle*, *Glue Code*, *Jumbled Model Architecture*, *Unwanted Debugging Code*. Tuttavia, la maggior parte degli sviluppatori ritiene un discreto tasso di frequenza solo per due delle istanze definite: *Pipeline Jungle* e *Glue Code*.

5.2.3 RQ2.2: Quali sono le tipologie di code debt e architectural debt più problematiche secondo la percezione degli sviluppatori AI?

Gli sviluppatori hanno riportato tre particolari istanze di AI Technical Debt come problematiche all’interno del sistema. *Undeclared Consumers*, la quale presenta una moda di severità pari al massimo nella distribuzione dei dati,

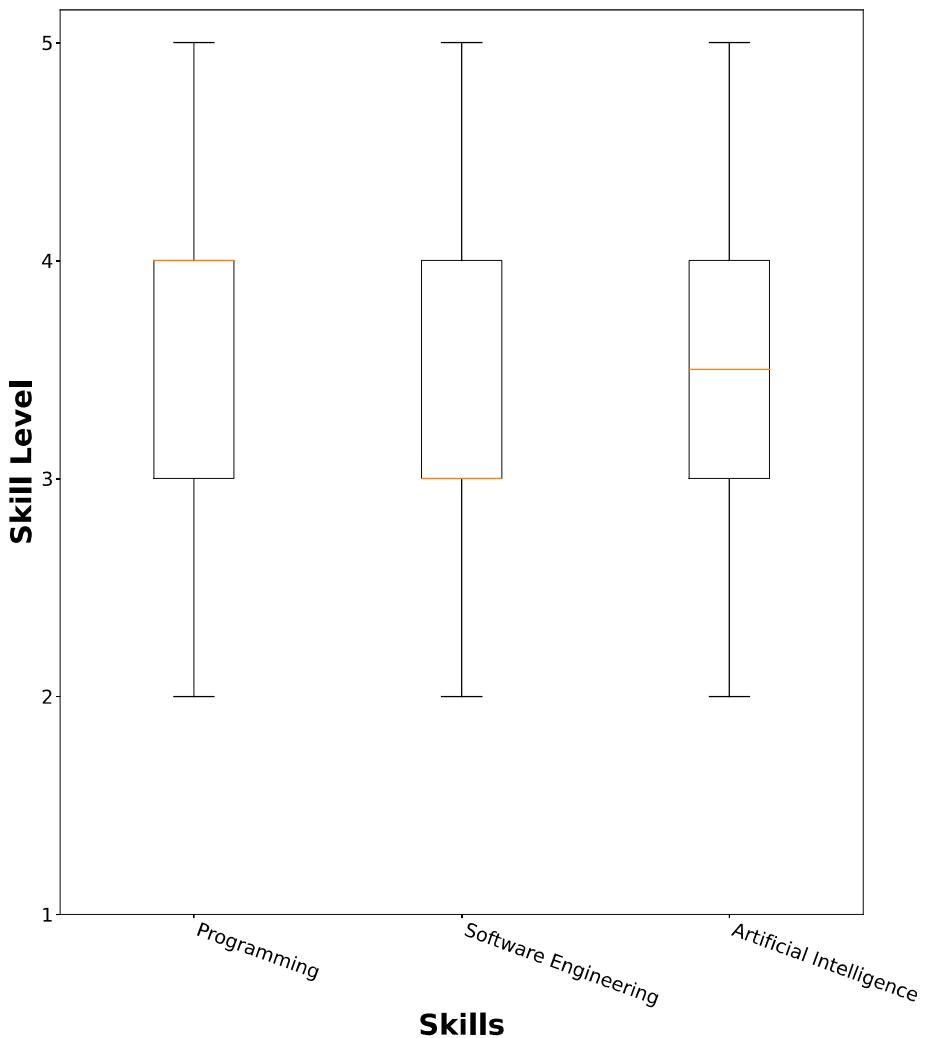


Figura 5.5: Competenze generali dei partecipanti inclusi nel questionario

è considerato per gli sviluppatori il più problematico per i sistemi AI, come illustrato in Tabella 5.4. Le motivazioni raccolte dai partecipanti sono riguardanti l'aspetto di sicurezza, in quanto la presenza di questa particolare istanza di AI Technical Debt può essere sfruttata al fine di comprendere il funzionamento del modello di intelligenza artificiale e effettuare attacchi malevoli. In

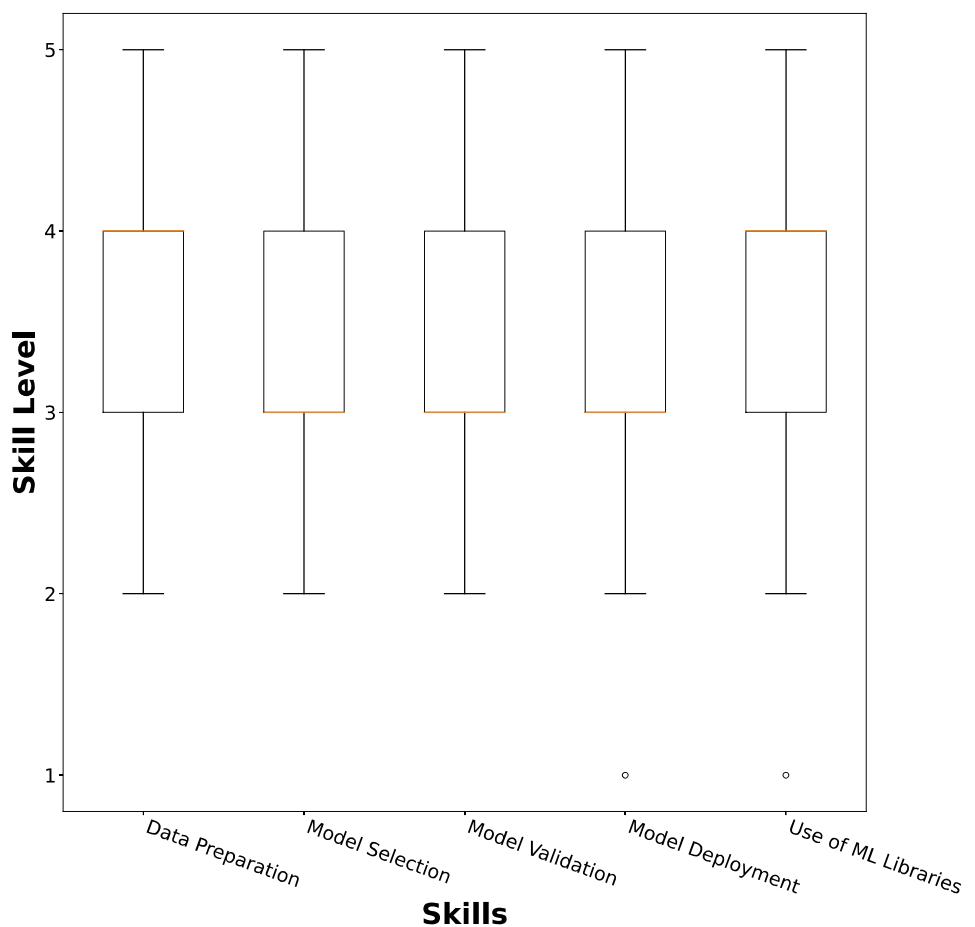


Figura 5.6: Competenze di intelligenza artificiale dei partecipanti inclusi nel questionario

particolare, un partecipante ha risposto spiegando che si trova molto spesso a lavorare con dati sensibili e questo può essere una grave falla di sicurezza utilizzabile per diffondere quei dati. Inoltre la percezione degli sviluppatori ha riscontrato un'alta severità per le istanze di *Pipeline Jungle* e *Jumbled Model Architecture*. Entrambe le istanze causano una crescita incontrollata del sistema, riportando gravi problematiche di complessità e manutenibilità. In particolare, un partecipante, in considerazione della presenza di un istanza

AI Technical Debt Instance	Median	Media	Dev. Std.	Moda
Glue Code	3.0	2.45	1.11	3.0
Multiple Language Smell	2.0	2.05	1.11	1.0
Undeclared Consumers	2.0	1.97	1.08	1.0
Pipeline Jungle	2.5	2.69	1.17	2.0
Correction Cascades	2.0	2.44	1.03	2.0
Scattered Use of ML Libraries	2.0	2.17	1.23	1.0
Jumbled Model Architecture	2.0	2.53	1.02	2.0
Unwanted Debugging Code	2.0	2.5	1.24	2.0
Deep God File	2.0	2.18	1.03	2.0

Tabella 5.3: Distribuzione della diffusione di AI Technical Debt secondo la percezione degli sviluppatori

di *Jumbled Model Architecture* ha risposto che la problematica nasce quando si ha a che fare con un sistema di grandi dimensioni, dove uno sviluppatore in fase di manutenzione del sistema deve affrontare una mancanza di chiarezza e di riconoscibilità delle componenti.

AI Technical Debt Instance	Median	Media	Dev. Std.	Moda
Glue Code	3.0	2.92	1.32	2.0
Multiple Language Smell	3.0	3.0	1.47	4.0
Undeclared Consumers	4.0	3.73	1.32	5.0
Pipeline Jungle	4.0	3.47	1.13	4.0
Correction Cascades	3.0	3.22	1.22	4.0
Scattered Use of ML Libraries	3.0	3.02	1.38	4.0
Jumbled Model Architecture	4.0	3.47	1.0	4.0
Unwanted Debugging Code	2.0	2.32	1.22	1.0
Deep God File	3.0	3.11	1.45	2.0

Tabella 5.4: Indice di severità delle diverse istanze di AI Technical Debt secondo la percezione degli sviluppatori.

In aggiunta lo sforzo di identificazione e lo sforzo di mitigazione delle istanze di AI Technical Debt percepito dagli sviluppatori è riportato rispettivamente in Tabella 5.5 e in Tabella 5.6. Anche sotto questo aspetto, gli sviluppatori

ritengono che l'identificazione e la mitigazione per le istanze di *Undeclared Consumers*, *Pipeline Jungle* e *Jumbled Model Architecture* richiede un'alto effort. Essendo istanze di AI Technical Debt che interferiscono sul sistema a livello architetturale, queste istanze portano un calo degli attributi di qualità su tutta l'architettura del sistema. Questa granularità così alta, a differenza dei dati e del codice, comporta ad aumentare la difficoltà di identificare e mitigare un problema presente. Inoltre, in caso di presenza di istanze come *Correction Cascades* o *Deep God File*, gli sviluppatori ritengono che la mitigazione di queste particolari istanze possa richiedere un'alto effort.

AI Technical Debt Instance	Median	Media	Dev. Std.	Moda
Glue Code	3.0	2.92	1.32	2.0
Multiple Language Smell	3.0	3.0	1.47	4.0
Undeclared Consumers	4.0	3.74	1.33	5.0
Pipeline Jungle	4.0	3.47	1.13	4.0
Correction Cascades	3.0	3.22	1.22	4.0
Scattered Use of ML Libraries	3.0	3.03	1.38	4.0
Jumbled Model Architecture	4.0	3.47	1.02	4.0
Unwanted Debugging Code	2.0	2.32	1.22	1.0
Deep God File	3.0	3.12	1.45	2.0

Tabella 5.5: Sforzo di identificazione delle diverse istanze di AI Technical Debt secondo la percezione degli sviluppatori.

5.2.4 RQ2.3: Qual è l'impatto causato dai code debt e architectural debt secondo la percezione degli sviluppatori AI?

La Tabella 5.7 riporta i risultati delle analisi relative alle caratteristiche di qualità di impatto sulla altre componenti dei sistemi AI, sulla comprensibilità e sull'evoluzione. Secondo il punto degli sviluppatori, l'impatto più alto sulle altri componenti del sistema è causato dalla presenza di *Undeclared Consumers*, considerando che un libero accesso ai dati in output del modello,

AI Technical Debt Instance	Mediana	Media	Dev. Std.	Moda
Glue Code	3.0	2.95	1.23	2.0
Multiple Language Smell	4.0	3.24	1.57	4.0
Undeclared Consumers	4.0	3.53	1.29	4.0
Pipeline Jungle	4.0	3.33	1.17	4.0
Correction Cascades	4.0	3.5	1.18	4.0
Scattered Use of ML Libraries	3.0	3.0	1.41	2.0
Jumbled Model Architecture	4.0	3.82	1.03	5.0
Unwanted Debugging Code	2.0	2.09	1.24	1.0
Deep God File	4.0	3.71	1.19	4.0

Tabella 5.6: Sforzo di mitigazione delle diverse istanze di AI Technical Debt secondo la percezione degli sviluppatori.

possia permettere ad applicazioni esterne e sconosciute di avere un impatto su tutte le componenti del sistema.

La comprensibilità del codice del software è, secondo il punto di vista degli sviluppatori, un aspetto di qualità che può essere influenzato da diverse istanze di AI Technical Debt. In particolare, dalle risposte riscontriamo *Deep God File* con una mediana pari a 5, una media pari a 4.32 e una moda pari a 5. Inoltre, *Pipeline Jungle*, *Jumbled Model Architecture* e *Correction Cascades* sono considerate molto problematiche e possibili cause di un calo della comprensibilità all'interno del sistema.

Molte delle istanze proposte agli sviluppatori sono ritenute invece come possibili minacce all'evoluzione del sistema. In particolare, *Deep God File*, la quale distribuzione delle risposte presenta una mediana pari a 4, una media pari a 4.21 e una moda pari a 5, è l'istanza di AI-Technical Debt più problematica per l'evoluzione di un sistema AI. Inoltre, un'alta influenza sull'evoluzione del sistema è causata da *Jumbled Model Architecture*, *Correction Cascades*, *Pipeline Jungle* e *Multiple Language Smell*.

La Tabella 5.8 riporta i risultati delle analisi relative alle caratteristiche di qualità di perfomance, accoppiamento e manutenzione dei sistemi AI. Secondo il punto degli sviluppatori, le performance del modello possono

Quality aspect	AI TechnicalDebt Instance	median	mean	std	mode
Impact	Glue Code	3.0	2.89	1.27	4
	Multiple Language Smell	3.0	3.03	1.42	4
	Undeclared Consumers	4.0	3.76	1.24	4
	Pipeline Jungle	4.0	3.44	1.08	4
	Correction Cascades	3.0	3.31	1.24	4
	Scattered Use of ML Libraries	2.5	2.64	1.33	1
	Jumbled Model Architecture	3.0	3.18	1.11	4
	Unwanted Debugging Code	2.0	2.32	0.98	2
	Deep God File	3.0	3.29	1.31	3
Understandability	Glue Code	3.5	3.05	1.41	4
	Multiple Language Smell	4.0	3.32	1.68	5
	Undeclared Consumers	3.0	3.03	1.22	4
	Pipeline Jungle	4.0	4.08	1.0	5
	Correction Cascades	4.0	3.94	1.07	4
	Scattered Use of ML Libraries	3.0	3.14	1.44	4
	Jumbled Model Architecture	4.0	4.06	1.01	4
	Unwanted Debugging Code	2.0	2.24	1.13	2
	Deep God File	5.0	4.32	0.84	5
Evolution	Glue Code	3.0	2.97	1.44	2
	Multiple Language Smell	4.0	3.61	1.5	5
	Undeclared Consumers	3.0	3.08	1.46	5
	Pipeline Jungle	4.0	3.86	1.15	5
	Correction Cascades	4.0	3.75	1.13	4
	Scattered Use of ML Libraries	3.5	3.28	1.41	4
	Jumbled Model Architecture	4.0	3.94	1.18	4
	Unwanted Debugging Code	2.0	2.18	1.03	2
	Deep God File	4.0	4.21	0.95	5

Tabella 5.7: Effetti sull'impatto sulle componenti del sistema, comprensibilità e evoluzione delle diverse istanze di AI Technical Debt secondo la percezione degli sviluppatori.

essere influenzate da diverse istanze di AI Technical Debt. La presenza di una *Pipeline Jungle*, compromettendo il controllo e l'accessibilità alle fasi di gestione e di esecuzione della pipeline, inficiano sulle performance finali del

modello. I risultati mostrano per questa istanza una mediana pari a 4, una media pari 3.97 e una moda pari a 4. Inoltre, gli sviluppatori AI ritengono che anche la presenza di *Correction Cascades* e *Undeclared Consumers* sono possibili cause di un decremento delle performance del modello.

La presenza di *Correction Cascades*, all'interno di un sistema AI, porta ad incrementare l'accoppiamento delle componenti, presentata dalle risposte che in media hanno riportato un'indice pari a 4.33.

Infine, analizzando l'aspetto di manutenibilità di un sistema AI, la presenza di *Jumbled Model Architecture* e minacce all'architettura del sistema possono aumentare la difficoltà di poter effettuare operazioni di manutenzione al sistema AI. Anche la presenza di *Deep God File* può rallentare nel processo di effettuare operazioni di manutenzione da parte degli sviluppatori.

5.2.5 RQ 2.4: Quali sono le strategie utilizzate dagli sviluppatori AI per l'identificazione e la mitigazione di code debt e architectural debt?

La Tabella 5.9 illustra i risultati delle risposte forniti dagli sviluppatori AI sui possibili approcci che utilizzano per la identificazione di AI Technical Debt. Per ogni istanza di AI Technical Debt è stato riportato il numero e la percentuale di sviluppatori che hanno indicato di utilizzare come approcci di identificazione una revisione manuale, un'ispezione automatica, un team specializzato all'analisi di questi problemi o se non utilizzano nessun'approccio per la identificazione della specifica istanza di AI Technical Debt. Dalla tabella è possibile notare come attualmente la maggior parte degli sviluppatori non ha identificato un tool o una tecnica automatica per la identificazione di AI Technical Debt all'interno del sistema, ma effettuano una revisione manuale. In particolare, in presenza dell'istanza di *Pipeline Jungle*, uno sviluppatore in particolare aggiunge che "regolari review vengono effettuate dal team al fine di avere sotto controllo la gestione della Pipeline.". Possibili soluzioni automatizzate quindi potrebbero permettere agli sviluppatori che attualmente

Quality aspect	AI TechnicalDebt Instance	median	mean	std	mode
Performance	Glue Code	4.0	3.39	1.35	4
	Multiple Language Smell	4.0	3.32	1.54	5
	Undeclared Consumers	4.0	3.61	1.41	5
	Pipeline Jungle	4.0	3.97	0.94	4
	Correction Cascades	4.0	3.81	1.12	4
	Scattered Use of ML Libraries	3.5	3.36	1.2	4
	Jumbled Model Architecture	3.0	3.29	1.09	3
	Unwanted Debugging Code	3.0	2.91	1.33	2
	Deep God File	3.0	3.09	1.29	3
Coupling	Glue Code	4.0	3.32	1.23	4
	Multiple Language Smell	4.0	3.37	1.44	4
	Undeclared Consumers	4.0	3.63	1.32	4
	Pipeline Jungle	4.0	3.61	1.1	4
	Correction Cascades	4.0	4.33	0.59	4
	Scattered Use of ML Libraries	3.0	3.28	1.16	4
	Jumbled Model Architecture	4.0	3.47	1.19	4
	Unwanted Debugging Code	2.0	2.32	1.15	1
	Deep God File	3.0	3.21	1.3	2
Maintenance	Glue Code	3.0	3.0	1.21	4
	Multiple Language Smell	4.0	3.39	1.52	5
	Undeclared Consumers	3.0	3.37	1.34	3
	Pipeline Jungle	4.0	3.75	1.13	4
	Correction Cascades	4.0	3.69	1.28	5
	Scattered Use of ML Libraries	3.0	3.17	1.21	3
	Jumbled Model Architecture	4.0	4.26	0.79	5
	Unwanted Debugging Code	2.0	2.21	0.95	2
	Deep God File	4.0	4.06	1.04	5

Tabella 5.8: Effetti sulle performance, accoppiamento e manutenzione delle diverse istanze di AI Technical Debt secondo la percezione degli sviluppatori.

effettuano ispezioni manuali di accelerare il processo. Parte degli sviluppatori utilizza tool di analisi statica per la identificazione delle istanze proposte. Uno sviluppatore, per la identificazione di istanze di *Undeclared Consumers*, indica di applicare uno Static Application Security Testing (SAST) Tool, il quale

può aiutare ad analizzare il codice sorgente per identificare difetti di sicurezza. È possibile quindi adoperare l'utilizzo di questa tipologia di tool al fine di identificare gli accessi che vengono effettuati al sistema AI. Gli approcci di refactoring identificati dagli sviluppatori AI prevedono infine, come indicato dalle risposte alle domande S2D8 del questionario, la riprogettazione della componente o la sostituzione di essa con un'altra soluzione.

AI TechnicalDebt Instance	M.I.		A.I.		P.T.		N/A	
	N	(%)	N	(%)	N	(%)	N	(%)
Glue Code	15	45.45	3	9.09	0	0	15	45.45
Multiple Language Smell	11	33.33	1	3.03	6	18.18	15	45.45
Undeclared Consumers	10	32.26	3	9.68	7	22.58	11	35.48
Pipeline Jungle	19	46.34	5	12.2	10	24.39	7	17.07
Correction Cascades	29	67.44	2	4.65	7	16.28	5	11.63
Scattered Use of ML Libraries	17	42.5	1	2.5	8	20	14	35
Jumbled Model Architecture	22	55	4	10	6	15	8	20
Unwanted Debugging Code	15	40.54	5	13.51	3	8.11	14	37.84
Deep God File	9	40.91	2	9.09	2	9.09	9	40.91

M.I. = Manual Inspection
A.I. = Automated Inspection
P.T. = Professional Team
N/A = Don't Identify

N = Numero di risposte
(%) = Percentuale delle risposte

Tabella 5.9: Tipologie di approcci utilizzati dagli sviluppatori AI per l'identificazione di AI Technical Debt.

🔗 Percezione degli sviluppatori I risultati riportano una bassa diffusione ma un'alta pericolosità dalla percezione degli sviluppatori per le istanze appartenenti alla tipologia di *architectural debt*. *Undeclared Consumers*, *Pipeline Jungles* e *Jumbled Model Architecture* sono le istanze riportate come le più pericolose e che causano la maggiore influenza sugli aspetti di qualità del sistema AI.

Capitolo 6

Conclusioni ed Implicazioni

Questo lavoro di tesi ha effettuato un'esplorazione completa della definizione, le minacce e le possibili soluzioni esistenti per la gestione, identificazione e mitigazione di istanze di Technical Debt che sono specifiche per i sistemi AI. Dall'analisi della letteratura si evince un'alta diffusione delle istanze relative ai dati, alla configurazione del sistema, al codice e all'architettura e in particolare la diffusione di istanze come *Pipeline Jungles*, *Unstable Data Dependencies* e *Epsilon Features*.

Successivamente, lo studio ha condotto un'analisi sulla percezione di 54 specialisti di sistemi AI con lo scopo di estrarre informazioni sulle possibili minacce che possono causare gli AI Technical Debt. In particolare, sono state analizzate la diffusione, la severità e l'impatto di 9 istanze di *AI Technical Debt*, appartenenti alle tipologie di *code debt* e *architectural debt*. I risultati mostrano in generale una discreta diffusione delle istanze nei sistemi AI, ma un'alta severità e impatto. In particolare, dal punto di vista degli sviluppatori si evince che le istanze di *Pipeline Jungle*, *Undeclared Consumers* e *Jumbled Model Architecture* sono considerate davvero pericolose per i sistemi AI, riscontrando un'alto indice di severità e conseguenzialmente un'alto sforzo necessario al fine di identificare e mitigare queste minacce. Inoltre, i risultati hanno dimostrato che la presenza di AI Technical Debt può influenzare decisivamente su tutti gli aspetti di qualità che sono stati proposti agli sviluppatori. Quindi, la

presenza di un’istanza di AI Technical Debt ha un’influenza diretta sulle altre componenti del sistema, sulla sua comprensibilità, sulla sua evoluzione, sul suo accoppiamento e sullo sforzo di manutenzione del sistema e sulle performance del modello, Infine, solo una piccola parte degli sviluppatori utilizzano tecniche automatiche al fine di identificare e effettuare refactoring, applicabili inoltre solo per alcune istanze di AI Technical Debt. La più grande parte degli sviluppatori, invece, utilizza approcci di ispezione e revisione manuale, coinvolgendo più componenti al fine di aumentare l'affidabilità del processo. I risultati di questo lavoro ha dato luce a diverse importanti informazioni. Lo stato attuale della ricerca non permette alla comunità accademica e scientifica di ottenere conoscenza della presenza di possibili minacce alla qualità del software AI. I risultati preliminari definiti dall’analisi della letteratura dimostrano che non è ancora presente un numero sufficiente di articoli di ricerca utile a diffondere l’argomento di ricerca e incrementare il contributo scientifico. I risultati di questo lavoro di tesi, in particolare le informazioni estratte dalla percezione degli sviluppatori, sono un ulteriore incentivo a investire lo sforzo della ricerca in questo campo. In particolare, le informazioni estratte da **RQ2.2** e **RQ2.3** sono importanti fonti utili a ridirezionare gli studi futuri, in quanto definiscono una base di selezione delle istanze di AI Technical Debt su cui approfondire e investire la ricerca al fine di identificare tecniche di identificazione e mitigazione per le industrie che operano nella produzione di sistemi AI. Inoltre, i risultati di questo lavoro fondano una base di definizioni utili a contribuire allo sviluppo della tassonomia di AI Technical Debt e, contemporaneamente, apre la strada a numerose sfide per i ricercatori. Un aspetto importante sul quale sarà necessario intervenire in futuro è investigare sulla diffusione e sulla severità di istanze appartenenti alla tipologia di *data debt*. Le istanze di *data debt* sono minacce alla qualità dei dati, e la loro introduzione all’interno dei sistemi AI possono non solo danneggiare componenti di gestione e preparazione dei dati, ma propagare il debito introdotto verso tutte le altri componenti del sistema. Inoltre, lo studio condotto ha collegato le informazioni utili fornite dallo

stato dell'arte alla percezione degli specialisti AI. Per poter avere una visione completa sui possibili danni che possono causare la presenza delle definite istanze, sarà necessario includere nelle analisi l'estrazione di informazioni e lo studio di progetti di sistemi di intelligenza artificiale. Questo ulteriore passo darà la possibilità di analizzare nella pratica l'effetto che causa l'introduzione di un particolare AI Technical Debt all'interno dei progetti sotto analisi. Infine, un aspetto fondamentale su cui investire gli sforzi sarà sicuramente sulla definizione di tecniche di identificazione e di refactoring automatiche, al fine di fornire supporto agli sviluppatori AI, i quali attualmente tentano di identificare e ripagare il debito introdotto attraverso la conduzione di ispezioni manuali e revisioni.

Bibliografia

- [1] G. Anthes, «Artificial Intelligence Poised to Ride a New Wave», *Commun. ACM*, vol. 60, n. 7, pp. 19–21, giu. 2017, ISSN: 0001-0782. DOI: 10.1145/3088342. indirizzo: <https://doi.org/10.1145/3088342>.
- [2] L. Fischer, L. Ehrlinger, V. Geist et al., «Ai system engineering—key challenges and lessons learned», *Machine Learning and Knowledge Extraction*, vol. 3, n. 1, pp. 56–83, 2020.
- [3] *Uber's self-driving car saw the pedestrian but didn't swerve*, <https://www.theguardian.com/technology/2018/may/08/ubers-self-driving-car-saw-the-pedestrian-but-didnt-swerve-report>, 2018.
- [4] W. Cunningham, «The WyCash Portfolio Management System», *SIGPLAN OOPS Mess.*, vol. 4, n. 2, pp. 29–30, dic. 1992, ISSN: 1055-6400. DOI: 10.1145/157710.157715. indirizzo: <https://doi.org/10.1145/157710.157715>.
- [5] «IEEE Standard Glossary of Software Engineering Terminology», *IEEE Std 610.12-1990*, pp. 1–84, 1990. DOI: 10.1109/IEEESTD.1990.101064.
- [6] I. Sommerville, *Software Engineering*, 9^a ed. Harlow, England: Addison-Wesley, 2010, ISBN: 978-0-13-703515-1.
- [7] ISO/IEC 25010, *ISO/IEC 25010:2011, Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models*, 2011.

- [8] G. Suryanarayana, G. Samarthyam e T. Sharma, *Refactoring for Software Design Smells: Managing Technical Debt*, 1st. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2014, ISBN: 0128013974.
- [9] A. Ampatzoglou, A. Ampatzoglou, A. Chatzigeorgiou e P. Avgeriou, «The financial aspect of managing technical debt: A systematic literature review», *Information and Software Technology*, vol. 64, pp. 52–73, 2015, ISSN: 0950-5849.
DOI: <https://doi.org/10.1016/j.infsof.2015.04.001>. indirizzo: <https://www.sciencedirect.com/science/article/pii/S0950584915000762>.
- [10] M. Fowler, *Refactoring: Improving the Design of Existing Code*. Boston, MA, USA: Addison-Wesley, 1999, ISBN: 0-201-48567-2.
- [11] F. Palomba, G. Bavota, M. Di Penta, R. Oliveto e A. De Lucia, «Do They Really Smell Bad? A Study on Developers' Perception of Bad Code Smells», in *2014 IEEE International Conference on Software Maintenance and Evolution*, 2014, pp. 101–110.
DOI: [10.1109/ICSME.2014.32](https://doi.org/10.1109/ICSME.2014.32).
- [12] McCabe, «A Complexity Measure», *IEEE Transactions on Software Engineering*, vol. 2, pp. 308–320, 1976.
- [13] M.-W. Dictionary, *Artificial Intelligence definition*, Accessed = 2022-08-10, 2020. indirizzo: <https://www.merriam-webster.com/dictionary/artificial-intelligence>.
- [14] C. E. Dictionary, *Artificial Intelligence definition*, Accessed = 2022-08-10, 2012. indirizzo: <https://www.dictionary.com/browse/artificial-intelligence>.
- [15] J. McCarthy, «What is AI?», 2004. indirizzo: <http://jmc.stanford.edu/articles/whatisai/whatisai.pdf>.

- [16] *Artificial intelligence — Wikipedia, The Free Encyclopedia*, [Online; accessed 10-August-2022], 2022. indirizzo: <https://en.wikipedia.org/w/index.php?title=Artificial-intelligence>.
- [17] R. S. Michalski e J. R. Anderson, «Machine learning - an artificial intelligence approach», in *Symbolic computation*, 1984.
- [18] Y. LeCun, Y. Bengio e G. Hinton, «Deep learning», *nature*, vol. 521, n. 7553, p. 436, 2015.
- [19] T. Menzies, «The Five Laws of SE for AI», *IEEE Software*, vol. 37, n. 1, pp. 81–85, 2020.
DOI: [10.1109/MS.2019.2954841](https://doi.org/10.1109/MS.2019.2954841).
- [20] D. Sculley, G. Holt, D. Golovin et al., «Hidden technical debt in machine learning systems», *Advances in neural information processing systems*, vol. 28, 2015.
- [21] D. Sculley. «Machine Learning, Technical Debt and You», Google. (2015),
indirizzo: <https://www.youtube.com/watch?v=V18AsBIHlWs>.
- [22] «Scikit Learn, Machine Learning in Python». (2022),
indirizzo: <https://scikit-learn.org/stable/>.
- [23] N. Novielli, D. Girardi e F. Lanubile, «A Benchmark Study on Sentiment Analysis for Software Engineering Research», ser. MSR '18, Gothenburg, Sweden: Association for Computing Machinery, 2018, ISBN: 9781450357166. DOI: [10.1145/3196398.3196403](https://doi.org/10.1145/3196398.3196403). indirizzo: <https://doi.org/10.1145/3196398.3196403>.
- [24] R. Wirth e J. Hipp, «CRISP-DM: Towards a standard process model for data mining», *Proceedings of the 4th International Conference on the Practical Applications of Knowledge Discovery and Data Mining*, gen. 2000.

- [25] A. Rajaraman e J. D. Ullman, «Data Mining», in *Mining of Massive Datasets*. Cambridge University Press, 2011, pp. 1–17. DOI: 10.1017/CBO9781139058452.002.
- [26] F. Khomh, B. Adams, J. Cheng, M. Fokaefs e G. Antoniol, «Software Engineering for Machine-Learning Applications: The Road Ahead», *IEEE Software*, vol. 35, n. 5, pp. 81–84, 2018. DOI: 10.1109/MS.2018.3571224.
- [27] H. Foidl, M. Felderer e R. Ramler, *Data Smells: Categories, Causes and Consequences, and Detection of Suspicious Data in AI-based Systems*, 2022. DOI: 10.48550/ARXIV.2203.10384. indirizzo: <https://arxiv.org/abs/2203.10384>.
- [28] T. Sharma, M. Fragkoulis, S. Rizou, M. Bruntink e D. Spinellis, «Smelly Relations: Measuring and Understanding Database Schema Quality», in *2018 IEEE/ACM 40th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*, 2018, pp. 55–64.
- [29] M. Al-Barak e R. Bahsoon, «Database Design Debts through Examining Schema Evolution», in *2016 IEEE 8th International Workshop on Managing Technical Debt (MTD)*, 2016, pp. 17–23. DOI: 10.1109/MTD.2016.9.
- [30] J. Bogner, R. Verdecchia e I. Gerostathopoulos, «Characterizing technical debt and antipatterns in ai-based systems: A systematic mapping study», in *2021 IEEE/ACM International Conference on Technical Debt (TechDebt)*, IEEE, 2021, pp. 64–73.
- [31] H. Jebnoun, H. Ben Braiek, M. M. Rahman e F. Khomh, «The Scent of Deep Learning Code: An Empirical Study», in *Proceedings of the 17th International Conference on Mining Software Repositories*, ser. MSR '20,

- Seoul, Republic of Korea: Association for Computing Machinery, 2020, pp. 420–430, ISBN: 9781450375177. DOI: 10.1145/3379597.3387479. indirizzo: <https://doi.org/10.1145/3379597.3387479>.
- [32] J. Gesi, S. Liu, J. Li et al., *Code Smells in Machine Learning Systems*, 2022. DOI: 10.48550/ARXIV.2203.00803. indirizzo: <https://arxiv.org/abs/2203.00803>.
- [33] B. Kitchenham, «Procedures for Performing Systematic Reviews», *Keele, UK, Keele Univ.*, vol. 33, ago. 2004.
- [34] C. A. Bailey, *A Guide to Qualitative Field Research*. SAGE Publications, 2017.
- [35] B. Kitchenham e S. Charters, «Guidelines for performing systematic literature reviews in software engineering», Department of Computer Science, University of Durham, Durham, UK, rapp. tecn. EBSE-2007-01, 2007.
- [36] R. Solingen e E. Berghout, «The Goal/Question/Metric Method: A Practical Guide for Quality Improvement of Software Development», gen. 1999.
- [37] M. Ciolkowski, O. Laitenberger, S. Vegas e S. Biffl, «Practical Experiences in the Design and Conduct of Surveys in Empirical Software Engineering», in *ESERNET*, 2003.
- [38] C. Atzmüller e P. Steiner, «Experimental vignette designs for factorial surveys», *Kolner Zeitschrift fur Soziologie und Sozialpsychologie*, vol. 58, pp. 117–146+200, mar. 2006.
- [39] C. Wohlin, «Guidelines for Snowballing in Systematic Literature Studies and a Replication in Software Engineering», ser. EASE '14, London, England, United Kingdom: Association for Computing Machinery, 2014, ISBN: 9781450324762. DOI: 10.1145/2601248.2601268. indirizzo: <https://doi.org/10.1145/2601248.2601268>.

- [40] G. Recupito, *Github replication package*, 2022. indirizzo: <https://github.com/gilbertrec/TDStateOfArt-DevPerception-RP>.

Lista delle Sorgenti

- S01 - Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., ... Dennison, D. (2015). Hidden technical debt in machine learning systems. *Advances in neural information processing systems*, 28.
- S02 - Tang, Y., Khatchadourian, R., Bagherzadeh, M., Singh, R., Stewart, A., and Raja, A. (2021). An empirical study of refactorings and technical debt in Machine Learning systems. *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, 238–250. IEEE.
- S03 - Belani, H., Vukovic, M., and Car, Ž. (2019). Requirements Engineering Challenges in Building AI-Based Complex Systems. *2019 IEEE 27th International Requirements Engineering Conference Workshops (REW)*, 252–255. doi:10.1109/REW.2019.00051
- S04 - Foidl, H., Felderer, M., and Biffl, S. (2019). Technical Debt in Data-Intensive Software Systems. *2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 338–341. doi:10.1109/SEAA.2019.00058
- S05 - Alahdab, M., and Çalıklı, G. (2019). Empirical analysis of hidden technical debt patterns in machine learning software. *International Conference on Product-Focused Software Process Improvement*, 195–202. Springer.
- S06 - Liu, J., Huang, Q., Xia, X., Shihab, E., Lo, D., and Li, S. (2020). Is using deep learning frameworks free? characterizing technical debt in deep learning frameworks. *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Society*, 1–10.
- S07 - Malakuti, S., Borrison, R., Kotriwala, A., Kloepper, B., Nordlund, E., and Ronnberg, K. (2021). An Integrated Platform for Multi-Model

BIBLIOGRAFIA

Digital Twins. 11th International Conference on the Internet of Things, 9–16.

- S08 - Bogner, J., Verdecchia, R., and Gerostathopoulos, I. (2021). Characterizing technical debt and antipatterns in ai-based systems: A systematic mapping study. 2021 IEEE/ACM International Conference on Technical Debt (TechDebt), 64–73. IEEE.