UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Laurea Triennale in Informatica

TESI DI LAUREA

Studio e analisi dei problemi riguardanti l'utilizzo dei bot per lo sviluppo software

RELATORI

Prof. **Fabio Palomba**

Dr. **Stefano Lambiase**

Università degli Studi di Salerno

CANDIDATO

Otino Pio Santosuosso

Matricola: 0512107364

Anno Accademico 2021-2022

Un computer meriterebbe di essere definito intelligente se potesse ingannare un essere umano facendogli credere di essere umano.

Alan Turing.

Sommario

Al giorno d'oggi, la maggior parte delle persone hanno integrato nella loro vita i bot. Questi ultimi si occupano appunto di affiancare l'utente rendendogli i lavori, o lo svolgimento di alcune attività sempre più semplici. Durante gli anni si è resi conto dei diversi atteggiamenti che assumono gli agenti in base al loro sviluppo.

Pertanto, questa tesi, si occupa proprio di analizzare quelli che sono gli aspetti negativi dei bot, portando l'attenzione sugli agenti che vengono affiancati agli sviluppatori, nello sviluppo software. Durante questo lavoro sono state analizzate e categorizzate le maggiori *challenge* che si presentano nello sviluppo e utilizzo di questi ultimi.

Per raggiungere tale obiettivo è stato necessario svolgere un lavoro di ricerca sulla letteratura primaria e secondaria, in modo da analizzare tutti i progressi che hanno portato i bot negli ultimi anni.

Tale ricerca ha consentito sopratutto di focalizzare l'attenzione sugli aspetti negativi e quindi cercare una soluzione a questi ultimi. La soluzione a questi problemi è stata Stephen, un agente conversazione capace di fornire *best-practice* per lo sviluppo di bot.

Indice

Elenco delle figure

Elenco delle tabelle

CAPITOLO 1

Introduzione

1.1 Motivazioni e Obiettivi

Al giorno d'oggi, i bot sono adoperati in qualsiasi ambito, dal settore medico [13], al settore educativo, fino ad affiancare gli sviluppatori nello sviluppo *software*. Per questo motivo, la progettazione e lo sviluppo degli agenti, è un attività importate affinché questi non creino dei disagi nelle loro applicazioni quotidiane.

Lo stato dell'arte ha individuato la maggior parte degli aspetti positivi e negativi che vi sono dietro gli agenti conversazionali. Più in dettaglio si è occupato di :

1. Osservare le varie applicazioni dei bot;
2. Cosa gli agenti portano di buono;
3. Le sfide che si rivelano dietro lo sviluppo e la progettazione di questi ultimi;
4. I tool che aiutano nell'affrontare le *challenge* dei bot;

La seguente tesi è stata realizzata poiché è stato deciso di approfondire ancora di più le tutte le sfide che ci sono dietro i bot, calandosi nei panni di chi "utilizza" gli agenti e di chi li "sviluppa". Pertanto è stata condotta una ricerca del tipo *Multivocal Literature Review*, con la quale è stato possibile svolgere sia una ricerca nella *white literature*, ovvero documenti realizzati da conferenze, ma anche nella *grey literature*, ovvero documenti estratti da *blog* e *community*.

Questa tecnica ha permesso difatti di categorizzare tutte le *challenge* in:

- *Development challenge*
- *Use challenge*

Dopo l'analisi è stato ritenuto essenziale realizzare un agente, Stephen, in grado di affiancare gli sviluppatori nello sviluppo dei bot. Stephen di fatti è stato realizzato applicando tutto ciò che è stato appreso durante questo periodo, cercando di realizzare appunto un agente di qualità.

La realizzazione di quest'ultimo è stata permessa grazie al framework AZURE BOT FRAMEWORK, il quale ha facilitato lo sviluppo dell'agente. Inseme ad esso è stato usato anche BOT FRAMEWORK EMULATOR, il quale ha permesso di testare Stephen in locale. Infine il bot è stato affiancato da un modello di *Natural Language Processing*, con lo scopo di riconoscere l'intento di una frase. Il modulo appena citato è stato implementato usando due algoritmi:

- Il primo si basa sulle *Artificial neural network*, esse difatti hanno un comportamento simile alle reti neurali biologie. Prendono in input una frase, e tramite i nodi cercano quale può essere l'intento che rappresenta la frase.
- Il secondo modulo si basa sul teorema di *Naive Bayes*, infatti utilizza la probabilità condizionata per valutare quanto la frase rappresenta un determinato intento in termini probabilistici. Per il problema in questione è stata usata una variante, ovvero quella "multinomiale", la quale è più adatta per la classificazione del testo.

Dopo la loro implementazione, sono state applicate delle metriche di valutazione in modo da esaminare quale dei due moduli affiancare al bot.

1.2 Risultati

In conclusione i risultati che sono stati ottenuti in questo lavoro di tesi sono i seguenti:

1. Analisi dei vari aspetti dei bot, in particolare focalizzandosi su quelli negativi. L'analisi è stata svolta sotto più punti di vista, in modo da categorizzare quante più *challenge* possibili.
2. Implementazione di Stephen, un agente conversazionale capace di offrire *best-practice* per lo sviluppo di questi ultimi.
3. Realizzazione di un *survey* per la creazione di un *dataset* con il quale allenare Stephen. Il seguente sondaggio è stato sottoposto agli studenti dell'Università degli Studi di Salerno.
4. Implementazione di un modulo per la comprensione del linguaggio naturale, adoperando tecniche di *data cleaning*.
5. Il modulo è stato affiancato a Stephen, ed è stato pubblicato un *repository* [9] nel quale è presente il codice sorgente dell'agente e tutta la documentazione generata in questo lavoro di tesi.

1.3 Struttura della tesi

Per la divulgazione di tutti questi concetti appena menzionati sono stati previsti cinque capitoli. Di seguito è presente una sintesi dei vari capitoli:

- **Capitolo 2**: Stato dell'arte, il quale illustra tutti gli studi che sono stati effettuati sui bot di recente. In questo capitolo principalmente è stato discusso delle varie applicazioni degli agenti, in ambito quotidiano e in quello aziendale. Poi è stata presentata una generica architettura interna dei bot, analizzando da quali componenti sono formati e soprattutto come essi lavorano insieme. Infine l'obiettivo della tesi è discutere delle problematiche dei bot

nello sviluppo software, quindi è stata fatta una categorizzazione degli agenti che vengono utilizzati in questo ambito.

- **Capitolo 3**: Un'analisi della letteratura per le challenge dei chatbot nello sviluppo software, questo capitolo discute di tutta la metodologia che è stata applicata durante la ricerca. Infine in questo capitolo sono stati espressi i risultati della ricerca come una categorizzazione delle sfide che più interessano i bot.
- **Capitolo 4**: Stephen, in questo quarto capitolo sono state espresse tutte le scelte prese per l'implementazione di Stephen. Esse comprendevano *frameworks*, scelte architetturali e il modello di *Natural Language processing*.
- **Capitolo 5**: Conclusioni e lavori futuri, in questo ultimo capitolo sono stati discussi i risultati che ha portato il seguente progetto di tesi e quelli che possono essere i lavori futuri.

CAPITOLO 2

Stato dell'arte

2.1 Bot and Chatbot nello sviluppo

Sin dai primi programmi *software*, le persone sognavano di farli agire, pensare e parlare come gli esseri umani. Tutto questo con lo scopo non solo di automatizzare i compiti degli esseri umani, ma anche di lavorare con essi per risolvere compiti intellettuali che non possono essere completamente automatizzati.

Già nel 1966, la speranza era che questi programmi superassero il test di Turing, in cui le persone venivano ingannate facendogli credere di interagire con un essere umano più intelligente piuttosto che con un semplice *software*. Nello stesso anno, Joseph Weizenbaum, sviluppò un programma con il quale dimostrò che era possibile far comunicare un essere umano con un computer utilizzando il linguaggio naturale. Da quel momento nacque la prima generazione di "bot/chatbot".

Oggi, l'impiego dei bot può essere applicato in vari ambiti, spesso però vengono utilizzati in quelle attività ripetitive e noiose. In particolare nello sviluppo software vengono utilizzati per aiutare gli sviluppatori a prendere decisioni più ragionate e per supportare gli individui nelle attività di comunicazione. Un esempio di applicazione degli agenti per lo sviluppo di applicativi *software* è l'esecuzione dei test una volta che vengono soddisfatte determinate condizioni, o addirittura la stesura della documentazione dopo una modifica effettuata al *software*. In generale tutti quei compiti che uno sviluppatore potrebbe evitare in modo da potersi concentrare su altri aspetti più importanti.

Portando l'attenzione allo stesso argomento, ma al di fuori dello sviluppo *software*, le aziende usufruiscono dei bot per estendere i propri servizi in modo da soddisfare al meglio le esigenze dei propri clienti. In particolare l'assistente di ricerca di Google si occupa proprio di migliorare l'esperienza dell'utente. Perfezionare l'esperienza vuol dire aiutare a effettuare delle ricerche, oppure pianificare eventi e promemoria, ai quali poi si allegano tantissimi altri servizi.

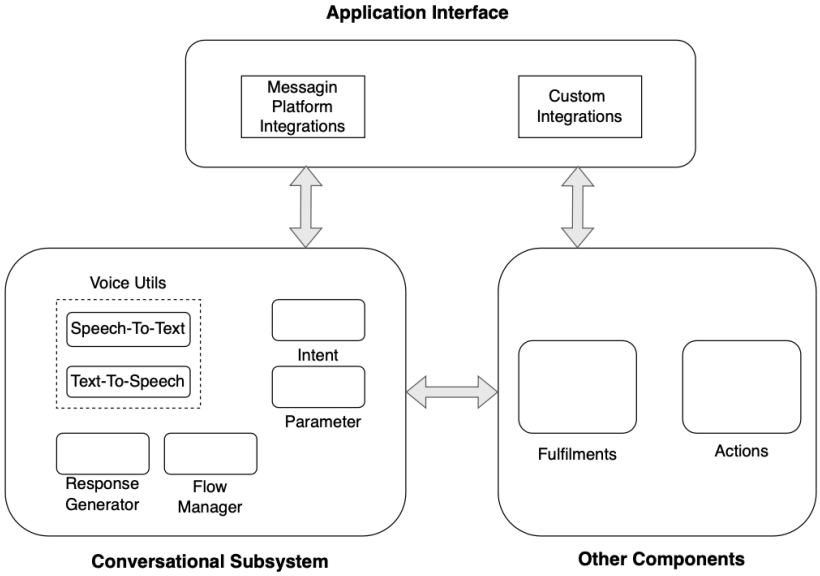

Figura 2.1: Bot Architecture.

Per ciò che concerne lo sviluppo software, l'utilizzo degli agenti conversazionali è aumentato vertiginosamente in quanto, come è stato detto, vanno a snellire molte delle attività noiose e ripetitive.

A livello di team, i bot hanno migliorato molto la comunicazione e l'efficienza in ogni stato del ciclo di vita dello sviluppo *software*. Tutto questo è stato concesso proprio grazie agli strumenti come SLACK o HIPCHAT che permettono l'integrazione dei loro servizi con i Bot. Permettendo ciò, gli agenti conversazionali hanno acquisito un nuovo punto di forza, ovvero essere integrati nelle chat di comunicazione. Tutto ciò ha permesso di migliorare la comunicazione fra sviluppatori e bot.

2.1.1 Bot architecture

Per comprendere meglio il funzionamento dei bot bisogna introdurre alcuni concetti sulla loro architettura interna. In letteratura è stato realizzato uno studio sull'architettura di riferimento per applicazioni con componenti conversazionali [14].

Nella Figura 2.1 è possibile notare come l'architettura degli agenti conversazionali è stata divisa in 3 sottosistemi:

1. **Application Interface:** questo primo sottosistema si occupa di offrire al consumatore un interfaccia conversazionale con l'agente. Ci sono diverse tipologie di integrazione:

   Custom integrations: sono delle integrazioni in nuove applicazioni, esse possono variare tra portali web, applicazioni Android e applicazioni IOS.

   Messaging Platform Integrations: sono integrazioni con piattaforme di messaggistica popolari che permettono di integrare il bot realizzato. Un esempio è proprio TELEGRAM o MESSENGER.

2. **Conversational Subsystem:** in questo secondo sottosistema fanno parte tutte le componenti che hanno l'incarico di processare il linguaggio naturale. Si può notare che ne fanno parte:

Voice Utils: è una componente essenziale per le applicazioni che hanno un interfaccia vocale, in quanto i bot cercano di lavorare sempre con un input testuali anziché con input vocali. Questa componente permette appunto di convertire il messaggio vocale in messaggio testuale, e si tratta di *Speech-to-Text*. Permette inoltre di trasformare il messaggio testuale in vocale ad esempio per fornire una risposta, e si tratta di *Text-to-Speech*.

Intent and Parameter: gli intenti sono le diverse classi di input che l'agente conversazionale può prevedere, mentre i parametri sono attributi della richiesta fatta al bot ed entrambi sono necessari per produrre una risposta. Di solito un agente possiede una lista predefinita di intenti. Mentre un classificatore si occupa di classificare una frase in uno di questi. Un aspetto importante è che nella lista degli intenti ce ne sia uno che viene utilizzato da "*fallback*", nel caso in cui il classificatore non riesca a categorizzare l'input. Per i parametri esiste un altro classificatore apposito che si occupa di estrarre gli attributi dalla frase in input. Questo perché i parametri possono essere presenti letteralmente nella frase, o derivare dai frammenti della medesima.

Response Genereto: questa componente permette di generare una risposta da dare all'utente, in modo da fornirgli un feedback al precedente input. Le risposte che vengono generate possono essere statiche, come ad esempio "Ciao, come stai?". Se invece si parla di risposte dinamiche, esse sono frasi che comprendono i parametri che l'utente ha fornito in input. Per le risposte dinamica il generatore ha bisogno di altre componenti, che sono definite in base all'ambiente in cui il bot offre il servizio.

Flow Manager: questa componente ha un compito fondamentale, poiché si occupa del flusso della conversazione. Anche se sembra banale per una conversazione naturale, per una "umano-bot" non lo è affatto. Questa componente si occupa di tener traccia dello stato attuale della conversazione, e in base agli input forniti capire qual'è il passo successivo. Inoltre per il *Flow Manager* è molto importante il contesto degli oggetti che persistono in più cicli di risposta. Difatti essi vanno a cambiare il proprio stato in base alla conversazione corrente.

3. **Other Components:** per altre componenti si intende quell'insieme di elementi che devono fornire le funzionalità dell'applicazione. Per realizzare un ponte fra il sottosistema conversazionale e le funzionalità offerte dal bot vengono utilizzate azioni, *Actions*, e gli adempimenti , *Fulfilments*.

   Fulfilments: gli adempimenti sono i punti di connessione dove il sottosistema conversazionale si connette al resto dell'applicazione. Il *Flow Manager* sceglie un particolare adempimento in base all'evento accaduto.

   Actions: le azioni sono attivate dagli adempimenti, e vengono rappresentate come una pipeline che viene eseguita per soddisfare la richiesta dell'utente.

2.1.2 Tipi di bot

È stato discusso di come i bot agevolano gli sviluppatori nella comunicazione, ma insieme ad essi esistono altre classi di agenti conversazionali con diversi scopi, ad esempio per lo sviluppo.

Molto interessante è stato uno studio [15] sull'utilizzo dei bot nelle varie fasi dello sviluppo *software*. Da questa ricerca sono emersi svariate classi di agenti:

Bot di codice: esistono tantissimi agenti conversazionali che aiutano a rendere le attività di codifica più efficienti ed efficaci. Difatti questi ultimi vengono utilizzati molto spesso per affrontare quelle sfide che si presentando durante lo sviluppo.

Ad esempio quando uno sviluppatore deve sincronizzare le attività su due o più flussi di lavoro separati. Ossia modificare il codice su GITHUB e aggiornare elementi di lavoro e attività su TRELLO. Eseguire tutte queste mansioni spesso risulta noioso, però si può usufruire di un bot che faccia proprio questo.

Un ulteriore esempio che porta una migliore comprensione del ruolo dei "bot di codice" è quello del *refactoring-bot* [21]. Infatti è stato realizzato uno studio su come creare un agente conversazionale che agevolasse il processo di *refactoring* agli sviluppatori, ovvero modificare la strutture interna di un *software*.

Questa soluzione è stata presa in considerazione in modo che si integrasse nel modo più naturale possibile al team. L'obiettivo dell'agente era di creare richieste *pull* per proporre agli sviluppatori le possibili modifiche nel codice. Ciò che ha spinto la creazione di *refactoring-bot* è stato semplicemente il voler migliorare la qualità del codice con il minor sforzo aggiuntivo per gli sviluppatori. Difatti quando arriva la richiesta *pull* di modifica gli sviluppatori possono approvare o rifiutare la modifica proposta.

Bot di test: i Bot svolgono un ruolo fondamentale nei test. Prima dell'utilizzo degli agenti conversazionali molti strumenti di analisi del codice, erano macchinosi da usare, però integrandoli ad essi sono molto più accessibili. In effetti strumenti come FINDBUGS, CHECKSTYLE e PMD sono stati integrati nel Bot Freud, un agente fornito dall'ecosistema Atlassian. Quando uno di questi strumenti trova un problema nella qualità del codice, Freud va automaticamente a creare una richiesta *pull* con la corrispondente modifica del codice suggerita.

Ci sono altri agenti conversazionali di Atlassian che assistono il test del *software*, ad esempio Dr.Code, il quale monitora e visualizza lo stato di salute del progetto nel corso del tempo.

Un agente che lavora in un modo simile è Repairnator [17], anche esso lavora su un progetto nel tempo, ma si focalizza maggiormente sugli errori di test che si verificano nell'integrazione continua, ovvero quando viene fatta una modifica al codice.

Come si può notare dal *workflow* di Repairnator, Figura 2.2, risulta interessante come l'agente conversazionale offre più output: il primo è per risolvere gli errori dei test, il secondo è per raccoglie dati preziosi sulla riparazione dei programmi per delle eventuali ricerche future in questo settore.

DevOps Bot: i *DevOps Bot*, anche chiamati *ChatOps* o *ChatBot*, vengono utilizzati per velocizzare l'implementazione del codice o affrontare i loop di feedback lenti tra sviluppatori, infrastruttura e personale.

Un ulteriore utilizzo è quello di apprendimento di concetti sconosciuti. Un esempio interessante è PagerBot, un agente creato da *Stripe*, che si occupava tramite PAGERDUTY di avvisare le persone giuste di un guasto di un servizio, in modo da ridurre il rumore di avviso.

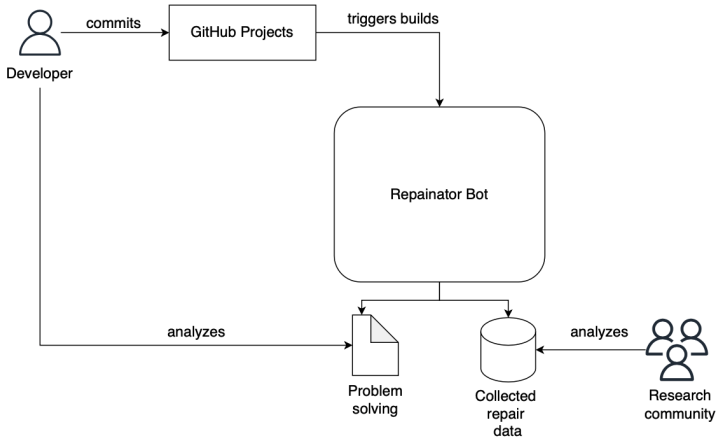

Figura 2.2: Workflow Repairnator.

È stato discusso anche di *DevOps Bot* per l'apprendimento di concetti sconosciuti, questo può essere utilizzato in diversi ambiti come nella didattica, oppure stesso nel sviluppo per i *Software Engineer* che sono alle prime armi. Per quest'ultima applicazione è stato realizzato TutorBot che si occupa di guidare in modo interattivo i *Software Engineer* su nuovi argomenti o concetti che desiderano padroneggiare [16]. La realizzazione di questo agente conversazionale comprende un'interfaccia realizzata con DIALOGFLOW di Google. Questo perché il *framework* citato cerca di realizzare interfacce quanto più familiari e intuitive possibili in modo che l'utente non si annoi durate l'utilizzo.

Inoltre *TutorBot* tiene traccia dei progressi dell'utente dopo l'apprendimento, in modo da potersi focalizzare sui concetti meno acquisiti. Dallo studio effettuato su questo bot, è stato notato come ha portato notevoli vantaggi soprattutto in termini di tempo risparmiato.

Bot di supporto: i bot di supporto servono per colmare il divario di comunicazione che c'è tra utenti e sviluppatori. Una delle sfide principali è quella di gestire i messaggi provenienti dagli utenti, e gli agenti possono automatizzare le interazioni con gli utenti rispondendo alle domande più frequenti. L'applicazione dei bot di supporto oltre al contesto dello sviluppo software, può essere applicata anche in altri ambiti come ad esempio nell'istituzione.

Un esempio è proprio *Tashi-Bot* che si occupa di aiutare gli studenti universitari ad ottenere informazioni su processi accademici e amministrativi [5].

Il vantaggio che portano i bot di supporto, tra cui anche *Tashi-Bot*, è che sono disponibili 24 ore su 24, 7 giorni su 7, distribuiti su più canali di comunicazione, e vanno a ridurre notevolmente i costi di lavoro.

Bot di documentazione: spesso la documentazione è una delle attività più costose in termini di tempo per gli sviluppatori, come ad esempio la creazione di note di rilascio di una nuova versione del software. Difatti sempre *Atlassian*, offre un agente conversazionale che tramite le informazioni dei *commit* riesce a realizzare delle note di rilascio per la nuova versione del *software*.

Negli anni passati è stato realizzato un agente simile a quello proposto da Atlassian, il quale genera documentazione basandosi sul *refactoring* [12]. Il seguente bot non crea soltanto docu-

mentazione sul *refactoring* del codice, ma anche sui *refactoring architettonici*. Esso è costituito principalmente da una componente che si occupa di analizzare le modifiche effettuate. Una seconda componente certifica la documentazione scritta dal committente, in modo da verificare l'eventuale documentazione mancante o errata. Infine l'ultima componete si occupa della generazione di documentazione basata su regole. Una volta generata la documentazione di *refactoring*, lo sviluppatore può interagire con l'agente per accettare, rifiutare o modificare la documentazione generata.

Si può notare come ci sono molte classi di bot differenti, e molto spesso gli sviluppatori utilizzano agenti conversazionali programmabili tramite framework, come BOT FRAMEWORK di Microsoft, che permettono di scegliere la miglior configurazione del Bot.

Difatti anche nella scelta del *framework*bisogna fare attenzione, in quanto ogni *tool* ha degli aspetti positivi e negativi. Ci sono framework che permettono di analizzare il linguaggio naturale più facilmente di altri, oppure che permettono facilmente di integrare i bot ad altri software, difatti bisogna tener in considerazione questo *trade-off*.

D'altro canto può sembrare che gli risolvono gran parte dei problemi degli sviluppatori, però come riescono a risolvere queste difficoltà ne portano di nuove. In particolare un esempio che fa comprendere questi problemi è proprio il sovraccarico di informazioni che si crea nei canali di comunicazione fra team diversi. Un secondo problema che si presenta frequentemente è la scarsa elaborazione del linguaggio naturale, o addirittura le difficoltà di integrazione dell'agente.

Queste sono solo alcune delle tante *challenge* che gli sviluppatori si trovano ad affrontare quando realizzano un agente conversazionale. Alcune di queste vengono risolte dai vari *framework* utilizzati per l'implementazione dei bot, alcune invece vengono affrontate utilizzando *best practice*. Prima di sviluppare un bot però bisogna fare uno studio sulle *challenge* maggiormente presenti nello sviluppo degli agenti conversazionali.

2.2 Tool per lo sviluppo dei bot

Tool e *framework* sono essenziali per lo sviluppo dei bot, questo perché come già è stato detto aiutano ad affrontare molte delle *challenge* che si incontrano nello sviluppo degli agenti conversazionali. Negli ultimi anni con la grandissima espansione dell'utilizzo dei bot, sono nati tantissimi *framework* che supportano lo sviluppo di questi ultimi.

Ci sono tool realizzati da grandi colossi come Microsoft con "Microsoft Bot Framework", "Dialogflow" di Google fino a "IBM Watson Assistant" di IBM. A questi si aggregano tantissimi altri *framework Open Source* che offrono degli ottimi servizi. Un esempio è proprio "Rasa" che oltre ad offrire un tool per l'apprendimento fornisce un supporto per l'integrazione.

Come si osservare dalla Tabella 2.1, esistono svariati *framework* che supportano gli sviluppatori durante la fase di progettazione e sviluppo degli agenti conversazionali. Analizzandoli in modo più accurato:

Microsoft Bot Framework: è uno dei *framework* più sfruttati nel campo dei Bot. Esso fornisce una vasta gamma di servizi, SDK e strumenti per la realizzazione di agenti intelligenti.

Integrazioni:

Framework	Linguaggio di programmazione
Microsoft Bot Framework	C#, Java, JavaScript e Python
Rasa	Python
Wit AI	Python,Ruby
Dialogflow	Node js
IBM Watson	Java, C++
Botpress	JavaScript
BotKit	Node js
ChatterBot	Node js,JavaScript

Tabella 2.1: Framework per lo sviluppo di agenti conversazionali.

- Slack
- Facebook Messenger
- Website
- Cortana
- Skype
- Microsoft teams

Rasa: "Rasa" è un *framework Open Source* e possiede due componenti principali:

1. Il primo è "Rasa NLU", il quale si occupa della comprensione del linguaggio naturale.
2. Il secondo invece è "Rasa CORE" che riesce a fornire al bot quella caratteristica intelligente che lo differenzia dagli altri.

Uno degli aspetti positivi di questo *framework* è che agevola gli sviluppatori anche nel cercare dati per il bot. Infatti Rasa riesce proprio ad eseguire l'addestramento con pochissimi dati.

Integrazioni:

- Slack
- Facebook Messenger
- Telegram

Wit.AI: "Wit.AI" anche esso è un *framework Open Source*, ma dedicato al processo di *Natural Language Processing*.

Difatti permette agli sviluppatori di creare facilmente entità e intenti.

Integrazioni:

- Website
- Facebook Messenger
- Slack

Dialogflow: questo *framework* è stato lanciato sul mercato da *Google*. Esso offre molti servizi come ad esempio apprendimento automatico, *Natural Language Processing* e integrazioni con molte piattaforme.

Uno degli aspetti più interessanti di questo tool è il poter utilizzare i modelli di apprendimento automatici forniti da *Google*. Questo infatti permette più facilmente di addestrare il bot.

Integrazioni:

- Slack
- Facebook Messenger
- Alexa
- Cortana
- Skype
- Telegram
- Viber

IBM Watson: *Watson* è una delle piattaforme più conosciute e utilizzate dagli sviluppatori. Ciò che lo rende così noto sono i suoi strumenti che aggiungono flessibilità e facilitano l'integrazione del bot con altre piattaforme.

Integrazioni:

- Slack
- Facebook Messenger

Botpress: è un *framework Open Source* e si basa su un'architettura modulare. Possiede molte funzionalità come editor, *Natural Language Processing*, analisi e utilizzo multicanale. Il vantaggio che porta questo tool è che gli sviluppatori possono lavorare al bot in locale, testarlo e poi eseguire il servizio di *hosting* su un server.

Integrazioni:

- Telegram
- Facebook Messenger
- Website

BotKit: BotKit è una piattaforma di chatbot *Open Source* ormai acquisita da Microsoft. Essa si basa su un motore di elaborazione del linguaggio naturale, andando a semplificare il lavoro per gli sviluppatori.

Integrazioni:

- Slack
- Cisco Webex
- Microsoft Teams
- Facebook Messenger

ChatterBot: "ChatterBot" è un *framework* che utilizza algoritmi di *Machine Learning* che automatizzano l'intero flusso di lavoro. La scelta di questi algoritmi permette al bot di essere istruito su qualsiasi lingua.

Integrazioni:

- Facebook

Si può notare che ci sono tantissimi *framework* e tool che permettono facilmente di creare un bot o un chatbot. Bisogna semplicemente analizzare bene il problema in questione e capire quale *framework* possa essere migliore.

2.3 Challenge dei bot

Dopo più di 50 anni che Weinzebaum ha introdotto il primo *software* in grado di avere una conversazione con gli essere umani, i bot sono diventati il canale di comunicazione principale tra servizio ed essere umano. La potenza degli agenti è stata alimentata dall'intelligenza artificiale e dalla migliore elaborazione del linguaggio naturale. Difatti ai giorni d'oggi vengono applicati in assistenti personali, assistenza sanitaria, assistenti di aziende disponibili h24. Tutta questa popolarità però implica che lo sviluppo e il mantenimento dei chatbot richiedono sempre più attenzioni.

Lo sviluppo di un agente conversazionale richiede delle competenze specializzate, come ad esempio l'apprendimento e l'elaborazione del linguaggio naturale, che lo rende differente dallo sviluppo del software tradizionale. Queste complessità sono presenti sia a livello di sviluppo sia a livello di utilizzo dei bot e in letteratura sono narrate molte delle *challenge* che gli sviluppatori si trovano ad affrontare. Alcune di queste sono agevolate dai vari *framework* utilizzati per lo sviluppo,

come ad esempio Microsoft Bot Framework oppure Rasa. Entrambi vanno a semplificare il lavoro di comprensione del linguaggio naturale. Questi da soli però non bastano poiché bisogna impiegare anche delle *best practice* per far si che l'agente abbia dei comportamenti adeguati al luogo che lo circonda.

Per comprendere al meglio le *challenge* relative ai bot, inizialmente è stata fatta una ricerca su vari *paper*, in particolare è stato interessante come in uno lavoro di ricerca [2] è stata effettuata una suddivisione in macro-aree delle *challenge* che interessavano gli agenti conversazionali. In questo documento la suddivisione è stata fatta per cinque argomenti: "integrazione", "sviluppo", "Natural-language understanding", "integrazione utente", "input utente". È stato deciso inoltre di approfondire queste *challenge*, analizzando altri paper trovati in letteratura. Per di più è stato alzato il livello di astrazione in modo da dividere le *challenge* in due grandi macro-aree:

Development challenges: le challenge di "sviluppo" possono essere viste come *challenge* che riguardano ad esempio il processo del linguaggio naturale, testare i bot, costi di sviluppo, assicurare la sicurezza dei dati e l'integrazione.

Use challenges: le challenge di "utilizzo" includono concetti relativi all'accessibilità intesa come accessibilità possibile a tutti gli utenti, concetti etici, il rumore e l'interruzione provocata dagli agenti conversazionali fino ad azioni sbagliate.

Dalla suddivisione in macro-aree, è stata presa la decisione di analizzare tutte le possibili *challenge* che appartengono all'area di *Development challenges* e tutte quelle che fanno parte dell'area *Use challenges*.

CAPITOLO 3

Un'analisi della letteratura per le challenge dei chatbot nello sviluppo software

3.1 Metodologia

Per affrontare lo studio sulle challenge dei bot, è stata condotta una ricerca su diverse tipologie di letteratura. Per la "Letteratura primaria" sono stati presi in considerazione i *paper* pubblicati dalle conferenze o da riviste importanti. Nonostante ciò è stato necessario basarsi anche sulla "Letteratura secondaria", infatti sono state approfondite le sfide utilizzando *community* e *blog* dove l'argomento principale di discussione era lo sviluppo di agenti conversazionali.

L'approccio usato durante la ricerca di documenti è definito *Multivocal Literature Review*. Esso infatti è stato utile a perfezionare i risultati ottenuti durante fase di raccolta, ma in particolare ha fornito più punti di vista dello stesso problema.

Per di più è stata adottata questa strategia per cercare di fondere tutti i risultati ottenuti dall'analisi della "Letteratura primaria" e della "Letteratura secondaria". In questo modo è stato possibile categorizzare la maggior parte delle *challenge* che riguardano lo sviluppo dei bot.

Lo studio effettuato sulle *challenge*, come si può notare dalla Figura 3.1, comprende 4 fasi:

1. **Raccolta:** in questa fase come già accennato è stata svolta una ricerca sulla "Letteratura primaria" usando *paper* di conferenze e riviste. Inoltre è stata svolta una ricerca anche sulla "Letteratura secondaria", consultando blog e articoli postati da community di sviluppatori. Durante questo *step*, a seguito della letteratura di riferimento, sono stati impiegati diversi approcci:

   Letteratura primaria: per la "Letteratura primaria" come motore di ricerca è stato sfruttato *Google Scholar*. Il vantaggio è che è stato possibile effettuare ricerche utilizzando *query* e *keyword*. Come *output*, GOOGLE SCHOLAR, fornisce paper selezionati da interessanti

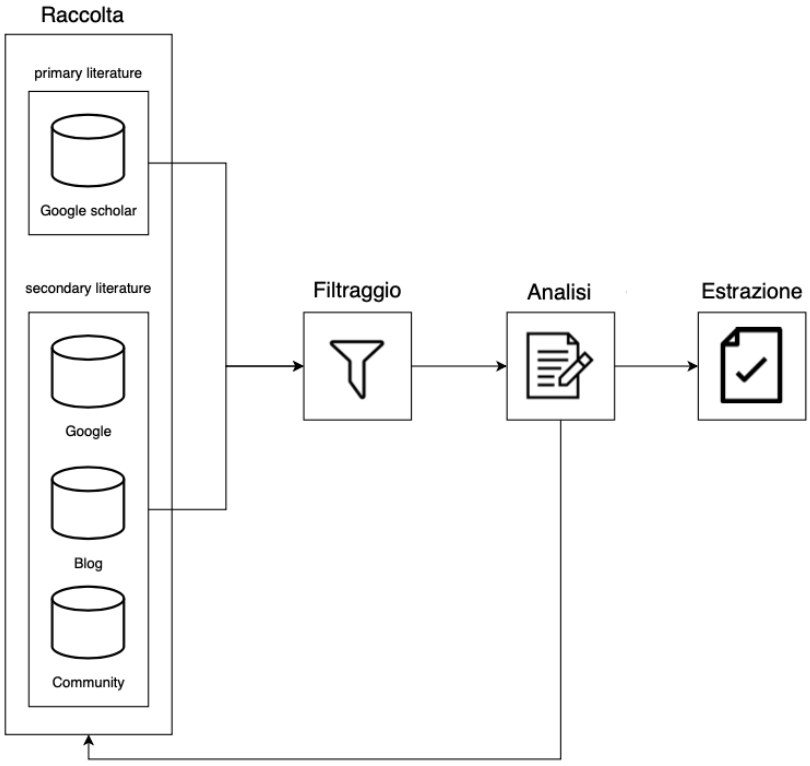

Figura 3.1: Lavoro svolto.

database come IEEEXPLORE o SCOPUS. Per la ricerca dei vari paper è stata usata la seguente *query*:

> ('development challenge' OR 'challenge' OR 'problem' OR 'issue') AND ('bot' OR 'chatbot' OR 'convesional agent')

La query usata per la ricerca ha fornito paper che discutevano di come i bot affiancano gli sviluppatori durante lo sviluppo *software*[20] [3].

Inoltre però ha fornito documenti interessanti, i quali discutevano del argomento principale dal quale è stata avviata l'attività di ricerca : "le challenge dei bot nello sviluppo software" [11] [2].

Letteratura secondaria: per la "Letteratura secondaria" come motore di ricerca principalmente è stato utilizzato GOOGLE, ma sono state scelte anche delle community o blog dove approfondire ricerche. Una delle community dove sono stare riscontrate più informazioni è quella di MICROSOFT.

Inoltre un esempio di blog molto interessante è stato "Chatbots Magazine" nel quale sono stati trovati vari articoli su come realizzare bot e da qui sono state estrapolate delle *challenge* interessanti.

In questo caso oltre alla query usata nella "Letteratura primaria", sono state utilizzate ulteriori frasi per migliorare la ricerca:

> Best practices for bot development

> Best practices for addressing bot challenges

> Positive and negative aspects of bots in software development

2. **Filtraggio:** per filtrare la documentazione sono stati scelti dei criteri di inclusione ed esclusione. In questo modo sono stati scelti i documenti che più erano opportuni al problema in questione. Come per la raccolta, anche in questo caso, i criteri utilizzati per filtrare i documenti si dividono in base alla tipologia di letteratura di appartenenza.

   Per quanto riguarda la "Letteratura primaria" sono stati adottati vari criteri per l'inclusione di un *paper* come:

   - Articoli che nel *abstract* centravano in pieno la ricerca che si stava effettuando.
   - Articoli che discutevano con spirito critico dei difetti degli agenti conversazionali [19].

   Invece sono stati adottati criteri di esclusione come:

   - Articoli che nel *abstract* facevano capire che nel paper veniva solo riportato qualche citazione ai bot.

   Portando l'attenzione sulla "Letteratura secondaria", il criterio più importante è stato cercare delle fonti attendibili. Per far ciò è stato valutato l'articolo trovato, il blog o la community dove è stata recuperata l'informazione.

3. **Analisi:** durante la fase di analisi dei documenti, veniva esaminato tutto il paper per comprendere quel era l'argomento centrale.

   Contemporaneamente alla lettura se qualche concetto risultava intrigante veniva annotato in modo da riprenderlo in un secondo momento. Inoltre se si trattava di "Letteratura primaria" si faceva particolare attenzione alle citazioni fatte. In questo modo se veniva individuato un documento interessante veniva riportato nella fase di "Raccolta" per poi proseguire con la fase di "Filtraggio".

   Un ulteriore aspetto fondamentale che è stato analizzato sono gli autori del paper, in quanto potevano aver scritto ulteriori articoli potenzialmente interessanti sullo studio corrente. Questa fase è stata molto importante in quanto ha permesso la collezione di molti paper grazie all'analisi delle citazioni. In più è stata necessaria per l'ultima fase ovvero quella di estrazione, dove è stata realizzata una categorizzazione delle *challenge* reperite durante l'analisi.

4. **Estrazione:** come step finale della ricerca, c'è la fase di estrazione. In quest'ultima fase l'obiettivo è stato estrarre le *challenge* dai paper/articoli.

   In particolare si esaminavano le parti più interessanti dei documenti trovati nella "Letteratura primaria" e si procedeva in due modi:

   - Se nel paper venivano esplicitamente elencate le *challenge* da affrontare venivano solo categorizzate.
   - Altrimenti se nel paper si discutevano di problematiche degli agenti conversazionali in linea generale, quindi senza definire una vera e propria *challenge*, si procedeva a comprendere il problema da affrontare per poi classificarlo.

Portando un esempio di estrazione, in un paper si discuteva di *testing* di bot per il *Natural Language Processing* [4]. In questo documento venivano riportate diverse *challenge* come: *Regression testing*, *Performance testing* e *Usability and Human Computer Interaction testing*. Tutte queste sfide sono relative ai vari test che vengono effettuati sugli agenti, ma si è preferito avere un'unica *challenge* ovvero "Test di Bot/Chatbot".

Per quanto riguarda la "Letteratura secondaria", l'estrazione delle informazioni risultava più immediata. Difatti applicando i filtri elencati durante la fase di "Raccolta", la maggior parte dei documenti ottenuti erano degli elenchi puntati di *challenge*. L'unico aspetto a cui si doveva fare più attenzione erano le sfide analoghe tra di loro. Un esempio è proprio questo:

- Varietà nel modo in cui le persone digitano i loro messaggi.
- I diversi modi in cui gli esseri umani parlano.
- Comportamento umano imprevedibile, stati d'animo ed emozioni.

Nel esempio riportato, si può notare come queste tre sfide possono far parte della *challenge* del "Natural Language Processing".

3.2 Risultati

Questa sezione riassume i risultati della ricerca effettuata sulla letteratura primaria e secondaria.

Dalla revisione della "Letteratura primaria" e "secondaria" è stato riscontrato come tutte le *challenge* estratte si potevano categorizzare in due macro-aree. Come si può notare dalla Tabella 3.1, la quale è stata divisa in *"Gruppo"* e *"Nome"*, i gruppi sono due:

1. **Development challenge:** la macro-area *development challege* è stata ricavata in quanto durante la fase di "Analisi" del materiale sono state riscontrate tantissime *challenge*, le quali riguardano lo sviluppo degli agenti conversazioneli.

   In quest'area sono state identificate circa nove sfide :

   High on data/high data quality: la definizione di questa *challenge* è sorta proprio durante l'analisi di uno studio, nel quale si erano occupati di realizzare un bot conversazionale che aiutava i nuovi sviluppatori ad integrarsi a progetti *Open Source* [7].
   Purtroppo in conclusione nella ricerca era sorto un problema. La difficoltà in questione era che l'agente conversazionale si basava su conversazioni fra sviluppatori. Queste ultime però erano poche affinché il bot desse risultati positivi.
   Il seguente problema, portato ad un livello più alto di astrazione, quindi fuori da quel contesto applicativo ha portato ad un'osservazione. La riflessione era che tutti i bot, per dare una buona predizione, devono essere allenati con una grande mole di dati, e sopratutto con dei dati di alta qualità.

   Natural Language Processing: la seguente *challenge* è stata categorizzata in quanto è stato notato che un qualsiasi bot si troverà prima o poi a discutere con un essere umano. Per far si che l'esperienza dell'essere umano sia unica, vi è bisogno di un agente conversazionale che riesca a capire ciò che l'utente *intende*.

Gruppo	Nome
Development challenge	High on Data/high data quality
	NLP(Natural Language Processing)
	Test of Bot/ChatBot
	Maintenance
	Development cost
	Secure data
	Machine learning
	Respond slowly
	Integration
Challenge for use	Privacy concerns
	Accessibility
	Noise
	Ethics
	trust/reliability
	Interruption
	Choosing a bot / Configuring a bot
	Lack of understanding of intent
	Too long answers
	Solve everything
	Lack of information about the bot
	Wrong actions/wrong information
	Usability

Tabella 3.1: Challenge ottenute dalla revisione della letteratura.

Il *Natural Language Processing* si occupa proprio di questo, ed è conosciuto come l'insieme delle tecniche che permettono di processare il linguaggio naturale.
Durante l'analisi difatti sono emersi molteplici *blog* nel quale tra le sfide più importanti veniva inserita proprio questa. Difatti un *blog* [G9] discuteva di quanto importante è decidere il modello di *Natural Language Processing*, prima che l'agente venga realizzato. Ciò vuol dire che bisogna prima esaminare quali modelli per l'elaborazione del linguaggio esistono sul mercato, e poi in base alla decisione presa implementare il giusto bot.

Test of Bot/ChatBot: testare i bot risulta essere un processo molto complesso. Questo perché gli agenti conversazionali sono in continua evoluzione grazie all'aggiornamento delle loro componenti interne. Quindi il processo di *testing* dei bot deve essere fatto frequentemente, in modo da tener conto nel tempo dell'accuratezza di questi ultimi.
Per agevolare il lavoro di *testing*, sono state proposte da un articolo [G1] delle piattaforme che permettono l'automatizzazione dei test sugli agenti. Sono state citate piattaforme come DIMON, ZYPNOS e BOT TESTING, le quali consentono agli sviluppatori di ottenere *report* e risultati dettagliati utili per i vari casi di test.

Maintenance: la manutenzione è una *challenge* molto importante. Si sa bene che i bot lavorano con dati, e banalmente può succedere che modificando questi ultimi si buca anche l'interfaccia dell'agente stesso. Questo citato è solo uno dei tanti esempi che fa capire quanto è importante la manutenzione del bot dopo la sua distribuzione.
Inoltre in un articolo dove è stata studiata la potenza dei bot nei progetti *Open Source*, fra le *challenge* più importanti c'è proprio la manutenzione [18]. Nel *paper* appena menzionato, i ricercatori infatti avevano affermato che spesso gli agenti vengono sviluppati in modo instabile, quindi richiedono una manutenzione elevata affinché non generano disagi.

Development cost: è stato notato come i costi di sviluppo vengono considerati come una *challenge*, in quanto la necessità di formazione di un bot negli anni sta diventando sempre più costosa. Questo perché come già è stato detto l'agente deve avere un comportamento quanto più simile a quello umano.
Durante l'analisi [G9], è stato notato come i costi si presentano anche in altre forme, poiché gli agenti conversazionali possono essere integrati a tantissime piattaforme, come siti web o applicazione. Se si desidera implementare diversi bot per le diverse applicazioni, è richiesta ancora più codifica e di conseguenza alte spese.
Questa sfida non è assolutamente da sottovalutare, difatti durante l'analisi sono emersi molti *blog* e *community* [G2][G8] nei quali veniva discussa questa *challenge*.

Secure data: nell'ambito dello sviluppo dei bot, la sicurezza è uno dei argomenti principali. Infatti durante lo sviluppo degli agenti conversazionali, si deve garantire che abbiano almeno lo stesso livello di sicurezza del sito/applicazione in cui essi si trovano.
Questo argomento è molto importante poiché spesso le conversazioni possiedono dati altamente sensibili e informazioni personali. Per salvaguardare questi dati, durante l'analisi di un blog [G9], sono stati citate delle tecniche utili come :

- Utilizzare protocolli "http" nel bot;

- Assicurarsi che le piattaforme che ospitano l'agente siano sicure.
- Controllare che il bot possieda meno informazioni possibili al suo interno, in modo da evitare che le persone possano manipolarlo per ottenere tali informazioni.

L'importanza che viene posta alla sicurezza dei dati è dovuta in quanto questi ultimi portano a problemi di alto calibro. Difatti ci sono articoli che consigliano vivamente di possedere quanti meno dati sensibili all'interno del bot [G10].

Machine learning: il *Machine learning* è un ulteriore *challenge* da affrontare durante lo sviluppo dei bot. Quando si parla di *Machine learning* applicato nell'ambito dei bot, si intende realizzare un agente che, con il tempo, riesce a raffinare le risposte da dare agli utenti con cui interagisce.

Molti *paper* cercano difatti di espandere questi concetti di intelligenza artificiale, affinché si possano realizzare agenti capaci di apprende in modo autonomo [11].

Respond slowly: la generazione di risposte è un ulteriore sfida che gli sviluppatori devono affrontare durante lo sviluppo. Spesso gli utenti hanno un intervallo di tempo limitato per le loro domande e si aspettano risposte fulminee. Inoltre risulta molto difficile far mantenere l'attenzione dell'utente fino alla fine della discussione.

Ciò richiede lo sviluppo di agenti con abilità e funzionalità straordinarie, in grado di generare risposte brevi ma allo stesso tempo con un senso logico. Da un *blog* è emerso come l'interfaccia utente svolge un ruolo fondamentale [G6], in quanto va a rendere una conversazione molto più umana sotto alcuni punti di vista, e va ad alleviare i tempi di risposta.

Integration: è stato notato che spesso la realizzazione di un bot avviene per alleggerire i carichi di lavoro, difatti l'agente viene del tutto affiancato ai flussi di lavoro. Un esempio sono le potenzialità che offre un bot nella realizzazione di progetti *Open Source* [7].

Con il passare del tempo l'integrazione degli agenti conversazionali avviene da qualsiasi parte, come siti web, social network, oppure canali di comunicazione per lo sviluppo, ad esempio *Slack*. Questi sono solo alcuni dei tanti esempi di integrazione, da ciò si può notare come i bot, quasi sempre, sono affiancati ai *software*. L'integrazione però non risulta del tutto semplice. Difatti i risultati di una ricerca hanno dimostrato come quest'ultima fosse una fonte di frustrazione per gli sviluppatori [6].

Tutto ciò rende l'integrazione ad essere una "challenge".

2. **Use challenge:**la macro-area *use challenge* è stata individuata durante l'analisi della documentazione, calandosi nei panni di coloro che utilizzano i bot. La maggior parte di queste sfide comprendono concetti di "privacy", "concetti etici" e "cattive azioni". Di seguito sono riportate tutte le *challenge* ricavate dalla ricerca.

   Privacy concerns: i problemi di *privacy* come si può notare sono stati inseriti anche nella macro-area *Use challenge*. In tal caso i bot durante le loro applicazioni spesso raccolgono feedback e dati comportamentali in modo da migliorare la conversazione.

   In oltre è stato realizzato una ricerca sugli assistenti personali, come Siri, e del loro impatto con gli utenti [6]. Da qui sono nate molte sfide, una delle quali riguardava

i problemi di *privacy*. Difatti, analizzando il documento, è stato notato che gli utenti classificavano le informazioni sanitarie e bancaria come dati sensibili.

Queste ultime non sono state però classificate sensibili tanto quanto i contatti personali o ad esempio l'annuncio di un evento importante. Questo fa notare che non è semplice far utilizzare nella vita quotidiana un agente conversazionale ad un qualsiasi utente, poiché si ha a che fare con questi problemi.

Accessibility: l'accessibilità viene intesa come sfida in quanto possono esserci tantissimi utilizzatori di un bot. Quindi bisogna garantire l'accessibilità a tutti i potenziali utenti, indipendentemente dalla situazione lavorativa, delle capacità o delle competenze linguistiche.

Nella ricerca delle *challenge*, un blog [G5], consigliava di adottare delle *best practices* di progettazione per far si che l'interfaccia dell'agente soddisfi tutti i requisiti appena citati.

Noise: fin ora è stato discusso parzialmente del rumore generato dagli agenti, sia nella comunicazione umana che nel flusso di lavoro.

Un esempio di "rumore" è stato anche analizzato in letteratura. In particolare sono stati posti dei quesiti agli sviluppatori sull'integrazione dei bot nei progetti e del loro impatto [20]. Le risposte ottenute hanno affermato come effettivamente l'integrazione degli agenti, nei canali di comunicazione, portano non pochi problemi.

In quella determinata ottica i problemi più fastidiosi erano la verbosità e l'alta frequenza con cui i bot svolgevano le azioni. Tutto questo rumore introdotto andava a produrre ulteriori problemi, ad esempio l'interruzione del flusso lavorativo dei sviluppatori, per controllare cosa stava accadendo.

Ethics: durante l'analisi dei *paper*, è stato osservato come i problemi "etici" erano un altro grande ostacolo degli agenti. Uno dei tanti problemi può essere generato dal comportamento intrusivo del bot.

Infatti è stato notato che spesso gli agenti venivano integrati in progetti *Open Source*. Da ciò quindi potevano modificare i commit o le richieste pull [20], ma questo era solo uno dei piccoli problemi "etici" da affrontare. Si può immaginare che questo problema sia ancora più grave se ad esempio un agente iniziasse a diffondere commenti razzisti su un canale di comunicazione o su un *social network*. Proprio per questo i problemi etici sono stati categorizzati come *challenge* da affrontare.

Trust/reliability: fiducia e affidabilità sono la base con cui i bot devono lavorare. Questo perché di solito gli agenti vengono utilizzati anche dalle aziende come guida per i nuovi clienti che visitano il loro sito. Durante tutto ciò l'azienda deve essere sicura di avere un agente affidabile, che riesca a realizzare delle conversazioni nelle quali fornisce risposte sensate.

A quest'esempio è possibile legarne un altro, difatti durante la l'analisi delle *challenge* è stato di particolare interesse un esempio di affidabilità che richiedono le aziende nei confronti degli agenti conversazionali [8]. Le aziende in effetti cercavano di ottenere agenti che in qualsiasi situazione, addirittura di guasti *hardware* e *software*, continuassero

a ricoprire il loro ruolo.

Quest'ultimo esempio fa riflettere molto su come è importante affrontare questa *challenge* durante lo sviluppo di questi ultimi.

Interruption: l'interruzione è spesso causata dall'alto tasso di messaggi che genera un agente. Difatti questa *challenge* proviene proprio dalla sfida relativa al "rumore" creato dai bot [20].

Un esempio per comprendere al meglio ciò che accade, è proprio quando ad uno sviluppatore viene affiancato un agente con lo scopo di lavorare insieme ad un progetto *software*. Quando questo agente inizia a inviare messaggi continui allo sviluppatore, porta lo stesso a interrompere il proprio lavoro per dedicarsi a tutti i messaggi ricevuti. L'interruzione generata in effetti porta solo aspetti negativi nello sviluppo, di conseguenza genera della frustrazione nello sviluppatore, la quale si propaga nel codice. Quindi è importantissimo cercare di affrontare questa sfida, in modo da migliorare il comportamento del bot e successivamente di alleviare il grado di frustrazione degli sviluppatori.

Choosing a bot/configuring a bot: scegliere un Bot o una configurazione può sembrare semplice ma non lo è. Quando si vuole creare un agente, la prima grande sfida da affrontare è capire dove sarà utilizzato. Questo perché possono esserci tantissimi modi per configurare un agente conversazionale in base al suo utilizzo. Infatti possono esserci delle scelte progettuali come la selezione del modulo per "processare il linguaggio naturale", il quale verrà allenato in base al contesto in cui sarà utilizzato il bot.

La scelta della configurazione infatti è una delle tante strade che genera il fallimento di un agente, poiché esso deve allinearsi alle priorità e obiettivi del contesto che lo circonda [G7]. Dopotutto gli agenti conversazionali sono responsabili del successo a cui si punta, e non scegliere un giusto bot porta al non raggiungimento di determinati obiettivi.

Lack of understanding of intent: nonostante il loro accurato sviluppo, gli agenti spesso non riescono a riconoscere l'intento dell'utente. Sicuramente anche questa è una delle *challenge* più importanti. Difatti può succedere che l'utente interagisce con il bot in uno stato di frustrazione [G11]. Ciò può causare un peggioramento della comprensione dell'intento da parte dell'agente.

A seconda dell'ambito in cui il bot si trova, questa vulnerabilità può rilevarsi dannosa per il proprietario. Ad esempio una conversazione deludente, con un agente che funge da assistente alle vendite può allontanare i clienti.

Too long answers: un'ulteriore *challenge* è la generazione di risposte troppo lunghe da parte del bot. Difatti un *paper* [3] discuteva di come risposte troppo lunghe, che contengono più informazioni nascoste, possono provocare lo scoraggiamento dell'utente e l'interruzione della conversazione.

Quindi bisogna affrontare questa sfida magari aiutandosi con delle *best practice* che migliorano la creazione delle frasi. Difatti bisogna generare frasi più compatte in modo da invogliare l'utente a proseguire la conversazione.

Solve everything: i bot non sanno "ancora" risolvere tutto. Ciò significa che di solito gli

agenti vengono realizzati per essere affiancati al lavoro umano. Infatti hanno uno specifico contesto applicativo fuori dal quale non hanno nessun potere. Questa *challenge* è stata realizzata per far comprendere all'utente che utilizza il bot, che questi ultimi possiedono dei limiti che non possono oltrepassare, ancora per ora.

Questa sfida è stata discussa anche da un blog [G3], è una delle soluzione che venivano proposte era aggiungere un aiuto umano per il bot. Per far fronte a questa *challenge*, è stato pensato di inoltrare le richieste che non sono gestibili stesso dal bot ad un umano, in modo da fornire una risposta al cliente che aveva richiesto tale quesito.

Lack of information about the bot: avere poche informazioni sul bot che si sta utilizzando porta a non usarlo nel modo giusto [19]. Infatti durante la progettazione del agente si cerca di affrontare questa sfida. Uno dei tanti modi per far ciò è generare un messaggio per l'utente in modo da fornire una sintesi di tutto ciò che il Bot può fare.

Wrong actions/wrong information: spesso si pensa che gli agenti siano innocui, ma non è sempre cosi. Spasso succede che un bot, progettato male, durante una conversazione inizia a fornire informazioni sbagliate o addirittura a svolgere cattive azioni. Tutto questo spesso è causato da una cattiva progettazione, ma come è stato detto ci sono alcuni agenti capaci anche di apprendere dalle conversazioni.

Difatti inevitabilmente possono esserci delle conversazioni che creano questi problemi, ovvero alimentano gli agenti di informazioni errate.

Durante la ricerca è emerso un blog [G4], il quale evidenziava che molti bot possiedono dei dati non corretti. Pur essendo che questi dati vengono divulgati tramite una bella interfaccia rimangono però contenuti sbagliati. Quest'ultimo esempio fa capire bene come è importante non dare informazioni errate di base al bot, questo perché stesso lui continuerà a divulgare tali informazioni e ad apprenderne nuovamente informazioni sbagliate.

Usability: l'ultima *challenge* presente in questa macro-area ha come obiettivo di ampliare le funzionalità dei bot. Questo perché gli agenti devono aggiungere valore al contesto in cui sono utilizzati. Purtroppo spesso accade che quando si utilizza un bot per acquistare qualcosa, quest'ultimo riporta il cliente al sito web per entrare nella fase di pagamento. Quest'azione di solito non è molto gradita dagli utenti e porta ad avere un lunga conversazione senza un obiettivo finale.

Questo citato è solo uno degli *use-case* che un agente può affrontare, difatti un articolo ha fornito ha fornito ulteriori esempi di applicazione di un bot come:

- Programmazione e progettazione degli appunti;
- Servizi di identificazione e autenticazione;
- Sondaggi;
- Pagamenti e transazioni;

Gli esempi appena citati fanno ragionare molto su quella che è l'usabilità dei bot, la quale non viene utilizzata al 100%.

Fra tutti i *paper* visionati per la categorizzazione delle *challenge*, è risultato molto interessante un sondaggio basato su 43 utenti Github, i quali erano coinvolti in progetti *software bot*[10]. Di

questa ricerca è risultato interessante il feedback della domanda "*What challenges bot developers face when developing bots?*", in quanto gli sviluppatori hanno segnalato diverse problematiche. Le più importanti erano legate agli strumenti usati per lo sviluppo degli agenti conversazionali, e sopratutto alla ricerca di informazioni per l'implementazione di questi ultimi. Da ciò è stato dedotto che pur essendoci i mezzi che supportano lo sviluppo degli agenti, vengono a mancare però le informazioni che permettono di realizzare un bot di qualità.

Per tali motivi, è stato deciso di realizzare un agente conversazionale che si occupava di colmare alcune lacune e aspetti poco evidenti dello sviluppo di questi ultimi. Infatti l'obiettivo del bot era in primis di affiancare gli sviluppatori e allo stesso tempo essere una fonte di informazioni per gli stessi.

L'agente infatti tramite una semplice interfaccia utente, era capace di dare *best practices* per affrontare *challenge* appartenenti alle macro-aree definite. Inoltre l'utilizzo del bot è stato progettato per integrare nuove sfide che vengono a galla ogni giorno e metterle a disposizione di altri sviluppatori.

Il fulcro della questione era che essendo un *tool* condiviso fra sviluppatori, loro stessi potevano contribuire all'incremento della quantità e qualità di informazioni nel bot. Di conseguenza far si che l'agente desse consigli sempre migliori.

CAPITOLO 4

[Stephen] ConverSional agenT to provide bEst Practices for developing otHer convErsioNal agent

4.1 Obiettivo del tool

Nei capitoli precedenti è stato discusso di come viene realizzato un bot, dalle varie tipologie di agenti che si possono trovare e sopratutto del loro impatto nell'utilizzo quotidiano. Di particolare interesse sono state le *challenge* che vengono affrontate dagli sviluppatori nell'utilizzare o realizzare un agente.

Nel terzo capitolo è stata svolta un'analisi in letteratura, sia primaria che secondaria, in modo da poter classificare le *challenge* che più interessano i bot. Svolta la ricerca e confrontato i risultati ottenuti, sono state ottenute ben 22 sfide. La grande quantità di queste *challenge* ha suscitato l'idea di implementare un bot che affianchi gli sviluppatori durante lo sviluppo di questi ultimi.

L'obiettivo del tool infatti è di affiancare gli sviluppatori durante lo sviluppo, fornendo informazioni e *best-practice* per i bot. Un esempio che può rendere le idee più chiare è il seguente: può accadere che durante lo sviluppo di un agente, che poi dovrà essere integrato in un *social network*, uno sviluppatore si imbatte in problemi etici. Senza essere affiancato da Stephen, dovrebbe interrompere lo sviluppo per cercare delle tecniche per affrontare questo problema. Però se allo sviluppatore viene affiancato Stephen, tramite delle semplici domande, riesce immediatamente ad ottenere delle *best-practice* per affrontare quella *challenge*.

Come si può notare, l'utilizzo di questo tool permette al programmatore innanzitutto di risparmiare tempo, ma sopratutto permette di non allontanarsi troppo dal focus principale, ovvero sviluppare l'agente.

Il tool in questione per ognuna delle *challenge* trovate possiede delle *best-practices* che aiuteranno lo sviluppatore a realizzare un bot che soddisfi tutti i requisiti necessari.

	Azure Bot Framework	Rasa
Linguaggio di programmazione	.NET, Java, JavaScript, Python	Python
Pro	-Facile integrazione con altri software. - Facile integrazione con il servizio di NLP Luis. - Per sviluppare il bot viene utilizzato un approccio programmatico.	- La componente Rasa NLU aiuta la comprensione del linguaggio naturale. - La componente Rasa Core aiuta a creare chatbot intelligenti. - Altamente personalizzabile.
Contro	- Non adatto ai principianti.	- Richiede di avere delle conoscenze di NLP.

Tabella 4.1: Framework.

4.2 Tecnologie utilizzate

Lo sviluppo di Stephen ha implicato l'analisi e la scelta di varie tecnologie.

Linguaggio di programmazione: come linguaggio di programmazione è stato scelto PYTHON, in quanto risulta essere un linguaggio molto semplice ma allo stesso tempo potente. È stato scelto poiché fra tutti i linguaggi è quello maggiormente usato dai vari framework per lo sviluppo di agenti conversazionali. Infatti nella Tabella 4.1, si può notare come entrambi i framework scelti per lo sviluppo usano "Python" come linguaggio di programmazione.

Framework: per lo sviluppo del bot è stato utilizzato un framework che facilitasse lo sviluppo dello stesso. Come si può notare dalla tabella 4.1 la scelta ricadeva su :

Azure bot framework: esso fornisce tutti gli strumenti per la compilazione, distribuzione e gestione di bot intelligenti. Questo tool ha un approccio più programmatico, infatti le conversazioni si basano su dei *dialog*, ovvero classi che hanno delle funzioni a cascata, le quali vanno a creare un flusso di conversazione.

Inoltre AZURE BOT FRAMEWORK permette l'integrazione di LUIS, ovvero un servizio di *Natural Language Processing* offerto sempre da MICROSOFT. Questo servizio permette al bot di elaborare il linguaggio naturale, per poi estrarre l'intento e l'entità che verranno usati per eseguire delle azioni.

Oltre ad elaborare il linguaggio naturale, LUIS permette di rendere l'agente più intelligente, infatti con il tempo riesce a migliorare le proprie prestazioni.

Rasa: questo è stato il secondo *framework* candidato per l'implementazione del bot. Rasa anche semplifica lo sviluppo degli agenti, ma a differenza di AZURE BOT FRAMEWORK è un tool meno programmatico. Questo vuol dire che invece di basarsi su dei "dialogi a cascata", esso si basa su delle "storie" e da lì gestiste le varie conversazioni.

Inoltre Rasa possiede 2 componenti, "Rasa NLU" che aiuta la comprensione del linguaggio naturale, più in dettaglio si occupa di analizzare la grammatica e la logica di una frase. In *output* fornisce le parti più interessanti che verranno poi usate da "Rasa CORE" per l'elaborazione della risposta.

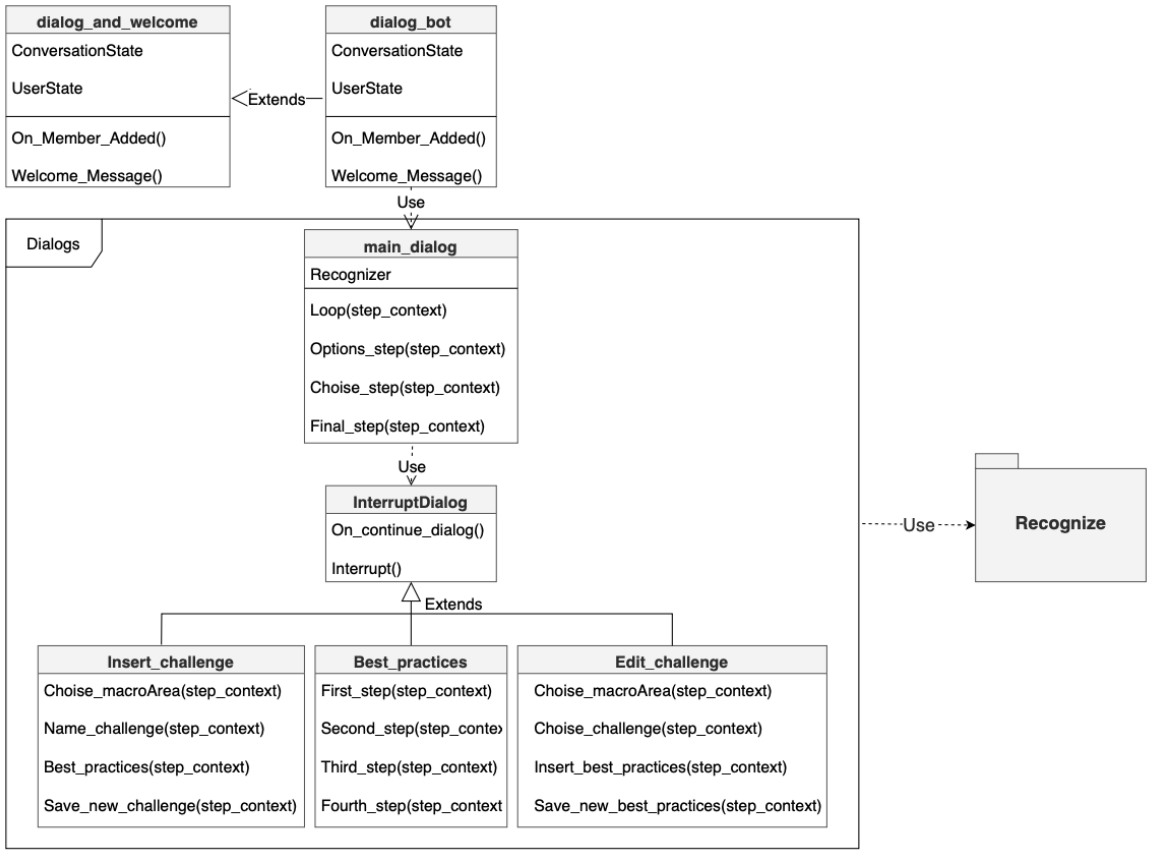

Figura 4.1: Architettura interna di Stephen.

Dopo un'analisi dei *framework*, considerando aspetti positivi e negativi, la scelta è ricaduta su AZURE BOT FRAMEWORK, in quanto utilizza un approccio più "programmatico". È stata presa questa decisione poiché si sono volute affrontare alcune delle *challenge* riscontrare nel capitolo precedente, calandosi nei panni di chi sviluppa i bot. In più la scelta su questo *framework* è ricaduta in quanto offre un tool, BOT FRAMEWORK EMULATOR, per testare l'agente in locale.

Tool per testare il bot: per testare l'agente è stato usato BOT FRAMEWORK EMULATOR. Esso è un *software* che consente facilmente agli sviluppatori di testare i bot in locale, offrendo un interfaccia conversazionale con quest'ultimo.

Prima di poter iniziare una conversazione con l'agente, il *framework* chiede di inserire il numero di porta sulla quale è in ascolto il bot ovvero **localhost:3978**. Una volta aperta la connessione, tramite JSON, è possibile avviare la conversazione.

Questo *framework* è molto comodo, infatti permette di testare l'agente in locale senza distribuirlo sul cloud.

4.3 Architettura

Per la realizzazione di Stephen è stata progettata un'architettura abbastanza classica. Osservando la Figura 4.1, si può notare come l'implementazione del bot usa i `dialogs`, ovvero delle classi

che a cascata vanno a richiamare man mano i propri metodi per creare una conversazione con un utente. Entrando nel dettaglio dell'architettura si può notare che ci sono varie classi:

Dialog_and_welcome: tale classe si occupa di mantenere aggiornato lo "stato della conversazione" e lo "stato dell'utente". Questi ultimi sono molto importanti in quanto permettono la scrittura e la lettura di coppie `chiave-valore` nel livello di archiviazione del bot. La differenza fra questi è:

- Lo "stato della conversazione" è sempre disponibile, indipendentemente dall'utente con cui si ha il dialogo.
- Lo "stato dell'utente" è disponibile in qualsiasi momento in cui l'agente sta avendo una conversazione con l'utente in un canale specifico.

I metodi definiti in questa classe come si può notare sono 2:

- **On_Member_Added():** il quale viene richiamato ogni qual volta un nuovo utente si unisce alla conversazione.
- **Welcome_Message():** il quale viene richiamato da "On_Member_Added()" per allegare un messaggio di benvenuto al nuovo utente.

Dialog_bot: questa classe estende "dialog_and_welcome", essa è fondamentale per la gestione dei turni nella conversazione. In questa classe sono presenti 2 metodi:

- **On_Turn():** questo metodo si occupa di chiamare uno dei gestori in base al tipo di attività ricevuta.
- **Welcome_Message():** il quale viene richiamato da `On_Message_Activity()`. Esso è un gestore della attività e si occuperà di richiamare il dialogo opportuno in base all'azione.

Main_dialog : come abbiamo già detto, Stephen si basa sui *dialogs*, e la classe principale dalla quale iniziano tutti i *use-case* è proprio "main_dialog".

In questa classe sono presenti 4 metodi:

- **Loop():** questo metodo si occupa di prendere in input una frase, e tramite il modulo *Recognize*, ottenere l'intento dell'input. Il modulo in questione si occupa di processare il linguaggio naturale per ricavare l'intento dell'utente. Dopo il quale viene generata una frase contenente le informazioni che esso desidera. Qualora l'utente non sappia come interagire con Stephen, potrà digitare "menù" e varrà richiamato il metodo "Options_step()", il quale aiuta l'utente tramite dei *buttons* nella ricerca, inserimento o modifica di una *challenge*.
- **Optins_step():** in questo metodo viene creato un messaggio con il quale è possibile far scegliere all'utente quale azione si voglia effettuare. Le azioni possibili sono 3: ricerca di una *challenge* con le relative *best-practice*, inserire o modificare una *challenge*.
- **Choise_step():** questo metodo prende l'input dell'utente, ricava l'intento, e in base ad esso fa avanzare la conversazione verso altri *dialogs*: "Insert_challenge", "Best_practices" o "Edit_challenge".
- **Final_step():** quest'ultimo verrà richiamato alla fine di ogni *dialog*, e invierà un messaggio di fine conversazione all'utente.

Interrupt_dialog: questa classe è stata implementata per gestire le possibili interruzioni dell'utente. Essa viene estesa da "Insert_challenge", "Best_practices" e "Edit_challenge". In questa classe sono presenti 2 metodi:

- **On_continue_dialog():** il quale richiamerà il metodo "Interrupt()" per verificare se l'utente ha interrotto il flusso di conversazione. Se il flusso viene interrotto allora viene richiamato il metodo "Final_step()" di "main_dialog".
- **Interrupt():** ha una duplice funzionalità, la prima è se l'utente digita "?" o "help" quest'ultimo invia un messaggio per indicargli cosa può fare. Altrimenti se digita "cancel" o "quit" l'intera conversazione viene terminata.

Best_practices: questa classe va a implementare la *feature* principale di Stephen, ovvero dare delle *best-practice* in base alla *challenge* che si vuole affrontare nello sviluppo degli agenti conversazionali. Per implementare ciò sono stati realizzati 4 metodi:

- **First_step():** questo metodo ha il compito di realizzare un messaggio nel quale viene illustrato allo sviluppatore in generale come si dividono le *challenge* relative ai bot, *development challenges* e *use challenges*, con i relativi aspetti.
- **Second_step():** in questo secondo metodo, in base all'area interessata dallo sviluppatore vengono allegate ad un messaggio le *challenge* che fanno parte di quella particolare area.
- **Third_step():** il terzo metodo invece si occupa di prendere in input la specifica *challenge* che si vuole affrontare e fornire all'utente delle *best-practice*. A queste poi viene allegato un messaggio per chiedere all'utente se ha bisogno di ulteriori informazioni su un'altra *challenge*.
- **Fourth_step():** l'ultimo metodo invece si occupa di gestire la richiesta dell'utente. Se l'utente non desidera altre *best-practice* allora viene terminata la conversazione. Altrimenti viene riportato al metodo "First_step()".

Insert_challenge : questa classe permette all'utente di inserire una nuova *challenge* e di aggregarla ad una delle macro-aree definite, development challenge e use challenge.

Per implementare questo dialogo sono state previsti 4 metodi:

- **Choise_macroArea():** questo è il primo metodo che viene richiamato nella classe, e ha il compito di presentare all'utente un messaggio nel quale chiede a quale macro-area faccia parte la nuova *challenge*.
- **Name_challenge():** il secondo si occupa di salvare la macro-area di appartenenza, e crea un messaggio nel quale viene chiesto il nome della nuova *challenge*.
- **Best_practices():** questo terzo metodo invece ha il compito di salvare il nome della nuova *challenge*, e in particolare di chiedere all'utente le *best-practice* che vengono usate per affrontarla.
- **Save_new_challenge():** l'ultimo metodo ha il compito di prendere tutte le informazioni fin ora salvate come coppie "chiave-valore". Difatti recuperare la macro-area di appartenenza, nome della *challenge* e *best-practice*, e inserire la nuova sfida e le relative *best-practice* nel gruppo di appartenenza.

Edit_challenge: quest'ultima classe implementa un flusso di dialogo che permette all'utente di modificare, quindi di aggiungere nuove *best-practice* ad una *challenge*. Quest'ultima classe è stata ideata in quanto affiancando il bot allo sviluppatore, si vuole che quest'ultimo sia altamente personalizzabile.

Per comprendere meglio questo concetto di personalizzazione si può osservare il seguente scenario. Durante lo sviluppo di un agente, uno sviluppatore utilizzando Stephen, per cercare delle *best-practice*, non risulta soddisfatto. Una volta che lo sviluppatore ha speso del suo tempo per cercarne di nuove, può magari proporle al bot in modo che in un futuro non debba effettuare di nuovo quella ricerca.

Come si può notare questa classe ha una duplice funzionalità, la prima è che permette allo sviluppatore di non perdere le nuove *best-practice* cercate. La seconda appunto che incrementa il numero di informazioni che l'agente possiede sulle *challenge*.

L'implementazione di questa classe è molto simile a quella descritta precedentemente, "Insert_challenge", e sono stati realizzati 4 metodi:

- **Choise_macroArea():** questo primo metodo appunto come nella classe "Insert_challenge" chiede all'utente di quale macro-area fa parte la *challenge* da modificare.
- **Choise_challenge():** il secondo invece si occupa di far selezionare all'utente la *challenge* da modificare.
- **Insert_best_practices():** il terzo metodo si occupa di prendere in input la *challenge* che si vuole modificare, e salvarla tramite una coppia chiave-valore. Inoltre in "Insert_best_practices" viene creato un messaggio per l'utente chiedendogli di inserire le nuove *best-practices*.
- **Save_new_best_practices():** infine l'ultimo metodo è fondamentale in quanto prende in input le nuove *best-practices* e apporta le nuove modifiche alla *challenge*. Il metodo si occupa di recuperare il nome della *challenge* precedentemente salvata tramite una coppia chiave-valore e apportare la nuova modifica.

4.4 Intent and entities

Gli intenti e le entità sono la base su cui si realizza una conversazione. Difatti quando si ha una conversazione con un bot, gli intenti permettono all'agente di capire l'intenzione della frase e quindi cosa "intende" dire l'utente.

Spesso nella frase che bisogna analizzare ci sono delle informazioni, chiamate "entità", che saranno poi necessarie al bot per intraprendere delle azioni. Banalmente i concetti di "intento" e "entità" possono essere compresi più facilmente tramite un esempio. Se si considera la frase "Devo prenotare un treno per Roma", l'intento della frase è "prenotare un treno", mentre l'entità in questo caso è la destinazione, ovvero "Roma".

Compreso il concetto di intento ed entità, in questa sezione si analizzeranno gli intenti riconosciuti da Stephen, in particolare la Tabella 4.2 riporta 3 intenti con i relativi esempi:

Ricerca di challenge: il primo intento presente nella tabella è "ricerca di challenge", questo intento proviene dalla *feature* principale dell'agente. L'obiettivo che ha portato alla creazione di

Intento	Esempi
Ricerca di challenge	- "Esistono delle best-practices per l'integrazione dei bot?" - "Hai dei consigli su come testare il bot?" - "Non so come rendere il bot più sicuro!"
Inserimento di una nuova challenge	- "Durante l'implementazione del bot ho riscontato un problema" - "Durante un meeting con colleghi abbiamo parlato di una nuova challenge dei bot ..."
Modifica di una challenge	- "Ho approfondito meglio questa challenge, e dopo varie ricerche ho notato che ci sono nuovi modi per affrontarla "

Tabella 4.2: Intenti riconosciuti da Stephen.

questo intento è il dover cercare delle *best-practice* per una determinata *challenge*. In tabella sono stati riportati anche alcuni esempi come "Hai dei consigli su come testare il bot?". Nell'esempio appena citato, l'intento dell'utente è proprio quello di ricevere dei consigli o magari anche dei tool che facilitano il *testing* degli agenti conversazionali.

Inserimento di una nuova challenge: questo secondo intento è stato realizzato in quanto negli ultimi anni l'utilizzo dei bot, soprattutto per scopi commerciali, è aumentato notevolmente. L'espansione degli agenti implica problemi non del tutto banali, in quanto tutti vogliono un bot che si distingua dalla massa. Con lo sviluppo di nuove tecnologie tutto questo è anche possibile, ma insieme ad esse emergono nuove *challenge* da affrontare.

In conclusione uno sviluppatore può trovarsi difronte ad una *challenge* del tutto nuova anche per il bot. In questo caso lo sviluppatore può approfondire questa sfida, quindi capire di quale macro-area fa parte e quali *best-practice* usare per affrontarla. Da ciò lo sviluppatore può fornire questa nuova *challenge* a Stephen.

In tabella sono riportati alcuni esempi come: "Durante un meeting con colleghi è stato discusso di una nuova challenge dei bot...". In questo caso l'intento è di suscitare nel bot un interesse per questa *challenge*, quindi di inserire la nuova sfida alle altre. Questo poi si svilupperà in un dialogo dove verrà chiesto allo sviluppatore di specificare macro-area di appartenenza, nome della *challenge* e *best-practice* associate.

Modifica di una challenge: l'ultimo intento riconosciuto da Stephen è quello di modificare una *challenge*. Riflettendo sul discorso relativo all'intento precedente, "Inserimento di una nuova challenge", è emerso che negli anni nascono nuove sfide. Spesso però succede che con il passar del tempo emergono anche nuovi modi migliori per affrontare una *challenge* già esistente. Da ciò si può dedurre che è fondamentale che un agente rimanga aggiornato, quindi con questo intento viene data la possibilità allo sviluppatore di aiutare il bot, nell'apprendere nuove *best-practices*.

Un esempio che migliora la comprensione di questo intento è : "Ho approfondito meglio questa *challenge*, e dopo varie ricerche ho notato che ci sono nuovi modi per affrontarla".

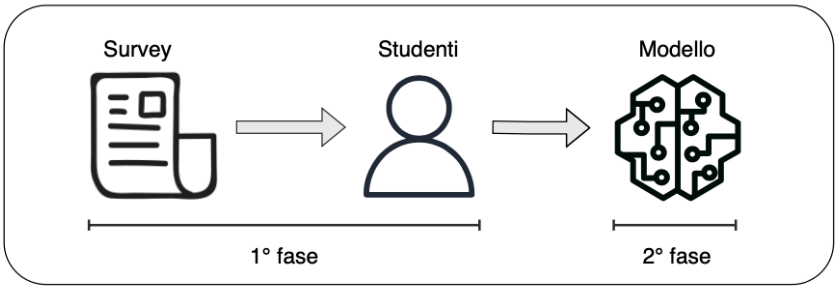

Figura 4.2: Pipeline di implementazione del modello di *Natural Language Processing*

.

Questo fa scattare il dialogo per l'apprendimento di nuove *best-practice*, quindi di conseguenza modificare una *challenge*.

4.5 Natural Language Processing

Come è stato già accennato nei capitoli precedenti, i bot spesso hanno a che fare con utenti, e hanno la necessità di comprendere ciò che essi intendono. Per far si che un agente possa comprendere il senso di una frase, esso deve avere un modulo al suo interno, tramite il quale riesca a processare il linguaggio naturale.

Il modulo appena citato deve essere formato da algoritmi di intelligenza artificiale, che tramite delle tecniche, riescano a comprendere ciò che un umano intende. Da ciò l'insieme di tutte le tecniche e algoritmi usati per elaborare il linguaggio naturale viene anche chiamato *Natural Language Processing* (NLP).

Il *Natural Language Processing* comprende numerosi task come:

- Il riconoscimento della lingua;
- Scomposizione delle frasi in unità elementari;
- Trasformazioni delle singole unità nella loro radice;

Nel caso di Stephen, è stato usufruito del *Natural Language Processing* per individuare gli intenti degli sviluppatori.

Si può osservare dalla Figura 4.2, che per l'implementazione del modulo relativo al riconoscimento del linguaggio naturale sono state previste due fasi:

1. **Prima fase:** nella prima fase si può notare come è stato realizzato un *survey*, nel quale sono state previste tre domande. Lo scopo era raccogliere quante più frasi possibili per poter allenare l'agente, in modo da avere una conversazione con gli sviluppatori. Nella stessa fase è stato condiviso il *survey* con gli studenti, chiedendo loro di immedesimarsi nei panni di uno sviluppatore, e creare le frasi in base "agli intenti" presenti nel questionario.
2. **Seconda fase:** nella seconda fase sono state raccolte le frasi fornite dagli studenti e sono stati implementati due modelli. Il primo è un modello di *Deep Learning* e per l'implementazione di quest'ultimo sono state usate le "reti neurali". Difatti è stata usata la potenza di queste

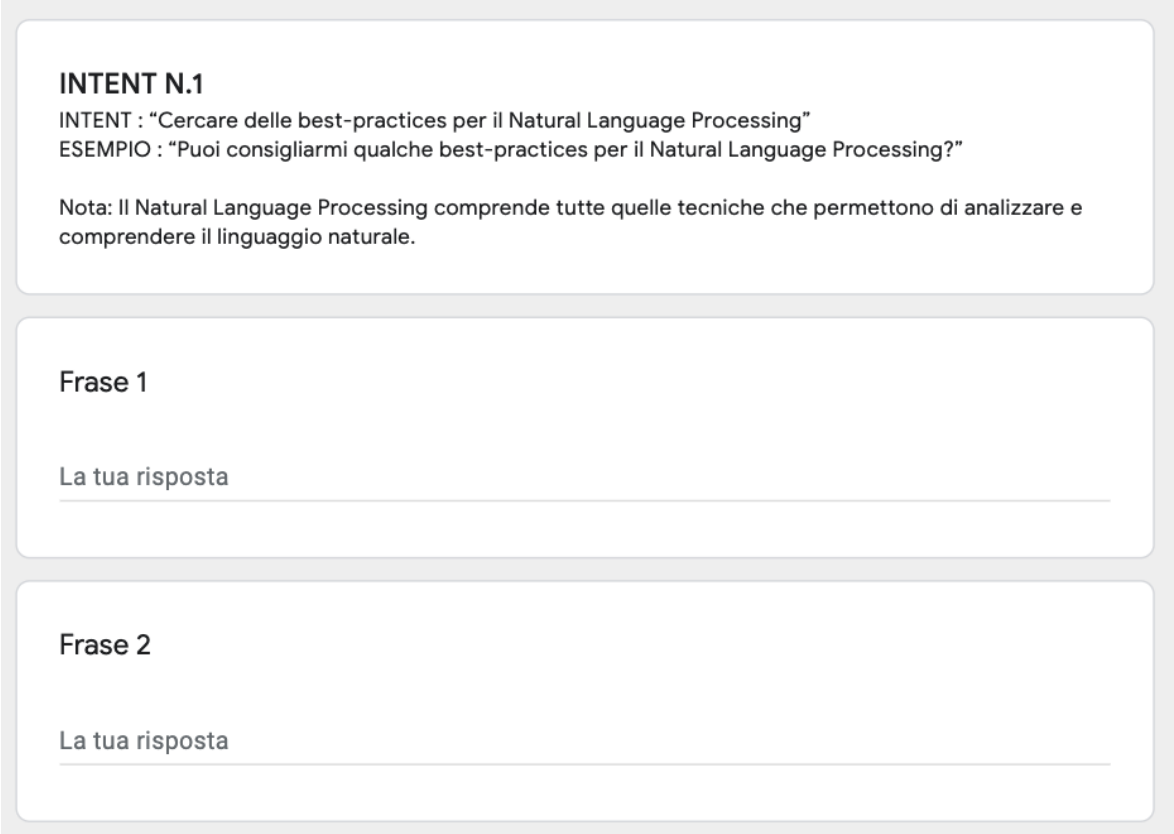

Figura 4.3: Esempio di *intent* presente nel *survey*.

ultime per individuare l'intento dell'utente. Il secondo modello usato fa parte degli algoritmi di *Machine Learning*, ed è "Naive Bayes". Esso è un algoritmo di classificazione basato sul teorema di "Bayes", in questo caso fornisce la probabilità che la frase in input può essere rappresentata da un determinato intento.

Alla fine della seconda fase, sono state usate le frasi ricavate dal *survey* per allenare il modello.

Di seguito verranno analizzate le varie fasi più in dettaglio:

1. **Prima fase:** come si può notare dalla Figura 4.2, la prima fase è stata suddivisa in :

   Survey: il questionario è stato realizzato tramite i "form di GOOGLE". Esso presentava un messaggio generale, nel quale veniva spiegato l'obiettivo del *survey*, quindi il progetto a cui si stava lavorando. Inoltre è stata fatta una piccola introduzione al significato di "intento", in modo che durante la compilazione ci fossero stati pochissimi dubbi da parte degli studenti.

   Il corpo del questionario era formato da tre *intent*:

   - *Natural Language Processing;*
   - *Testing* di Bot;
   - Concetti di *privacy*;

   Per ognuno di questi è stato esplicitato l'intento, un esempio di possibile formulazione e sono state richieste due frasi per ognuno di essi.

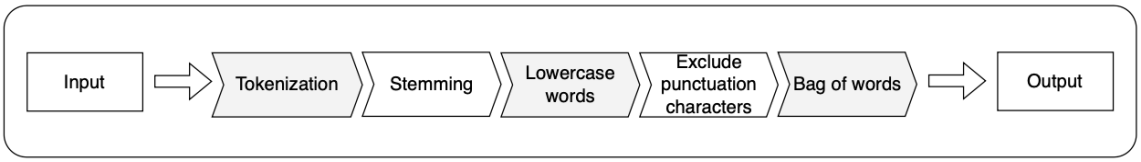

Figura 4.4: Pipeline *data cleaning*

Osservando la Figura 4.3 è possibile notare un esempio di domanda presente nel questionario.

Studenti: per la raccolta delle informazioni è stato chiesto un aiuto agli studenti dell'Università degli Studi di Salerno. Il *survey* infatti è stato condiviso nella maggior parte dei gruppi di comunicazione, in modo da ricevere quanti più *feedback* possibili. In *output* sono state ricevute circa 30 risposte, circa 60 frasi per intento, e per un totale di circa 180 frasi con cui allenare il modello.

Prima di procedere ad allenare i modelli, le frasi sono state filtrate e sono stati usati dei criteri di esclusione come:

- Frasi incoerenti;
- Frasi già ripetute;

2. **Seconda fase:** nella seconda fase sono stati implementati due algoritmi. Prima di discutere di come sono stati realizzati, bisogna accennare alcuni aspetti del *Natural Language Processing*.

   Difatti prima di allenare i modelli con le frasi ricavate, bisogna adottare delle tecniche di *data cleaning* per la pulizia dei dati.

   Nella Figura 4.4 si può notare come i dati attraversano 4 fasi:

   (a) **Tokenization:** la prima fase del processo di *data cleaning* è proprio *tokenization*. È stata necessaria in quanto ha consentito di dividere le frasi in singole parole. I token possono essere singole parole, numeri o punteggiatura.

   Di seguito è stato riportato un esempio di applicazione della medesima tecnica:

   > **Input:** "Come si gestiscono gli aspetti di privacy del bot?"
   > **Output:** ["Come", "si", "gestiscono", "gli", "aspetti", "di", "privacy", "del", "bot", "?"]

   Come è possibile notare, la frase in *input*, è stata divisa in singole parole in modo che è possibile focalizzarsi sui singoli elementi. Difatti quando si lavora con il linguaggio naturale, bisogna prima di tutto identificare tutte le parole e le punteggiature che compongono una frase.

   Per questa tecnica è stato usata una libreria PYTHON, *Native Language Tool Kit* (NLTK), la quale si occupa tramite il metodo "word_tokenize(input)" di applicare la "tokenizzazione" alla frase in input.

   (b) **Stemming:** la seconda fase che è possibile osservare dalla Figura 4.4 è lo *stemming*. Questa tecnica è stata fondamentale in quanto riduceva la forma della parola nella sua

radice. Questo nell'ambito del *Natural Language Processing* è importantissimo, in quanto spesso nelle conversazioni vengono usate tantissime "variazioni" di parole. Difatti questa procedura come già è stato detto permette la riduzione della parola nella sua radice.
Di seguito è stato riportato un esempio di *stemming*:

> **Input:** "organize", "organizes", "organizing"
> **Output:** ["organ", "organ", "organ"]

Anche in questo caso per implementare tale tecnica è stata usata la libreria PYTHON, *Native Language Tool Kit* (NLTK).

(c) **Lowercase words:** come terza fase sono state rese tutte le parole in "minuscolo". Questo perché spesso non è chiaro come l'utente scrive una frase, ed è stato deciso di lavorare con parole in *lowercase*.

(d) **Exclude punctuation characters:** come quarta fase c'è l'esclusione dei simboli di punteggiatura. Infatti è stato necessario eliminarli dalle frasi ricavate dal *survey* per poter addestrare il modello.
Di seguito è riportato un esempio di questa tecnica:

> **Input:** ["Ci", "sono", "delle", "linee", "guida", "per", "testare", "un", "Bot", "?"]
> **Output:** ["Ci", "sono", "delle", "linee", "guida", "per", "testare", "un", "Bot"]

Come è possibile notare dall'esempio la frase in *input*, divisa in più elementi, contiene il simbolo "?", mentre in *output* quest'ultimo non è più presente.

(e) **Bag of words:** l'ultima fase presente nella *pipeline di data cleaning* è *bag of words*. Questa tecnica è stata usata in quanto permette la trasformazione di una lista di parole in una lista formata da 0 e 1. La tecnica seguente assegna 1 al termine, se è presente nel "sacco" di parole, 0 altrimenti. Essa è usata poiché molti algoritmi di classificazione lavorano con *input* formati da 0 e 1, un esempio è il modulo di *artificial neural network* implementato per Stephen.
Di seguito è stato riportato un esempio della tecnica di *bag of words*:

> **All words:** ["Bot", "practices", "per", "best", "NLP", "un", "Quale"]
> **Input:** "Quale best practices posso usare per NLP?"
> **Output:** [1 , 1 , 1 , 0 , 0 , 1 , 1]

Dall'esempio riportato si può notare come è reso disponibile una lista di parole, *all words*, per applicare la tecnica. Mettendo in pratica tale tecnica alla frase in *input* è possibile

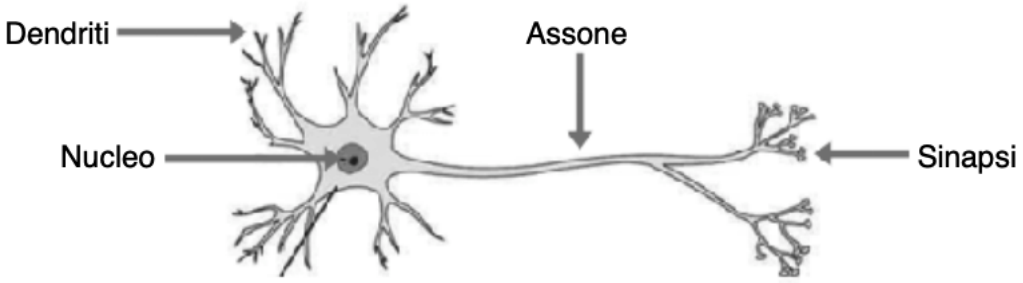

Figura 4.5: Rete neurale biologica.

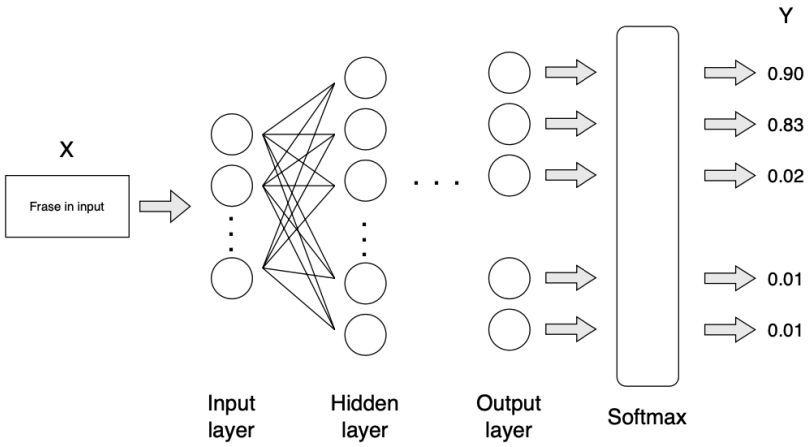

Figura 4.6: Pipeline *artificial neural network.*

notare che l'*output* possiede solo due occorrenze di "0". Esse infatti coincidono proprio con "posso" e "usare", ovvero i due termini che non fanno parte del "sacco" di parole.

Tutte le fasi precedentemente elencate sono state necessarie per l'implementazione dei due modelli di *Natural Language Processing*. In particolare, come già accennato, sono stati scelti ben due algoritmi su cui basare i modelli.

Artificial Neural Network: il primo algoritmo fa parte del *Deep Learning*.

I modelli che si basano sulle reti neurali si ispirano al cervello umano, e sono applicati a svariati progetti come il riconoscimento verbale, vocale o di immagini. Come si può notare sono modelli che si dedicano al concetto di "classificazione". Difatti vale la pena approfondire le reti neurali, il cui funzionamento è molto simile al cervello. Si sa bene che il cervello possiede neuroni interconnessi tra di loro, e questi ultimi sono presenti anche nelle reti neurali artificiali sotto forma di nodi.

È possibile osservare la somiglianza fra le reti neurali biologiche, Figura 4.5, e le reti

neurali artificiali Figura 4.6. Si può notare come i "dendriti" della prima corrispondono agli input che può ricevere una rete neurale artificiale. In più il "nucleo" corrisponde perfettamente ai nodi, e le sinapsi rappresentano i pesi che vengono dati in *output* dal modello.

Da ciò che concerne, le reti neurali artificiali sono state applicate al modello in modo da permettergli di processare il linguaggio naturale.

Dalla Figura 4.6 è possibile notare che l'*artificial neural network* possiede vari layer:

- **Input layer:** accetta i vari input.
- **Hidden layer:** questo è il livello di computazione delle reti neurali.
- **Output layer:** l'ultimo layer si occupa di propagare l'*output* fornito dal livello intermedio.

Di seguito sono illustrate le classi che hanno permesso la realizzazione del modello:

NeuralNet: questa classe è stata realizzata in quanto rappresenta la vera e propria rete neurale. Difatti questa classe estende "Module", ovvero una classe presente nel modulo PyTorch. Questo libreria è stata inclusa in quanto offre diverse funzioni per il *machine learning* e per il *deep learning*. In "NeauralNet" sono stati definiti i 3 layer già citati precedentemente, con il proprio numero di nodi interni. Bisogna sottolineare che per il livello di *output layer*, il numero di nodi è pari al numero di intenti che il bot può riconoscere.

Infine in questa classe è stato realizzato il metodo *forward*, il quale si occupa di prendere un input e farlo processare dai vari *layer*.

Alla fine di ognuno di essi è stato richiamata la funzione "ReLU", la quale è una funzione di "attivazione". L'obiettivo di quest'ultima è che se l'input è un numero positivo viene restituito come numero positivo, mentre se è negativo viene restituito come zero:

$$\text{ReLU(X)} = \{\ 0 \text{ se } x < 0,\ x \text{ se } x > 0\ \}$$

Infine nel metodo verrà automaticamente usato "softmax", il quale si occuperà di normalizzare tutti gli *output* nell'intervallo (0,1) in modo da poter essere usati come probabilità.

Train: questa classe si occupa di allenare il modello. Difatti in questa classe sono state:

- Prelevate le frasi con cui allenare il Bot.
- Applicate tutte le tecniche di *data cleaning* citate.
- Definito i parametri da dare in input a "NeuralNet".
- Allenato il modello con i dati ripuliti.

Si può osservare, Figura 4.6, come il funzionamento del modello risulti semplice. Infatti una volta dato in input la frase e processata, verranno generate delle probabilità per ogni "intento" che il bot possiede. Da qui poi sarà preso "l'intento" con la probabilità più alta.

Naive Bayes: anche *Naive Bayes* è un algoritmo di classificazione.

	precision	recall	f1-score	support
BestPracticesNLP	0.94	0.94	0.94	17
BestPracticesPrivacy	0.92	0.92	0.92	13
BestPracticesTest	0.94	0.94	0.94	18
accuracy			0.94	48
macro avg	0.94	0.94	0.94	48
weighted avg	0.94	0.94	0.94	48

Figura 4.7: Report di valutazione delle reti neurali.

Quest'ultimo si basa sul teorema di *Bayes*:

$$P(A/B) = \frac{P(B|A) * P(A)}{P(B)}$$

L'algoritmo basandosi sul teorema di *Bayes* riesce a calcolare la probabilità che una frase appartenga ad un determinato "intento".
Per l'implementare è stata importata la versione "multinomiale" di *Bayes* presente nella libreria di *sklearn.naive_bayes*. Questa decisione è stata presa in quanto il classificatore "multinomiale Naive Bayes" è adatto per il conteggio delle parole per la classificazione del testo.
Detto ciò la *pipeline* di addestramento è simile a quella delle reti neurali:

- Sono state prelevate le frasi per addestrare il bot.
- Sono state applicate alcune delle tecniche di *data cleaning* citate.
- Infine è stato allenato il modello con i dati ripuliti.

Metriche di valutazione: La scelta dell'algoritmo da adottare è stata presa analizzando le metriche di valutazione. Tali metriche permetto infatti di validare il lavoro di un modello, ovvero la sua "bontà". Per valutare i due algoritmi sono stati usati vari approcci:

- **Accuracy:** l'accuratezza di un algoritmo di classificazione è un modo per misurare la frequenza con cui il modello classifica correttamente. Essa è il rapporto fra il numero di predizioni corrette e il numero totali di predizioni.

$$Accuracy = \frac{Numero\ dipredizioni\ corrette}{Numerototali\ di\ predizioni}$$

Come è possibile notare dalla Figura 4.7, *l'accuracy* del modello che si basa sulle reti neurali si aggira intorno al 94%. Mentre per il modello di *Naive Bayes*, Figura 4.8, l'accuratezza è di circa 85%

	precision	recall	f1-score	support
BestPracticesNLP	0.76	0.94	0.84	17
BestPracticesPrivacy	0.92	0.85	0.88	13
BestPracticesTest	0.93	0.78	0.85	18
accuracy			0.85	48
macro avg	0.87	0.86	0.86	48
weighted avg	0.87	0.85	0.85	48

Figura 4.8: Report di valutazione del modello di *Naive Bayes*.

- **Precision:** anche la precisione è stata usata come indicatore di prestazioni di un modello. Essa si occupa di indicare la qualità di una previsione positiva effettuata dall'algoritmo. Matematicamente viene indicata come il rapporto fra i "veri positivi" e "tutti i positivi":

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

  Si può osservare come, Figura 4.7 e Figura 4.8, per le reti neurali la precisione fra i tre "intenti" è molto equilibrata.
  Questo però non succede per il modello basato su *Naive Bayes*. Difatti si può notare la differenza di precisione per l'intento *BestPracticesNLP*, la quale si aggira intorno al 76% e *BestPracticesPrivacy* o *BestPracticesTest* le quali superano la soglia del 90%.
- **Recall:** la *recall* indica la capacità che un modello ha nell'identificare i "veri positivi". Matematicamente è il rapporto fra i "veri positivi" e la somma dei "veri positivi" e i "falsi negativi":

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

  È interessante notare come anche per questa metrica le reti neurali, Figura 4.7, sono molto stabili nella predizione.
  Mentre per *Naive Bayes*, Figura 4.8, si può notare che il modello è più propenso alla classificazione di intenti per il *Natural Language Processing* e per la *Privacy*.
- **F1-score:** la quarta metrica usata è *F1-score*. Essa è una misura di valutazione che unisce *precision* e *recall*. Un buon valore generato da questa metrica porta ad avere pochi "falsi negativi e positivi". Matematicamente viene rappresentata come:

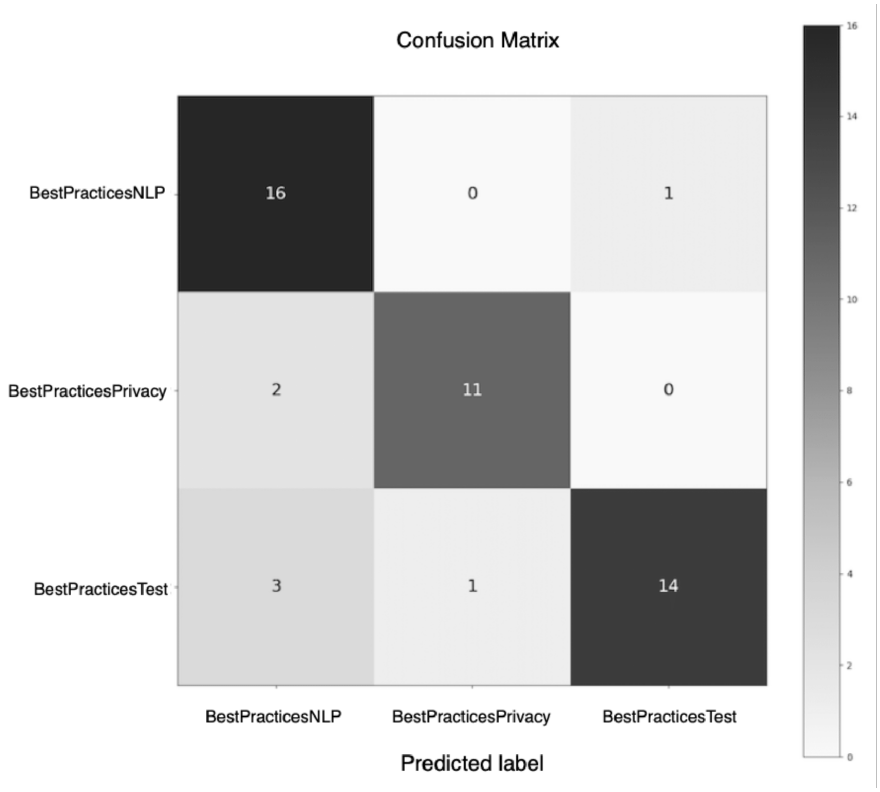

Figura 4.9: *Confusion Matrix* modello di *Naive Bayes*

$$F1 - score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

In linea generale è possibile osservare dalla Figura 4.8 e dalla Figura 4.7, come il modello delle reti neurali riesce ad identificare meglio i "falsi positivi e negativi" a differenza dei *Naive Bayes*.

- **Matrice di confusione:** questa metrica è molto usata per la valutazione delle prestazioni degli algoritmi. Difatti la matrice di confusione, come dal nome utilizza una matrice o tabella per la rappresentazione dei valori reali e i valori che sono stati ottenuti dal modello.
  Sulla "diagonale" della matrice sono presenti tutte le risposte giuste, mentre quelle sbagliate si trovano al di sotto o al di sopra della diagonale.
  Il punto di forza della "matrice di confusione" è proprio questo, poiché è possibile individuare dove il modello si confonde, osservando le celle che non fanno parte della diagonale.
  Da ciò poi si può lavorare sul *dateset* in modo da migliorare le predizioni del modello.
  Il dataset è stato diviso per il 70% in *training set* e il restante 30% per il *test set*. Dalla Figura 4.10 è possibile notare come per il modello delle reti neurali, su un totale di

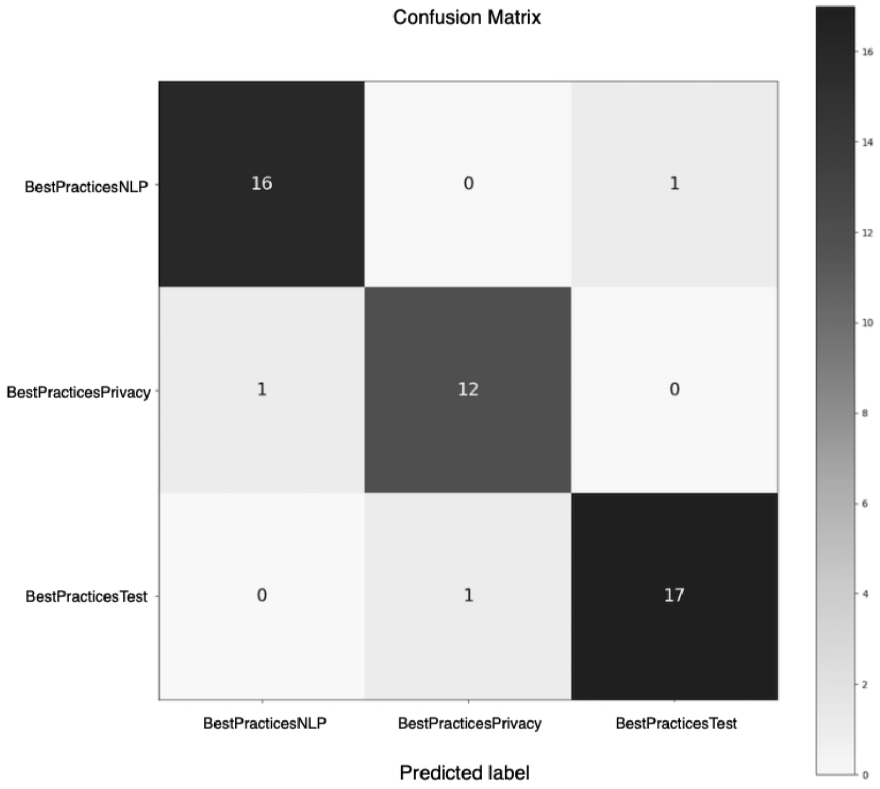

Figura 4.10: *Confusion Matrix* modello sulle reti neurali

48 test, solo 3 predizioni sono errate. Mentre per il modello di *Naive Bayes*, Figura 4.9, è possibile osservare come sulle stesse istanze il numero di errori è nettamente più alto, 7 errori.

- **Size of dataset:** un'ultima valutazione adottata è stata quella di osservare come reagivano i due modelli in base alla dimensione del dataset.
  Difatti sono stati addestrati e testati i modelli più volte suddividendo il dataset come segue:
  - 20% *test set* / 80% *training set*
  - 30% *test set* / 70% *training set*
  - 40% *test set* / 60% *training set*
  - 50% *test set* / 50% *training set*
  - 60% *test set* / 40% *training set*
  - 70% *test set* / 30% *training set*
  - 80% *test set* / 20% *training set*

  Per ognuna delle configurazioni è stata calcolata *l'accuracy* in modo da poter riportare i dati in *output* su un grafico.
  Dalla Figura 4.11, e dalla Figura 4.12 è possibile osservare i comportamenti dei due modelli. Ci sono varie considerazioni che sono sorte a seguito dei test usando le varie configurazioni elencate precedentemente. Una prima considerazione è che i due modelli reagiscono entrambi molto bene fin quando la *size* del *test set* è del

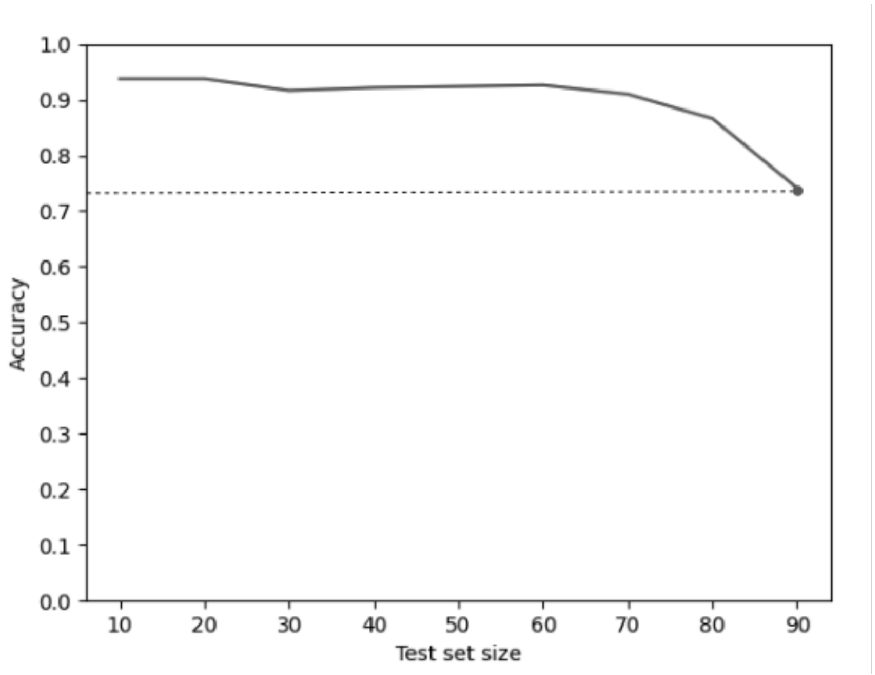

Figura 4.11: Rappresentazione "accuratezza - dimensione test set", per il modello di *Artificial Neural Net*.

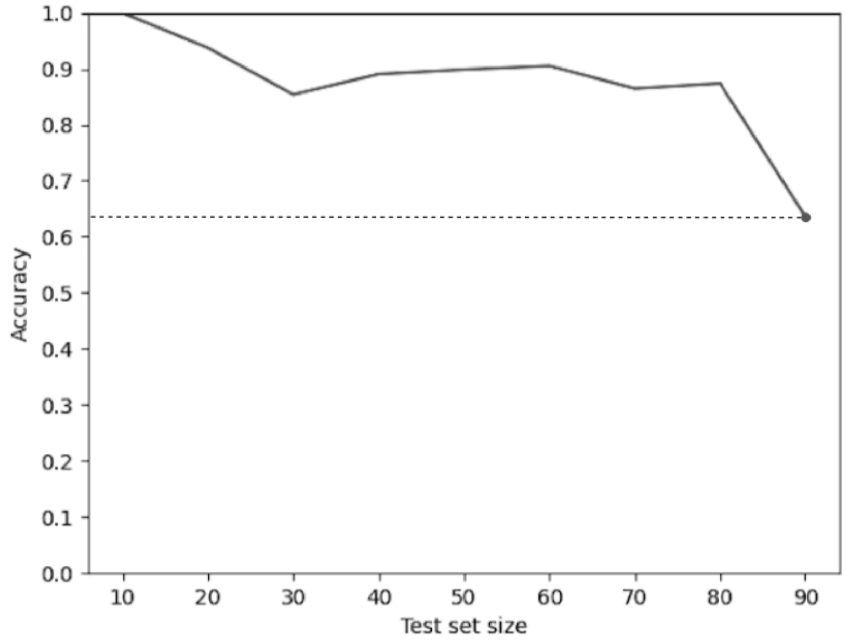

Figura 4.12: Rappresentazione "accuratezza - dimensione test set", per il modello di *Naive Bayes*.

70%-80% di quello totale. Superata tale soglia si può notare come nel modello di *Naive Bayes*, avendo solo il 30%-20% del *dataset* come *training set*, il valore dell' *accuracy* scende drasticamente. Il modello di *artificial neural network* invece grazie alla potenza di computazione dei singoli nodi riesce a mantenere un'*accuracy* alta. Difatti questa è stata la caratteristica che ha portato a scegliere l'applicazione del modello delle reti neurali.
Difatti porta vari vantaggi:

- Se il *dataset* non è molto grande, è possibile sempre applicare il modello in quanto i risultati portati sono abbastanza soddisfacenti.
- Qualora vi è a disposizione un *dataset* molto grande, è stato notato come i risultati ottenuti sono più che soddisfacenti, avendo un accuratezza che supera addirittura il 90%.

CAPITOLO 5

Guida di installazione del tool

Questo capitolo illustra tutti gli *step* per l'installazione e l'utilizzo di Stephen.

5.1 Prerequisiti

L'implementazione dell'agente è stata svolta usando come linguaggio di programmazione PYTHON.

L'unico prerequisito che viene chiesto per l'utilizzo del bot difatti è avere installato una versione >= PYTHON 3.0

5.2 Step per l'installazione

L'implementazione del bot e la documentazione relativa usata per questo lavoro di tesi sono presenti al seguente *repository* GitHub [9].

Di seguito sono riportati i relativi step per l'installazione e utilizzo del tool:

Clonare la repository: il primo step è proprio quello di clonare la *repository*.

Di seguito è riportato l'appisito comando:

```
git https://github.com/Pio57/Stephen.git
```

Installazione dei package necessari per Stephen: tutti i pacchetti necessari per l'agente si trovano in "requirements.txt", difatti bisogna semplicemente aprire un terminale nel progetto clonato, e digitare :

```
pip install -r requirements.txt
```

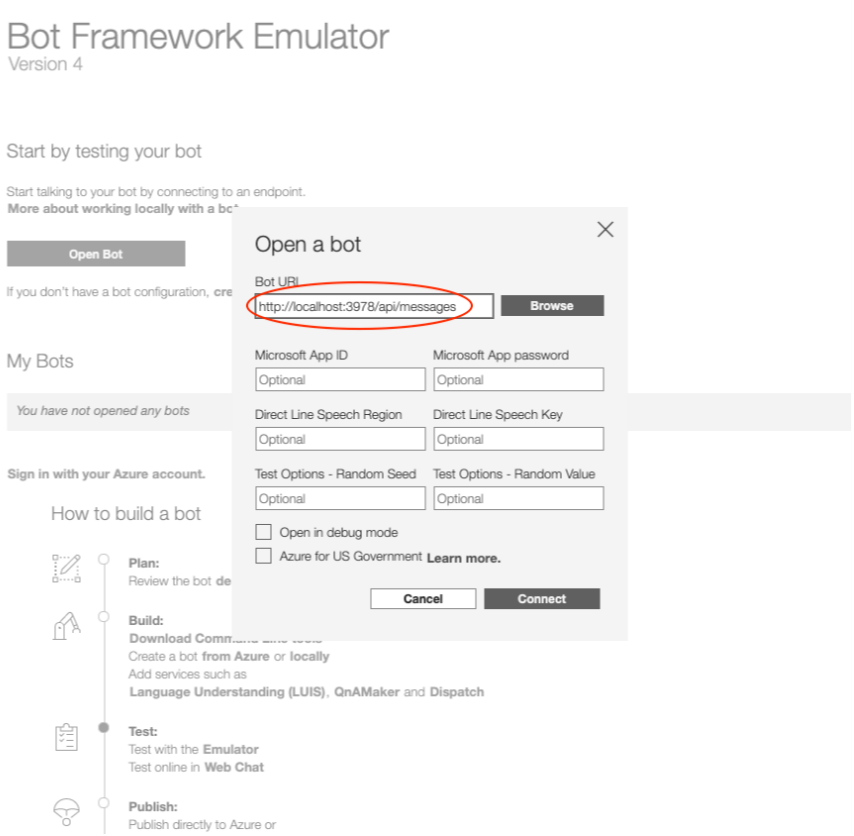

Figura 5.1: Avvio di Stephen usando Bot Framework Emulator.

Avviare il bot: l'ultimo step è quello di avviare Stephen, difatti stesso nel terminale basta digitare:

```
python app.py
```

5.3 Testare il bot

Per testare Stephen in locale è stato usato BOT FRAMEWORK EMULETOR[1], un tool di MICROSOFT, il quale permette appunto di avere una conversazione con l'agente.

Una volta avviato Stephen, bisogna avviare BOT FRAMEWORK EMULETOR, e in "Bot URL" inserire:

```
http://localhost:3978/api/messages
```

È possibile osservare un esempio di avvio del bot dalla Figura 5.1

Una volta avviato è possibile iniziare una conversazione con il bot.

CAPITOLO 6

Conclusioni e lavori futuri

6.1 Conclusioni

La realizzazione di questa tesi ha portato l'analisi di tutte le sfaccettature delle *challenge* che vengono affrontate nello sviluppo e nell'utilizzo dei bot.

Gli obiettivi che hanno portato alla realizzazione di questa tesi sono stati vari:

- Il primo motivo era capire le potenzialità e debolezze degli agenti conversazionali.
- Il secondo è stato quello di analizzare in modo più dettagliato le debolezza, e come esse sono affrontate dagli sviluppatori.
- Il terzo motivo è stato quello di realizzare un agente conversazionale capace di fornire *best-practices* per affrontare le sfide relative ai bot.

Per la realizzazione di questi scopi è stata condotta una ricerca del tipo *Multivocal Literature Review*. La tecnica appena citata difatti si occupa di analizzare sia fonti estratte da conferenza, *white literature*, sia fonti divulgate da *community* o *blog*, *grey literature*.

Dall'analisi effettuata la maggior parte delle informazioni sono state raccolte dalla letteratura secondaria, *grey literature*.

Difatti è possibile osservare dalla Figura 6.1, come la maggior parte dei documenti, circa il 64.4% appartiene alla *grey literature*, mentre il restante 35.6% fa parte della *white literature*.

Da ciò che concerne questa tecnica ha portato la raccolta di un ottimo numero di *paper*, dai quali come citato nel capitolo 2, sono emerse circa 22 *challenge*.

Nello sviluppo della tesi è stato ritagliato un periodo nel quale cercare documenti interessanti, che discutevano delle *challenge* dei bot. In questo periodo è stato interessante annotare le date di pubblicazione dei *paper*, in modo da poterne trarre delle considerazioni.

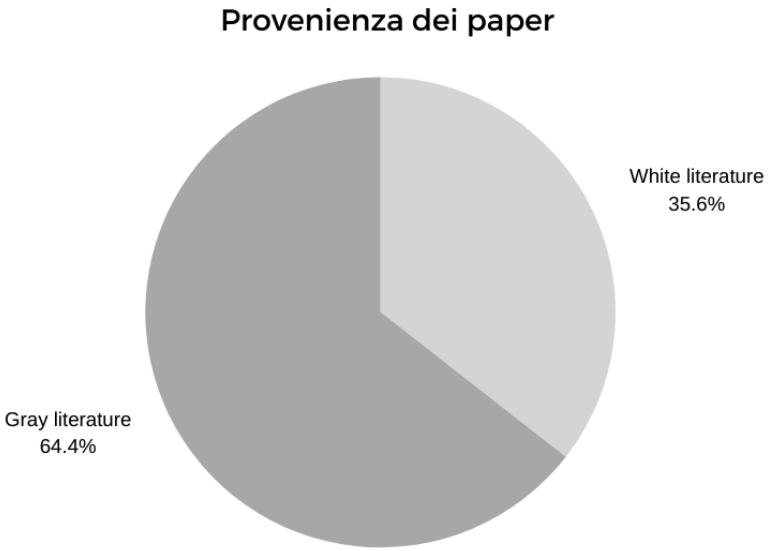

Figura 6.1: Provenienza dei paper raccolti.

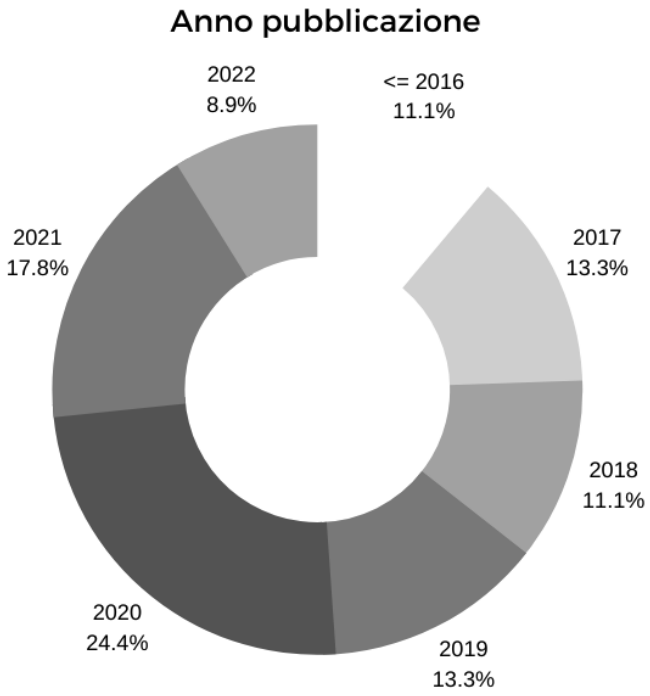

Figura 6.2: Anno di pubblicazione dei paper.

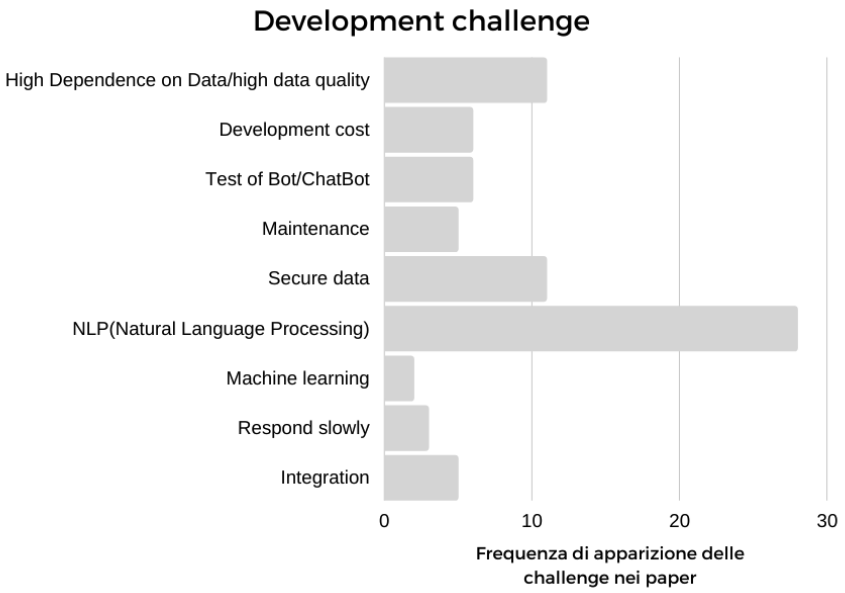

Figura 6.3: Frequenza di discussione delle *development challenge* nei paper.

Dalla Figura 6.2 è possibile notare come dal 2019, i bot e le loro *challenge* sono state approfondite in letteratura.

Dopo un accurata analisi e categorizzazione delle sfide degli agenti conversazionali in letteratura, è stata effettuata un suddivisione in macro-aree:

- La prima è "*development challenge*", nella quale fanno parte tutte le sfide relative allo sviluppo come ad esempio il *Natural Language Processing*, o problemi di integrazioni degli agenti.
- La seconda macro-area è "*use challenge*", nella quale fanno parte tutte le sfide relative all'utilizzo dei bot. Un esempio di sfida è rendere l'agente quanto più "eticamente" corretto, o addirittura prevedere un interfaccia intuitiva per l'utente finale che poi utilizzerà l'agente.

Soffermando l'attenzione sulla prima macro-area, *development challenge*, è possibile notare dalla Figura 6.3 come la *challenge* più riscontrata durante la ricerca è stata quella relativa al *Natural Language Processing*.

A seguire vi sono *High Dependence on Data/high data quality* e *Sacure data*, anche esse rappresentano delle significative sfide da affrontare nello sviluppo dei bot.

Nella seconda macro-area, osservando la Figura 6.4, è possibile notare come le sfide più frequentemente riscontrare riguardano i *privacy concerns*, il rumore (*noise*) generato dall'utilizzo degli agenti e la scelta della configurazione di questi ultimi.

Effettuata l'analisi delle sfide, e osservando come queste vengono affrontate dagli sviluppatori, l'ultimo obiettivo che è stato affrontato è stato quello di progettare un agente capace di fornire consigli sullo sviluppo dei bot.

La realizzazione di quest'ultimo è stata fondamentale in quanto è stato possibile mettere in campo tutto ciò che è stato appreso nell'ambito dello sviluppo degli agenti conversazionali. Difatti per la realizzazione di Stephen è stato necessario adottare *best-practices* relative all'accessibilità, in modo che tutti gli utenti riescano facilmente ad utilizzarlo. Di particolare interesse è stato il modulo

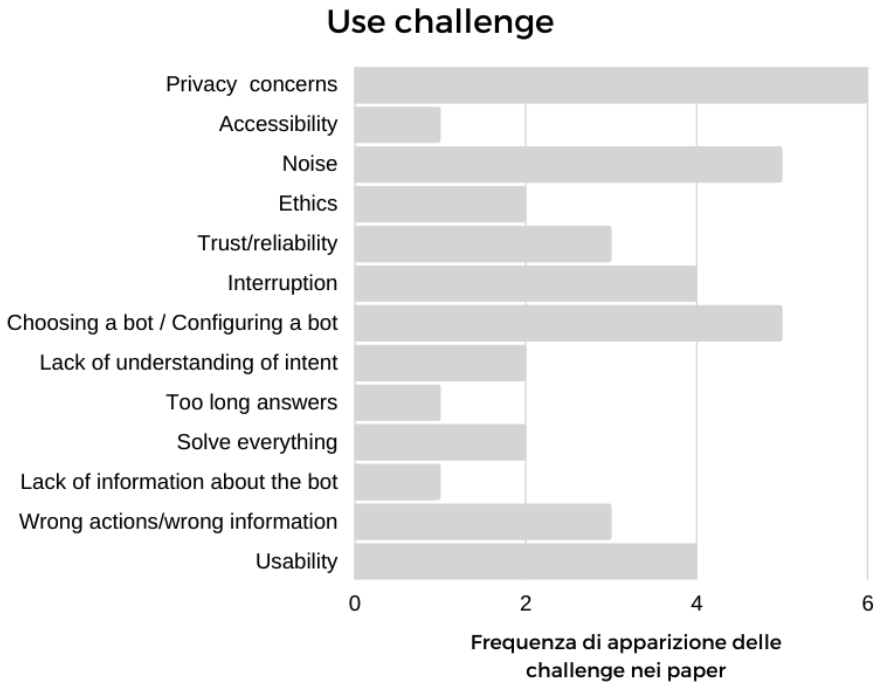

Figura 6.4: Frequenza di discussione delle *use challenge* nei paper.

di *Natural Language Processing*, per ricavare l'intento di una frase. Esso infatti ha permesso di approfondire la ricerca e l'applicazione di algoritmi e tecniche per analizzare il linguaggio naturale.

Riassumendo, il lavoro effettuato ha portato i seguenti contributi:

- È stata realizzata un analisi e categorizzazione delle sfide maggiormente presenti durante lo sviluppo e l'utilizzo dei bot.
- È stato realizzato un agente conversazionale capace di affiancare gli sviluppatori durante lo sviluppo di questi ultimi.
- Sono stati realizzati due modelli per riconoscere l'intento di una frase, il primo si basava sulle reti neurali, mentre il secondo su *Naive Bayes*. Entrambi sono stati confrontati usando delle metriche che tenevano conto di *accuracy*, *recall* e dimensione del *training set*.
- Infine l'agente e tutto il materiale è disponibile al *repository GitHub* [9]

6.2 Lavori futuri

La seguente tesi ha portato sicuramente ottimi risultati, nonostante ciò di seguito sono stati riportati alcuni lavori futuri:

Approfondire le challenge: come già è stato citato nei capitoli precedenti, le sfide che devono affrontare gli sviluppatori durante l'implementazione di un agente conversazionale sono tante. Con il passar del tempo vengo scoperti nuovi modi per affrontare queste sfide ed è importante approfondirli.

Incremento degli intenti riconosciuti da Stephen: in Stephen è stato integrato un modulo di *Natural Language Processing* per l'estrazione dell'intento da una frase. Durante l'implementazione di questo modulo, è stato deciso di considerare un piccolo sottoinsieme di "intenti" da far riconoscere al bot. Questa decisione è stata presa poiché dopo l'analisi delle *challenge* in letteratura, sono emerse ben 22 sfide, e cercare delle frasi per tutte queste non era al quanto semplice. Difatti Stephen è stato allenato considerando le sfide più frequenti delle macro-aree. Un lavoro futuro potrebbe essere quello di raccogliere nuove frasi, in modo da poter allenare l'agente e riconoscere le nuove richieste sulle restanti sfide.

Utilizzo degli agenti conversazionali in altri ambiti: La seguente tesi ha discusso dell'applicazione dei bot nell'ambito dello sviluppo *software*. Una buona iniziativa potrebbe essere quella di approfondire cosa gli agenti portano di buono in altri ambiti, come ad esempio in quelli economici, aerospaziali o addirittura in quelli medici. In questo modo è possibile che nei prossimi anni, con lo sviluppo di bot più intelligenti e con il loro contributo, gli umani riescano a raggiungere obiettivi che fin'ora erano impensabili.

Ringraziamenti

Il seguente lavoro di tesi insieme al percorso universitario è giunto al termine, in conclusione vorrei spendere delle parole per coloro che hanno permesso di raggiungere tale obiettivo.

Sono grato al Prof. Fabio Palomba, relatore di questo lavoro di tesi. Durante il mio percorso di laurea si è sempre dimostrato competente e sopratutto disponibile ad aiutare. Inoltre la ringrazio in quanto, tutto ciò che lei mi ha insegnato, è stato fonte di ispirazione per la realizzazione di tale tesi.

Ringrazio Stefano Lambiase, mio correlatore di tesi. Persona gentilissima, amichevole ma sopratutto competente e appassionata. È stato presente in tutte le fasi del mio lavoro di tesi, dandomi consigli e motivandomi. Sicuramente tutti i suoi consigli faranno parte del mio bagaglio.

Ringrazio mia madre, la quale mi ha sempre supportato in questo percorso, dal primo all'ultimo giorno. Con lei ho condiviso momenti felici e tristi, ma grazie ai suoi incoraggiamenti ho raggiunto questo bellissimo traguardo.

Ringrazio mio padre, il quale dal primo giorno ha avuto fiducia in me, dicendomi sempre di stare tranquillo, anche quando non lo ero per paura di deludere tutti. Proprio grazie a lui e le sue parole, ho raggiunto tale obiettivo.

Ringrazio Agostino, mio fratello. Durante questi anni è stato presente, ad appoggiarmi per qualsiasi decisione. Questa tesi va anche a lui, il quale in questo percorso ha sempre cercato di rendermi la strada libera.

Ringrazio Michele, con il quale condivido un forte legame. Grazie per avermi sempre ascoltato e motivato anche quando un esame non andava come volevo. Ti sono grato per tutto ciò che hai fatto per me, non lasciandomi mai solo in questo percorso.

Ringrazio Antonio, ormai compagno di una vita. Ti sono grato per tutti i tuoi consigli che

mi hai dato, i quali per me in questi anni sono stati essenziali, in più ti ringrazio poiché riesci sempre a strapparmi un sorriso, anche nelle giornate più brutte.

Infine ringrazio Alessandra e Alessandro, compagni di corso, ma anche altro. Alessandra, persona fantastica, la quale in questo percorso è stata sempre presente ad aiutarmi, e innanzitutto a supportare i miei capricci. Alessandro, anche lui una persona fantastica, sempre pronto a sollevarmi l'umore prima degli esami. Con loro ho condiviso momenti felici, ma anche momenti brutti i quali ci hanno fatto legare ancora di più.

Bibliografia

[1] microsoftbotframeworkemulator a desktop application that allows users to locally test and debug chat bots built with the bot framework sdk. `https://github.com/microsoft/BotFramework-Emulator`, 06 2018.

[2] Ahmad Abdellatif, Diego Costa, Khaled Badran, Rabe Abdalkareem, and Emad Shihab. Challenges in chatbot development: A study of stack overflow posts. In *Proceedings of the 17th International Conference on Mining Software Repositories*, pages 174–185, 2020.

[3] Eleni Adamopoulou and Lefteris Moussiades. Chatbots: History, technology, and applications. *Machine Learning with Applications*, 2:100006, 2020.

[4] Jordi Cabot, Loli Burgueno, Robert Clarisó, Gwendal Daniel, Jorge Perianez-Pascual, and Roberto Rodriguez-Echeverria. Testing challenges for nlp-intensive bots. In *2021 IEEE/ACM Third International Workshop on Bots in Software Engineering (BotSE)*, pages 31–34. IEEE, 2021.

[5] Henriquez Carlos, Sánchez-Torres German, and Salcedo Dixon. Tashi-bot: A intelligent personal assistant for users in an educational institution. 2021.

[6] Benjamin R Cowan, Nadia Pantidi, David Coyle, Kellie Morrissey, Peter Clarke, Sara Al-Shehri, David Earley, and Natasha Bandeira. " what can i help you with?" infrequent users' experiences of intelligent personal assistants. In *Proceedings of the 19th International Conference on Human-Computer Interaction with Mobile Devices and Services*, pages 1–12, 2017.

[7] James Dominic, Jada Houser, Igor Steinmacher, Charles Ritter, and Paige Rodeghero. Conversational bot for newcomers onboarding to open source projects. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, pages 46–50, 2020.

[8] James Lester, Karl Branting, and Bradford Mott. Conversational agents. *The practical handbook of internet computing*, pages 220–240, 2004.

[9] Fabio Palomba, Stefano Lambiase, and Otino Pio Santosuosso. Stephen. `https://github.com/Pio57/Stephen`, 2022.

[10] André M Pinheiro, Caio S Rabello, Leonardo B Furtado, Gustavo Pinto, and Cleidson RB de Souza. Expecting the unexpected: distilling bot development, challenges, and motivations. In *2019 IEEE/ACM 12th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, pages 51–52. IEEE, 2019.

[11] AM Rahman, Abdullah Al Mamun, and Alma Islam. Programming challenges of chatbot: Current and future prospective. In *2017 IEEE Region 10 Humanitarian Technology Conference (R10-HTC)*, pages 75–78. IEEE, 2017.

[12] Soumaya Rebai, Oussama Ben Sghaier, Vahid Alizadeh, Marouane Kessentini, and Meriem Chater. Interactive refactoring documentation bot. In *2019 19th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, pages 152–162. IEEE, 2019.

[13] Prakhar Srivastava and Nishant Singh. Automatized medical chatbot (medibot). In *2020 International Conference on Power Electronics & IoT Applications in Renewable Energy and its Control (PARC)*, pages 351–354. IEEE, 2020.

[14] Saurabh Srivastava and TV Prabhakar. A reference architecture for applications with conversational components. In *2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS)*, pages 1–5. IEEE, 2019.

[15] Margaret-Anne Storey and Alexey Zagalsky. Disrupting developer productivity one bot at a time. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 928–931, 2016.

[16] Venkatesh Subramanian, Nisha Ramachandra, and Neville Dubash. Tutorbot: contextual learning guide for software engineers. In *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*, pages 16–17. IEEE, 2019.

[17] Simon Urli, Zhongxing Yu, Lionel Seinturier, and Martin Monperrus. How to design a program repair bot? insights from the repairnator project. In *2018 IEEE/ACM 40th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*, pages 95–104. IEEE, 2018.

[18] Mairieli Wessel, Bruno Mendes De Souza, Igor Steinmacher, Igor S Wiese, Ivanilton Polato, Ana Paula Chaves, and Marco A Gerosa. The power of bots: Characterizing and understanding bots in oss projects. *Proceedings of the ACM on Human-Computer Interaction*, 2(CSCW):1–19, 2018.

[19] Mairieli Wessel and Igor Steinmacher. The inconvenient side of software bots on pull requests. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, pages 51–55, 2020.

[20] Mairieli Wessel, Igor Wiese, Igor Steinmacher, and Marco Aurelio Gerosa. Don't disturb me: Challenges of interacting with software bots on open source software projects. *Proceedings of the ACM on Human-Computer Interaction*, 5(CSCW2):1–21, 2021.

[21] Marvin Wyrich and Justus Bogner. Towards an autonomous bot for automatic source code refactoring. In *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*, pages 24–28. IEEE, 2019.

Gray literature

[G1] Ai chatbot development challenges its future by ronak patel chatbots journal. https://chatbotsjournal.com/ai-chatbot-development-challenges-its-future-4614be2f4c4e.

[G2] Implementing chatbot in educational institutes. https://ijrar.org/papers/IJRAR1AOP010.pdf.

[G3] 5 chatbot challenges and how to overcome them by allan stormon chatbots magazine. https://chatbotsmagazine.com/5-chatbot-challenges-and-how-to-overcome-them-caccc3a26d7c, 09 2017.

[G4] Chatbot challenges. https://contentrules.com/chatbot-challenges/, 10 2017.

[G5] Top 4 challenges to adopting conversation bots in the enterprise techrepublic. https://www.techrepublic.com/article/top-4-challenges-to-adopting-conversation-bots-in-the-enterprise/, 11 2018.

[G6] 4 biggest challenges in chatbot development and how to avoid them. https://insights.daffodilsw.com/blog/4-biggest-challenges-in-chatbot-development-and-how-to-avoid-them, 11 2020.

[G7] 4 reasons for enterprise chatbot failure. https://servisbot.com/four-reasons-for-enterprise-chatbot-failure/, 08 2020.

[G8] Conversational bots accenture. https://www.accenture.com/_acnmedia/pdf-77/accenture-research-conversational-ai-platforms.pdf, 04 2020.

[G9] Overcome these 6 major chatbot challenges with ease without coding engati. https://www.engati.com/blog/overcoming-chatbot-challenges-the-right-way, 08 2020.

[G10] The key challenges and benefits of utilizing chatbots for business infographic social media today. https://www.socialmediatoday.com/news/the-key-challenges-and-benefits-of-utilizing-chatbots-for-business-inf 594486/, 02 2021.

[G11] 7 reasons chatbots fail and how we can fix it customer service blog from happyfox. https://blog.happyfox.com/7-reasons-chatbots-fail-and-how-we-can-fix-it/, 04 2022.