Towards Understanding and Improving the Value of Chatbots in Software Engineering

Ahmad Abdellatif

A Thesis

in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy (Software Engineering) at

Concordia University

Montréal, Québec, Canada

December 2021

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Ahmad Abdellatif**

Entitled: **Towards Understanding and Improving the Value of Chatbots in Software Engineering**

and submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy (Software Engineering)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

_____ Chair
Dr. Ahmed Soliman

_____ External Examiner
Dr. Andy Zaidman

_____ Examiner
Dr. Ferhat Khendek

_____ Examiner
Dr. Juergen Rilling

_____ Examiner
Dr. Nikolaos Tsantalis

_____ Supervisor
Dr. Emad Shihab

Approved by

_____
Dr. Leila Kosseim, Graduate Program Director
Department of Computer Science and Software Engineering

December 17, 2021

_____
Dr. Mourad Debbabi, Dean
Gina Cody School of Engineering and Computer Science

Abstract

Towards Understanding and Improving the Value of Chatbots in Software Engineering

Ahmad Abdellatif, Ph.D.

Concordia University, 2022

Software chatbots have been around since 1966, where the first computer interacted with a human. Recently, advances in artificial intelligence and natural language processing and understanding led to the rise and widely use of chatbots in a variety of services (e.g., healthcare, e-commerce, customer service). Nowadays, chatbots have become the main conduit between humans and services. Through natural language, chatbots enable users to communicate with different services intuitively. Another reason for the increasing popularity of chatbots in several domains is their benefits in saving time, effort, and cost. These benefits and wide adoption of chatbots attract practitioners to implement chatbots that support software engineering tasks. Chatbots play an important role in various software development tasks from answering development questions to running tests and controlling services. While there are numerous chatbots and their capability of supporting software practitioners is encouraging, little is known about the development challenges and usage benefits of software engineering chatbots.

This thesis presents series of empirical studies that aim to understand the challenges of developing chatbots for the software engineering domain, highlight the value of using chatbots in software development, and proposes novel approaches to support developers at developing more efficient software engineering chatbots. More specifically, we tackle three aspects of chatbots in software engineering. First, we present an empirical study to explore the chatbot development challenges. We find that chatbot developers face several challenges that are related to chatbot integration, development, natural language understanding platforms (NLUs), user interaction, and user input.

Second, we propose a chatbot layered on top of software repositories to answer software project related questions in order to showcase the potential of chatbots in software development. We find

that practitioners are able to complete their tasks more accurately (65.6% more completed tasks) and in less time (83.3% faster) when using chatbots compared to using conventional tools. During this work we find two critical challenges in chatbot development which are selecting an NLU model for the chatbot implementation and curating a high-quality dataset to train the NLU model.

Third, we propose guidelines and approaches to improve chatbots in the software engineering domain. First, we assess the performance of multiple widely used NLUs using representative software engineering tasks to guide chatbot developers in designing more efficient chatbots. We report a guideline for chatbot developers on the best performing NLUs for intents classification and entity extraction. Finally, we investigate an approach that combines synonyms replacement and paraphrasing techniques to augment the training dataset of SE chatbots, which helps chatbot developers create high-quality datasets for training the NLU models. We find that augmenting the dataset using the combined approach does not improve the NLU's performance for intents classification. Also, the results show that using the combined approach has a negligible effect on the NLU's confidence in its classification.

Related Publications

The following publication are related to the materials presented in this thesis:

- **A. Abdellatif**, K. Badran, D. E. Costa, and E. Shihab, "A Comparison of Natural Language Understanding Platforms for Chatbots in Software Engineering", IEEE Transactions on Software Engineering (TSE), Accepted 2021.
- **A. Abdellatif**, K. Badran, and E. Shihab, "MSRBot: Using Bots to Answer Questions from Software Repositories", Springers Journal of Empirical Software Engineering, Accepted 2020
- **A. Abdellatif**, D. E. Costa, K. Badran, R. Abdalkareem, and E. Shihab, "Challenges in Chatbot Development: A Study of Stack Overflow Posts", In Proceedings of the 17th International Conference on Mining Software Repositories (MSR), Accepted 2020.

The following publications are not directly related to the material presented in this thesis but were conducted as parallel work to the research presented in this thesis.

- M. Wessel, **A. Abdellatif**, I. Wiese, T. Conte, E. Shihab, M. Gerosa, and I. Steinmacher, "Bots for Pull Requests: The Good, the Bad, and the Promising", In the 44th International Conference on Software Engineering (ICSE), Accepted 2022.
- **A. Abdellatif**, Y. Zeng, M. Elshafei, E. Shihab, and W. Shang, "Simplifying the Search of npm Packages", Elseviers Journal of Information and Software Technology (IST), Accepted 2020.
- **A. Abdellatif**, M. Alshayeb, S. Zahran, and M. Niazi, A measurement framework for software product maturity assessment, Journal of Software: Evolution and Process, Accepted 2019

Statement of Originality

I, Ahmad Abdellatif, hereby declare that I am the sole author of this thesis. All ideas and inventions attributed to others have been properly referenced. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners. I understand that my thesis may be made electronically available to the public.

Dedications

To my mom, dad, Mohammad, Saja, Rawan, Omar, Zein, Taim, and grandfather

Acknowledgments

First, I thank Almighty Allah for providing me the strength and the ability to pursue my Ph.D. thesis.

I would like to express my extremely grateful to my supervisor Dr. Emad Shihab for all of his guidance, support, and patience through out my Ph.D. journey. His immense knowledge and plentiful experience have encouraged me in all the time of my academic research and daily life. Dr. Emad, words fail me to thank you for all your effort and comments to make me a good researcher.

I want to thank my committee members, Dr. Andy Zaidman, Dr. Ferhat Khendek, Dr. Nikolaos Tsantalis, and Dr. Juergen Rilling. Also, I extend my gratitude to Dr. Weiyi Shang for his fruitful feedback about my work.

During my journey, I was lucky to collaborate and discuss my research with brightest researchers. I would like to thank Drs Rabe Abdalkareem, Diego Costa, Marco Gerosa, Igor Steinmacher, Weiyi Shang, Essam Mansour and Mairieli Wessel for sharing their insights and advices.

I would like to extend my thanks to my collogues that I was very lucky to work and collaborate with, Khaled Badran, Suhaib Mujahid, Makram Adaime, Mohamed Elshafei, Jinfu Chen, Sayed-Hassan Khatoonabadi, Jasmine Latendresse, Haya Samaana, Abbas Javan, Farbod Farhour, Mahsa Arani, Patrick Ayoup, Atique Reza, Xiaowei Chen, Riya Dutta, Nicholas Nagy, Olivier Noury, Giancarlo Sierra, Hosein Nourani, Juan Hoyos, Yi Zeng and everyone else in the Data-driven Analysis of Software (DAS) Lab for the many fruitful discussions and collaborations.

Some special words of gratitude go to my dear friends who have always been a major source of support, guidance, and love. I would like to thank Mutasim, Saif, Ammar, Ahmad, Mohammad, and Khader for being marvelous friends. Thanks guys for always being there for me.

I am extremely grateful to my mom, dad, brother, and sister for your love, prayers, and caring. Without your endless support, I would not have been able to complete my Ph.D journey. I am blessed to have you. My beloved wife and the little three musketeers (Omar, Zein, and Taim), thank you for your scarifies, love, and support through the journey. You provided me with the courage and motivation all the time to reach the goal. I dedicated this thesis to you.

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Research Statement

1.1 Introduction

The idea of chatbots was pitched in the 1960s to allow humans to interact with machines. Eliza Weizenbaum (1966) proposed the first natural language interaction between users and machines in 1966. Chatbots are a sub-category of software bots which automate tasks and have the ability to communicate with users through natural language C. Lebeuf, Zagalsky, Foucault, and Storey (2019a); C. R. Lebeuf (2018). Nowadays, chatbots are becoming more popular and attracting the attention of many practitioners and large organizations (e.g., Google, Microsoft, IBM) Daniel, Cabot, Deruelle, and Derras (2020). Moreover, they are frequently used to perform various tasks. For example, Ni, Lu, Liu, and Liu (2017) developed a chatbot that diagnoses patients and generates a report to the doctor with possible causes of the patients' symptoms. A. Xu, Liu, Guo, Sinha, and Akkiraju (2017) developed a customer service chatbot to answer users' questions on social media (e.g., Twitter, Facebook).

Prior work shows that the recent rise of chatbots is due to the advancement of artificial intelligence and natural language processing techniques C. Lebeuf, Storey, and Zagalsky (2018c). Those improvements allow chatbots to better understand the users' intentions and establish more efficient conversations, thus allowing chatbots to support more tasks and be more user-friendly. Through natural language, chatbots allow users to easily communicate with different services to perform specific tasks. This simple method of interaction enables the chatbot to serve as a suitable conduit

between users and services C. Lebeuf et al. (2018c). In other words, chatbot users can monitor and control different services without needing technical knowledge/skills to perform those tasks. In fact, previous studies show that chatbots reduce the cost, time, and effort required to achieve certain goals by automating difficult and tedious tasks Storey and Zagalsky (2016a).

Given its benefits, chatbot adoption in the Software Engineering (SE) domain is increasing Paikari and van der Hoek (2018). Beside automating repetitive tasks, chatbots help developers in their software development tasks by extracting information easily and efficiently Daniel and Cabot (2021); Storey and Zagalsky (2016b), resolving code conflicts Paikari et al. (2019), and answering development questions using Stack Overflow Murgia, Janssens, Demeyer, and Vasilescu (2016b); Romero, Parra, and Haiduc (2020a).

While there are numerous chatbots and their capability of supporting software practitioners is encouraging, little is known about chatbots in the Software Engineering domain. Specifically, the challenges of developing chatbots that assist software practitioners in different tasks and the chatbot usage-benefits in the SE domain. To identify the chatbot development challenges, we examine the difficulties that face chatbot practitioners to ease developing and integrating chatbots in the software development process. Moreover, we develop a software engineering based chatbot that leverages code and issues tracking software repository data to answer software project related questions to highlight the value of using chatbots in SE. In this thesis, we aim to 1) identify the challenges that face practitioners when developing chatbots, 2) understand the benefits of using chatbots in the software development, and 3) improve the SE-based chatbots.

1.2 Problem Statement

Recent studies show that chatbots are becoming more popular in the SE domain due to their benefits in many other domains (e.g., customer services, e-commerce) such as improving productivity and saving resources Daniel and Cabot (2021). However, the challenges that face practitioners in developing SE-based chatbots and their usage benefits in the SE domain are still unexplored areas. Understanding these aspects is of paramount importance because they enable to design of more effective chatbots and increase their adoption by software practitioners. This concern led to the

formulation of this thesis problem statement, which is stated as follows:

Given that chatbots are beneficial in several domains, we believe that the full potential of using chatbots in the software development process remains untapped. Therefore, we conduct empirical studies to gain insights about the chatbot development challenges, understand the value of using chatbots in the software development, and propose approaches to design more effective chatbots for software engineering.

1.3 Thesis Organization

First, the thesis provides a background and discusses work related to chatbots (Chapter 2). The remainder of the thesis is divided into two parts. Each part focuses on studying one aspect of this thesis goal:

- **Part I: Understanding the chatbot development challenges and benefits of using chatbots in the software development process.** We present a study that highlights the development challenges that face chatbot developers. Our work illustrates the most pressing and difficult challenges to develop chatbots. In addition, we present a chatbot that answers software project related questions. And we evaluate the proposed chatbot in a real-life scenario by asking SE practitioners to use it. Our work illustrates the potential benefits of using chatbots to assist practitioners in their development tasks. [Chapter 3 and 4]

- **Part II: Improving the chatbots in the SE domain.** We present a guideline on the best performing NLUs for intents classification and entity extraction using SE tasks to develop more efficient SE-based chatbots. In addition, we evaluate an approach to help developers augment training datasets for their chatbots. Our work illustrates how to improve chatbots in the SE domain. [Chapter 5 and 6]

1.4 Thesis Overview

In this section, we provide an overview of the work presented in this thesis and highlight the main results of each work.

Chapter 2: Background and Related Work.

Before diving into the chatbot development challenges, we first present a background of the chatbots and the used terminologies throughout this thesis. Then, we discuss the existing work related to using software bots in software engineering. Moreover, we present studies related to developing chatbots that assist practitioners in their daily development tasks. Finally, we discuss the work that envisions the use of chatbots in the SE domain.

Part I: Understanding the chatbot development challenges and benefits of using chatbots in the software development process.

A large body of prior work focuses on proposing chatbots to assist developers in their daily development tasks Bradley, Fritz, and Holmes (2018); Dominic, Houser, Steinmacher, Ritter, and Rodeghero (2020a); Paikari et al. (2019); Qasse, Mishra, and Hamdaqa (2021); Wessel and Steinmacher (2020). However, the challenges that face practitioners in developing chatbots and their usage-benefits in the SE domain are unexplored areas. In this part, we explore the development challenges by examining the chatbot developers' posts on Stack Overflow. In addition, we develop an SE-chatbot to highlight the potential benefits of using chatbots in software development.

Chapter 3: Understanding the challenges of chatbot development.

Using chatbots improves the productivity of software practitioners by automating redundant tasks, controlling services through natural language, and notifying practitioners about critical events in the software development (e.g., build or service failures) C. Lebeuf et al. (2018c); Storey and Zagalsky (2016b). While using chatbots has a positive impact on the software development process, little is known about the challenges that encounter practitioners during designing and developing SE-based chatbots and adopting them in the development process. Prior work shows that developing chatbots

requires special expertise from developers which makes chatbot development different than the traditional software development Daniel et al. (2020). For example, chatbot development requires machine learning and natural language processing knowledge to analyze and process the user's input, and perform actions based on that input Jha (2019). Moreover, developing chatbots requires knowledge in designing the chatbot's conversation flow with the users. Therefore, as a first step, our goal is to have a general understanding of the chatbot development process and its challenges. More specifically, we examine the challenges related to chatbot development by analyzing the chatbot developers' questions on the development Q&A website (Stack Overflow).

In Chapter 3, we perform one of the first in-depth studies to understand the development challenges that face chatbot practitioners. We mine Stack Overflow posts related to chatbots and leverage topic modeling to understand the chatbot topics that are being by developers on Stack Overflow. We find that developers discuss topics related to chatbot integration, development, natural language understanding platforms (NLUs), user interaction, and user input. Also, our results show that developers are looking for specific features implementation. More importantly, we find that the most challenging topics are related to training chatbot models. This work was accepted as a full paper in the International Conference on Mining Software Repositories Abdellatif, Costa, Badran, Abdelkareem, and Shihab (2020).

Chapter 4: Determining the value of SE-context chatbots.

Software repositories contain a plethora of software development data that helps to improve the software quality Hassan (2008); Khomh, Adams, Dhaliwal, and Zou (2015). Software practitioners struggle to extract this data to answer questions related to their software project (e.g., "Which commits fixed the bug 5012?"). Practitioners need special expertise to perform such tasks, especially when the data is scattered in different repositories (e.g., code and issue repositories). Moreover, even if the practitioners have the required skills to extract data from software repositories, this task requires effort and time.

To understand and showcase the potential of chatbots in the software development, we develop a chatbot that answers software project related questions. We layered a chatbot on top of software

repositories to help practitioners extract software repository information related to their development and maintenance tasks. Then, we perform a study with software practitioners to evaluate the effectiveness of the proposed chatbot and its impact on the practitioners' productivity. We find that practitioners using the chatbot completed 90.8% of the tasks accurately (90.8%) on median of 40 seconds per task, compared to 25.2% of the tasks were completed on median of 240 seconds per task when not using the chatbot. Moreover, most of the participants (90.8%) find the chatbot to be either useful or very useful. We observe that selecting and training a chatbot's model is challenging for SE-based chatbots. This is because selecting a suitable model (i.e., NLU) is crucial for the chatbot to understand the user's queries. Another observations is that training a chatbot's model requires a lot of resources and efforts to curate a high-quality training dataset. The results of this research have been published in the Journal of Empirical Software Engineering Abdellatif, Badran, and Shihab (2020a).

Part II: Improving the chatbots in the SE domain.

In Part I, we highlight the chatbot development challenges and explore the benefits of using chatbots in the SE domain. In this part, we shift our focus towards helping chatbot developers to design more effective SE chatbots. To achieve this, we target two of the main challenges related to the NLUs that face chatbot developers, which we uncoverd in Part I (Chapter 3 and 4) of this thesis. More specifically, we evaluate the performance of NLUs using SE tasks to guide chatbot practitioners in the NLU selection for their SE-based chatbots. Then, we evaluate a data augmentation approach that helps practitioners craft training datasets for chatbots in the SE-domain.

Chapter 5: Can we help developers to design more effective chatbots for the SE domain?

At the heart of all chatbots lies a Natural Language Understanding (NLU) component that enables chatbots to understand the user's input. One of the main tasks of NLU is to extract structural information from unstructured language input posted by the user Abdellatif, Badran, and Shihab (2020a). The chatbot's response highly depends on whether the NLU extracts the information correctly. For example, if the NLU misclassifies the user's intention from the question, the chatbot performs an

incorrect action and/or returns an incorrect reply. Developing an NLU from scratch is a very difficult task as it needs AI and NLP expertise, and thus, chatbot developers resort to a handful of widely-used NLUs that they leverage in their chatbots Marbot (2019); Munoz, Araque, Llamas, and Iglesias (2018); Murgia, Janssens, Demeyer, and Vasilescu (2016a); Toxtli, Monroy-Hernández, and Cranshaw (2018). However, there is no consensus on the best NLU to use in the chatbot implementation Damir (2020); T. G (2016); Nick (2018). Moreover, developers of chatbots that operate in the SE domain report facing challenges when selecting the best NLU for the domain Abdellatif, Badran, and Shihab (2020a); Dominic et al. (2020a). Choosing a suitable NLU for the chatbot implementation is a critical task as it directly impacts the user satisfaction Lastra (2016); C. Lebeuf, Storey, and Zagalsky (2018d).

To guide chatbot practitioners in the NLU selection for their SE-based chatbots, we perform an empirical study to assess the performance of four widely-used NLUs, namely IBM Watson, Rasa, Google Dialogflow, and Microsoft LUIS in intents classification using representative SE tasks. Also, we evaluate the NLUs using different features (i.e., list and prediction features) for extracting a piece of information (e.g., bug ticket ID) from the user's query. We find that IBM Watson achieves the best in intents classification with an F1-measure greater than 84%. For the entity extraction, Microsoft LUIS and IBM Watson outperform other NLUs in the two SE tasks. Finally, we find that NLUs tend to better classify intents that have a higher number of training examples compared to intents with fewer numbers of training examples. This work was published in IEEE Transactions on Software Engineering journal Abdellatif, Badran, Costa, and Shihab (2021b).

Chapter 6: Improving the SE chatbot's accuracy.

Chatbot developers train the NLU model on queries that present the different ways users rephrase their queries to the chatbot. Training the NLUs on more queries yields better performance in terms of intents classification Abdellatif et al. (2021b). Nevertheless, building and collecting a training dataset for chatbots in the SE domain is a costly and time-consuming task Abdellatif, Badran, and Shihab (2020a); Dominic et al. (2020a). This is because the chatbot developers dedicate time and effort to brainstorm and create training queries for the chatbot. Moreover, there is a lack of user-chatbot interaction datasets that are available for training the SE chatbots.

In Chapter 6, we evaluate an approach that combines NLP techniques to help practitioners augment datasets for SE-based chatbots. The approach takes as an input the dataset used to train the NLU, and outputs a dataset with more augmented queries. The main goal of the approach is to augment new queries that have new keywords, brand new sentence structure, and maintains the semantic (intent) of the queries in the original dataset. To evaluate the impact of the approach on the NLU's performance, we conduct an empirical study using three SE datasets. We find that using the combined approach to augment the dataset for SE chatbots does not improve the NLU's performance in intents classification. Moreover, the results show that the approach does not improve the NLU's confidence in its classification. Our results should alarm the research community on the limitations of current augmentation approaches when applied on software engineering datasets.

1.5 Thesis Contributions

The main contributions of this thesis are as follows:

- We discover and report that developers face different chatbot developments challenges that are related to chatbot integration, development, natural language understanding platforms (NLUs), user interaction, and user input.
- We develop a chatbot that leverages code and issue tracking software repository data to answer software project related questions to highlight the value of using chatbots in SE.
- We compare the performance of four widely-used NLU on tasks from the SE domain. Also, we assess the NLUs using different features for extracting entities from the user's query.
- We evaluate the value of a dataset augmentation approach for SE chatbots to enhance the NLU's performance and reduce the cost of manual augmentation of the training dataset.
- We provide a guideline for practitioners to improve the performance of their SE-based chatbots. Moreover, we make the tools, approaches, and datasets that used in this thesis publicly available to accelerate the future research for chatbots in the SE domain.

Chapter 2

Background and Related Work

In this chapter we provide an overview of the relevant background to our research which includes a brief description chatbots, and related terminology used throughout the report. Also, we present the work to software bots in the software engineering domain, using chatbots to assist developers in their development tasks, and visions for the future use of chatbots.

2.1 Background

Storey and Zagalsky defined bots as tools that perform repetitive predefined tasks to save developer's time and increase their productivity Storey and Zagalsky (2016b). They outlined five areas where they see bots as being helpful: code, test, DevOps, support, and documentation. A recent study showed that 25% of the project on Github are using at least one bot Wessel et al. (2018a). For example, Greenkeeper (2019) bot updates the dependencies in the project if they pass the CI tests, and the bot opens an issue in case the test fails to notify the developers about which package update breaks the build. Kubernetes Prow Kubernetes (2016) tests the pull requests and merges them if the tests passed. Snyk (2019) searches for vulnerabilities in the project dependencies and submits a pull request to fix them.

Chatbots are a sub-category of software bots which have the ability to communicate with users through natural language C. Lebeuf et al. (2019a); C. R. Lebeuf (2018). Chatbots serve as the

conduit between their users and automated services C. Lebeuf et al. (2018d). Through natural language, users can ask the chatbot to perform a specific task or inquire about a piece of information. At the heart of all chatbots lies a natural language platform (NLU). NLUs are essential for the chatbot's ability to understand and act on the user's input. The main goal of the NLU is to extract structured data from unstructured language input. In particular, it extracts intents and entities from users' queries. **Intents** represent the user intention/purpose of the question, while **entities** represent important pieces of information in the query. For example, in the query "How to detect HTML elements movements ?", the intent is to know how to detect the movements of the elements (i.e., looking for code sample). On the other hand, the 'HTML' is an entity of type programming language. Chatbots uses both the intent and entities to perform the action that answers the user's question. In this example, the chatbot searches for HTML code sample that detects the elements' movements and returns it to the user.

To use an NLU on a specialized domain, chatbot developers should define a set of custom entities and intents. Then, for each custom intent, the NLU needs to be trained on a set of queries that represents the different ways a user could ask for that intent. For example, the following two queries "how many commits happened in the last month?" and "show me the number of commits between 1-09-2019 and 30-09-2019" have the same semantics but different syntax. Both questions can be used to train the NLU on the different ways a user could ask a question with 'get the number of commits by date' intent. Similarly to custom intents, NLUs need to be trained to properly recognize custom entities. To do that, developers can label the entity types and their values in the queries. For example, in the following query "what is the fixing commit for bug HHH-8501?", the entity 'HHH-8501' is labeled as a Jira ticket type.

2.2 Software Bots

Software bots are getting more attention from Software Engineering practitioners Storey and Zagalsky (2016b). For example, Phaithoon et al. (2021) developed FixME bot to help project maintainers detect and track the self-admitted technical debts that wait a specific bug resolution before fixing the debt. Wyrich and Bogner (2019) implemented a bot that performs code refactoring for

incorrect order of language modifiers, missing override annotation, commented-out code smells, and unused method parameters and creates a pull request with the new changes. Basu and Banerjee (2021) implemented a bot that assigns Jira tickets among the engineers at Walmart in a round-robin manner to improve the ticket resolution time. Urli, Yu, Seinturier, and Monperrus (2018a) developed the Repairnator which is a repair bot for Java programs. The Repairnator monitors the CI of the project and in the case of a test failure, the bot reproduces the bug and generates a fix using NPEFix Cornu, Durieux, Seinturier, and Monperrus (2015), Nopol Xuan et al. (2017), and Astor Martinez and Monperrus (2016). Carvalho et al. (2020) designed the C3PR (Code Check and Correction via Pull Requests) bot that finds static analysis issues in the code. Then, it generates the fix using ESLint, TSLint, and WalkMod Sonar plugin and submits it as a pull request to be reviewed by the developers. Şerban, Golsteijn, Holdorp, and Serebrenik (2021) developed a bot that suggests fixes to the static analysis warnings for the project maintainers. Kumar et al. (2019) implemented the Sankie bot which recommends reviewers for pull requests. Sankie's developers evaluated it on 50 repositories at Microsoft for 2 weeks and their results showed that 72% of the reviewers added by Sankie ended up interacting in the pull request. On the other hand, researchers proposed taxonomies to better understand and characterize of the existing software bots Erlenhov, de Oliveira Neto, Scandariato, and Leitner (2019); C. Lebeuf et al. (2019a). C. Lebeuf et al. (2019a) designed a multi-faceted taxonomy that is related to the environment, intrinsic, and interaction properties of bots. Erlenhov et al. (2019) proposed a DevBots taxonomy to support the process of classifying and understanding the different bots used by software practitioners. Erlenhov, Neto, and Leitner (2020) identified three personas of DevBots based on their autonomy, chat interfaces, and smartness.

The vast amount of research on developing software bots to perform software development tasks (e.g., resolve code conflicts) motivate our work. In this research proposal, we propose a chatbot framework to answer users' questions using project repository data.

2.3 Using Chatbots to Assist Developers

Chatbots were proposed in the 1960s to enable interaction between humans and machines. For example, Weizenbaum (1966) in 1966 developed Eliza chatbot that uses pattern matching and substitution methodology to act as a psychotherapist and answers patients' questions. In 1973, Cerf (1973) developed Parry chatbot that mimics a patient with paranoid schizophrenia. Later in the 1990s, Wallace (1995) developed ALICE that imitates a human over the internet and engages in a conversation with a user. The recent advances in artificial intelligence and natural language processing led to the current rise and wide use of chatbots to perform different tasks C. Lebeuf et al. (2018c) such as assisting customer service A. Xu et al. (2017), answering student admission questions Agus Santoso et al. (2018), and helping health care workers Cameron et al. (2018). Recently, chatbots has attracted much attention in the SE domain to support software developers in their daily tasks Bradley et al. (2018); Dominic et al. (2020a); Qasse et al. (2021); Wessel and Steinmacher (2020). For example, Bradley et al. (2018) developed Devy to assist developers in their basic development tasks (e.g., commit a code). Then, the authors evaluate Devy through asking 21 software practitioners to complete two tasks using Devy. The results show that all participants completed both tasks successfully. Qasse et al. (2021) developed a chatbot, called iContractBot, that assists developers in developing and modeling contracts (i.e., self-executed program codes) in blockchain platforms. Dominic et al. (2020a) developed a chatbot to assist in the project onboarding process for newcomers. Wessel and Steinmacher (2020) introduced meta-bot to serve as an intermediate between developers and different bots on Github. With that, developers can manage Github bots and their notifications by simply interacting with the meta-bot through natural language. Paikari et al. (2019) developed a chatbot prototype to detect and resolve code conflicts that arise when multiple developers work on the same file.

A number of studies have focused on implementing chatbots to answer software development related questions Chun-Ting Lin and Huang (2020); Romero, Parra, and Haiduc (2020b); Tian, Thung, Sharma, and Lo (2017a); B. Xu, Xing, Xia, and Lo (2017a). B. Xu et al. (2017a) developed AnswerBot to answer software development questions using Stack Overflow. AnswerBot extracts the

answers using related Stack Overflow posts and summarize the answers to respond to user's question. Vale and Maia (2021) developed an assistant to answer development related questions using a transformer-based model (GPT-2). Tian et al. (2017a) developed APIBot, a chatbot that is able to answer developers' questions about a specific API using that API's documentation. Chun-Ting Lin and Huang (2020) developed MSABot to help developers in the development and maintenance of Microservices-based systems. Romero et al. (2020b) proposed a GitterAns, a chatbot that answers development questions in online Gitter chatting platform.

Prior work focused on proposing chatbots that answer users' questions using external sources of information (e.g., Stack Overflow and API documentation) and support software practitioners in their development tasks. However, none of this work has used an internal source of information (repositories data) to support developers. One of the main goals of this thesis is to empirically validate the challenges of developing SE chatbots. Moreover, we showcase a chatbot that uses an internal source of information to answer software project related questions.

2.4 Visions for the Future Use of Chatbots

In addition to the visionary work by Storey and Zagalsky Storey and Zagalsky (2016c) which presented a cognitive support framework in the bots landscape, a number of other researchers proposed work that laid out the vision for the integration of bots in the Software Engineering domain. In many ways, this visionary work motivates our bot framework. Acharya, Parnin, Kraft, Dagnino, and Qu (2016) proposed the idea of code drones, a new paradigm where each software artifact represents an intelligent entity. The authors outline how these code drones interact with each other, updating and extending themselves to simplify developers' lives in the future. They see the use of bots as a key to bringing their vision to life. Similarly, Matthies, Dobrigkeit, and Hesse (2019) envisioned a bot that analyzes and measures the software project's data to help development teams track their project progress. Researchers also envisioned bots that generate fixing patches and validate the refactoring and bug fixes van Tonder and Goues (2019), and explain those fixes to the developers Monperrus (2019). Dominic, Houser, Steinmacher, Ritter, and Rodeghero (2020b) proposed a vision of a chatbot to help in the onboarding of newcomers to OSS projects by providing resources

and recommending experienced developers for assistance.

Beschastnikh, Lungu, and Zhuang (2017) presented their vision of an analysis bot platform, called Mediam. The idea of Mediam is that developers can upload their projects to GitHub and allow multiple bots to run on them, which will generate reports that provide feedback and recommendations to developers. The key idea of this vision is that bots can be easily developed and deployed, allowing developers quick access to new methods developed by researchers. Robillard et al. (2017) envisioned a future system (OD3) that produces documentation to answer user queries. The proposed documentation is generated from different artifacts i.e. source code, Q&A forums, etc.

The gap between this visionary work and industrial use, motivates our work as we aim to bridge this gap by bringing the visionary work to life. In our research proposal, we propose a chatbot framework to support software developers that have different levels of technical knowledge in their daily tasks. And, we examine the chatbot usages on GitHub from software developers' perspective.

2.5 Chapter Summary

This chapter provides a background about the chatbots and their relevant definitions. Then, it surveys previous work on software bots in the software engineering domain. Next, it presents prior research on using chatbots to help developers in their tasks and discusses visionary work on of integrating chatbots in the software engineering domain. In the next chapter, we describe our empirical study to identify the chatbot development challenges. Also, we present our proposed chatbot framework to help answering software project related question.

Chapter 3

Understanding the challenges of chatbot development

Chatbots are becoming increasingly popular due to their benefits in saving costs, time, and effort. This is due to the fact that they allow users to communicate and control different services easily through natural language. Chatbot development requires special expertise (e.g., machine learning and conversation design) that differ from the development of traditional software systems. At the same time, the challenges that chatbot developers face remain mostly unknown since most of the existing studies focus on proposing chatbots to perform particular tasks rather than their development. Therefore, we answer this question by examining the Q&A website, Stack Overflow, to provide insights on the topics that chatbot developers are interested and the challenges they face. In particular, we leverage topic modeling to understand the topics that are being discussed by chatbot developers on Stack Overflow. Then, we examine the popularity and difficulty of those topics. Our results show that most of the chatbot developers are using Stack Overflow to ask about implementation guidelines. We determine 12 topics that developers discuss (e.g., Model Training) that fall into five main categories. Most of the posts belong to chatbot development, integration, and the natural language understanding (NLU) model categories. On the other hand, we find that developers consider the posts of building and integrating chatbots topics more helpful compared to other topics. Specifically, developers face challenges in the training of the chatbot's model. We believe that our study

guides future research to propose techniques and tools to help the community at its early stages to overcome the most popular and difficult topics that practitioners face when developing chatbots. The result of this research question appears in the International Conference on Mining Software Repositories Abdellatif, Costa, et al. (2020).

3.1 Introduction

More than 50 years after Weinzebaum introduced the first computer program to have a conversation with humans Weizenbaum (1966), chatbots have become the main conduit between humans and services Storey and Zagalsky (2016b). Potentialized by the recent advances in artificial intelligence and natural language processing C. Lebeuf et al. (2018c), chatbots are the primary interface in a variety of services, from smart homes Baby, Khan, and Swathi (2017); Valtolina, Barricelli, and Gaetano (2020) and personal assistants Apple (2020); Google (2020a), to health care Care (2020) and E-commerceSumo (2020). Given how chatbots reduce the operational costs of services, the usage of chatbots will only increase - experts predict that 85% of users' interactions with services will be done through chatbots by 2021 Milenkovic (2019).

Due to their importance and popularity, developing and maintaining chatbots is becoming more important. In addition, the development of chatbots requires expertise in specialized areas, such as machine-learning and natural language processing, which, distinguishes it from traditional software development Jha (2019). While recently introduced chatbot frameworks (e.g., Microsoft Bot Framework Microsoft (2020b)) have reduced the barrier to entry of creating chatbots, e.g., by providing the components for user interaction and natural language understanding platforms, little is known about the specific challenges that chatbot developers face when developing chatbots. Understanding such challenges is of paramount importance, helping the research community provide more effective tools for chatbot development, improving their quality, and ultimately increasing their adoption and usefulness among users.

In this chapter, we provide the first attempt at understanding the challenges of chatbot development by investigating what chatbot developers are asking about on Stack Overflow. We study Stack

Overflow since it is the most prominent code-centric Q&A website and used constantly by the development community to communicate their challenges and issues, provide solutions and foment discussions about all aspects in software development Abdalkareem, Shihab, and Rilling (2017); Overflow (2019b). Our investigation dives into the chatbot-related posts on Stack Overflow to pinpoint the major topics surrounding the discussions on chatbot development. We use well-known topic modeling techniques to group the posts into cohesive topics and apply a series of quantitative analyses, both through metrics and manual analysis. Specifically, our work investigates the following research questions:

- **RQ1: What topics are chatbot developers asking about?** We find that chatbot developers ask about 12 main topics that can be grouped into 5 main categories. The categories are related to chabot integration, development, natural language understanding (NLU), user interaction, and User Input. The most popular questions include those related to chatbot creation, integration, and user interface.

- **RQ2: What types of questions are chatbot developers asking?** Chatbot developers use Stack Overflow primarily as a source of guidance for specific implementation routines, working examples, and troubleshooting. This shows a need for better documentation that provides real-scenarios and more information about the NLU models used by chatbots.

- **RQ3: Which topics are the most difficult to answer?** The most difficult topics are related to training the chatbot NLU models. On the other hand, posts related to traditional software development, e.g., chatbot development framework, are more frequently answered, albeit, we did not find any statistically significant correlation between the popularity and difficulty of the chatbot topics in our study.

In addition to the identified chatbot topics in Stack Overflow, we discuss the evolution of the chatbot topics on Stack Overflow and find that the chatbot-related discussions have increased substantially since 2016. The activity of some categories are linked to the releases of chatbot platforms. Also, we compare the chatbot topics to other mature SE fields (e.g., mobile and security) in terms of popularity and difficulty. Our results show that the chatbot community needs more effort to reach the maturity level of similar SE fields.

Our findings show that platform owners need to improve their current documentation and integration with popular third-parties. Moreover, we believe that our study guides future research to focus on the most popular and challenging chatbot topics.

3.1.1 Organization of the Chapter

The rest of the chapter is organized as follows. Section 3.2 describes our methodology. Section 3.3 reports our empirical study results. Section 3.4 discusses our results and the implications of our findings. Section 3.5 discusses the threats to validity, and Section 3.6 concludes the chapter.

3.2 Methodology

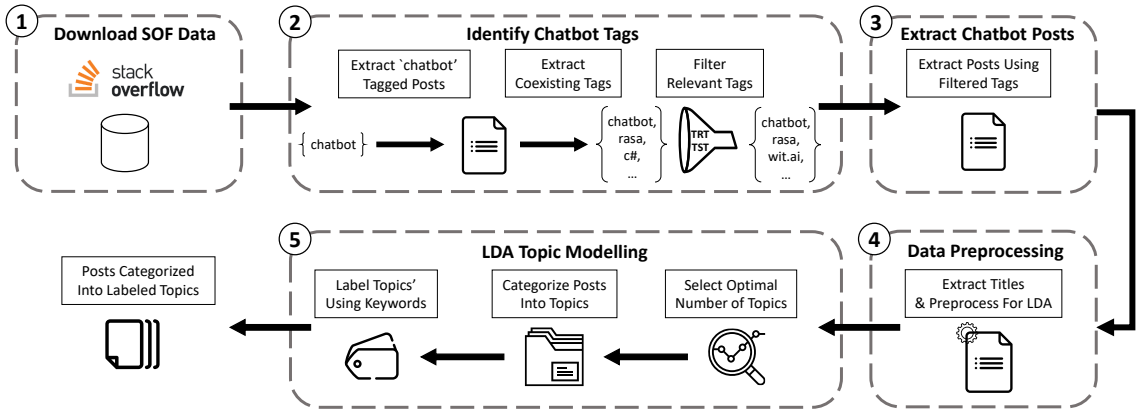


Figure 3.1: Overview of the methodology of our study.

The main *goal* of our study is to examine what chatbot developers are asking about. To achieve this goal, we resort to analyze the developers' discussions on Stack Overflow as it provides a rich dataset and have been used by similar investigations in other domains, such as concurrency S. Ahmed and Bagherzadeh (2018), cryptography APIs Nadi, Krüger, Mezini, and Bodden (2016), and deep learning Han, Shihab, Wan, Den, and Xia (2019). While providing structured data with questions, answers and their respective metadata (e.g., accepted answers), Stack Overflow does not contain any fine-grained topic information related to chatbots. Hence, we first need to identify the posts from Stack Overflow that are related to chatbots, group them according to their dominant topic, and then conduct our analysis. As Figure 3.1 shows, we perform the selection of chatbot related posts

in a methodology of five-steps, which will be detailed further in this section.

Step 1: Download & extract Stack Overflow dump. We download the entire Stack Overflow dump (last updated 4 September 2019) Exchange (2019), containing user questions, answers, and the metadata of the posts (e.g., view count, creation date) for the period between August 2008 and September 2019. The initial dataset contains approximately 18 million questions and 28 million answer posts.

Step 2: Identify chatbot tags. Stack Overflow holds posts on a myriad of different software development topics (e.g., Java, security, and blockchain). Posts are typically tagged by their authors with commonly used tags (e.g. chatbot, web) to improve the posts' visibility and chances of being answered Barua, Thomas, and Hassan (2014). To identify the most relevant chatbot-related tags, we follow the approach used by prior work Bagherzadeh and Khatchadourian (2019); Rosen and Shihab (2016), and create a *tag set* using the following procedure. First, we retrieve all posts with the 'chatbot' tag, yielding a set of 2,116 posts. We refrain from adding any other tags in this inital step to reduce the chances of introducing noise, as this will be used to identify other chatbot-related tags. Second, we extract all the tags that co-exist with the 'chatbot' tag from the chatbot-tagged posts. Next, we use two heuristic metrics used in prior work to obtain a bigger set of chatbot-related tags Rosen and Shihab (2016); Wan, Xia, and Hassan (2019). The first metric is the *tag relevance threshold (TRT)*, a measure of how related a specific tag is to the chatbot-tagged posts. This measure calculates the ratio of the chatbot-related posts (posts that include the 'chatbot' tag) for a specific tag compared to the total number of posts for that tag. Specifically, the TRT is measured using the equation $TRT_{tag} = \frac{No.\,of\,chatbot\,posts\,for\,the\,tag}{Total\,no.\,of\,posts\,for\,the\,tag}$. For example, 'rasa' is a tag with a TRT of 21.2%, which means that 21.2% of the posts tagged with 'rasa' are also tagged with 'chatbot'. By using the TRT we are able to eliminate the irrelevant tags from our set.

However, some tags that have a small number of posts (e.g., the 'botlibre' tag has only 3 posts) can have a high TRT of (33.3%) because a single one of their posts is chatbot-related, and this may introduce insignificant tags. Therefore, we use a second metric, the *tag significance threshold (TST)*, which is a measure of how prominent a specific tag is in the chatbot-tagged posts Rosen and Shihab (2016); Wan et al. (2019). This metric is measured by using the total number of the chatbot posts for that tag and the total number of the chatbot posts for the most popular tag ('chatbot' tag with

2,116 posts.) as follows $TST_{tag} = \frac{No.\,of\,chatbot\,posts\,for\,the\,tag}{No.\,of\,chatbot\,posts\,for\,the\,most\,popular\,tag}$. For example, the 'rasa' tag has a TST of 0.3% which means that the total number of the posts that are tagged with 'rasa' and 'chatbot' at the same time are equal to 0.3% of the total number of chatbot-related posts for the 'chatbot' tag.

We consider a tag to be significant and relevant to the chatbot posts if its corresponding TRT and TST are above a certain threshold. The first three authors, with varying degrees of chatbot development experience, independently examined the tags with different TRT and TST thresholds. For each tag, we inspect a randomly selected sample of posts, to identify when the tags become less relevant and less specific to chatbots, to identify the most appropriate TRT and TST thresholds. This method has been used by several previous similar studies Bagherzadeh and Khatchadourian (2019); Rosen and Shihab (2016) and has the goal of selecting tags relevant to chatbots without including too much noise in the dataset. Then, we discussed the chosen thresholds to reach a consensus on the optimal TRT and TST values. The first three authors independently evaluated the optimal TRT and TST thresholds that yield the best results and discussed their choices to reach a consensus. We find that tags with a TRT value higher than 11% and a TST value higher than 0.14% value yield an appropriate balance between the inclusion of more posts related to chatbots (i.e,. more representative dataset) and the filtering of posts that are unrelated to chatbots (i.e., less noise). It is important to note that our thresholds are in-line with the thresholds used by previous studies that adapted the same approach S. Ahmed and Bagherzadeh (2018); Bagherzadeh and Khatchadourian (2019); Yang, Lo, Xia, Wan, and Sun (2016). Finally, we use the selected TRT and TST thresholds to identify our tag set. Table 3.1 shows the tags obtained in our tag set and their respective TRT and TST values.

Step 3: Extract chatbot posts. After obtaining the chatbot-related tag set, we use those tags (see Table 3.1) to extract the posts that will constitute our chatbot dataset throughout this study. We extract this corpus by querying all posts on Stack Overflow that are tagged with one of the tags in our tag set. This process yielded a dataset containing 3,890 chatbot posts and their respective metadata.

Step 4: Preprocessing chatbot posts. We filter out the irrelevant information before applying the topic modeling techniques. In this analysis, we focus only on the posts' titles, as opposed to their

Table 3.1: The tag set used to identify the chatbot related posts. The TRT and TST are expressed in percentages.

Tag Name	TRT	TST	Tag Name	TRT	TST
chatbot	100	100	aws-lex	14.3	0.6
facebook-chatbot	42.1	6.2	sap-conversational-ai	50	0.5
amazon-lex	22.2	4.3	chatfuel	26.3	0.5
rasa-nlu	18.4	2.9	pandorabots	41.2	0.3
aiml	27.6	2.6	rasa	21.2	0.3
rasa-core	22.6	2.4	chatbase	18.2	0.3
wit.ai	13.1	1.9	chatscript	30.8	0.2
chatterbot	25.4	1.6	rivescript	28.6	0.2
api-ai	11.4	0.8	program-o	37.5	0.1
web-chat	13.6	0.8	botpress	33.3	0.1
gupshup	27.1	0.6	lita	25	0.1

body contents, as the content in the posts' bodies can introduce noise to our analysis. This approach of using the posts' titles has been used in the prior investigations Rosen and Shihab (2016), as a post's title has been shown to be representative of the post body Chen, Chen, Xing, and Xu (2016); B. Xu et al. (2017a). After extracting the posts' titles, we prepare the data to be used in the topic modelling process. To do so, we leverage the Python NLTK (NLTK) (2019a) and Gensim Gensim (2019) tools to perform the preprocessing steps on our dataset. First, we remove the stopwords, such as 'how', 'a' and 'can', using the NLTK stopwords corpus (NLTK) (2019b) as those words hinder the process of differentiating between topics. Next, we build a bigram model using Gensim since we notice that some words commonly appear together (e.g., 'Rasa NLU' and 'Bot Framework') and the topic modelling technique should consider them together. Moreover, we lemmatize the words to map them to their origin (e.g., 'development' is mapped to 'develop'). Those steps output a preprocessed dataset that is ready to be inputted to the topic modelling technique in our next step.

Step 5: Identify chabot topics. To identify the topics that are discussed by chatbot developers on Stack Overflow, we use the Latent Dirichlet Allocation (LDA) modeling technique Blei, Ng, and Jordan (2003), which has been widely used in Software Engineering studies Bagherzadeh and Khatchadourian (2019); Rosen and Shihab (2016). LDA groups the posts of our dataset into a set of topics based on the word frequencies and their co-occurrences in the posts. In particular, LDA

Table 3.2: The chatbot topics, categories, and their popularity.

Main Category	Topic	# Posts	Avg. Views	Avg. Favourites	Avg. Scores
Integration	API Calls	264	354.2	1.2	0.5
	Messenger Integration	**463**	638.0	1.4	0.7
	NLU Integration/Slots	388	406.0	1.1	0.8
Development	General Creation/Integration	250	**671.6**	**3.1**	0.6
	Development Frameworks	375	513.3	1.6	0.8
	Implementation Technologies	320	619.2	1.5	0.7
NLU	Intents & Entities	437	516.3	1.7	**1.0**
	Model Training	347	524.3	1.4	0.7
User Interaction	Chatbot Response	253	409.1	1.2	0.7
	Conversation	278	510.5	1.9	0.6
	User Interface	208	536.8	2.6	0.8
User Input	User Input	307	402.7	1.2	0.6

assigns to each post a series of probabilities (one per topic) that indicate the chances of a post being related to a topic. The topic with the highest probability for a particular post (i.e., the post that contains more keywords of a particular topic) is considered to be the post's dominant topic. We use the Mallet implementation of LDA in our methodology McCallum (2002).

The main challenge of using LDA is to identify the optimal number of topics K, that the LDA uses to group the posts. If the K value is too high, topics may become too specific to draw any relevant analysis. On the other hand, if K value is small, the yielded topics may be too generic, encompassing posts of many different aspects. To overcome this issue, we examine different K values ranging between 5 to 20 in steps of 1 and calculate the coherence metric value of the topics. The coherence metric measures the understandability of the topics resulting from the LDA using different confirmation measures, and has been shown to be highly correlated with human understandability Röder, Both, and Hinneburg (2015). Thus, the first two authors run the LDA with varying K values and then stored the resulting coherence score from each run. We find that K values in the range of 10 to 14 have very similar coherence scores (i.e., the difference is very small). To ensure that we select the best K value, the first two authors examined a randomly selected sample of 30 posts from each topic for K values from 10 to 14. Based on this examination, we find that a K value of 12 (i.e., 12 topics) provides an optimal set of topics that balances the generalizability and the specificity (i.e., most cohere posts) of the resulting chatbot topics.

3.3 Case Study Results

In this section, we present the analysis of the chatbot posts and topics to answer our research questions.

3.3.1 RQ1: What topics are chatbot developers asking about?

Motivation: Chatbot development has some particularities that distinguish it from traditional software development Jha (2019). For example, chatbot developers require specific expertise in natural language processing, machine learning, and conversation design, which are often unnecessary or overlooked in most conventional software development tasks. Hence, the challenges faced by chatbot developers are likely to differ from the challenges of traditional software development. Since developers use Q&A websites to communicate both problems and solutions, the goal of this research question is to dive into the invaluable data of Stack Overflow to identify the most common and pressing chatbot topics and the issues that are more frequently encountered by the chatbot community. Moreover, identifying the widely discussed chatbot topics is the initial step to highlight the topics that are gaining more traction and difficult to answer by the chatbot community.

Approach: We use the LDA as a method to identify the different topics that developers discuss on Stack Overflow as mentioned in Section 3.2. The first three authors (annotators) labelled the set of topics based on the posts overall theme. In particular, each of the annotators individually inspected the top 20 keywords and a random sample of at least 30 posts from each topic in order to label it with a title that best represents the posts of that topic. Then, the authors discuss each of the 12 topics' labels to reach a consensus about the titles of all topics. We observe that some topics that discuss similar aspects of the chatbot development process or are related to the same chatbot component can be further grouped into categories. For example, one topic with keywords related to 'response', 'webhook', and 'card' and another topic that has 'display', 'trigger', and 'prompt' keywords are related to chatbot user interaction. Therefore, we further categorize those topics to have a hierarchical view on the chatbot discussions on Stack Overflow. We also examine the most popular chatbot topics among developers. To investigate that, we use three different complementary measurements of popularity that have been adopted in prior work S. Ahmed and Bagherzadeh

(2018); Bagherzadeh and Khatchadourian (2019); Bajaj, Pattabiraman, and Mesbah (2014); Nadi et al. (2016):

(1) **The average number of views (avg. views)** of the post from both registered and unregistered users. Our intuition here is that if a post is viewed by a large number of developers, then this post is popular among chatbot developers. Overall, this metric measures the interest of the community by telling us how often a post is visualized.

(2) **The average number of posts marked as favourite (avg. favourites)** by Stack Overflow users. This metric measures the issues and solutions that developers deemed to be helpful and having a high chance of recurring during the development of chatbots.

(3) **The average score (avg. scores)** of the posts. Stack Overflow allows it's members to up-vote posts that they consider to be interesting and useful. The votes are then aggregated as a score, which we use as a metric of perceived community value.

Results: Table 3.2 shows the 12 topic titles, which are grouped into 5 main categories. It also shows the number of posts that belong to each topic and the topics' popularity through our popularity metrics: views, favourites, and the scores received by developers on Stack Overflow. As seen from the table, the developers ask about different topics in chatbot development and the number of posts varies across the topics.

The 12 chatbot topics can be mainly grouped into five categories: 'Integration', 'Development', 'NLU', 'User Interaction', and 'User Input'. Next, we discuss those categories in more details.

Integration: This category contains three topics, namely Messenger Integration, NLU Integration/Slots, and API Calls. This category deals with the integration between chatbot platforms, APIs, and websites. About 28.6% of posts in our dataset belong to this category. We also see that the Messenger Integration topic has the highest number of posts in our dataset. In this topic, developers mainly ask about how to create and integrate chatbots to messenger applications. One of the reasons of the widespread of chatbots is the global adoption of messaging platforms (e.g., Slack) C. Lebeuf et al. (2018c). For example, Facebook reported that there are more than 300,000 active chatbots in 2018 that are deployed on its Messenger platform Boiteux (2018). An example of posts

under this topic is a developer asking on Stack Overflow "Facebook Chatbot (PHP webhook) sending multiple replies"Woodman (2018). As chatbots are used to integrate various services C. Lebeuf et al. (2018c), chabot developers are more exposed to the challenges of multi-service and platform integration.

Development: The posts of this category are related to building chatbots using different development frameworks, asking about special configurations and features, and specific implementations using those frameworks. For example, a developer posted on Stack Overflow "How to start a conversation from Nodejs client to Microsoft bot"Hovel (2017). The posts of Development Frameworks, Implementation Technologies, and General Creation/Integration topics form this category. In our study, this category is the second largest, containing 24.3% of the posts in our dataset. This shows that developers tend to heavily rely on chatbot frameworks.

Natural Language Understanding (NLU): This category contains posts related to the definition of intents (the purpose/intention behind the user's input) and entities (important pieces of information in the user's input such as city names), handling and manipulating those intents and entities, customizing and configuring NLUs, and improving the performance of the NLU models. This category comprises 20.2% of the posts in our dataset. It has Intents & Entities and Model Training topics. Those topics are related to the chatbot capability of understanding the users' input and replying accordingly, which has a direct impact on user satisfaction Abdellatif, Badran, and Shihab (2020a). The post "How can I improve the accuracy of chatbot built using Rasa?" Ratan (2017) is an example of posts from this category. Currently, large IT companies are investing to build NLUs (e.g., Microsoft developed LUIS platform Microsoft (2021b)), which is an indicator of their importance and popularity. Moreover, NLU platforms nowadays are considered to be one of the critical components of chatbots Rychalska, Glabska, and Wroblewska (2018a). Leveraging an NLU platform allows developers to focus on the core functionalities of their chatbots rather than having to analyze the user input and manage the conversation with the user.

User Interaction: This category contains posts about conversation design, generating reply messages to users, and designing the chatbot's graphical user interface. For example, developers ask "How to resume or restart paused conversation in RASA?"Shuvro (2019) and "How to add custom choices displayed through Prompt options [...] using C#?"C. N. G (2019). This category

includes User Interface, Chatbot Response, and Conversation topics and forms 19% of the posts in our dataset. We believe that managing the conversation flow with the user is not an easy task since the chatbot users might deviate (i.e., change to other topic) from the designed conversation flow.

User Input: The posts of this category are related to checking/validating and storing the user input, e.g., "How to store and retrieve the chat history of the dialogflow?"Casagrande (2019). There is only one topic that is included in this category and it contains 7.9% of posts in our dataset. Having a single topic as a group indicates that parsing and storing chatbot users' input is a more independent problem among the chatbot topics.

From our results, we observe that the categories cover the end-to-end development of chatbots. The User Interaction category covers the creation of the chatbot interface, while the User Input category covers the manipulation of the users' input received through the User Interaction component. The NLU category includes posts about understating the users' input and optimizing the NLU Model of the chatbot, the Development category covers the back-end development of the core functionalities of the chatbot, and finally, the Integration category covers the integration of all the chatbot components together (User interface, NLU, backend, etc.). This shows that developers are facing various challenges and seeking knowledge about each phase of the chatbot development process. Moreover, the topics within each category reflect specific concerns and issues within that category. For example, in the NLU category, developers are asking questions about defining/handling intents and entities, and improving the performance of the NLU model.

In the second part of our analysis, we investigate the popularity of the chatbot topics. We find that the most popular topics fall into the Development and NLU categories. Table 3.2 shows that the topic General Creation/Integration contains the most viewed and most favourited posts by chatbot developers. This topic contains posts with basic questions about chatbot creation and its high popularity can be explained by the introductory nature of the topic, that is, any newcomer will look for these posts to start developing their first chatbot. Another aspect of this topic's popularity might be due the lack of proper chatbot introductory documentation and support for newcomers. The most viewed and favourited post in our dataset is "Any tutorials for developing chatbots?" with more than 71,565 views and 104 members marking it as a favorite post, evidences the lack of documentation concern. Interestingly, our findings suggest that the chatbot development community should give

special attention to providing a more extensive and accessible documentation on how to develop chatbots from scratch. Intents & Entities is the topic with highest average of post score, the process of handling intents and entities is one of the most specialized aspects of chatbot development, which might explain why developers have a higher (relative) praise for posts from this particular topic.

> *Chatbot developers ask about every aspect and phase of the chatbot development process including Integration, NLU, Development, User Input, and User Interaction. The most popular topics in the chatbot dataset are related to General Creation/Integration.*

3.3.2 RQ2: What types of questions are chatbot developers asking?

Motivation: After understanding the most interesting topics to chatbot developers, we set out to examine the types of posts that they ask in each chatbot category. Prior work Rosen and Shihab (2016) shows that developers ask different types (i.e., how, why, what) of questions to address distinct challenges, hence, this analysis will help us identify the nature of the challenges encountered during chatbot development.

Approach: To achieve that, we follow a similar approach used by prior work to identify the types of the posts on Stack Overflow Rosen and Shihab (2016); Treude, Barzilay, and Storey (2011). In particular, we randomly sample posts from each of the five main chatbot categories with a confidence level of 95% and a confidence interval of 5%. Our random sample size for each category yields a total of 1241 posts: 286 Integration posts, 273 Development posts, 258 NLU posts, 253 User Interaction posts, and 171 User Input posts. Overall, the annotators achieve substantial agreement (kappa=0.62) on the 1241 classified posts. Our level of agreement is higher than the agreement reached in similar studies Rosen and Shihab (2016). For the cases that all annotators failed to agree on, the annotators revisit the questions together and discussed them to reach an agreement. Then, the first three authors individually examine the sample posts' titles and bodies and label each post using one of following types that were used by prior work Rosen and Shihab (2016):

- **How**: Used for posts that ask about a method or technique to implement something Rosen and Shihab (2016). Posts with this type differ from the 'why' posts as in here the developer

Table 3.3: Chatbot posts types on Stack Overflow.

Main Categories	% How	% Why	% What	% Other
Integration	66.4	22.7	10.8	0.0
Development	57.9	23.4	18.3	0.4
NLU	54.3	29.5	15.9	0.4
User Interaction	66.8	22.5	10.3	0.4
User Input	68.4	14.6	14.6	2.3
Chatbots (all)	61.8	25.4	11.7	1.2

has a particular goal in mind, and asks for the steps to achieve this goal (e.g., "how to get user name in Microsoft bot framework in C# using V4?").

- **Why**: Posts where the developer asks about the reason, cause, or purpose of something Rosen and Shihab (2016). Posts of 'why' type are often related to troubleshooting where the developer expects an explanation of a particular (and unexpected) behavior (e.g., "why is Wordpress blocking the js livechat window?").

- **What**: Posts where the developer is asking for a particular information Rosen and Shihab (2016). Often, the user wants to clarify a doubt that is useful to make more informed decisions (e.g., "what are "implicit triggers" in a Google Action package?").

- **Other**: We assign this type to posts that do not fall under any of the above types (e.g., "chatbot conversation objects, your approach?").

To measure the quality of our classification of the random sample, we use Cohen's Kappa McHugh (2012) to measure the level of inter-agreement among the annotators.

Results: Table 3.3 shows the percentage of the posts types for each chatbot category. We see that more than half of the posts (61.8%) are of 'how' type, followed by 'why' (25.4%) and 'what' (11.7%). This shows that the developers are looking for more working examples, debugging, and information. The User Interaction category has the most 'how' posts (66.8%), showing a need for more sources of guidance to design and manage the conversation flow between the user and chatbot. The NLU category has the most 'why' posts (29.5%), suggesting the need for discussion forums and better documentation on how the NLU models work, especially given that most NLUs are closed source. The Development category has the most 'what' posts (18.3%), suggesting that providing

Table 3.4: The difficulty per topic.

Topic	Posts w/o Accepted (%)	Median Time (h)
General Creation/Integration	72.0	8.2
Intents & Entities	71.4	19.5
User Interface	70.7	7.0
Model Training	70.2	22.4
Messenger Integration	70.0	22.6
User Input	66.8	9.3
NLU Integration/Slots	66.5	12.8
Conversation	65.5	6.9
Chatbot Response	65.2	11.3
Implementation Technologies	64.7	15.5
API Calls	63.7	16.2
Development Frameworks	63.7	15.6

general information about the supported features of the chatbot frameworks is appreciated by the community.

> *Chatbot developers mainly (61.8%) look for implementation guidance by posting how posts, followed by why (25.4%) and what (11.7%). Developers are concerned about the how aspect of the User Interaction category, whereas most the highest share of why posts are from the NLU category, and what posts from the Development category.*

3.3.3 RQ3: Which topics are the most difficult to answer?

Motivation: Given that we know the popular topics and their types of posts. Now, we want to investigate the difficulty of answering posts in each topic. Finding whether some topics are harder to answer than others will help us identify the topics that need more attention from the community. Also, it allows us to highlight the topics where there is a need for better tools/frameworks to support developers at addressing chatbot development challenges.

Approach: We measure the difficulty of each topic by applying two metrics that have been used in prior work Bagherzadeh and Khatchadourian (2019); Rosen and Shihab (2016); Yang et al. (2016):

(1) **The percentage of posts of a topic without accepted answers (% w/o accepted answers)**.

Table 3.5: Correlation of topics popularity and difficulty.

Correlation Coeff. / p-value	Avg. Views	Avg. Score	Avg. Favourite
% w/o Accepted Answers	0.524/0.084	0.147/0.651	0.419/0.176
Median Time to Answer (Hrs.)	0.105/0.749	0.223/0.485	−0.335/0.287

For each chatbot topic, we measure the percentage of posts that have no accepted answers. While many answers can be issued in a post, the post's author has the sole authority to mark an answer as accepted if it satisfies and solves the original post's question. Therefore, topics with less accepted answers are considered more difficult Bagherzadeh and Khatchadourian (2019); Rosen and Shihab (2016).

(2) **The median time in hours for an answer to be accepted (Median Time to Answer (Hrs.))**. We measure the median time in hours for posts to receive an accepted answer. This metric considers the creation time of the accepted answer and not the time at which the answer is marked as accepted. The longer it takes for a post to be properly answered (receive an accepted answer), the harder the post isBagherzadeh and Khatchadourian (2019); Rosen and Shihab (2016).

Our dataset includes some posts that did not have sufficient time to receive an answer. In our dataset of chatbot-related posts, questions take a median of 14.8 hours to be answered, hence, we remove from this analysis posts that were created less than 14.8 hours before the data collection date (September 4, 2019).

Results: Table 3.4 shows the percentage of accepted answers and median time (in hours) to receive an accepted answer for each of the identified topics in Section 3.3.1. The topics in Table 3.4 are ordered based on the percentage of accepted answers they received. The most popular topic General Creation/Integration is also the one with the largest share of posts without accepted answers. The posts in this topic, however, take a median time of only 8.2 hours to receive an accepted answer, which is the third fastest median time in our topics. To understand the reason behind the high percentage of posts with no accepted answers (72%), we examine the posts of this topic. We find that the posts without an accepted answer are given low scores (on average 0.17) from developers on Stack Overflow. This might be due to unclear or ill-formed questions, which effectively reduces

the chances of getting an accepted answer.

If we analyze the median time to answer a topic, we see a higher variation among the topics. Messenger Integration, Intents & Entities, and Model Training are the most difficult topics based on their time to receive accepted answers. Interestingly, Intents & Entities, and Model Training are related to the NLU category which discusses how to load and train NLU models, and identify and handle intents and entities. The results show that the topics related to the NLU are harder to answer by the Stack Overflow community. This may be due to the black box implementation of most popular NLUs, which prevents chatbot developers from fully understanding and solving NLU related issues.

On the other hand, posts that are related to Development Frameworks have the highest percentage of accepted answers and a median time to answer in-line with the overall chatbot topics (15.6 hours). This topic includes posts on how to implement chatbot routines using a certain technology (e.g., "How to send location from Facebook messenger platform?") or comparing of different platforms (e.g., "Comparison between Luis.ai vs Api.ai vs Wit.ai?"). These are also tasks that are more closely related to traditional software development, which could explain why the Stack Overflow respondents tend to answer this topic faster and more frequently.

To have a full view of the chatbot-related posts, we want to examine if there is a statistically significant correlation between the difficulty and popularity. In particular, we use the Spearman Rank Correlation Coefficient Spearman (2008) to verify the correlations between the three popularity metrics (avg. views, avg. favourites, and avg. scores) and the two difficulty metrics (% w/o accepted answers and median time to answer). We choose Spearman's rank correlation since it does not have any assumption on the normality of the data distribution. As shown in Table 3.5, we do not find any statistically significant correlation between the popularity and difficulty metrics since all correlations have $p - value > 0.05$. In other words, the difficult topics are not necessary popular among developers, and vice versa.

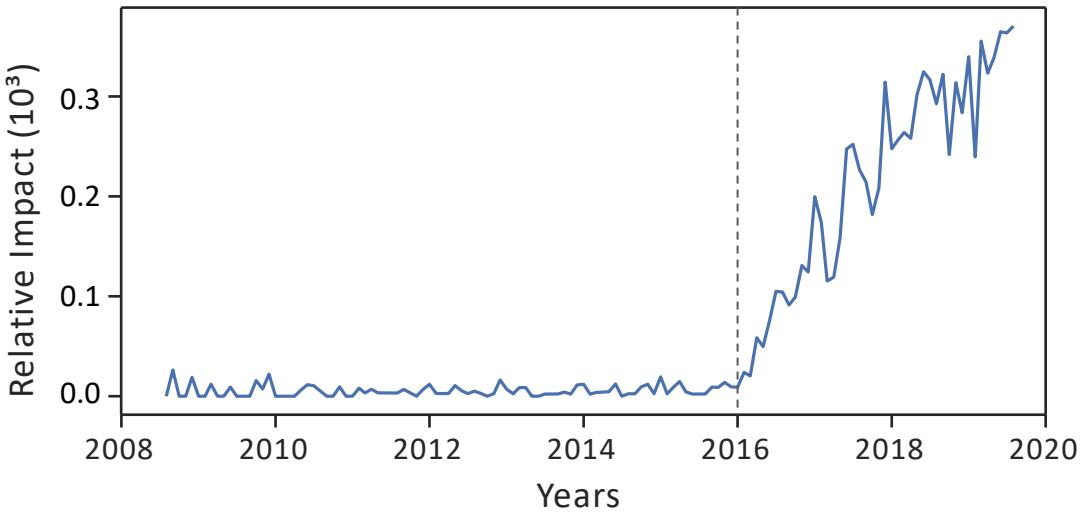


Figure 3.2: Relative growth of chatbot related posts over time.

Topics related to training chatbot models are the most difficult in chatbot development. While the most popular topic, General Creation/Integration, contains the largest share of unanswered posts. On the other hand, posts related to the Development Frameworks topic tend to be answered more frequently.

3.4 Discussion & Implications

In this section, we discuss the chatbot topics evolution and compare our findings with the findings in prior work. Then, we delve into the data to identify the prevalent topics on different platforms and discuss the implications of our results.

3.4.1 Chatbot Topics Evolution

Chatbots are an emerging topic that is getting more attention from developers in different domains (e.g. security ”Dutta, Joyce, and Brewer (”2018”), software engineering Toxtli et al. (2018)). To examine the evolution of a topic, we utilize two measures; the absolute growth, which measures the change in the total number of posts over time; and the relative growth, which represents the relative change in the total number of posts for a specific topic compared to the change in the total

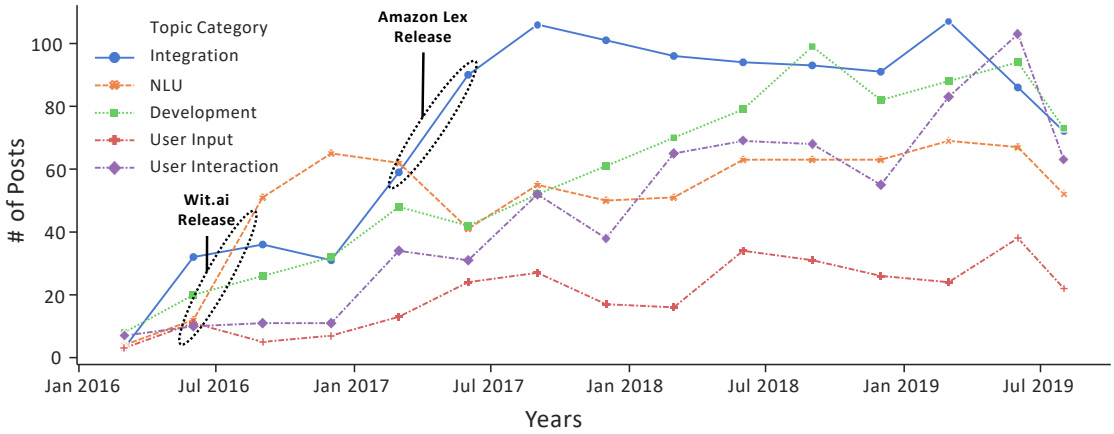


Figure 3.3: Chatbot categories evolution over time.

number of posts for the entire Stack Overflow dataset. To highlight the evolution of the chatbot topics, we examine the relative growth of all chatbot topics compared to Stack Overflow over time, from August 2008 to September 2019. Figure 3.2 shows the evolution of the chatbot in terms of relative growth compared to Stack Overflow. As seen from the Figure, the relative growth of the chatbot topics has an increasing trend that started in 2016. This increase in the last few years shows that chatbots are gaining more attention from the community over time.

To better understand the evolution of the different chatbot development activities, we measure the absolute growth of each of the five categories over time. We find that all of our categories are growing positively over time as shown in Figure 3.3. This means that the number of posts for every category is increasing overtime, which in turn indicates the increasing trend of the various chatbot development activities represented by the different categories.

We further investigate the reasons behind the sudden increases (i.e., hikes) in the number of posts during specific periods of time and find two interesting cases as shown in Figure 3.3. The first case is related to the Integration category which has the highest spike (46 posts) on June 2017. We find that most of the discussions during this spike are related to the integration of the Amazon Lex platform Amazon (2019) that was released in April 2017 AWS (2019). The second sudden increase can be observed in the NLU category during November 2016. Posts of that spike are asking about the intents and entities in the Wit.ai platform Facebook (2019), which was released in April 2016 TechCrunch (2017).

Table 3.6: Comparison of popularity and difficulty between different fields

Metrics	Chatbot	Mobile	Security	Big Data
# of Posts	3,890	1,604,483	94,541	125,671
Avg. ViewCount	512.4	2,300	2,461.1	1,560.4
Avg. FavoriteCount	1.6	2.8	3.8	1.9
Avg. Score	0.7	2.1	2.7	1.4
Avg. AnswerCount	1.0	1.5	1.6	1.1
% w/o Answers	67.7	52	48.2	60.3
Med. TimeToAnswer (Hrs.)	14.8	0.7	0.9	3.3

Although we show the results of the chatbot categories' evolution over time, we share the evolution results of each of the topics in a publicly available online dataset A. A. D. C. K. B. R. A. E. Shihab (2020). In general, we can see a trend of chatbot development activities gaining traction among developers. Our findings also show that the chatbot community tends to pick up the new platforms as shown in the cases of Amazon Lex and Wit.ai.

3.4.2 Chatbot Compared to Other SE Fields

In the previous sections, we find that chatbot discussions only started to become more active in 2016. As a new and emerging field, we set out to investigate how the topics of chatbot compares against discussions of more consolidated Software Engineering (SE) fields such as mobile, big data and security (topics that were similarly studied in the past). To answer this question, we examine the difficulty and popularity of the chatbot topics and compare it against other disciplines, by including data from similar studies on Stack Overflow, focused on the topics of mobile apps Rosen and Shihab (2016), security Yang et al. (2016), and big data Bagherzadeh and Khatchadourian (2019). Those studies were conducted in a different time frame, therefore, we use their reported keywords to construct an updated dataset and calculate the popularity and difficulty metrics for each of those fields.

Table 3.6 shows the results of the popularity and difficulty metrics among the four fields. From the sheer number of posts, the chatbot topic is, by a few orders of magnitude, smaller than mobile, security and big data. Second, the chatbot posts are consideranly more difficult compared to the other fields, which is also a consequence of having a small and niche crowd. There is a big gap in the time to receive an accepted answer for the chatbot-related posts compared to other topics. Most mobile and security posts are answered in less than an hour, while most chatbot posts take at least

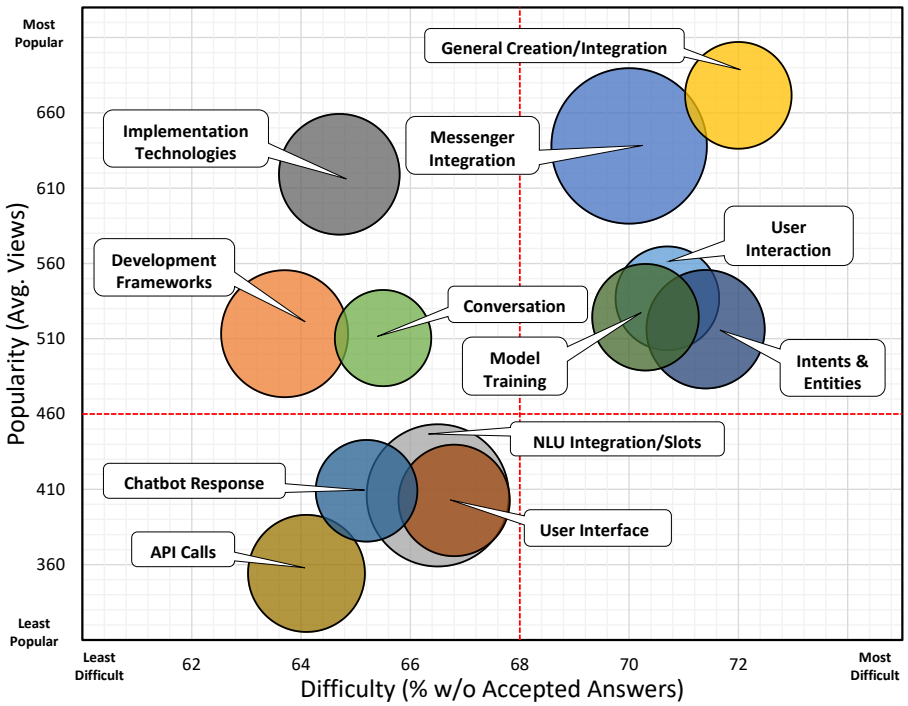


Figure 3.4: Chatbot topics' popularity vs. difficulty

14 hours. This corroborates with the emerging nature of the chatbot topic and indicates that much needs to be done to put the chatbot development community on pair with other mature fields such as mobile and security.

3.4.3 Implications

The results of our study can help chatbot community at better focusing their efforts on the most pressing issues in chatbot development. In the following, we describe how our results can be used to better guide practitioners, researchers and educators at improving the practice and learning of chatbots development.

To help identify the most pressing issues, we present in Figure 3.4 a bubble plot that positions the topics in terms of their popularity and difficulty. The size of the bubble represents the number of posts for a particular topic and we visually split the figure into four quadrants to show the relative importance and difficulty of the topics. We use the average number of views as a proxy for popularity and the percentage of posts without accepted answers as a proxy for difficulty.

Implication for Practitioners. As shown in Figure 3.4, albeit being the most popular topic, beginner questions on how to build Chatbots (General Creation/Integration) remain largely unanswered.

The development community should use this finding to devise better tutorials and documentation aiming at reducing the entry-barrier for developing chatbots.

Our findings can help chatbot developers better prioritize their work by taking into account the areas of the most difficult topics in chatbot development. Topics related to NLU, such as Model Training and Intents & Entities, are among the topics with the highest share of posts without accepted answers. Software managers can take that into account by assigning more resources (development time) to tasks that involve training NLU models, especially given that NLU has the highest share of troubleshooting posts (Section 3.3.2), indicating that developers experience issues more frequently with this kind of tasks.

The evidence of the difficulty of NLU related topics can be used to motivate better and more intuitive NLU frameworks. Practitioners can improve the current documentation of the NLU frameworks and companies that develop and publish NLU platforms should focus on improving the expressiveness of their current framework APIs. For instance, some platforms (e.g., Google DialogFlow Google (2020a) and Microsoft LUIS Microsoft (2021b)) offer graphical interface for training the NLU model, in an attempt to extend the model training to users less familiar to software programming G. Dialogflow (2020); Microsoft (2020c).

Figure 3.4 also shows that Messenger Integration is the largest topic in our dataset. In fact, Integration is the category with the highest number of posts in Stack Overflow. Chatbots are expected to communicate between multiple services and integrate with messengers to make use of already existing Social Networks platforms (e.g., Facebook). Practitioners should invest more resources into facilitating integration of their platforms and tools with other services. For instance, Dialogflow offers developers a one-click integration feature to some of the most popular chatting platforms, such as Slack, Twitter and Skype Google (2020b). As chatbot developers find integration a pressing issue, providing straightforward approaches to integration would allow developers to focus on the core chatbot functionalities, reducing the time and effort overhead of developing multi-service chabots.

Implication for Researchers. Our findings confirm that chatbot developers discuss topics such as Conversation, NLU Integration/Slots, and Chatbot Response, that differ chatbot development from traditional software development. As shown in Figure 3.4, NLU related topics are notoriously

difficult and research can be put into some of the problems faced by chatbot developers at training their NLU models. One such problem is the acquisition of a high-quality dataset, frequently asked by developers in Stack Overflow Overflow (2017, 2019a). A high-quality dataset that represents well the intents and entities supported by the chatbot is paramount for the chatbot performance. New comprehensive datasets and approaches that focus on generating labelled data can help alleviate this challenge faced by developers. Another problem is related to methods for extracting intents and entities, which has received some attention by the research community Gao, Chen, Zhang, He, and Lin (2019); B. Xu et al. (2017a); Ye et al. (2016); Zamanirad, Benatallah, Chai Barukh, Casati, and Rodriguez (2017); Q. Zhang, Fu, Liu, and Huang (2018), but remains a challenging problem in chatbot development.

Implication for Educators. Educators can use our topics and categories as a roadmap to design their chatbot-related courses. The category development also has a high number of discussions looking for the most appropriate framework and best practices ('what' posts), hence, educators can introduce their audience to the several existing chatbot development frameworks and discuss best practices and standards to be followed during the chatbot development phase. As mentioned before, special attention should be given to the NLU topics, which has shown to be difficult (Figure 3.4). In particular, since NLU has the highest share of 'why' posts, this indicates that chatbot developers are in need of theoretical explanations of NLU machine-learning algorithms and models.

There are many aspects that practitioners, researchers, and educators can take into consideration when deciding where to focus their efforts. Nevertheless, we believe that our findings and implications can help improve this decision-making process.

3.5 Threats to validity

Internal Validity: Internal validity concerns factors that could have influenced our results. We use tags from Stack Overflow to identify chatbot-related posts and it might be the case that some chatbot-related posts are mislabelled (i.e., missing tags or having incorrect tags) and therefore are omitted from our dataset. We mitigate this threat by examining all tags that coexist with the 'chatbot' tag and selecting a set of tags that are related to chatbots using the TST and TRT measures. Those

measures have been used in prior work to have a better coverage of a certain topic's posts and limit the noise in the dataset Bagherzadeh and Khatchadourian (2019); Rosen and Shihab (2016); Wan et al. (2019); Yang et al. (2016). Moreover, we find that the TST and TRT thresholds that we obtain in our study are in-line with previous studies S. Ahmed and Bagherzadeh (2018); Bagherzadeh and Khatchadourian (2019); Yang et al. (2016).

One potential threat is that we select $K = 12$ as the optimal number of topics for the LDA topic modelling technique. The number of topics (K) has a direct influence on the quality of the resulting topics from the LDA, and selecting an optimal number is known to be difficult. To alleviate this threat, we follow the approach used in similar studies to select the number of topics Han et al. (2019); Wan et al. (2019). Specifically, we experiment with different values of K and we examine the coherence of topics to select the optimal K value that balances the generalizability and relevance of the chatbot topics.

The labelling of posts types is another threat to the validity of our results, due to the subjectivity of the process. We mitigate this threat by performing three independent classifications and evaluating the interrater-agreement using the Cohen-Kappa test, that indicated substantial agreement among the annotators .

Construct Validity: Construct validity considers the relationship between theory and observation, in case the measured variables do not measure the actual factors. Labelling the resulting topics from the LDA might not reflect the posts associated with the topics. To minimize this threat, the first three authors individually examine the keywords and more than 30 posts randomly from each topic, then they discuss each topic's label to reach a consensus on the label that reflects the posts of that topic. We use different metrics to measure the popularity and difficulty of the chatbot topics which might be a threat to construct validity. These metrics have been used in similar studies S. Ahmed and Bagherzadeh (2018); Bagherzadeh and Khatchadourian (2019); Bajaj et al. (2014); Nadi et al. (2016); Rosen and Shihab (2016); Yang et al. (2016).

External Validity: Threats to external validity concern the generalization of our findings. Our study was focused on and collected data from posts on Stack Overflow, however, there are other forums that may host developers' discussions regarding chatbots. We believe that using Stack Overflow allows for the generalizability of our results as Stack Overflow is a very popular platform that

hosts a large number of questions and answers from developers with a wide variety of domains and expertise. We also believe that this study can be improved by including discussions from different forums or surveying actual software developers about issues that they face when building chatbots.

The focus of this study is on chatbot which is considered to be a sub-category of software bots C. Lebeuf et al. (2019a); C. R. Lebeuf (2018). Therefore, our observations and results cannot be generalized to other types of bots, such as agents. However, we believe that our observations are still relevant and contribute to the larger community (software bot). We encourage other researchers to conduct similar studies on other types of bots and compare the results from the different types to paint a full picture about bots in general.

3.6 Chapter Summary

In this chapter, we analyze Stack Overflow posts to identify the most pressing issues facing chatbot development. We find that developers discuss 12 chatbot-related topics that fall under five main categories, namely Integration, Development, NLU, User Interaction, and User Input. Chatbot developers are highly interested in posts that are related to chatbot creation and integration into websites. On the other hand, training the NLU model of the chatbot proves to be challenging task for developers. We also find that chatbot practitioners show considerable interest in understanding the behavior of NLUs, while also seeking good recommendation regarding chatbot development platforms and best practices. We believe that our results are useful to the chatbot community as they guide future research to focus on the more pressing and difficult aspects of chatbot development. Moreover, our findings help platform owners to understand the issues faced by chatbot developers when using their platforms, and to overcome those challenges. Chatbot educators can take into consideration the discussed topics and categories and their perspective difficulty to better design their courses.

Our study opens the door for chatbot researches and practitioners to further understand the chatbot development challenges. Nevertheless, we plan in the future to examine developers' discussion from other forums to draw more accurate and generalizable conclusions. We also plan to investigate the developers discussions regarding bots in general, which would allow us to compare our results

with with other bot types. Finally, we intend to investigate chatbot repositories and analyze the commits and bug reports to obtain further insights regarding the various issues faced by chatbot developers and their attempts to solve it.

In the following chapter, we validate our results from Section 3.4.3 and endorse the chatbot potential to revolutionize SE. Specifically, we develop a chatbot framework that extracts data from software repositories to assist developers in their tasks. Mining data from software repositories to answer development/maintenance questions is a tedious and difficult task. We implement a chatbot on top of software repositories to answer those questions facing developers. Then, we perform a user study to evaluate the effectiveness and efficiency of the proposed chatbot.

Chapter 4

Determining the value of SE-context chatbots

Software repositories contain a plethora of useful information that can be used to enhance software projects. Prior work has leveraged repository data to improve many aspects of the software development process, such as, help extract requirement decisions, identify potentially defective code and improve maintenance and evolution. However, in many cases, project stakeholders are not able to fully benefit from their software repositories due to the fact that they need special expertise to mine their repositories. Also, extracting and linking data from different types of repositories (e.g., source code control and bug repositories) requires dedicated effort and time, even if the stakeholder has the expertise to perform such a task. Therefore, we use bots to automate and ease the process of extracting useful information from software repositories. Particularly, we lay out an approach of how bots, layered on top of software repositories, can be used to answer some of the most common software development/maintenance questions facing developers. We perform a preliminary study with 12 participants to validate the effectiveness of the bot. Our findings indicate that using bots achieves very promising results compared to not using the bot (baseline). Most of the participants (90.0%) find the bot to be either useful or very useful. Also, they completed 90.8% of the tasks correctly using the bot with a median time of 40 seconds per task. On the other hand, without the bot, the participants completed 25.2% of the tasks with a median time of 240 seconds per task. Our work

has the potential to transform the MSR field by significantly lowering the barrier to entry, making the extraction of useful information from software repositories as easy as chatting with a bot. The result of this research question appears in Journal of Empirical Software Engineering Abdellatif, Badran, and Shihab (2020a).

4.1 Introduction

Software repositories contain an enormous amount of software development data. This repository data is very beneficial, and has been mined to help extracts requirements (e.g., Ali, Guhneuc, and Antoniol (2013); Mordinyi and Biffl (2017)), guides process improvements (e.g., Gupta, Sureka, and Padmanabhuni (2014); Siddiqui and Ahmad (2018)) and improves quality (e.g., Hassan (2008); Khomh et al. (2015)). However, we argue that even with all of its success, the full potential of software repositories remains largely untapped. For example, recent studies presented some of the most frequent and urgent questions (e.g., "Where do developers make the most mistakes?" and "Which mistakes are the most common?") that software teams struggle to answer Begel and Zimmermann (2014a). Many of the answers to such questions can be easily mined from repository data.

Although software repositories contain a plethora of data, extracting useful information from these repositories remains to be a tedious and difficult task Banerjee and Cukic (2015); Bankier and Gleason (2014). Software practitioners (including developers, project managers, QA analysts, etc.) and companies need to invest significant time and resources, both in terms of personnel and infrastructure, to make use of their repository data. Even getting answers to simple questions may require significant effort.

More recently, bots were proposed as means to help automate redundant development tasks and lower the barrier to entry for information extraction Storey and Zagalsky (2016a). Hence, recent work laid out a vision for how bots can be used to help in testing, coding, documenting, and releasing software Beschastnikh et al. (2017); Tian, Thung, Sharma, and Lo (2017b); Wessel et al. (2018b); B. Xu, Xing, Xia, and Lo (2017b). To bridge the gap between the visionary works and practicality, and to bring those visionary works to life, we devise a framework of using bots over software repositories. Although different bots have been developed for the software engineering

domain, no prior work has applied bots to answer the developers questions using the stored data from software repositories.

Although it might seem like applying bots on software repositories is the same as using them to answer questions based on Stack Overflow posts, the reality is there is a big difference between the two. One fundamental difference is the fact that bots that are trained on Stack Overflow data can provide general answers, and will never be able to answer project-specific questions such as "how many bugs were opened against my project today?". Also, we would like to better understand how bots can be applied on software repository data and highlight what is and what is not achievable using bots on top of software repositories.

Therefore, our goal is to design and build a bot framework for software repositories and perform a case study to examine its efficiency and highlight the challenges facing our framework. The approach contains five main components, a **user interaction** component, meant to interact with the user; **entity recognizer** and **intent extractor** components, meant to process and analyze the user's natural language input; a **knowledge base** component, that contains all of the data and information to be queried; and a **response generator** component, meant to generate a reply message that contains the query's answer and return it to the user interaction component. To evaluate our bot approach, we add support for 15 of the most commonly asked questions by software practitioners mentioned in prior work Begel, Khoo, and Zimmermann (2010); Begel and Zimmermann (2014a); Fritz and Murphy (2010); Sharma, Mehra, and Kaulgud (2017); Sillito, Murphy, and Volder (2008). To evaluate our framework, we perform a case study with 12 participants using the Hibernate and Kafka projects. In particular, we asked those participants to perform a set of tasks using the bot then evaluate it based on its replies. We examine the bot in terms of its effectiveness, efficiency, and accuracy and compare it to a baseline where the survey participants are asked to do the same tasks without using the bot. We also perform a post-survey interview with a subset of the survey participants to better understand the strengths and areas of improvements of the bot approach.

Our results indicate that bots are useful (as indicated by 90.0% of answers), efficient (as indicated by 84.17% of answers) and accurate (as indicated by 90.8% of tasks) in providing answers to some of the most common questions. In comparison to the baseline, the bots significantly outperform the manual process of finding answers for their questions (the survey participants were able to

only answer 25.2% of the questions correctly and took much longer to find their answers). Based on our post-survey interviews with the participants, we find that bots can be improved if they enable users to perform deep-dive analysis and help compensate for user errors, e.g., typos. Based on our results, we believe that applying bots on software repositories has the potential to transform the MSR field by significantly lowering the barrier to entry, making the extraction of useful information from software repositories as easy as chatting with a bot.

In addition to our findings, the chapter provides the following contributions:

- To the best of our knowledge, this is the first study to use bots on software repositories. Also, our framework allows project stakeholders to extract repository information easily using natural language.
- We perform an empirical study to evaluate our bot framework and compare it to a baseline. Also, we provide insights on areas where bot technology/frameworks still face challenges in being applied to software repositories.
- We make our framework implementation Abdellatif, Badran, and Shihab (2019a) and datasets Abdellatif, Badran, and Shihab (2019c) publicly available in an effort to accelerate future research in the area.

4.1.1 Organization of the Chapter

The rest of the chapter is organized as follows. We detail our framework and its components in Section 4.2. We explain the questions supported by the bot and questionnaire survey to evaluate our framework in Section 4.3. In Section 4.4, we report our findings, detailing the usefulness, speed, and accuracy of the bot. Section 4.5 discusses the bot evaluation and the implications of our results. Section 4.6 discusses the threats to validity, and Section 4.7 concludes the chapter.

4.2 MSRBot Framework

Our goal is to build a bot that users can interact with to ask questions (such as questions presented in Table 4.1) to their software repositories. To enable this, our framework is divided into

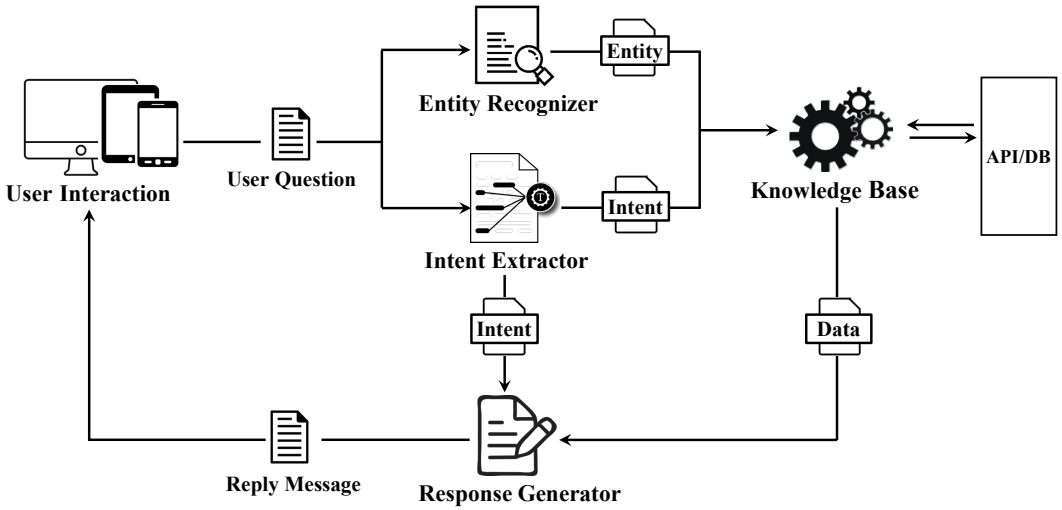


Figure 4.1: Overview of the MSRBot Framework and its Components' Interactions

five main parts (as shown in Figure 4.1), namely 1) user interaction 2) entity recognizer 3) intent extractor 4) knowledge base and 5) response generator. In the following subsections, we detail each of these parts and showcase a working example of our framework.

User Interaction

Users of bot frameworks need to be able to effectively interact with their information. This can be done in different ways, e.g., through natural language text, through voice and/or visualizations. In addition to handling user input, the user interaction component also presents the output of the question to the user. This is done in the same window and appears as a reply to the user's question. Users are expected to pose their questions in their own words, which can be complicated to handle, especially since different people can pose the same question in many different ways. To help us handle this diversity in the natural language questions, we devise entity recognizer and intent extractor components, which extract structured information from unstructured language input (question) posted by the user. We detail those components in the next subsections.

Entity Recognizer

The entity recognizer component identifies and extracts a useful information (entity) that a user mentioned in the question using Named Entity Recognition (NER) Tjong Kim Sang and De Meulder

(2003). Also, it categorizes the extracted entities under certain types (e.g. city name, date, and time). There are two main NER categories: Rule-Based and Statistical NER. Prior work showed that statistical NER is more robust than the rule-based in extracting entities Liu, Zhang, Wei, and Zhou (2011); Mohit (2014); Ratinov and Roth (2009). In the rule-based NER, the user should come up with different rules to extract the entities while in the statistical NER the user trains a machine learning model on an annotated data with the named entities and their types in order to allow the model to extract and classify the entities. The extracted entities help the knowledge base component in answering the user's question. For example, in the question: "Who modified Utilities.java?", the entity is "Utilities.java" which is of type "File Name". Having the file name is necessary to know which file the user is asking about in order to answer the question correctly (i.e. bring the information of the specified file). However, knowing the file name (entity) is not enough to answer the user's question. Therefore, we also need an intent extractor component, which extracts the user's intention from the posed question. We detail this component in the next subsection.

Intent Extractor

The intent extractor component extracts the user's purpose/motivation (intent) of the question. In the last example, "Who modified Utilities.java?", the intent is to know the commits authors that modified the Utilities file. One of the approaches (e.g., Zamanirad et al. (2017)) to extract the intents is to use Word Embeddings, more precisely the Word2Vec model Mikolov, Sutskever, Chen, Corrado, and Dean (2013). The model takes a text corpus as input and outputs a vector space where each word in the corpus is represented by a vector. In this approach, the developer needs to train the model with a set of sentences for each intent (training set). Where those sentences express the different ways that the user could ask about the same intent (same semantic). After that, each sentence in the training set is represented as a vector using the following equation:

$$\vec{Q} = \sum_{j=1}^{n} \vec{Q}_{wj} \quad \text{Where } \vec{Q}_{wj} \, \epsilon \, VS \tag{1}$$

where $\vec{Q}$ and $\vec{Q}_{wj}$ represent the word vector of a sentence and vector of each word in that sentence in the vector space VS, respectively. Afterwards, the cosine similarity metric Jurafsky

and Martin (2009) is used to find the semantic similarity between the user's question vector (after representing it as a vector using Equation 1) and each sentence's vector in the training set. The intent of the user's question will be the same as the intent of the sentence in the training set that has the highest score of similarity. The extracted intent is forwarded to the response generator component in order to generate a response based on the identified intent. Also, the intent is forwarded to knowledge base component in order to answer the question based on its intent. We explain this component in the next subsection.

If the intent extractor is unable to identify the intent (low cosine similarity with the training set), it notifies the knowledge base and the response generator components, which respond with some default reply.

Knowledge Base

The knowledge base component is responsible for answering the user's questions (e.g. making an API call or querying a DB). It uses the passed intent from the previous component to map it with an API call or DB query that needs to be executed in order to get the answer of the question. And, it uses the extracted entities from the entity recognizer component as parameters for the query or call. For example, if a user asks the question "Which commits fixed the bug ticket HHH-11965?", then the intent is to get the fixing commits and the issue key "HHH-11965" is the entity. So, the knowledge base component uses the identified intent to retrieve the mapped query to the extracted intent and the entity as a parameter to that query. Therefore, the knowledge base component queries the database on the fixing commits (using SZZ Śliwerski, Zimmermann, and Zeller (2005)) that are linked to Jira ticket "HHH-11965". The component forwards the query's results to the response generator component to generate a reply message on the user's question. In case the intent extractor was unable to identify the intent, the knowledge base will do nothing and wait for a new intent and entities. Furthermore, the knowledge base component verifies the presence of the entities associated with the extracted intent and notifies the response generator in case of missing entities or unable to retrieve the data from the API. We describe the response generator component in the next subsection.

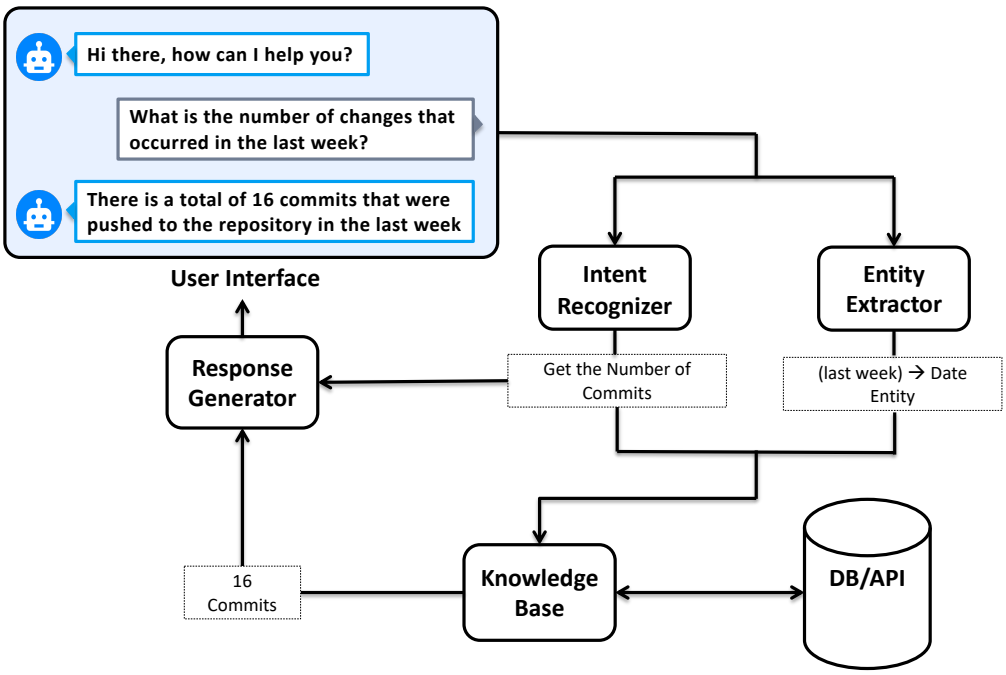


Figure 4.2: Example of MSRBot Framework Components Interactions to Answer User's Question

Response Generator

The response generator component generates a reply message that contains the answer to the user's question and sends it to the user interaction component to be viewed by the user. The response is generated based on the question asked, and more specifically, the extracted intent of the question. Finally, it sends the generated message to the user interaction component.

In some cases, the bot may not be able to respond to a question due to lack of information (i.e., intents and entities). For example, if it is not possible to extract the intent, the response generator returns a default response:"Sorry, I did not understand your question, could you please ask a different question?". And, in case of a missing entity, the response is "Could you please specify the *entity*?" and it mentions the entity in the message (e.g., file name).

It is worth mentioning that there are certain components that need to be customized to suit our approach. We customize the entity recognizer and intent extractor components to make our bot

applicable on software repositories data. For example, we need to train the entity recognizer on the software engineering entities that are specific to software repositories (e.g., Jira tickets and commit hashes) to be able to identify those entities in the posed query. Also, our knowledge base component is specific to software repositories in a number of ways. First, it extracts and links the data from the source code and issue tracking repositories. Second, the knowledge base is customized to retrieve the answer to the user's queries based on the defined intents and entities. On the other hand, the remaining components are applicable to any type of software bots (e.g., user interface).

Explanatory Example

Putting everything together, we showcase an end-to-end example of the interactions between the components of our proposed framework to answer the user's question. In Figure 2, the user interacts with the bot through the user interface component to inquire about the number of changes that happened in the last week. Next, the user interface component forwards the question to the entity recognizer and intent extractor components to parse the user's query. The entity recognizer component identifies the entity 'last week' (i.e., 21/01/2019 27/01/2019) of type 'Date' in the query. On the other hand, the intent extractor component classifies the intent of the posed query as "Get the Number of Commits". Then, the results of the entity recognizer and intent extractor are sent to the knowledge base component to be processed. Also, the extracted intent is forwarded to the response generator component to generate the reply message. The knowledge base component uses the forwarded intent to retrieve the SQL query that is mapped to it while the entity is used as the parameter for the query. Next, the knowledge base executes the SQL query and retrieves the data from the database. Then, the knowledge base sends the query execution result (i.e., 16 commits) to the response generator. Finally, the response generator uses the results from the knowledge base and the forwarded intent from the intent extractor to generate the reply message ("There is a total of 16 commits that were pushed to the repository in the last week"), and sends it back to the user interface component to present the answer of the question to the user.

Table 4.1: List of Questions Examined by MSRBot and the Rationale for Their Inclusion

Question	Rationale	Ref.
Q1. Which commits fixed the **bug id**?	To refer the commit to a developer who is working on similar bug.	Begel and Zimmermann (2014a)
Q2. Which developer(s) fixes the most bugs related to **File Name**?	To know which developer(s) have experience in fixing bugs related to a specific component or file.	Begel and Zimmermann (2014a)
Q3. Which are the most bug introducing files?	To refactor the buggy files in the repository.	Begel and Zimmermann (2014a)
Q4. Who modified **File Name**?	To know which developer(s) worked on the file in order to ask them about a piece of code.	Fritz and Murphy (2010); Sharma et al. (2017)
Q5. Which are the bugs introduced by commit **Commit Hash**?	To study the type of bugs introduced because of certain commit.	Begel and Zimmermann (2014a)
Q6. What is the number of commits in/on **Date**?	To track the progress of the team members at particular time (e.g., in the last day, week, month).	Fritz and Murphy (2010)
Q7. What commits were submitted on **Date**?	To know exactly what commits were done, to flag for review, testing, integration, etc.	Fritz and Murphy (2010)
Q8. What is/are the latest commit(s) to **File Name**?	Developers may want to know what are the last things that changed in a file/class to be up to date before they make modifications.	Begel and Zimmermann (2014b); Fritz and Murphy (2010); Sharma et al. (2017)
Q9. What are the commits related to **File Name**?	To track changes which are happening on the file which the developer is currently working on. Or to know the changes happening to a file that was abandoned by a developer.	Begel et al. (2010)
Q10. What is/are the most common bug(s)?	The team wants the most important/common bug (registered as watchers) so it can be addressed.	Begel and Zimmermann (2014a)
Q11. What are the buggy/fixing commits that happened in/on **Date**?	To know the buggy or fixing commits happened at a particular time e.g. before release date.	Begel and Zimmermann (2014a)
Q12. How many bugs have the status/priority?	Quickly view the number of bug reports with a specific status (e.g., open) or priority (e.g., blocker) plan a fix for it.	Begel and Zimmermann (2014a)
Q13. Who is the author of **File Name**?	Developers who have questions about a specific file or class may want to speak to the person who created it.	Sharma et al. (2017); Sillito et al. (2008)
Q14. Which developer(s) have the most unfixed bugs?	To determine the overloaded developers and possibly reassign bugs to others on the team.	Begel and Zimmermann (2014a)
Q15. What is the percentage of bug fixing commits that introduced bugs in/on **Date**?	To study the characteristics of the fixing commits which happened at a certain time and induced bugs.	Begel and Zimmermann (2014a)

4.3 Case Study Setup

To determine whether using bots really helps answer tasks based on repository data, we perform an experiment with 12 participants using Hibernate-ORM and Kafka repositories. We built a web-based bot application that implemented our framework and had users directly interact with the bot through this web-application. A screenshot of our bot's interface is shown in Figure 4.3. We divided the participants into two groups (each of 6 participants). Then, we sent emails that include links to the bot and online survey to both groups' members, asking each group to perform a set of tasks using the bot. Finally, we examine the bot in terms of its effectiveness, efficiency and accuracy and compare it to a baseline where both groups' members are asked to perform the same tasks without the bot. It is important to emphasize that each group performs the same set of tasks related to a specific repository. In other words, all members of group 1 performed the tasks using Hibernate-ORM, while the members of the group 2 used Kafka project.

To extract the intents and entities, we leveraged Google's Dialogflow engine Google (2019a). Dialogflow has a powerful natural language understanding (NLU) engine that extracts the intents and entities from a user's question based on a custom NLP model. Our choice to use Dialogflow was motivated by the fact that it can be integrated easily with 14 different platforms and supports more than 20 languages. Furthermore, it provides speech support with third-party integration and the provided service is free. These features make it easier to enhance our framework with more features in the future.

Any NLU model needs to be trained. Therefore, to train the NLU, we followed the same approach as Toxtli et al. (2018). Typically, the more examples we train the NLU on, the more accurate the NLU model can extract the intents and entities from the users questions Google (2019c). As a first step, we conducted a brainstorm session to create the initial training set which represents the different ways that the developers could ask for each intent in Table 4.1. Moreover, our bot can handle basic questions such as greeting users and asking general questions about the bot such as: "How are you?". Then, we used the initial set to train the NLU and asked two developers (each has more than 7 years of industrial experience) to test the bot for one week. During this testing period, we used the questions that the developers posed to the bot to further improve the training of the

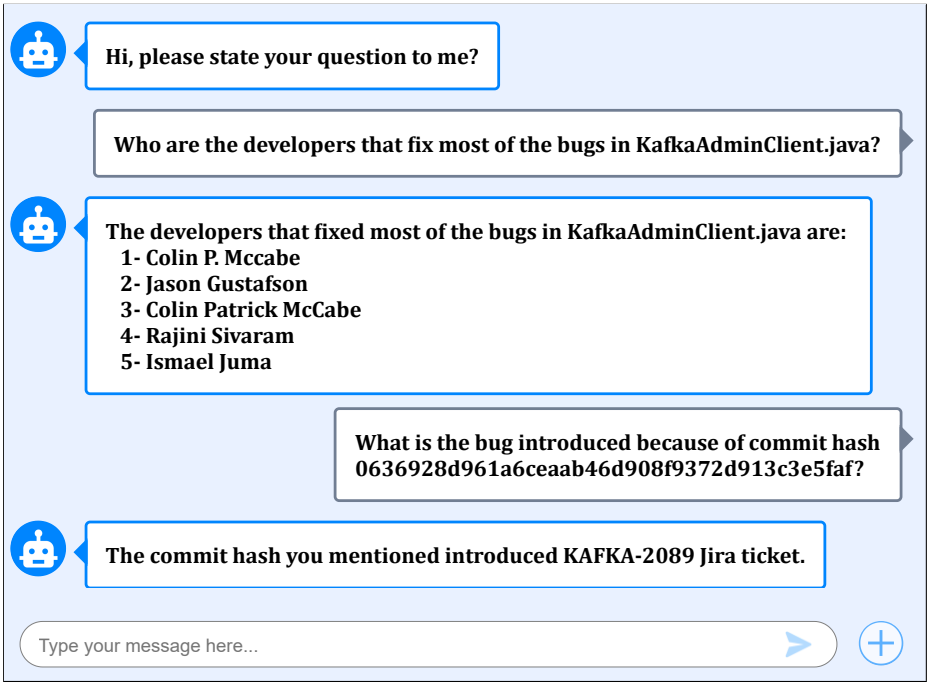


Figure 4.3: An Example User Conversation with MSRBot

NLU. The training data is publicly available on Abdellatif et al. (2019c).

Although we use Dialogflow in our implementation, it is important to note that there exist other tools/engines that one can use such as Gensim Gensim (2019), Stanford CoreNLP Manning et al. (2014), Microsoft's LUIS Microsoft (2019b), IBM Watson IBM (2019c), or Amazon Lex Amazon (2019).

To ensure that the usage scenario of the Bot is as realistic as possible, we asked the participants to perform a set of tasks using the bot. We used the questions that have been identified in the literature as being of importance to developers Begel et al. (2010); Begel and Zimmermann (2014a, 2014b); Fritz and Murphy (2010); Sharma et al. (2017); Sillito et al. (2008) to formulate the tasks in our experiment. The participants use the bot by posing any number of natural language questions needed to complete the task. Next, the participants evaluate the bot by answering a set of questions related to the task and bot's reply. To compare, we also ask the participants to perform the same tasks without the bot, which we call the baseline comparison. We detail all the steps of our case study in this section.

4.3.1 Questions Supported by the Bot

To perform our study, we needed to determine a list of questions that our bot should support. To do so, we surveyed work that investigated the most commonly asked questions of software practitioners (mostly developers and managers). We found a number of notable and highly cited studies, such as the study by Begel and Zimmermann Begel and Zimmermann (2014a, 2014b), which reported questions commonly asked by practitioners at Microsoft, the study by Fritz and Murphy Fritz and Murphy (2010) that conducted interviews with software developers to determine questions that developers ask and the study by Sharma *et. al* Sharma et al. (2017), which prioritized the importance of questions that developers ask. We also surveyed a number of other studies whose goal was not directly related to questions that developers ask, but which we found to be relevant for us to understand which questions we should ask (e.g., Begel et al. (2010); Sillito et al. (2008)). In most of these studies, *the authors reported that software practitioners are lacking support in answering these questions, which is exactly what our bot can help provide*. After going through the literature, we selected arbitrarily 15 questions that our bot can support and *can be answered using repository data.*

Table 4.1 presents the questions we use in the case study, the rationale for supporting the question and the study where the question was mentioned/motivated from. Each question represents an intent and the bold words represent the entities in the question. For example, the user could ask Q11 as: "What are the buggy commits that happened last week?", then the intent is "Determine Buggy Commits" and the entity is "last week". It is important to emphasize that the bot's users can ask the questions in different ways other than what is mentioned in Table 4.1. In the last example the user can ask the bot "What are the changes that introduced bugs on Dec 27, 2018" where the intent remains the same although the question is asked in a different way and the entity is changed to a specific date (Dec 27, 2018).

Although we support 15 questions in our prototype at this time, it is important to note that the bot framework can support many more questions and we are extending it to do so now. We opted to focus on these 15 questions since our goal is to evaluate the bot in this research context and wanted to keep the evaluation manageable.

Table 4.2: Participants' Knowledge on Version Control Repositories and Issue Tracking System

Likert Scale	Number of Participants	
	Version Control Repositories (Git)	Issue Tracking System (Jira)
1 (No Knowledge)	0	1
2 (Entry Level)	2	4
3 (Intermediate)	3	4
4 (Competent)	5	1
5 (Expert)	2	1

4.3.2 Study Participants

Once we decided on the 15 questions that we are able to support, we want to evaluate how useful the bot is. Since bots are meant to be used by real software practitioners, we decided to evaluate our bot through a user study.

Our user study involved 12 participants. For each participant, we asked them about their main occupation, background, software development experience and their knowledge of software repositories. All participants were graduate students (4 Ph.D. and 8 master students). Of the 12 participants, 75% have more than 3 years of professional software development experience and 25% have between 1-3 years of development experience. The participants' experience using Version Control Repositories (e.g., Git) and Issue Tracking System (e.g., Jira) are shown in Table 4.2.

We deliberately reached out to graduate students to conduct our user study for a few specific reasons. First, we knew that most of these graduate students (80%) have worked in a professional software development environment in the past. Second, since this is one of the first studies using bots, we wanted to interview some of the participants in person, which provides us with invaluable feedback about aspects of using bots that may not come out in the user study. Expecting developers from industry, who are already busy and overloaded, dedicate this much time to our study would be difficult and if they did, we would not be able to go as deep in our study with them. Also, as prior work has shown, students can be a good proxy for what developers do in professional environments, especially if the participants are experienced and the technology under study is new, which is our case Höst, Regnell, and Wohlin (2000); Salman, Misirli, and Juristo (2015). Lastly, the main goal of our case study is to evaluate the proposed framework, i.e., this is a judgment study

rather than a sample study, using students is completely valid. The real important factors for us is not the characteristics of the participants (students), rather the evaluation of the framework. Once we recruited our participants, we devised a questionnaire survey to evaluate the bot and baseline approaches. We detail our survey next.

4.3.3 Questionnaire Survey

We devised a survey that participants answer to help us understand the usefulness of the bot. To make the situation realistic, we mined the data from the Git and Jira repositories of the Hibernate-ORM and Kafka projects. Hibernate is a Java library that provides Object/Relational Mapping (ORM) support. And, Kafka is a Java platform that supports streaming applications. We setup our bot framework to be able to answer all the supported questions on Hibernate's and Kafka's repository data. There was no specific reason for choosing those projects as our case study, however, they did meet some of the most common criteria - they are large open source projects (Hibernate with 177 releases and Kafka with 97 releases) that uses Git and Jira, they have rich history (each has more than 6,500 commits, 8,800 bug reports), are popular amongst developers (each has more than 340 unique contributors) and have been studied by prior MSR-type studies (e.g., T. M. Ahmed, Bezemer, Chen, Hassan, and Shang (2016); Digkas, Lungu, Chatzigeorgiou, and Avgeriou (2017); Kabinna, Bezemer, Shang, and Hassan (2016); Sawant and Bacchelli (2017)).

Our survey was divided into three parts. The first section gathered information about the participants and asked questions related to experience, current role, and knowledge in mining software repositories, which is the information we presented in the section above. The second part of the survey was composed of 10 tasks that the participants are asked to perform. The task statements we gave the users all the needed information to complete the task. For example, the given task statement would say "ask about the commits that fix HHH-6574", and a user might use the bot to perform the task by asking: "which commits fixed the bug id?" In this case, the user is free to ask the question in any way they prefer, e.g., what are the fixing commits of HHH-6574 ticket? We provide a Jira ticket number that exists in Hibernate and Kafka, since we do not expect the user/participant to know this, however, someone related to any project is asking this question, s/he would indeed have such information. The remainder of the second part contained questions for the participants to

evaluate the bot based on the answer they receive. We discuss the questions used to evaluate the bot in the next section.

In addition to asking the participants to complete the tasks using the bot, we also got them to perform the same tasks manually (without using the bot) to have a baseline comparison. For the baseline evaluation, we gave the exact tasks formulated from the questions shown in Table 4.1 to the participants, so they know exactly what to answer to. The participants were free to use any technique they prefer such as writing a script, performing web searches, using tools (e.g., gitkraken *Git Client - Glo Boards — GitKraken* (2019) and Jira Client *Jira Client — Atlassian Marketplace* (2019)), executing Git/Jira commands, or searching manually for the answer in order to complete the tasks. Our goal was to resemble as close to a realistic situation as possible.

4.3.4 Evaluating the Bot

Bots are typically evaluated using factors that are related to both, accuracy and usability Vasconcelos, Candello, Pinhanez, and dos Santos (2017). Particularly, this work suggested two main criteria when evaluating bots:

- **Usefulness:** which states that the answer (provided by the bot) should include all the information that answers the question clearly and concisely Sankar, Greyling, Vogts, and du Plessis (2008); Zamora (2017).

- **Speed:** which states that the answer should be returned in a time that is faster than the traditional way that a developer retrieves information Zamora (2017).

In essence, bot should help developers in performing their tasks and do this in a way that is faster than if you were not using the bot. In addition to the two above evaluation criteria, we added another criteria, related to the accuracy of the answers that the bot provides. In our case, we define **accuracy** as the number of correctly completed tasks performed by the bot, where the task is marked as correct if the returned answer by the bot matches the actual answer the actual answer to the question Vasconcelos et al. (2017). We formalize our case study with three research questions that are related to the three evaluation measures used, in particular we ask:

RQ1: How useful are the bot's answers to users' questions?

RQ2: How quickly can users complete their tasks using the bot?

RQ3: How accurate are the bot's answers?

To address **RQ1**, we ask two sub-questions. First, we ask whether the bot was able to return an answer in the first place (in some cases it cannot) and we ask the participants whether they consider that the answer returned is useful in answering the question posed on a five point Likert's scale (from very useless to very useful).

For **RQ2**, we recorded the actual time the participants need to perform each of the tasks while using the bot through an online survey tool. For each task, we measure the time starting from the moment the participant is given the task until s/he submits the task. We use this time to quantitatively compare the time savings to the baseline, i.e., accomplishing the same tasks without using the bot. In addition, we ask the participants to indicate how fast the bot replies to their questions on a five point Likert's scale from very slow to very fast (to measure perceived speed). For the case of the baseline, we asked the participants to measure and report the time it took them to complete each task. We limited the maximum time to finish each task to 30 minutes when using the bot and in the baseline, i.e., in case they did not manage to get an answer within 30 minutes, we considered the task to be incomplete.

To address **RQ3**, we recorded all the interactions performed by the participants and, more importantly, the output of each bot's components including its responses. Then, we analyzed the tasks' results manually to determine if a task is completed correctly or not (by cloning the repositories and writing the scripts to answer the questions). For example, if the participant asks for the number of commits on a particular day, we would check if the returned answer was actually correct or not. This enables us to ensure that the entity recognizer, intent extractor, mapping process in the knowledge base, and the response generator components are working correctly. In the case of the baseline questionnaire, we asked the participants to provide us the answer of each task in the survey. This allowed us to determine the accuracy of the manually determined tasks. To ensure that the participants actually searched the answer for the asked task, we required that the participants briefly explain how they performed the task. Having this information also provided us with insight into how much work and what tools/techniques/commands practitioners typically use to perform such

tasks. Finally, at the end of the survey, we added an optional field to allow the participants to write their comments or suggestions, if they have any.

To avoid overburdening the participants, we divided them into two groups, where each group has 6 participants and is given 10 tasks to perform on a certain repository. The tasks were formulated from 10 randomly selected questions from the list of questions supported by the bot. The questions that were selected to formulate the tasks are Q1, Q2, Q3, Q5, Q6, Q7, Q9, Q11, Q14, and Q15 in Table 4.1. The first group performed the tasks on the Hibernate project while the members of the second group are instructed to do the same tasks on the Kafka project. Both groups (the Hibernate and Kafa groups) were asked to perform the tasks twice, once using the bot and another time without the bot (which we call the baseline). None of the participants knew the questions that the bot was trained on. This is to ensure they will use their own words when they are interacting with the bot and to monitor the questions that the participants ask the bot about. It is important to emphasize that each group received the same tasks in both, the baseline questionnaire and the bot-related questionnaire.

4.4 Case Study Results

In total, the 12 participants asked the bot 165 questions (some developers asked more than 10 questions) to perform the assigned tasks Abdellatif et al. (2019c). Of the 165 questions, we excluded 9 questions from our analysis because they were out of scope (e.g., "What's your name?", "What language are you written in?"), as the main focus of this work is to study bots on software repositories. Therefore, all of the presented results are based on the remaining 156 questions that are relevant.

4.4.1 RQ1: How useful are the bot's answers to users' questions?

As mentioned earlier, one of the first criteria for an effective bot is to provide its users with useful answers to their questions. Evaluating a bot by asking how useful its answers were commonly used in most bot-related research (e.g. Feng, Shaw, Kim, and Hovy (2006); B. Xu et al. (2017b); Zamora (2017)).

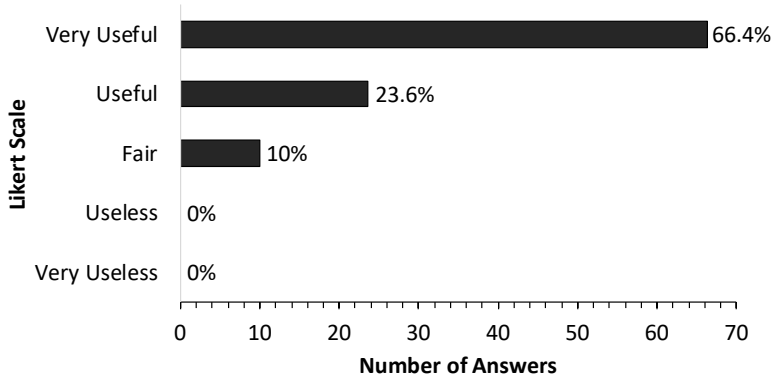


Figure 4.4: Usefulness of the Bot's Answers

Participants were asked to indicate the usefulness of the answer provided by the bot after each question they asked. The choice was on a five-point Likert's scale from very useful (meaning, the bot provided an answer they could actually act on) to very useless (meaning, the answer provided does not help answer the question at all). The participants also had other choices within the range, which were: useful (meaning, the answer was helpful but could be enhanced), fair (meaning, the answer gave some information that provided some context, but did not help the answer fully) and useless (meaning, the reply did not help with the question, but a reply was made).

Figure 4.4 shows the usefulness results in case they were correct. Overall, **90.0% of the participants indicated that the results returned by the bot were considered to be either useful or very useful**. Another 10.0% indicated that the bot provided answers that were fair, meaning the answers helped, but were not particularly helpful in answering their question. It is important to emphasize that when considering usefulness, we considered all tasks that were performed correctly.

Upon closer examination of the fair results, we found a few interesting reasons that lead users to be partially dissatisfied with the answers. First, in some cases, the users found that the information returned by the bot to not be easily understandable. For example, if a user asks for all the commit logs of commits that occurred in the last year, then the returned answer will be long and terse. In such cases, the users find the answers to be difficult to sift through, and accordingly indicate that the results are not useful. Such cases showed us that perhaps we need to pay attention to the way that answers are presented to the users and how to handle information overloading. We plan to address such issues in future versions of our bot framework. Another case is related to information

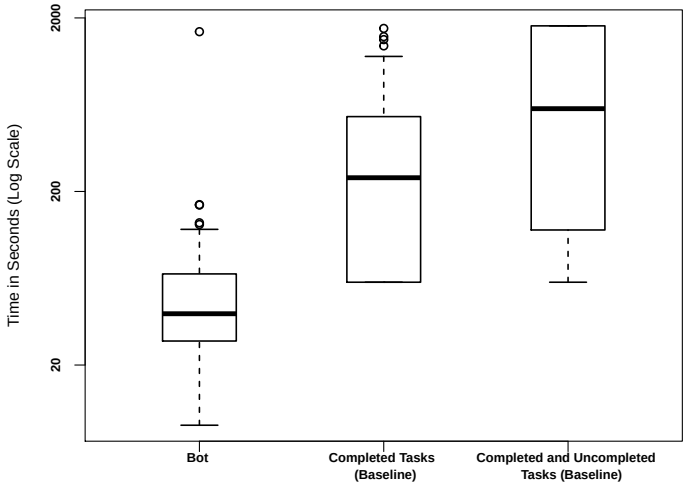


Figure 4.5: Time Required to Complete Tasks (bot vs. baseline)

that the users expected to see. For example, some users indicated that they expect to have the commit hash returned to them for any commit-related questions. Initially, we omitted returning the commit hashes (and generally, identification info) since we felt such information is difficult to read by users and envisioned users of the bot to be more interested in summarized data (e.g., the number of commits that were committed today). Clearly, the bot proved to be used for more than just summarized information and in certain cases users were interested in detailed info, such as a commit hash or bug ID. All of these responses provided us with excellent ideas for how we will evolve the bot.

> *The majority (90.0%) of the bot's users found it to be useful or very useful. Areas for improvement include figuring out how to effectively present the bot's answers to users.*

4.4.2 RQ2: How quickly can users complete their tasks using the bot?

Since bots are meant to answer questions in a chat-like forum, speed is of the essence. Therefore, our second RQ aims to shed light on how fast can users perform tasks related to their repositories using the bot and compare that to the time they need to complete the same tasks without the bot

(i.e., the baseline). We also ask the users to indicate their perceived speed of the bot.

Measured speed. We measure the exact time that the participants needed to complete each of the given tasks, once using the bot and another without the bot, which we call the baseline. This gives us better insights about the actual time savings when using the bot.

Figure 4.5 shows boxplots of the distribution of the time it took for the participants to perform the tasks, with and without the bot (note that the y-axis is log-scaled to improve readability). As evident from Figure 5, using the bot (the left most box plot) significantly outperforms the baseline approach, achieving a median task completion time of 40 seconds and a maximum of 1666 seconds. On the other hand, for the baseline approach, we have two results - one that considers all tasks that users were able to complete (labeled "Completed Tasks (Baseline)" in Figure 5) and the other considering all tasks, i.e., completed and uncompleted[1] (labeled "Completed and Uncompleted Tasks (Baseline)" in Figure 5). The median task completion time for the tasks in the baseline approach is 240 seconds with a maximum of 1,740 seconds. While, if all the completed and uncompleted tasks are considered, the time to perform a task is even higher, with a median of 600 seconds and a maximum of 1,800 seconds. To ensure that the difference between the bot and the two cases of the baseline is statistically significant, we performed a Wilcox test, and the difference in both cases (i.e., using the bot vs. completed tasks in the baseline and using the bot vs. all tasks in the baseline), and find that the difference is statistically significant (i.e., p-value≤ 0.01). It is obvious that using the bot to perform tasks is faster than the baseline approach. However, this research question investigates the amount of time that the bot saves compared to users manually doing the tasks, which in our case is more than 3 minutes/task, on median.

Perceived speed. The other side of the coin is to determine how users perceive the speed of the bot to be. To accomplish this, we asked users to indicate how fast they received the answer to their question from the bot. Once again, the choices for the users were given on a five point Likert's scale, from very fast (approx. 0 - 3 seconds) to very slow ($\geq$ 30 seconds). The participants also had other choices within the range, which were: fast (4 - 10 seconds), fair (11 - 20 seconds) and slow (21 - 30 seconds).

[1]Since we gave a maximum of 30 minutes for participants to complete a task, tasks that were not answered after 30 minutes were considered to be incomplete and also to have taken 30 minutes.

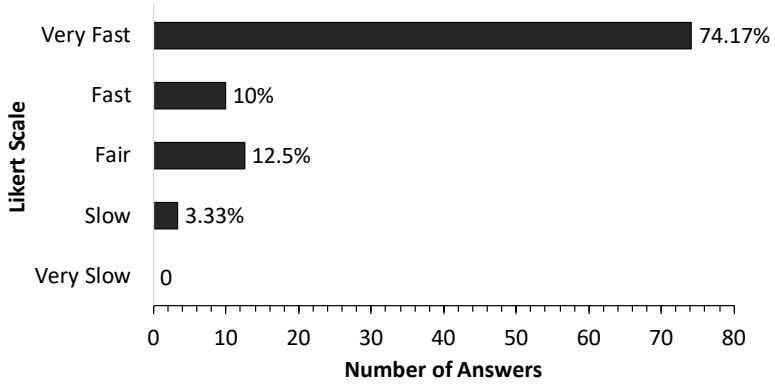


Figure 4.6: Speed of the Bot's Reply

Figure 4.6 shows the results of the survey participants. The majority of the responses (84.17%) indicated that the bot's responses were either, fast or very fast. The remaining 15.83% of the replies indicated that the bot's response was either fair or slow. Clearly, our answers show that the bot provides a significant speed up to users.

Deep-dive analysis. To better understand why some of the tasks took longer to perform using the bot, we looked into the logged data and noted 4 cases that may have impacted the response speed of the bot. We found that in those cases, Dialogflow took more than 5 seconds to extract intents and entities from the user's question. We searched for the reasons behind Dialogflow's delay and found that the way users ask questions can make it difficult for Dialogflow's algorithms to extract the entities and intents. On the other hand, there is one case where the participant required more than 30 minutes to complete their task using the bot. We followed up with the participant afterwards to determine the reason and were told that the participants simply "forgot" to complete the task since they were distracted.

As for the case where users took a long time to find that answers in the baseline case, we found that the main reason for such delays is that some tasks were more difficult to answer. Hence, users needed to conduct online searches (e.g., using Stack Overflow) of ways/techniques that they can use to obtain the answer.

That said, overall, the participants were fast in completing tasks using the bot. It is important to keep in perspective how much time using the bot saves. As we learned from the feedback of our baseline experiments, in many cases, and depending on the task being performed, a developer may

need to clone the repository, write a short script, and process/clean-up the extracted data to ensure that they complete the tasks correctly - and that might be a best case scenario. If the person looking for the information is not very technical (e.g., a manager), they may need to spend time to learn what commands they need to run or tools to use, etc., which may require several hours or days.

> *The participants take a median time of 40 seconds to perform a task using the bot. Moreover, the majority (84.17%) of the bot's users perceived the bot's responses to be fast or very fast. However, the way that the user frames the question may impact the speed of the bot's reply.*

4.4.3 RQ3: How accurate are the bot's answers?

In addition to using the typical measures to evaluate bots, i.e., usefulness and speed, it is critical that the bot returns accurate results. This is of particular importance in our case, since software practitioners generally act on this information, sometimes to drive major tasks.

Bot's performance. We measure accuracy by checking the tasks' results performed by the users using the bot and comparing it with the actual answer to the task if it was queried manually by cloning the repositories then writing a script to find the answer or executing git/Jira commands. For example, to get the developers who touched the "KafkaAdminClient" file, we ran the following git command: "git log –pretty=format:%cn – clients/src/main/java/org/apache/kafka/clients/admin /KafkaAdminClient.java". This RQ checks each component's functionality in the framework. Particularly, it checks whether the extraction of the intents and entities is done correctly from the natural language question posed by the users. Moreover, we check whether our knowledge base component queries the correct data and if the response generator produces the correct reply based on the intent and knowledge base, respectively. In total, the first two authors manually checked all the 120 completed tasks by the participants using the bot.

Our results showed that the users correctly completed 90.8% (109 of 120) of the tasks using the bot. Manual investigation of the correct tasks showed that the bot is versatile and was able to handle different user questions related to the tasks. For example, the bot was able to handle the questions *"tell me the number of commits last month"* asked by participant 1 vs. *"determine the number of*

Table 4.3: Reasons for Uncompleted Tasks by the Bot

Reason	Number of Questions
Extract Intent	5
Recognize Entity	5
Developer's Distraction	1
Out of scope	9

commits that happened in last month." asked by participant 2 vs. *"how many commits happened in the last month"* from participant 3, which clearly have the same semantics but different syntax.

Our findings indicate that the 10 tasks that the users fail to complete correctly were due to the incorrect extraction of intents or entities by our trained NLU model as shown in Table 4.3. For example, in one scenario the user asks "tell me the commit info between 27th May 2018 till the end of that month?" and our NLU model was unable to identify the entity (because it was not trained on the date format mentioned in the participant's question). Consequently, the knowledge base and the response generator components mapped the wrong entity and returned an incorrect result. Interestingly, in such cases, some of the participants had to ask the bot more than once to complete the task correctly. For example, P1 posed the following question "how many of June 2018 bugs are fixed" to perform task 10, the bot fails to extract the correct intent (the fixing commits that induce bugs) from the question, which lead to an incorrect reply (returned the developers that are expert in fixing bugs). Consequently, the participants rephrased the query to the bot as follows: "show me the percentage of bugs fixes that introduced bugs in June 2018" which allowed the bot to return the correct answer. Overall, the participants asked 1.3 question per task, on average. It is important to note that we consider the task where the participant is distracted as incomplete since s/he took more than 30 minutes.

Baseline performance. As mentioned earlier, we also conducted a baseline comparison where we asked users to perform the same tasks without the bot. Figure 7 shows a break down of 1) the number of completed tasks and 2) the number of *correct* tasks per completed tasks. On the positive side, we can see that the survey participants were able to provide some sort of answer for all tasks, albeit some of the tasks (e.g., T3, T6, T11 and T15) had less answers from participants. Across all tasks, the participants provided some sort of answer in 62.6% of the cases.

However, what is most interesting is that the number of correct answers is much lower. Across

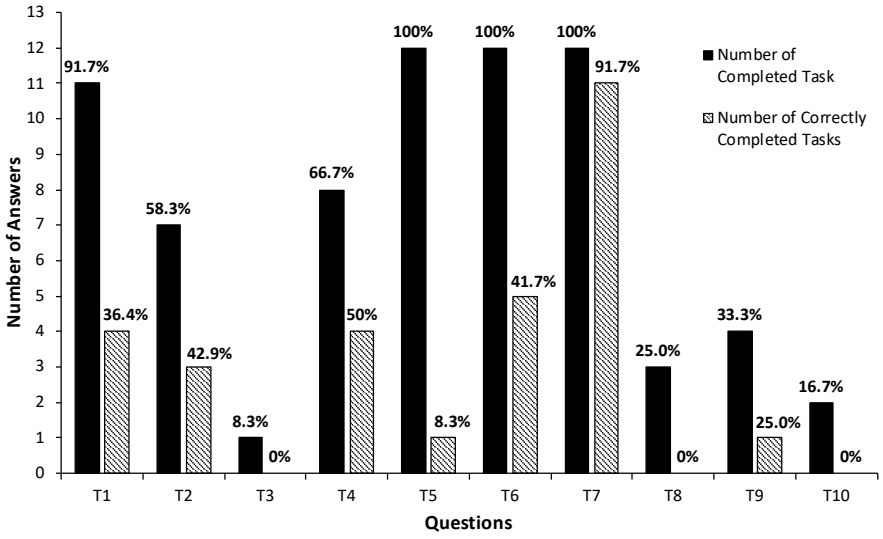


Figure 4.7: Number of Answers for Each Task in the Baseline

all tasks, the survey participants provided the correct answer in 25.2% of the cases. For example, for T3, T11 and T15, all of the provided answers were incorrect. On the other hand, T9's answers were all correct.

This outcome highlights another (in addition to saving time) key advantage of using the bot framework, which is that reduction of human error. When examining the results of the baseline experiments, we noticed that in many cases participants would use a wrong command or a slightly wrong date. In other cases where they were not able to provide any answer, they simply did not have the know how or failed to find the resources to answer their question within a manageable time frame.

> *Overall, the bot achieves an accuracy of 90.8% in answering user's questions, which is much higher than the baseline's accuracy of 25.2%. Techniques to make the NLU training more effective can help further improve the bot's accuracy.*

Follow-up Interviews

The survey results provided us with an excellent way to quantify the usefulness, efficiency, and accuracy of the bot. However, we wanted to obtain deeper insights, particularly related to

what the participants felt were the strengths and areas for improvement for the MSR-related bot framework. Therefore, we conducted semi-structured interviews, where we sat with 5 of the 12 survey participants and asked them: 1) What they believe are the strengths of using a bot framework? 2) What they believe can be improved? and 3) any general comments or feedback they had for us? We asked for permission to record the results of the interview, to enable us to perform deeper offline analysis of the results.

In terms of strengths, most of the points mentioned during the interview surrounded the benefits and applicability of the bot on top of software repositories in industry/practical settings. We elaborate on some of the points pointed by the interviewees below:

- **Bots are useful in software projects where personnel with many roles are involved.** Although we knew from the RQs that bots will be useful, however, our interview revealed to us that bots are probably useful to more than just developers. For example, P1 and P2, two of the most experienced participants indicated that usually large projects involve personnel with varying technical backgrounds. And it is usually the less technical personnel (e.g., project managers) that can significantly benefit from repository information, but often lack the know-how to extract such information.

- **Bots are very efficient, even when large and long-lived projects are being analyzed.** Another major strength pointed out is the ability of the bot to provide answers from a very rich history very quickly. For example, P4 mentions the fact that some repos "have more than 20,000 [or more] commits and if you are going through like 20,000 bugs or specific commits it will take so much time and probably you may even miss important data easily".

  Another participant, P5 had a different perspective and saw the potential for bots to help researchers that study software projects. Particularly, he pointed out that our bot framework can be used by researchers who may want to get a few quick answers about a project or do some deep-dive analysis based on some of their findings. For example, one can ask the bot how many changes or how many bugs were reported against a file that they found to yield interesting results in their analysis.

- **Bot's have a very low barrier-to-entry, especially since they can understand natural**

language. The participants pointed out that a major advantage is that they did not need to 'learn' any specific technology or language to interact with the bot. This significantly lowers the barrier to entry for adopters of our framework. For example, P3 says "I like that you type in natural language without thinking about building a query to do the thing [answer the question at hand]"

Other points mentioned reaffirmed our quantitative findings. For example, participants P2, P3 and P5 all mentioned the bot's speed in replying to questions and the fact that the bot helps complete the tasks at hand faster as a major benefit. We do not discuss this in detail here, since we believe our results already discussed such point and there is no point in repeating the same points again.

We also gained some valuable feedback about the potential areas of improvement for the bot framework. We elaborate on some of the points mentioned by the interviewees below:

- **Support deep-dive of answers provided.** The participants mentioned that although they appreciated the simplicity and clarity of the bot's answers, you see adding the ability of allowing the user to dive more into the results as a potential future feature. For example, participant P1 mentioned that it would be great to add hyperlinks for commit hashes or bug IDs that are returned. We believe that such a feature is indeed warranted and plan to (and believe we could easily) implement such a feature.

- **Make the bot more resilient and modifiable.** One clear limitation of our bot framework is that it supports a limited number of questions, even though we did that since our goal is to evaluate the viability of using a bot on top of software repositories. In any case, the participants P2 and P3 suggested that a clear improvement is to support more questions. We certainly plan to look into ways that can make our bot learn effectively from questions that are not yet supported, based on the questions users ask. Another participant, P4 suggested that we try and add a recommendation system to the bot so that typos are fixed with a "did you mean ..." type of messages. Also, questions that might be mistyped, e.g., how vs. who, can be provided with a suggested fix.

That said, all participants strongly showed support for the idea and mentioned they see a lot of potential for the combination of bots and software repositories. Our work is a step in the right

direction, showing the value and applicability of using bots to effectively extract useful information easily from software repositories.

4.5 Discussion

In this section, we present other perspectives in evaluating the software bots, and discuss the future and practical implications of our work.

4.5.1 Bots Evaluation

One of the main challenges in our work is the evaluation part since there is not much prior work that evaluates the use of bots on software repositories. One of the goals of using bots in the software engineering domain is to improve developers' productivity Storey and Zagalsky (2016a). Therefore, we believe that any proposed bot should be evaluated against the methods that developers usually use to perform their tasks.

As discussed in section 4.3, we evaluate the proposed framework according to its effectiveness and usefulness for developers in performing their tasks. However, there may be other ways to assess software bots. For example, usage, which can be measured by the number of times that a user uses the bot in a certain duration (e.g., a week). Other measures for speed can be related to the number of interactions needed for a user to complete a task, i.e., asking more questions to perform one task increases the time required to complete that task.

In our case study, there are 11 cases where the participants reworded the question because the bot failed to return the correct answer because of 1) incorrect extraction of intents and entities 2) missing entities in the posed questions 3) typos in the users questions (e.g., What commits happened between 27/5/208 - 31/5/2018) 4) the bot encountered connection issues to the internet. For example, one of the participants asked the bot "What the details of the commits between May 27th 2018 and May 31st 2018", the bot failed to extract the entity (May 27th 2018 and May 31st 2018) because it was not trained on date format that is specified in the posed question. Then, the user rephrased the question to the bot as follows: "What are the details of the commit between 27/5/2018 - 31/5/2018", which allowed the bot to extract the entity correctly.

That said, measuring the total time needed to perform a certain task may cover the number of interactions metric. One can also argue that the number of interactions is considered as a measure for users satisfaction rather than a speed. Because a large number of interactions to perform a task impacts the satisfactions of the bot users negatively Ask, Facemire, Hogan, and Conversations (2016); *The impact of conversational bots in the customer experience - Good Rebels* (2020); C. Lebeuf et al. (2018d).

In general, the evaluation of software bots varies based on their characteristics and the tasks they can perform. For example, if a bot is proactive (e.g., reminds the user to execute a certain task), then the number of interactions may be invalid as a speed metric since the conversation always comes from the bot. Storey and Zagalsky Storey and Zagalsky (2016a) suggested to measure the bot's efficiency and effectiveness in completing developers' tasks. However, we believe that there are different dimensions that bots' developers and researchers can use to assess the proposed bots regardless of their types such as accuracy and user satisfaction. For example, to measure user satisfaction, the bot can monitor and track the sentiment of a user through the conversation and take different actions based on the current user sentiment. On the other hand, bots can be evaluated based on their intelligence, such as how easily they can adapt to new contexts, their ability to explain the reasoning behind their behavior, and the degree of smoothness of the conversation flow C. Lebeuf et al. (2018d). We believe that bots evaluation is still an active area and we encourage researchers to investigate the various dimensions and measures that the bots community can use to assessdifferent bots.

4.5.2 Study Implications

Our framework has a number of implications on both, software engineering research and practice.

Implications for Future Research: Our study mainly focused on proposing and evaluating a framework to answer some of the most common developers' questions using software repositories. Based on our results, we find a number of implications for future research. First, there is a need for the MSRBot to support more complex queries, as indicated by the participants' comments.

For example, one of the participants stated that the bot is going to help a lot in answering more complex queries using the repositories data. Hence, our study motivates the need to examine more complex questions using data from different types of software repositories. Also, our study shows that there is a need to develop approaches that answers questions dynamically (i.e., removing the need to have predefined questions) using repositories to overcome the limited number of questions supported by the bot.

Second, the retrieved data in the bot's answers and the way it is displayed to the user can be improved. For example, one of the participants recommends providing a suggestion to the user in case the bot was unable to identify the user's intent from the posed question. Another participant indicated that the bot's reply should be short and suggested the use of pagination to enable the user to navigate more easily through the bot's reply. Using pagination when displaying the bot's reply (e.g., commits that happened in the last 3 months) needs careful consideration since questions such as how many records to display on each page, and how can users know in which page a specific record is, need to be considered. Furthermore, it is unconventional to implement the pagination in a chat interface, which might affect user satisfaction. Although there are many studies on software bots, we are not aware of any studies that discuss the best way to represent the bot's reply and kind of information that bots' users expect to see. Our findings motivate the need for such studies.

Practical Implications: A direct implication of our findings is that using the bot simplifies the extraction of useful information from software repositories in different ways. First, it helps project stakeholders that do not have technical skills to easily extract the information from different repositories using the natural language. And, although some developers have the skills to perform such tasks (i.e., mining and analyzing data from software repositories), our framework reduces the time to complete those tasks compared to the baseline (i.e., any tool other than MSRBot). Overall, we believe that our framework supports practitioners at different levels in performing their tasks by lowering the barrier to entry for extracting useful data from software repositories. For example, it helps project managers to track the project progress (based on the closed tickets) and assists developers in their daily tasks.

4.6 Threats to validity

In this section, we discuss the threats to construct, internal and external validity of our study.

Construct Validity: The 12 participants used to evaluate the bot framework may have reported incorrect results, which would impact our findings. However, we are quite confident in the results returned since 1) most of these students have professional software development experience and 2) in most cases, there was a clearly popular answer (i.e., very few outliers). We also interviewed a subset of the participants, and based on our discussions, all participants seemed very competent in evaluating the output of the bot.

We selected different questions from the literature to evaluate the proposed framework which might bias our evaluation by adding the questions that the bot can better answer. However, we mitigate this threat to validity in different ways. First, we selected the list of questions from different studies arbitrarily. Second, in the given tasks, the participants are free to ask any type of questions to the bot (that is related to the task). Lastly, the participants are unaware of the list of questions that the bot can answer or is trained on.

Internal Validity: We used Google's Dialogflow to extract the intents and entities from the posed questions. Hence, our results might be impacted by Dialogflow's ability to translate the user's questions. That said, RQ3, which examines the accuracy of the bot showed high accuracy, which makes us confident in the use of Dialogflow. However, using another framework, might lead to different results. In the future, we plan to examine the impact of such frameworks on our bot. Also, Dialogflow's NLU model is trained on examples that were provided and manually labeled (i.e., the intents and entities) by us. The quality and quantity of training data may impact the effectiveness of the NLU. That said, in our evaluation, Dialogflow was able to handle the majority of questions from our users. In the future, we plan to investigate ways that our bot can learn from user's input and automate the intent and entity extraction/training phase.

Another threat to internal validity is that we used the questions identified in the literature to formulate the tasks in our case study. In some cases, the statement provided in the task might bias the participants on how to pose questions to the bot. The reason for providing the statements to the participants is that we want to keep our study manageable and to evaluate the developed bot

using the supported types of questions. However, we mitigate this threat by providing the question as a statement. Also, we did not reveal the list of the questions that the bot was trained on to the participants. Albeit anecdotal, we believe that our results show a limited effect of the wording of the tasks on the participants because their questions were syntactically different than the provided statements, although they have similar semantics) (e.g., "how many commits in the last month" and "what are the details of the commits between 27/5/2018 - 31/5/2018?".

External Validity: Our study was conducted using Hibernate-ORM and Kafka projects and supported 15 common questions. This means that the study might yield different results when selecting other projects or supporting a different set of questions related to software repositories. These are threats to external validity as they may limit the generalisability of our results. However, we plan to expand our study to support more systems and questions in the future. Also, it is important to note that the point of this chapter is to design and evaluate the feasibility of using bots on top of software repositories to automate the answering of commonly asked questions.

Also, we used students to evaluate our framework. Therefore, using different participants (i.e., developers) might affect the generalizability of our results. However, we argue that the main purpose of the case study is to evaluate our approach rather than studying the participants characteristics. Moreover, most of the participants have more than 3 years of industrial experience which reduces the threats to external validity. Furthermore, we plan to conduct a large-scale study in the future by having more developers from the industry.

4.7 Chapter Summary

Software repositories contain numerous amounts of useful information to enhance software projects. However, not all project stakeholders are able to extract such data from the repositories since it requires technical expertise and time. In this chapter, we design and evaluate the feasibility of using bots to support software practitioners in asking questions about their software repositories. Our findings show that the bot provides answers that are considered very useful or useful (as indicated by 90% of the participants), efficient (participants take on median 40 seconds to complete the tasks), and accurately answers questions posed by its users (as indicated by 90.8% of completed

tasks). Also, our study highlighted some of the potential pitfalls when applying bots on software repositories. For example, our study finds that more attention is needed regarding the content and length of the information that the bot replies with, how to handle complex user questions, and how to handle user errors such as typos. Overall, our work showed that bots have the potential of playing a critical role by lowering the barrier-to-entry for software stakeholders to extract useful information from their repositories. Also, the bots are efficient when used on large projects with a long history, which saves the developers' time needed to extract the data from the repositories. Finally, we believe that our work encourages other researchers to explore different usages of bots on different types of software repositories, e.g., code review repositories.

In addition to addressing the issues found in the user study (e.g., handling user typos), the results in this chapter outline some directions for future work. First, since the entity recognizer and intent extractor components accuracy depend on the training dataset, we will examine different techniques to generate a dataset for those components (e.g., using Stack Overflow topics) to increase their accuracy. We want to compare the performance of different Natural Language Understanding Platforms (e.g., Dialogflow and Rasa) using software engineering datasets to help the bot's community identify the platform that best fits their context. Also, we are planning to support a wider range of repositories such as (e.g., Gerrit Code Review). Moreover, in the current implementation of our framework, we did not provide the users with the ability to configure the bot. However, allowing users to configure the bot may improve the flexibility and adaptability of our framework. In the future, we plan to allow the MSRBot to be configured through users' feedback by asking users for their preferences and suggesting possible configurations. Finally, we plan to evaluate our bot framework using industrial settings to gain more insights into the roles and scenarios in which the bot can be used.

The focus of this part is to identify the chatbot development challenges and present the chatbot potential in enhancing the software development. In the next chapter, we shift our focus to evaluate widely-used NLUs using SE tasks with an aim to guide the developers to design more effective SE chatbots by choosing an NLU for their chatbots based on their needs and the context of usage in the SE domain.

Chapter 5

Can we help developers to design more effective chatbots for the SE domain?

Chatbots are envisioned to dramatically change the future of Software Engineering, allowing practitioners to chat and inquire about their software projects and interact with different services using natural language. At the heart of every chatbot is a Natural Language Understanding (NLU) component that enables the chatbot to understand natural language input. Recently, many NLU platforms were provided to serve as an off-the-shelf NLU component for chatbots, however, selecting the best NLU for Software Engineering chatbots remains an open challenge.

Therefore, in this chapter, we evaluate four of the most commonly used NLUs, namely IBM Watson, Google Dialogflow, Rasa, and Microsoft LUIS to shed light on which NLU should be used in Software Engineering based chatbots. Specifically, we examine the NLUs' performance in classifying intents, confidence scores stability, and extracting entities. To evaluate the NLUs, we use two datasets that reflect two common tasks performed by Software Engineering practitioners, 1) the task of chatting with the chatbot to ask questions about software repositories 2) the task of asking development questions on Q&A forums (e.g., Stack Overflow). According to our findings, IBM Watson is the best performing NLU when considering the three aspects (intents classification, confidence scores, and entity extraction). However, the results from each individual aspect show that, in intents classification, IBM Watson performs the best with an F1-measure$>$84%, but in

confidence scores, Rasa comes on top with a median confidence score higher than 0.91. Our results also show that all NLUs, except for Dialogflow, generally provide trustable confidence scores. For entity extraction, Microsoft LUIS and IBM Watson outperform other NLUs in the two SE tasks. Our results provide guidance to software engineering practitioners when deciding which NLU to use in their chatbots.

5.1 Introduction

Software chatbots are increasingly used in the Software Engineering (SE) domain since they allow users to interact with platforms using natural language, automate tedious tasks, and save time/effort Storey and Zagalsky (2016b). This increase in attention is clearly visible in the increase of number of bots related publications Abdellatif, Badran, and Shihab (2019b); Abdellatif, Costa, et al. (2020); Chun-Ting Lin and Huang (2020); Dominic et al. (2020b); B. Xu et al. (2017a), conferences Conference (2019), and workshops BotSE (2019). A recent study showed that one in every four OSS projects (26%) on GitHub are using software bots for different tasks Wessel et al. (2018b). This is supported by the fact that bots help developers perform their daily tasks more efficientlyStorey and Zagalsky (2016b), such as deploy builds DeployBot (2020), update dependencies Dependabot (2020), and even generate fixing patches Urli, Yu, Seinturier, and Monperrus (2018b).

At the core of all chatbots lie the Natural Language Understanding platforms—referred hereafter simply as NLU. NLUs are essential for the chatbot's ability to understand and act on the user's input D. Braun, Hernandez-Mendez, Matthes, and Langen (2017); Rychalska, Glabska, and Wroblewska (2018b). The NLU uses machine-learning and natural language processing (NLP) techniques to extract structured information (the intent of the user's query and related entities) from unstructured user's input (textual information). As developing an NLU from scratch is very difficult because it requires NLP expertise, chatbot developers resort to a handful of widely-used NLUs that they leverage in their chatbots Abdellatif et al. (2019b); Marbot (2020); Munoz et al. (2018); Murgia et al. (2016b); Toxtli et al. (2018).

As a consequence of the diversity of widely-used NLUs, developers are faced with selecting the best NLU for their particular domain. This is a non-trivial task and has been discussed heavily in

prior work (especially since NLUs vary in performance in different contexts) D. Braun, Hernandez-Mendez, et al. (2017); Canonico and De Russis (2018); Gregori (2017). For instance, in the context of the weather domain, Canonico and De Russis (2018) showed that IBM Watson outperformed other NLUs, while Gregori (2017) evaluated NLUs using frequently asked questions by university students and found that Dialogflow performed best. In fact, there is no shortage of discussions on Stack Overflow about the best NLU to use in chatbot implementation Damir (2020); T. G (2016); Nick (2018) as choosing an unsuitable platform for a particular domain deeply impacts the user satisfaction with the chatbot Ask et al. (2016); Lastra (2016); C. Lebeuf et al. (2018d).

An important domain that lacks any investigation over different NLUs' performance is SE. Software Engineering is a specialized domain with very specific terminology that is used in a particular way. For example, in the SE domain, the word 'ticket' refers to an issue in a bug tracking system (e.g., Jira), while in other domains it is related to a movie (e.g., TicketMaster bot) or flight ticket. Moreover, there is no consensus amongst SE chatbot developers on the best NLU to use for the SE domain. For instance, TaskBot Toxtli et al. (2018) uses Microsoft Language Understanding Intelligent Service (LUIS) *LUIS (Language Understanding) Cognitive Services Microsoft Azure* (2019) to help practitioners manage their tasks. MSRBot Abdellatif et al. (2019b) uses Google Dialogflow NLU to answer questions related to the software repositories. MSABot Chun-Ting Lin and Huang (2020) leverages Rasa NLU to assist practitioners in developing and maintaining microservices. Given that no study has investigated which NLU performs best in the SE domain, chatbot developers can not make an informed decision on which NLU to use when developing SE-based chatbots.

Hence, in this chapter, we provide the first study to assess the performance of widely-used NLUs to support SE tasks. We evaluate NLUs on queries related to two important SE tasks: 1) Repository: Exploring projects' repository data (e.g.,"What is the most buggy file in my repository?"), and 2) Stack Overflow: Technical questions developers frequently ask and answer from Q&A websites (e.g., "How to convert XElement object into a dataset or datatable?").

Using the two SE tasks, we evaluate four widely-used NLUs: IBM Watson IBM (2019a), Google Dialogflow Google (2020b), Rasa Rasa (2020b), and Microsoft LUIS *LUIS (Language Understanding) Cognitive Services Microsoft Azure* (2019) under three aspects: 1) the NLUs'

performance in correctly identifying the purpose of the user query (i.e., intents classification); 2) the confidence yielded by the NLUs when correctly classifying and misclassifying queries (i.e., confidence score); and 3) the performance of the NLUs in identifying the correct subjects from queries (i.e., entity extraction).

Our results show that, overall (considering NLUs' performance in intents classification, confidence score, and entity extraction), IBM Watson is the best performing NLU for the studied SE tasks. However, the findings from evaluating the NLUs on individual aspects show that the best performing NLU can vary. IBM Watson outperforms other NLUs when classifying intents for both tasks (F1-measure $>$ 84%). Also, we find that all NLUs (except for Dialogflow in one task) report high confidence scores for correctly classified intents. Moreover, Rasa proves to be the most trustable NLU with a median confidence score $>$ 0.91. When extracting entities from SE tasks, no single NLU outperforms the others in both tasks. LUIS performs the best in extracting entities from the Repository task (F1-measure 93.7%), while IBM Watson comes on top in the Stack Overflow task (F1-measure 68.5%).

Given that each NLU has its own strengths in the different SE tasks (i.e., performs best in intent classification vs. entity extraction), we provide an in-depth analysis of the performance of the different NLU's features, which are the list feature, where the NLU extracts entities using an exact match from a list of synonyms; and the prediction feature, where the NLU predicts entities that it might not have been trained on before. Also, we analyze the characteristics of the intents in each task to better understand the intents that tend to be harder to classify by all of the evaluated NLUs.

The chapter makes the following contributions:

- To the best of our knowledge, this is the first work to evaluate NLUs on two representative tasks (i.e., software repositories data and Stack Overflow posts) from the SE domain.
- We evaluate the NLUs using different features for extracting entities (i.e., list and prediction features).
- We explore the impact of selecting different confidence score thresholds on the NLUs' intent classification performance.

- We provide a set of actionable recommendations, based on our findings and experience in conducting this study, for chatbot practitioners to improve their NLU's performance.
- We make our labelled dataset publicly available to enable replication and help advance future research in the field Abdellatif, Badran, Costa, and Shihab (2021d).

5.1.1 Organization of the Chapter

The rest of the chapter is organized as follows. Section 5.2 provides an overview about chatbots and explains related concepts used throughout this chapter. Section 5.3 describes the case study setup used to evaluate the performance of the NLUs. We report the evaluation results in Section 5.4. Section 5.5 discusses our findings and provides a set of recommendations to achieve better classifications results. Section 5.6 discusses the threats to validity, and section 5.7 concludes the chapter.

5.2 Background

Before diving into the NLUs' evaluation, we explain in this section the chatbot-related terminology used throughout the chapter. We also present an overview of how chatbots and NLUs work together to perform certain actions.

5.2.1 Definitions

Software chatbots are the conduit between their users and automated services C. Lebeuf et al. (2018d). Through natural language, users ask the chatbot to perform specific tasks or inquire about a piece of information. Internally, a chatbot then uses the NLU to analyze the posed query and act on the users' request. The main goal of an NLU is to extract structured data from unstructured language input. In particular, it extracts intents and entities from users' queries: intents represent the user intention/purpose of the question, while entities represent important pieces of information in the query. For example, take a chatbot like the MSRBot Abdellatif et al. (2019b), that replies to user queries about software repositories. In the query "How many commits happened in the last month of the project?", the intent is to know the number of commits that happened in a specific period (*CountCommitsByDate*), and the entity 'last month' of type *DateTime* determines the parameter for

the query. The chatbot uses both the intent and entities to perform the action that answers the user's question. In this example, the chatbot searches in the repository for the number of commits issued in the last month.

Most NLUs come with a set of built-in entities (e.g., currencies and date-time), which are pre-trained on general domain queries. To use an NLU on a specialized domain, developers should define a set of custom intents and entities. For each custom intent, the NLU needs to be trained on a set of queries that represents different ways a user could express that intent. Again, taking the former example, "How many commits happened in the last month?", this query can be asked in multiple different ways. For instance, "show me the number of commits between 1-03-2020 and 31-03-2020" is an example of a query with the same semantics but different syntax. Both queries can and should be used to train the NLU on how to identify the *CountCommitsByDate* intent. Similarly to custom intents, NLUs need to be trained to recognize custom entities. To do that, developers label the entity types and their values in the queries. For example, in the following query "what is the fixing commit for bug HHH-8501?", the entity 'HHH-8501' is labelled as a *JiraTicket* type.

The misclassification of intents and entities negatively impacts the user experience, although each in its own way. When an NLU misclassifies an intent, the chatbot fails to understand the query in a fundamental manner, leading the chatbot to reply to a different query or performing the wrong task. Misclassifying entities, on the other hand, causes the chatbot to reply about a wrong piece of information. For example, in the query "How to convert xml to json file in java" there are three entities: 'XML', 'Json' and 'Java'. If the NLU fails to extract the 'Java' entity, the chatbot loses the context of the question and might reply with an answer for converting XML to Json with code example from any other programming language (e.g., Python).

The last piece in the picture is the confidence score, which represents how confident the NLU is in classifying the intent Google (2019b); IBM (2019b); Microsoft (2019c); Rasa (2019a). The confidence score is given on a scale from 0 (i.e., not confident) to 1 (i.e., fully confident), which corresponds to the classified intent by the NLU. Chatbot developers use the confidence score to choose their next action, either by answering the user's question/request or triggering a fallback intent. The fallback intent is a response issued by the chatbot to give the user a chance to rephrase or clarify their initial query. Typically, the fallback intent is triggered when the returned confidence

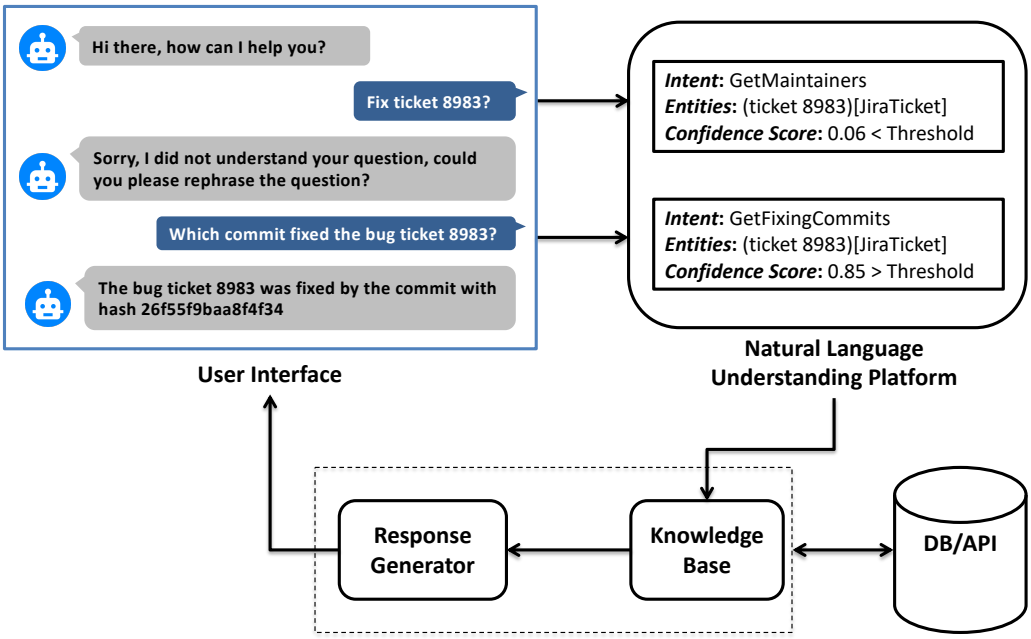


Figure 5.1: An overview of user-chatbot interaction

score is lower than a certain threshold. Choosing a suitable threshold for a chatbot is not an easy task, as a low value would make a chatbot answer to unclear questions more often (too confident), and a high threshold would trigger the fallback intent too often (insecure chatbot), annoying the user by asking it to rephrase the question frequently.

In our study, we want to investigate the NLUs' performance with regards to intents classification, confidence score, and entity extractions. All three aspects are critical to ensure that chatbots return correct and complete responses to the user.

5.2.2 Explanatory Example

To demonstrate how chatbots utilize NLUs to answer a user's query, we showcase an example of a user asking a repository related question to a chatbot as shown in Figure 5.1. In this example, we use a simplified architecture of the chatbot Abdellatif et al. (2019b) for illustration purposes. The NLU is trained on the queries (intents) related to mining software repositories and is trained

to extract repository entities from users' questions, such as a *JiraTicket* (e.g., HHH-8593). In this example, after the costumary greeting from the chatbot, the user asks the chatbot "Fix ticket 8983?" which is forwarded to the NLU where it classifies the user's question as having a *GetMaintainers* intent with a confidence score of 0.06. The low confidence score (lower than a predetermined threshold) triggers the fallback intent, thus, the chatbot asks the user to rephrase the question in a more understandable way (i.e., "Sorry, I did not understand your question, could you please rephrase the question?"). After the user rephrases the question "Which commit fixed the bug ticket 8983?", the NLU extracts the entity 'ticket 8983' of type *JiraTicket* and classifies the intent of the query as *GetFixingCommits* with a confidence score of 0.85. Finally, the chatbot performs the necessary action, querying the database to answer the posed question ("The bug ticket 8983 was fixed by the commit with hash 26f55f9baa8f4f34").

5.3 Case Study Setup

Since the main goal of this chapter is to evaluate the performance of different NLUs using SE tasks, we need to select the candidate NLUs that we want to examine and the SE tasks' data corpus to train and test those NLUs. In this section, we detail our selection of the NLUs, SE tasks used in the evaluation, and our experiment design.

5.3.1 Evaluated NLUs

There exists several widely-used NLUs that are easily integrated with third-party applications. To make our study comprehensive, we choose to examine the performance of four NLUs, namely IBM Watson, Dialogflow, Rasa, and LUIS. We select these NLUs since they are popular and widely used by both researchers and practitioners Munoz et al. (2018); Toxtli et al. (2018), and have been studied by prior NLU comparison work in other domains D. Braun, Hernandez-Mendez, et al. (2017); Gregori (2017); Koetter et al. (2018). Moreover, all selected NLUs can be trained by importing the data through their user interface or API calls, which facilitates the training process. In the following, we provide a description of those NLUs.

- **Watson Conversation (IBM Watson):** An NLU provided by IBM IBM (2019a). IBM Watson has prebuilt models for different domains (e.g. banking) and a visual dialog editor to simplify building the dialog by non-programmers.

- **Dialogflow:** An NLU developed by Google Google (2020b). Dialogflow supports more than 20 spoken languages and can be integrated with many chatting platforms such as Slack Google (2020b).

- **Rasa:** The only open-source NLU in our study, owned by Rasa Technologies Rasa (2020b). Rasa allows developers to configure, deploy, and run the NLU on local servers. Thus, increasing the processing speed by saving the network time compared to cloud-based platforms. In our evaluation, we use Rasa-nlu v0.14, which was the latest version when conducting the experiment.

- **Language Understanding Intelligent Service (LUIS):** An NLU cloud platform from Microsoft *LUIS (Language Understanding) Cognitive Services Microsoft Azure* (2019). LUIS has several prebuilt domains such as music and weather, and supports five programming languages: C#, Go, Java, Node.js, and Python.

5.3.2 SE Tasks and Data Corpora

To evaluate the performance of the NLUs in the Repository and Stack Overflow tasks, we select two representative data corpora, one for each task 1) *Repository corpus* Abdellatif et al. (2019b) used for the Repository task and includes questions posed to a chatbot by practitioners looking for information related to their projects' software repositories 2) *Stack Overflow corpus* Ye et al. (2016) used for the Stack Overflow task and contains a set of posts from Stack Overflow discussion threads. Our selection of these two tasks was motivated by two main reasons: Firstly, both tasks reflect realistic situations, as in both, developers are asking questions about issues they face or to get more information about their projects (e.g., fixing commit for a bug). In fact, the Repository task also covers questions that are commonly asked by project managers to grasp the state of the project repository. Hence, both tasks make our results more generalizable to the chatbots practitioners in the SE domain. Secondly, using two tasks in our evaluation gives us better insights on how each NLU

Table 5.1: Intents distribution in the Repository task.

Intent	Definition	Train (%)	Test (%)	Total (%)
BuggyCommitsByDate	Present the buggy commit(s) which happened during a specific time period.	66 (23.8)	13 (10.7)	79 (19.6)
BuggyCommit	Identify the bugs that are introduced because of certain commits.	52 (18.8)	9 (7.4)	61 (15.3)
BuggyFiles	Determine the most buggy files in the repository to refactor them.	37 (13.4)	13 (10.7)	50 (12.6)
FixCommit	Identify the commit(s) which fix a specific bug.	31 (11.2)	11 (9.0)	42 (10.6)
BuggyFixCommits	Identify the fixing commits that introduce bugs at a particular time	32 (11.6)	7 (5.8)	39 (9.8)
CountCommitsByDates	Identify the number of commits that were pushed during a specific time period.	11 (3.9)	21 (17.4)	32 (8.0)
ExperiencedDevFixBugs	Identify the developer(s) who have experience in fixing bugs related to specific file.	15 (5.4)	14 (11.6)	29 (7.3)
OverloadedDev	Determine the overloaded developer(s) with the highest number of unresolved bugs.	15 (5.4)	9 (7.4)	24 (6.0)
FileCommits	View details about the changes that are occurred on on a file.	10 (3.6)	12 (10.0)	22 (5.5)
CommitsByDate	Present the commit information (e.g., commit message) at a specific time.	8 (2.9)	12 (10.0)	20 (5.0)

Table 5.2: Entities distribution in the Repository task.

Entity Type	Definition	Train	Test
FileName	Name of the file (e.g., Transaction.java).	35,007	26
JiraTicket	Ticket ID number (e.g., KAFKA-3612).	21,012	11
DateTime	Specific/period data (e.g., during July of 2019).	117	52
CommitHash	Hash of a certain commit.	15,303	10

performs in different sub-contexts of SE. The Stack Overflow task uses a corpus that has a diverse set of posts from the top 8 tags in Stack Overflow, the most popular Q&A website in the developers community Abdalkareem, Shihab, and Rilling (2017); Mamykina, Manoim, Mittal, Hripcsak, and Hartmann (2011). On the other hand, the Repository task contains project specific information (i.e., "who touched file x?") rather than general programming questions.

Repository Corpus. This corpus was originally used to train and test the MSRBot Abdellatif et al. (2019b), a chatbot tailored for answering users' questions on project repository data. This corpus contains a set of 398 queries, with 10 different intents, as shown in Table 5.1. Each intent contains a set of different questions with the same semantic, indicating the different ways a user could ask the same query. Those intents have questions related to code repository (e.g., "List me the changes done in ClassA.java" from the *FileCommits* intent), issue tracker (e.g., "Who has the most bug assignments?" from the *OverloadedDev* intent), or a combination of both code and issue tracker (e.g., "Which commits fixed HHH-10956?" a query from the *FixCommit* intent). The corpus includes explicitly defined training and test sets that are labelled by the developers of the MSRBot. The training set includes the questions used to train the chatbot, acquired and curated by three developers involved in the project, while the test set is composed of questions posed by 12 software developers to test the MSRBot on a specified set of 10 different tasks. We use the training and test set as defined by the MSRBot authors in this experiment.

Each entity in the Repository corpus contains a so-called list feature Dialogflow (2020a); IBM (2020a); Microsoft (2020a); Rasa (2020c), a list of equivalent synonyms used by the NLU to recognize entities. With the list feature, the NLUs are limited to extract the exact match of the entities, but can use the specified synonyms in the extraction process. For instance, the entity 'HHH-7325' of

Table 5.3: List of the Stack Overflow task entities.

Entity	Definition	Total
ProgLanguage	Types of programming languages (e.g., Java) Ye et al. (2016).	96
Framework	Tools/frameworks that developers use (e.g., Maven) Ye et al. (2016).	85
Standards	"Refers to data formats (e.g., JSON), design patterns (e.g.,Abstract Factory), protocols (e.g., HTTP), technology acronyms (e.g., Ajax)" Ye et al. (2016).	20
API	An API of a library (e.g., ArrayList) Ye et al. (2016).	67
Platform	Software/Hardware platforms (e.g., IOS) Ye et al. (2016).	13

type *JiraTicket* has a list of synonyms containing 'bug7325', 'issue7325', and 'HHH7325', hence, any of these words can be used by the NLU to recognize the entity 'HHH-7325'. The Repository corpus contains four entity types using the list feature as shown in Table 5.2, covering the main artifacts in a repository, such as files (*FileName*), commits (*CommitHash*), and tickets from the JIRA bug-tracker (*JiraTicket*).

Table 5.4: List of the Stack Overflow task intents.

Intent	Description	Total (%)
LookingForCodeSample	Looking for information related to implementation. This includes looking for code snippets, the functionality of a method, or information specific to the user's needs.	132 (61.3%)
UsingMethodImproperly	A method or a framework is being used improperly causing an unexpected or unwanted behaviour of the program in hand. This can be related to code bugs or to performance issues.	51 (23.7%)
LookingForBestPractice	Looking for the recommended (best) practice, approach or solution for a problem.	12 (5.6%)
FacingError	Facing an error or a failure in a program, mostly in the form of an error message or a build failure.	10 (4.7%)
PassingData	Passing data between different frameworks or method calls.	10 (4.7%)

Stack Overflow Corpus. Stack Overflow is a popular Q&A website among developers and plays an important role in the software development life cycle Abdalkareem et al. (2017). Given its importance, data from Stack Overflow has already been used in prior work to train chatbots B. Xu

et al. (2017a). The titles of the questions in Stack Overflow represent a request for information by software practitioners (e.g., "How to create an JS object from scratch using a HTML button?"). Recently, Ye et al. (2016) developed a machine learning approach to extract software entities (e.g., the programming language name) from Q&A websites like Stack Overflow, and manually labelled entities from 297 Stack Overflow posts as shown in Table 5.3. We use the same corpus Ye et al. (2016) and extract the title of each post with its labelled entities. Unlike the Repository corpus, entities in the Stack Overflow corpus do not have a list feature, (list of entity synonyms), and need to be predicted by the NLUs (i.e., prediction feature). In other words, we train the NLUs only on the entities included in the training set. This allows us to evaluate the NLUs' ability to extract entities that they have not been trained on before, which emulates real-life scenarios where it is difficult for the practitioners to train the NLUs on all SE entities.

While the original Stack Overflow corpus contains manually labelled entities, it lacks the intent behind the posed questions. Hence, we need to manually label the intents of the queries before we can use the corpus for the Stack Overflow task. To achieve that, the first author used thematic analysis Cruzes and Dyba (2011) to categorize those queries (titles) according to their intents. Thematic analysis V. Braun and Clarke (2006) is a technique to extract common themes within a corpus via inspection, which was done manually in our case. This method is frequently used in qualitative research to identify patterns within collected data and has been used in prior work in the SE domain Lenberg, Feldt, and Wallgren (2015); Munir, Wnuk, and Runeson (2016). Initially, the first author categorized the queries into 19 intents. Then, we merged the categories that have a small number of examples (less than 10) but have a very similar rationale. This is because some NLUs recommend a training set of at least 10 queries for each intent Rasa (2020a). For example, the queries of *FacingException* and *FacingError* intents are quite similar in their goal as developers are looking for a solution to fix the crash and error.

To validate the manually extracted intents, we asked two additional annotators - the second author and one other Ph.D. student - to independently label the queries using the extracted intents. For each question, we asked the annotators to evaluate whether the query has a clear intent or not using the multiple choice ('Yes', 'May be', and 'No'). If the annotators answer the previous question with 'Yes' or 'May be', then they classify the title using one of the defined intents shown in Table

5.4. After both annotators finished the labelling process, we merged the labelled queries into one set to be used in the NLUs evaluation. We then use the Cohen's Kappa coefficient to evaluate the level of agreement between the two authors Cohen (1960). The Cohen's Kappa coefficient is a well-known statistic that evaluates the inter-rater agreement level for categorical scales. The resulting coefficient is a scale that ranges between -1.0 and +1.0, where a negative value means poorer than chance agreement, zero indicates agreement by chance, and a positive value indicates better than chance agreement. We find that the level of agreement between the two annotators on the occurrence of intent (i.e., whether a title has an intent or not) is +0.74, and the agreement on the intents classification is +0.71. Both agreements are considered to be substantial inter-rater agreements Fleiss and Cohen (1973). All three annotators discussed the disagreements and voted to the best fitting intent. After the merge, we discarded queries with unclear intent (total of 82), such as "JConsole Web Application". The final set includes 215 queries Abdellatif et al. (2021d), Tables 5.3 and 5.4 show the number of entities and intents included in our evaluation, respectively.

Our manual classification led to the creation of 5 intents shown in Table 5.4 with their definitions. The intents in the Stack Overflow Corpus represent different types of questions posted on Stack Overflow. For example, the *FacingError* intent contains questions asking for a solution for an encountered error: "PHP mysqli query returns empty error message" StackOverflow (2019). Table 5.3 shows entities used in the Stack Overflow corpus that are very specific to the SE domain. For example, *ProgLanguage* entity type has a set of different programming languages such as Python and JavaScript. Such entities can be used by code-centric chatbots Chun-Ting Lin and Huang (2020); Wyrich and Bogner (2019), such as chatbots that extract files that call a particular method in the code Chun-Ting Lin and Huang (2020).

5.3.3 Performance Evaluation of NLUs

We use the corpora from the Repository and Stack Overflow tasks to train IBM Watson, Dialogflow, Rasa, and LUIS and evaluate their performance. As each task has specific characteristics, we detail in the following how we train and test the NLUs for each task.

To evaluate the NLUs on the Repository task, we use the same training set from the Repository corpus, which includes 10 intents with their queries and entities with their lists of synonyms. To set

up the NLUs, we configure the NLUs to use the list feature for all entities, that is, the NLU will not attempt to extract any entities that are not present in the training set. This is thematically in-line with the nature of the Repository task where a chatbot answers questions about software repositories. In this context, an entity that does not exist in the repository (e.g., wrong Jira ticket number) is not useful for the chatbot and cannot be used to extract any information for the user. Then, using the NLUs' API, we define the entity types that exist in the Repository corpus, namely 1) CommitHash 2) JiraTicket 3) FileName, and use a fourth built-in entity type (DateTime).

In contrast to the Repository task, there is no original split in the Stack Overflow corpus, as this corpus was not originally intended to be used as training sets for chatbots. In fact, we augmented the dataset by manually labeling the intents in all queries, as discussed in Section 5.3.2. Therefore, to evaluate the NLUs' performance on the Stack Overflow task, we use a stratified tenfold cross validation Han, Pei, and Kamber (2011). The cross validation randomly divides the corpus into ten equal folds where nine folds are used for training and one fold for testing. Since we used the stratified version, the method maintains a consistent distribution of intents across all folds. It is important to note that we use the same process to train and test all the NLUs, hence, the queries remain consistent across the NLUs in each run of the tenfold cross validation. Moreover, we do not use the list feature to train the NLUs on all entities in the Stack Overflow task. Instead, we train the NLUs to extract the entities using the prediction feature. This allows us to evaluate the NLUs' ability to extract entities that they have not been trained on before, which better emulates real-life scenarios where practitioners cannot expect to train their chatbots in all possible entities.

We train Dialogflow, LUIS, and Rasa using their APIs, and train IBM Watson using its user interface. After the training process, we send each query in the test set through all NLU APIs and analyze their response, which includes the classified intent, the intent's confidence score, and the extracted entities. This response is then compared against the oracle to evaluate the performance of the NLU. To enable the replicability of our study, we make the scripts used to evaluate the NLUs performance and datasets publicly available Abdellatif et al. (2021d).

To evaluate the performance of the NLUs in each task, we calculate the standard classification accuracy measures that have been used in similar work (e.g., D. Braun, Hernandez-Mendez, et al. (2017)) - precision, recall, and F1-measure. In our study, precision for a class (i.e., intent or entity)

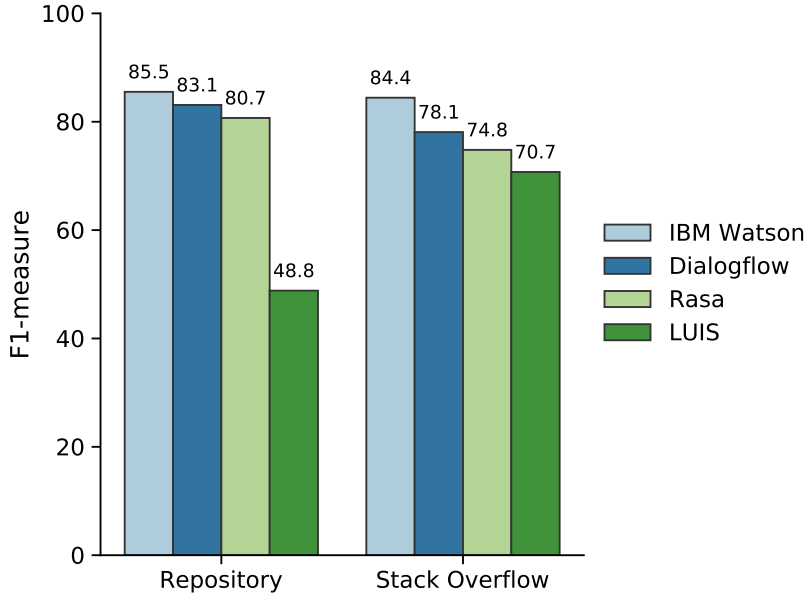


Figure 5.2: Intent classification performance as F1-measure of the four NLUs.

is the percentage of the correctly classified queries to the total number of classified queries for that class (i.e., Precision = $\frac{TP}{TP+FP}$). The recall for a class is the percentage of the correctly classified queries to the total number of queries for that class that exist in the oracle (i.e., Recall = $\frac{TP}{TP+FN}$). Finally, to present the overall performance of each NLU, we use the weighted F1-measure. In particular, we compute the F1-measure (i.e., F1-measure = $2 \times \frac{Precision \times Recall}{Precision+Recall}$) for each class and aggregate all classes F1-measure using weighted average, with the class' support as weights. This approach of calculating the F1-measure has been used in similar work Barash et al. (2019); Ilmania, Abdurrahman, Cahyawijaya, and Purwarianti (2018). While we evaluate all three metrics, we only showcase the weighted F1-measure in the chapter, whereas the precision, recall, and F1-measure for each intent and entity are presented in the Appendix.

5.4 Case Study Results

In this section, we present the comparison of the NLUs' performance in terms of intents classification, confidence score, and entity extraction on the Repository and Stack Overflow tasks.

Table 5.5: Intents' characteristics and classification performance as F1-measure of the four NLUs.

Task	Intent	Intent's characteristics			F1-measure				
		# Training Samples	% Queries w. Exclusive Words	Distinct Entity Types	IBM Watson	Dialogflow	Rasa	LUIS	Avg.
Repository	**BuggyFile**	37	92	-	96.0	96.0	**100.0**	83.3	93.8
	FixCommit	31	10	CommitHash	**100.0**	91.7	**100.0**	68.8	90.1
	BuggyCommit	52	7	JiraTicket	**94.7**	**94.7**	**94.7**	64.3	87.1
	OverloadedDev	15	88	-	**94.1**	80.0	88.9	58.1	80.3
	BuggyCommitByDate	66	39	-	72.7	69.6	80.0	**91.7**	78.5
	CountCommitsByDate	11	97	-	88.4	**91.3**	80.0	53.3	78.3
	BuggyFixCommit	32	74	-	**100.0**	85.7	63.2	48.0	74.2
	ExperiencedDevFixBugs	15	72	-	**92.3**	88.9	**92.3**	13.3	71.7
	CommitsByDate	8	70	-	70.0	50.0	**76.2**	0.0	49.1
	FileCommits	10	77	-	55.6	**80.0**	28.6	11.1	43.8
Stack Overflow	**LookingForCodeSample**	132	100	Platform	**90.9**	85.9	84.5	84.0	86.3
	LookingForBestPractice	12	92	-	81.3	81.3	78.0	**83.3**	81.0
	UsingMethodImproperly	51	100	-	**79.1**	66.5	64.4	57.5	66.9
	FacingError	10	100	-	**80.0**	60.0	33.3	10.0	45.8
	PassingData	10	70	-	36.7	**50.0**	36.7	0.0	30.9

5.4.1 Intents Classification

To evaluate the NLUs in intents classification, we train and test each of the NLUs using the corpus from each of the two SE tasks. When testing the NLUs, we only consider the top scoring intent as the classified intent for two reasons. First, to emulate real life use-cases where chatbots use the intent with the highest corresponding score, as it is the intent with the highest probability of being correct Google (2019b); IBM (2019b); Microsoft (2019c); Rasa (2019a). Second, to ensure that the evaluation of all NLUs is consistent, as Dialogflow only returns one intent with the corresponding confidence score in its response for a single query.

Results. Figure 5.2 shows the F1-measure for intent classification. The Figure presents the performance for each SE task, per NLU. The ranking is consistent across both tasks, showing that IBM Watson outperforms other NLUs, achieving an F1-measure of 85.5% in the Repository task and 84.4% in the Stack Overflow task. We also observe that for both SE tasks, LUIS comes last in intents classification.

To obtain a more detailed view of the NLUs' performance, we present the characteristics of all intents (columns 3-5) and the NLUs' F1-measure values (columns 6-9) in Table 5.5. We highlight in bold the best performing NLU for each intent. Our results show that IBM Watson is the best performing NLU for 8 intents, followed by Rasa, which performs best for 5 intents.

Moreover, we observe that some intents are more difficult to classify than others. For example, the *CommitsByDate*, *FileCommits*, *FacingError*, and *PassingData* intents are difficult to classify, with at least 2 of the four NLUs achieving an F1-measure $< 60\%$.

Given the performance of the NLUs on the different intents, we inspect the queries of each intent and find three main factors that contribute to the NLUs' varying performance. First, NLUs tend to generally perform well for intents that have a higher number of training samples (e.g., the *LookingForCodeSample* intent). Second, NLUs better classify intents that contain queries with exclusive words, that is, words that do not appear in any other intents (see the column % Queries w. Exclusive words in Table 5.5). For example, *FacingError* is the intent with the highest average score among all NLUs. This intent contains exclusive words, such as 'fail' and 'crash', in all its queries. Conversely, the NLUs misclassify intents that more frequently share words with other intents (i.e., those intents have less exclusive words). For example, *PassingData* intent has similar # Training Samples as *FacingError* intent but less % Queries w. Exclusive words. Third, intents that have distinct entity types in their queries (e.g., *Platform* entity type occurs only with *LookingForCodeSample* intent queries) are better classified by certain NLUs since such NLUs use the extracted entity types as input for the intent classification Dialogflow (2020c); IBM (2020b). This is clearly shown by the high F1-measure yielded by IBM Watson and Rasa in the classification of the intents *FixCommit* and *BuggyCommit*, which have specific entity types (i.e., *CommitHash* and *JiraTicket*).

> *NLUs rank similarly in both tasks in intents classification, with IBM Watson outperforming all other NLUs, followed by Dialogflow, Rasa, and LUIS. Aside from the training sample size, intents that contain exclusive words and distinct entity types are easier to identify by all NLUs.*

5.4.2 NLUs Confidence scores

As discussed earlier in Section 5.2, every intent classification performed by the NLU has an associated confidence score. The confidence score is used to determine how much trust one can put into the intent classification of the NLU. Typically, NLUs should provide high confidence scores for intents that are correctly classified. For example, if a query is asking "What is the number of

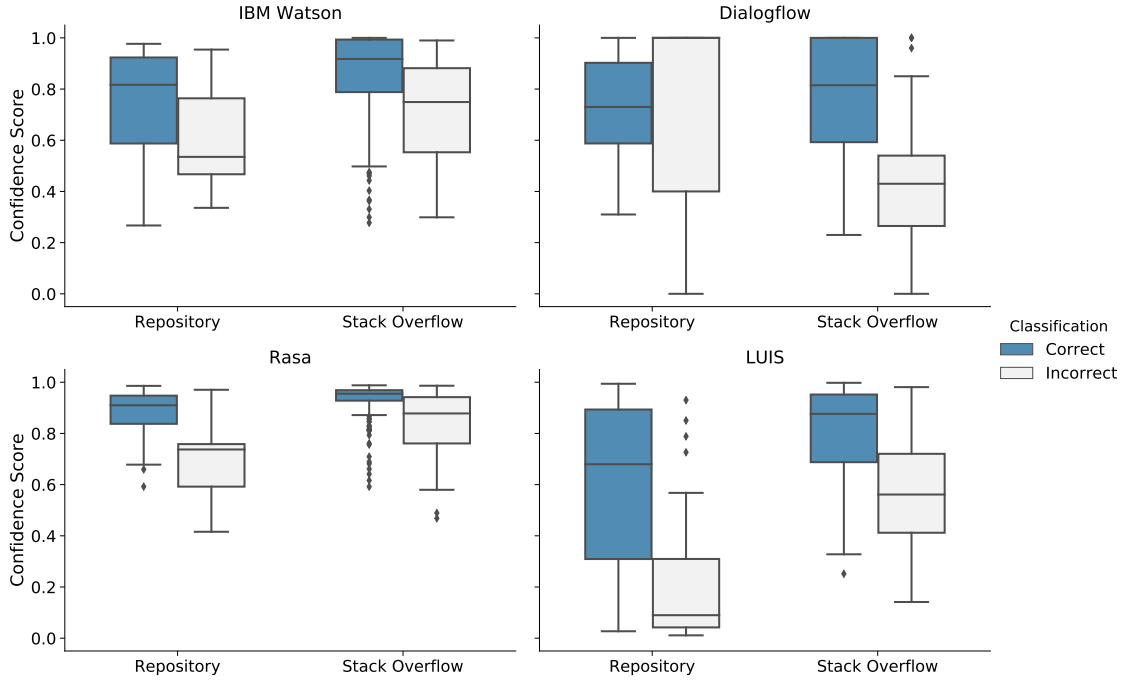


Figure 5.3: Confidence score distribution for all NLUs and tasks.

commits between 1-July-2020 to 1-August-2020?" and the NLU provides a high confidence score (e.g., 0.98) when attributing the query to the *CountCommitsByDate* intent, then the users of the NLU can trust these confidence scores. Also, the contrary is true, if an NLU provides high confidence scores to wrongly classified intents, then one would lose trust in the NLUs' produced confidence scores.

To compare the performance of the NLUs in terms of confidence scores, we queried each NLU with questions from the two tasks - Repository and Stack Overflow. For each query, we considered the highest confidence score to map to a specific intent. For example, for the query above ("What is the number of commits between 1-July-2020 to 1-August-2020?"), an NLU may return a confidence score of 0.98 for the intent *CountCommitsByDate*, and a confidence score of 0.80 for the intent *BuggyCommitsByDate*. If the correct intent is *CountCommitsByDate*, then we would consider this as a correctly classified instance and record the confidence score. In cases where the top confidence score does not indicate the correct intent, we record this an incorrect classification and record the intent confidence score.

We present the distributions of the confidence scores and compare these distributions for correctly and incorrectly classified intents. Ideally, NLUs should have high confidence scores for correctly classified intents (and vice versa). In addition, having clearly disjoint distributions of the confidence scores between the correctly and incorrectly extracted intents indicates that practitioners can rely on the confidence score of the NLU.

Results. Figure 5.3 shows the distributions of confidence scores returned by IBM Watson, Dialogflow, Rasa, and LUIS in the Repository and Stack Overflow tasks. From Figure 5.3, we observe that all NLUs return higher median confidence scores for the correctly classified intents compared to the incorrectly classified intents, for both tasks. The sole exception is Dialogflow, which has a higher median confidence score for incorrectly classified intents for the Repository task.

Among the evaluated NLUs, Rasa stands out as being the NLU with the highest corresponding confidence scores medians (higher than 0.91) in both the Repository and Stack Overflow tasks, for the correctly classified intents. Furthermore, Rasa has the most compact distribution of confidence scores among other NLUs and the least overlapping confidence scores between the correctly classified and misclassified intents. This means that when Rasa returns a high confidence score, it is highly likely to be correct.

To ensure that the difference in the confidence scores between the correctly and incorrectly classified intents across NLUs is statistically significant, we perform the non-parametric unpaired Mann-Whitney U test on each NLU results. We find that the differences are statistically significant (i.e., p-value < 0.05) in all cases and for both, the Repository and Stack Overflow tasks, except for the results of Dialogflow in the Repository task. Generally, our results show that developers can trust the confidence score yielded by NLUs to assess if the NLUs have correctly classified the intent, or the chatbot needs to trigger the fallback action, an action that is used when an NLU cannot determine a correct intent.

As mentioned earlier, the only outlier in our evaluation is the experiment from Dialogflow on the Repository task. Dialogflow returns a higher confidence score median for the misclassified intents (1.0) compared to the correctly classified intents (0.73), in the Repository task (see Figure 5.3). We searched the online documentation and forums to see if others have faced similar situations. We found multiple posts where developers raise issues with Dialogflow's high confidence scores for

incorrectly classified intents StackOverflow (2020); StackOverflow (2020). This indicates that our results are not an outlier and there might be an issue with Dialogflow that needs to be addressed. The developers reported that they rely on workarounds to overcome such issues, such as combining the confidence score with other measures (e.g., regular expression) StackOverflow (2020).

> *Overall, NLUs yield higher confidence scores for correctly classified intents. IBM Watson, Rasa, and LUIS provide higher median confidence scores, ranging between 0.68 - 0.96, for correctly classified intents.*

5.4.3 Entity Extraction

To correctly answers users' queries, chatbots need to also correctly extract the relevant entities. We consider the extracted entity to be correct only if its type and value exactly match the type and value of expected entity for that query in the oracle. The reason behind our criteria is that extracting entities that are only partially correct (i.e., have only correct values or correct types), causes the chatbot to incorrectly reply to the user's query. Since there can exist a varying number and types of entities in each query, we need a mechanism to ensure that our evaluation accounts for such variance. To mitigate this issue, we calculate the precision, recall, and weighted F1-measure based on the number of entities in each entity type, for the entity extraction results.

Results. Figure 5.4 presents the entity extraction results for the two tasks. To better interpret the results, we reiterate that extracting entities in the Repository and Stack Overflow tasks require different NLU features. For the Repository task, we configure the NLUs to extract entities using the list feature (i.e., list of synonyms) and entities are extracted as an exact match. In the Stack Overflow task, however, we configure the NLUs to use the prediction feature as discussed in Section 5.3.2, that is, requesting the NLUs to predict the correct entity. Given their differences, we discuss the results of each task separately, first describing the results for the Repository task and then the Stack Overflow task.

Repository. According to Figure 5.4, LUIS is the best performing NLU with an average F1-measure of 93.7%, followed by Rasa (90.3%). Both IBM Watson and Dialogflow perform similarly (70.7%) when extracting entities in the Repository task.

To better understand the factors influencing the performance, we examine the F1-measure of the NLUs in light of the different entity types. Table 5.6 shows the NLU's performance per entity type. We highlight, in bold, the best performing NLU for each entity type. Our results confirm that LUIS is the best performing NLU when extracting the *CommitHash*, *JiraTicket*, and *FileName* entity types from the Repository task. IBM Watson also performs well in the *CommitHash* and *JiraTicket* entity types, but it falls short when extracting *FileName* entities (F1-measure of 15.9%). This is due to IBM Watson's inability to differentiate between the normal words (e.g., 'To' and 'A') and entities of type *FileName* (e.g., 'To.java' and 'A.java'). In other words, it extracts entities based on their exact match with the training set without considering the entities' context when using the list feature IBM (2020c). For *DateTime* entities, Rasa performs best in extracting all the dates correctly (F1-measure of 100%) from the given queries. Rasa uses a probabilistic context-free grammar through their pipeline (Duckling) to extract the dates from the posed queries Rasa (2019b). For other NLUs, we notice two reasons behind the incorrect extraction of the *DateTime* entities: 1) the misspelling of the date from the users (e.g., "what are the commits I submit on 27/5/2018 $\sim$ 31/5/2018") 2) vague date formats in queries (e.g., "Tell me about the commits from 05-27 to 05-31 in 2018").

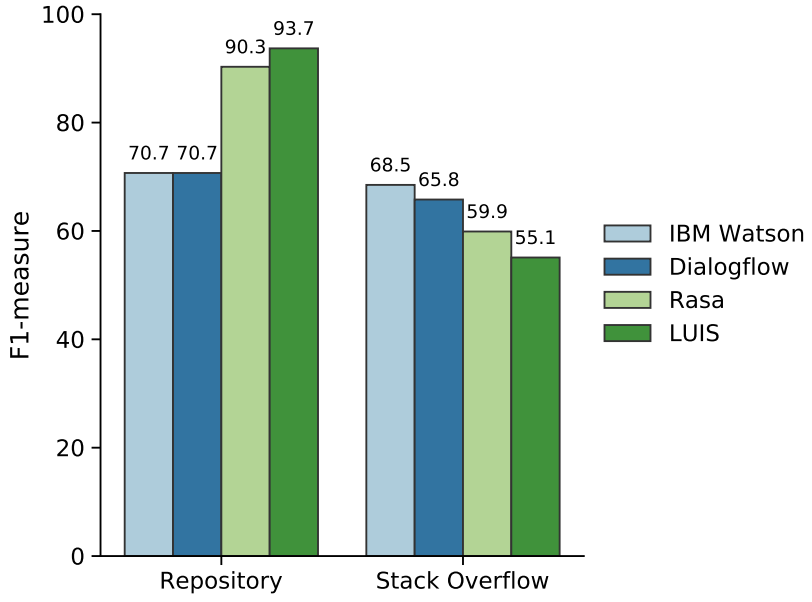


Figure 5.4: Entity extraction performance as avg. F1-measure of the four NLUs.

Stack Overflow. We observe in Figure 5.4 that IBM Watson yields the best results (F1-measure of

68.5%), followed by Dialogflow (65.8%), Rasa (59.9%), and LUIS (55.1%). Note that the performance of the NLUs on the Stack Overflow task is expectedly lower than the performance obtained in the Repository task, given that NLUs have to predict entities in a query.

Similar to the case of the Repository task, we also examine the performance of the NLUs when extracting the different entity types in the Stack Overflow task. Table 5.6 shows that the performance of the NLUs varies from one entity type to the other, and that no NLU outperforms the rest in extracting the majority of entity types. Table 5.6 shows that, in the Stack Overflow task, both IBM Watson and Dialogflow outperform other NLUs in extracting two different entity types. Furthermore, we observe that some entity types are more difficult to extract than others. In particular, entities of type *Framework*, *Standards*, and *API* are difficult to extract (i.e., the four NLUs achieve an F1-measure $< 60\%$). Upon closer investigation of the results, we find that more than 60% of the most difficult entities appear only once in the task, that is, they are unique entities. For example, the entity 'DOM' of type *Standards* in the query "How to tell if an element is before another element in DOM" is a unique entity as it occurs only once in the Stack Overflow task. This entity was not extracted by any of the evaluated NLUs. Consequently, NLUs tend to perform well when extracting entities which appear frequently in the training set (i.e., not unique).

> *Overall, NLUs perform differently in the two evaluated tasks, with LUIS and Rasa outperforming others when using the list feature (Repository task) for entity extraction, while IBM Watson and Dialogflow perform better when the entities need to be predicted (Stack Overflow task).*

5.4.4 Concluding Remarks

In the previous three sections, we compare the NLUs' performance in terms of three aspects (intents classification, confidence scores, and entity extraction). We find that the performance of each NLU can vary from one aspect to another, and between the two tasks. In other words, there is no NLU that outperforms all others in every aspect. Hence, in this section, we set up to rank the NLUs on their overall performance (considering the results from all aspects studied in the previous sections) in order to find the best NLU for chatbots in the SE domain. Having an overall measure is

challenging since we have different tasks (Repository and Stack Overflow) and are using different entity extraction features (i.e., list vs. prediction features). Therefore, we use an approach that has been used in prior work E. Shihab, Jiang, Adams, Hassan, and Bowerman (2011) to rank techniques evaluated against different datasets. In particular, we use F1-measures of each task for intents classification (Section 5.4.1) and entity extraction (Section 5.4.3) aspects. For the confidence score aspect, we rank the NLUs using the median confidence scores for the correctly classified intents of each task as shown in Section 5.4.2. To rank the NLUs, we compute the NLUs' average rank using their ranks in all aspects and tasks. The NLU with the lowest average rank is the best performing NLU.

Table 5.6: Entity extraction performance as F1-measure per entity of the four NLUs.

Task	Entity Type	F1- measure				
		IBM Watson	Dialogflow	Rasa	LUIS	Avg.
Repository	**CommitHash**	**100.0**	**100.0**	88.9	**100.0**	97.2
	JiraTicket	**100.0**	91.7	90.0	**100.0**	95.4
	DateTime	86.2	56.3	**100.0**	98.1	85.2
	FileName	15.9	79.2	71.4	**80.0**	61.6
Stack Overflow	**ProgLanguage**	92.0	**93.7**	91.4	86.8	91.0
	Platform	67.4	55.9	**75.1**	43.6	60.5
	Framework	**65.4**	56.0	56.1	54.4	58.0
	Standards	54.1	**56.9**	14.9	17.5	35.9
	API	**43.3**	42.8	29.9	23.5	34.9

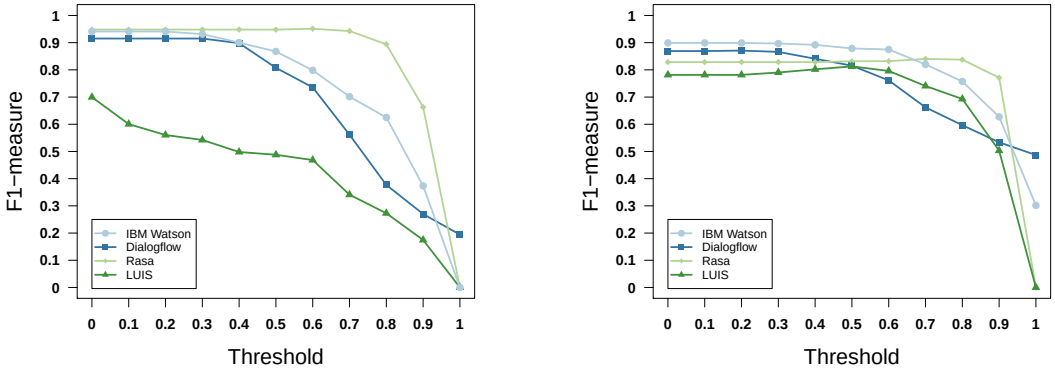
Table 5.7: NLUs' overall performance ranking.

NLU	Ranking in Repository Task			Ranking in Stack Overflow Task			Avg. Rank
	Intents Classification	Confidence Score	Entity Extraction	Intents Classification	Confidence Score	Entity Extraction	
IBM Watson	1	2	3*	1	2	1	1.7
Rasa	3	1	2	3	1	3	2.2
Dialogflow	2	3	3*	2	4	2	2.7
LUIS	4	4	1	4	3	4	3.3

* Same rank

Table 5.7 presents the NLUs' ranks in the different aspects on both tasks and their overall ranks. We find that IBM Watson is the best performing NLU when considering all aspects in both tasks, followed by Rasa, Dialogflow, and LUIS. That said, chatbot practitioners need to consider the most

important aspect (i.e., intents classification or entity extraction) and tasks performed by their chatbot when selecting the NLUs they want to use.



(a) Ranking in Repository Task

(b) Stack Overflow Task

Figure 5.5: Analysis of theshold sensitivity in terms of F1-measure of the NLUs in the Repository and Stack Overflow tasks.

5.5 Discussion

In this section, we dive into the evaluation results to gain more insights on the NLUs' confidence score sensibility as well as quantifying their abilities to extract unique entities. Finally, we provide a set of actionable recommendations to chatbot developers and researchers to achieve better intents classification and entity extraction results.

5.5.1 Examining the Impact of the Confidence Score Threshold on NLU Performance

One important impacting factor of the results from these NLUs is the confidence score threshold, i.e., at what confidence score value would an NLU be confident in classifying the returned intent as correct. In some cases, NLUs return high confidence scores for the incorrectly classified intents as shown in Section 5.4.2. Hence, defining the appropriate threshold to accept the classified intent or trigger the fallback action is still a challenge for chatbot developers. In essence, developers have to arbitrarily determine which confidence score value they will use to determine a correct classification.

Given that the confidence score threshold impacts the NLUs' performance, an important question is, how does the confidence score threshold impact the performance?

To maintain consistency with the results presented in Section 5.4, we use the same experimental settings. We vary the confidence score used by each NLU to determine the correct intent and plot its F1-measure performance.

Figure 5.5 shows the NLUs' F1-measures at varying confidence score thresholds. We perform this analysis for both tasks. For the Repository task, our results show that Rasa is the most robust, achieving consistently high performance for threshold values between 0-0.7. On the other hand, IBM Watson and Dialogflow start to see a reduction in performance for threshold values higher than 0.3. LUIS seems to generally follow a downward trend with higher confidence score thresholds. In the case of the Stack Overflow task, we again observe stable performance by Rasa for confidence score thresholds between 0-0.8, while IBM Watson, Dialogflow, and LUIS decrease in performance as the confidence score threshold increases.

From the results obtained, we observe that 1) the NLU's performance varies based on the task at hand and 2) that using lower confidence score threshold values tends to achieve better performance. This is because the selected threshold changes the NLU's performance only when it surpasses some of the confidence scores returned by that NLU. For example, Rasa has a high confidence score median (0.91), and hence, it will only be affected when the threshold increases significantly (threshold > 0.7). That said, one needs to be careful since using a low confidence score may also lead to the chatbot providing the wrong answer to the question being asked. Overall, we recommend that developers should investigate and carefully choose a proper confidence score threshold since such thresholds can have a significant impact on the chatbot's performance.

5.5.2 Unique Entities

The performance of the NLUs in extracting entities from the Stack Overflow task was affected by unique entities, as shown in Section 5.4.3. As the name suggests, unique entities appear just once in the dataset for the Stack Overflow task; thus, the NLUs have to predict their occurrences without prior training. It is important to note that there are no unique entities when evaluating the NLUs using the list feature because the NLUs have been trained on all entities that exist in the Repository

Table 5.8: Distribution of unique entities by entity type in the Stack Overflow task.

Entity Type	Unique Entities (%)
ProgLanguage	1 (1.3)
Framework	36 (48)
Standard	5 (6.7)
API	32 (42.7)
Platform	1 (1.3)

task. To better understand the NLUs' ability to extract unique entities, we investigate the results from the Stack Overflow task, examining the NLU performance on queries containing only unique entities. Hence, queries containing any non-unique entity were excluded from this investigation. We find 58 queries that fit our criteria, and they include a total of 75 unique entities that are distributed, as shown in Table 5.8. Similarly to the evaluation conducted in Section 5.4.3, we calculate the precision, recall, and weighted F1-measure of the NLUs when extracting unique entities.

Table 5.9 presents the NLUs' performance in extracting unique entities. We find that IBM Watson outperforms other NLUs with an F1-measure of 31.6% in extracting unique entities. We examine the results of extracting unique entities and find two factors that impact the NLUs' performance. First, the NLUs depend on the entities syntax similarity to recognize the entities in the queries of the testing set. For example, two NLUs (IBM Watson and LUIS) incorrectly extracted 'Receiving 0.0' as a *Framework* entity from the query "Receiving 0.0 when multiplying two doubles in an array" because 'Receiving 0.0' is syntactically similar to some other *Framework* entities such as 'Spring 4.0.2' and 'CodeIgniter 2.0.3'. Second, we find that the NLUs extract one of the words in the multi-worded entity (e.g., 'SAP crystal reports' is a single entity of type *Framework*) as a separate entity on its own, given that the NLUs are trained on such separate entities. For example, the multi-worded entity 'Python NameError' with an *API* type is extracted by all NLUs as the entity 'Python' of type *ProgLanguage*. In fact, the multi-worded entities in the Stack Overflow task are all unique entities, except for two entities (i.e., 'Internet Explorer' and 'WebSphere 8.5.5'). Hence, these results motivate the need for tools/techniques from the research community to extract the unique entities. Also, we encourage the chatbot developers to give special attention to unique entities by training the NLU on more examples by including those entities.

Table 5.9: Precision, Recall, and F1-measure of extracting unique entities in the Stack Overflow task.

NLU	Precision	Recall	F1-measure
IBM Watson	51.4%	28%	31.6%
Dialogflow	0%	0%	0%
Rasa	12%	1.3%	2.1%
LUIS	25.3%	5.3%	7.7%

5.5.3 Recommendations

Table 5.10: Recommendations for fine-tuning the NLUs when developing Chatbots.

Problem	Recommendation
Low accuracy on intents classification	R1. Train NLUs with multiple queries per intent R2. Merge intents with similar words and disambiguate with a follow-up action R3. Combine extra factors (e.g. regex, entity type) to aid to the confidence score
Low accuracy on entity extraction	R4. Use different entity features (list feature vs prediction feature) according to the entity type R5. When using entity prediction, focus on including entities in different query positions

Based on our findings and experience in conducting this study, we provide a set of actionable recommendations to help chatbot practitioners to improve the performance of the used NLU. Table 5.10 summarizes our recommendations to improve the NLUs' performance in intents classifications and entity extraction. While our results are based on SE tasks, some of the guidelines can be used to improve the NLU performance regardless of the domain. We discuss the recommendations in details in the following.

R1. Train NLUs with multiple queries per intent. Our results show that the NLUs perform better when classifying queries with intents that have more training examples. While more data is typically better in any machine-learning task, the focus on crafting a good training set needs to be put in the diversity of queries per intent. In fact, some NLUs recommend that each intent has 10 training examples or more Rasa (2020a) that represent different ways of querying that intent (e.g., having different sentence lengths Rasa (2019c)). NLUs like Dialogflow and LUIS provide an interactive

GUI Dialogflow (2020b) to allow the chatbot developers to edit and add training examples using the questions users pose to the chatbot. Chatbot developers should leverage this feature, especially at the early stages of the chatbot development, as a part of the debugging process of the NLU and as a way of fine-tuning the initial training set of the chatbot. We plan (and encourage others) to explore semi-supervised learning Zhu and Goldberg (2009) and weak supervision Z.-H. Zhou (2017) approaches to automate the NLU's retraining process.

R2. Merge intents with similar words and disambiguate with a follow-up action. The results show that NLUs better classify intents that contain exclusive words and distinct entity types, not shared with other intents. Hence, it is worth exploring the possibility of merging similar intents that share many common words and entity types into a single intent, as the NLUs can misclassify these intents. The merge of intents can be done in two ways: 1) during training, combining queries of similar intents into one intent, or 2) after the intents classification as a chatbot post-processing phase. Option 1 is only recommended when frequently misclassified intents have fewer training examples, as the merging can help boost the initial training dataset. Otherwise, option 2 is a more generally applicable solution as it does not introduce any noise by modifying the training dataset. Once the NLU classifies the merged intent, chatbot developers can employ some strategies to disambiguate the merged intent. The disambiguation can be done using entity extraction, regular expressions, or even relying on follow-up chatbot questions to help extract the target intent. For example, if there is a chatbot that does code refactoring, then the intents 'refactor class' and 'refactor method' have very similar training examples. In this case, the chatbot developers can merge both intents into one intent (e.g., 'refactor'). Then, to identify the target intent (class or method), the chatbot could ask the user about the type of refactoring to perform.

R3. Combine multiple factors to aid the confidence score. While most of the NLUs return high confidence scores when correctly classifying intents, as we discussed in Section 5.4.3, we recommend the chatbot developers to use other measures to ensure the intent was classified correctly. For example, chatbot developers can combine the confidence scores with regular expressions to check if specific keywords appear in the question. Another example is an intent that has a distinct entity

type where developers can check whether an entity of that type exists in the query and use that information to make sure that the intent was correctly classified (e.g., only queries of '*BuggyCommit*' intent have '*JiraTicket*' entities).

R4. Use different entity features according to the entity type. Developers should resort to different entity features (either list feature or prediction feature) together in the same chatbot, depending on the entities of their domain. Entities that are enumerable (e.g., months of the year) and have known finite values are better identified with the features containing all known synonyms. We observe this when evaluating the entity extraction performance in the Repository task, where NLUs extracted entities more accurately. However, in most cases, an entity could be expressed in various (and unknown) ways and developers need to resort to the prediction feature (e.g., *Framework* entities).

R5. When using the entity prediction feature, focus on including entities in different positions within the queries. When it comes to extracting entities using the prediction feature, developers need to diversify their queries so that they include entities in different positions (i.e., beginning, middle, and end of the query) in order to improve the NLU's ability to extract the entities in different contexts Microsoft (2019a); Rasa (2019d). Also, developers should expose the NLU with different variants of the entity (e.g., 'issues' and 'bugs') Microsoft (2019a). Finally, we believe that there is a need for a benchmark that contains SE terms (i.e., SE thesaurus) and their variations (e.g., 'commit' and 'change') as this helps the developers of SE chatbots to train the NLUs on SE entities and their synonyms.

5.6 Threats to validity

In this section, we discuss the threats to internal, construct, replicability, verifiability, conclusion, and external validity of our study.

Internal Validity: Concerns confounding factors that could have influenced our results. The Stack

Overflow tasks used to evaluate the NLUs' performance were manually labelled which might introduce human bias. To mitigate this threat, we had multiple annotators to label the tasks and used the discussion and voting to resolve all disagreements between the annotators. Another threat to internal validity is that we use the default configurations for all NLUs in our study, which could impact their performance. For example, we use the default NLU confidence score threshold to present our results in Section 5.4, but mitigate this threat by studying the impact of confidence score threshold on the NLUs' performance in Section 5.5.1. Other parameters could be tuned to enhance the NLUs' performance (spell-correction and training validation Google (2020)). We purposely decided to evaluate the NLUs under their default configurations, to evaluate the performance of an average user would encounter when deploying the NLUs in a chatbot. Also, the only common configuration across all NLUs is the confidence score threshold. In other words, some NLUs have configurations that could be tuned, which are not available in the other NLUs or have been implicitly defined in the NLU (since their internal implementation is closed source). Hence, modifying these configurations in one NLU might lead to different results and conclusions.

Construct Validity: Considers the relationship between theory and observation, in case the measured variables do not measure the actual factors. To evaluate the NLUs' performance in the Repository task, we use the MSRBot corpus that was created to evaluate the MSRBot. The MSRBot dataset might have some limitations, such as having questions (intents) that might be less popular than others in real settings. However, we argue that the questions that MSRBot supports were derived via a semi-structured process that collected the most common questions asked by software practitioners from previous studies Begel and Zimmermann (2014a); Fritz and Murphy (2010); Sharma et al. (2017). On the other hand, the MSRBot evaluation participants were free to word their questions to the chatbot. Finally, the list of questions used to train the MSRBot was not revealed to the participants.

Replicability Validity: Concerns the possibility of replicating the study Epskamp (2019a). In our study, we used three of the most popular NLUs that are closed source (except for Rasa). The NLU internal implementation might change without any prior notice to the users. Therefore, replicating the study might lead to different results. We mitigate this issue by providing the scripts and datasets

used in each task and the version used for Rasa (open-source). We encourage the scientific community to replicate the study through our replication package Abdellatif et al. (2021d) after a certain time (e.g., 6 months) to examine if there is a change in the NLUs performance. We believe that our study presents a snapshot of the NLUs' performance comparison in the SE domain. Moreover, our results serve as a starting point for the chatbot practitioners when selecting the NLUs to use.

Verifiability Validity: Concerns the verifiability of the study results Brundage et al. (2020). In our study, we compared the performance of different NLUs using two common SE tasks (i.e., Repository and Stack Overflow tasks). Using different NLUs or different tasks might lead to different conclusions. To mitigate this threat, we selected NLUs that are used in prior work D. Braun, Hernandez-Mendez, et al. (2017); Gregori (2017); Koetter et al. (2018) as a first step to benchmark the NLUs in the SE domain. Also, we described our case study setup in Section 5.3, studied each task characteristics and detailed our analysis and results in Section 5.4. Finally, we shared the dataset used to train/test NLUs, NLUs' responses Abdellatif et al. (2021d), and the used scripts Abdellatif et al. (2021d) with the community to allow for further investigation and accelerate the future research in the field.

Conclusion Validity: Concerns the relationship between the treatment and the outcome. In Section 5.4.4, we compute the overall NLUs' performance in both tasks by ranking the NLUs on each task based on their F1-measure then compute their average ranks in all tasks. The NLU's performance differs based on its intended usage (i.e., intents classification and entity extraction) and task (i.e., Repository and Stack Overflow). The main goal behind this analysis is to find the NLU that could be the best to use in the initial implementation of the SE chatbots. Moreover, this analysis has been used in similar studies E. Shihab et al. (2011) to obtain an overall performance for the NLUs. Finally, our results show consistency in the NLUs' ranks in all usages (except for entity extraction) for both tasks.

External Validity: Concerns the generalization of our findings. While we use four of the most commonly used NLUs to evaluate their performance in the SE domain, there exist other NLUs which are not included in our study. Since our goal is to find the best performing NLU in the SE domain, in our study we only select the NLUs that have been widely used by researchers and practitioners and they can be trained using their API calls and/or user interface.

Our study may be impacted by the fact that we evaluate the NLUs using the Repository and Stack Overflow tasks; hence our results may not generalize to other tasks in the SE domain. However, we believe that they cover very common tasks in SE, which could be improved with chatbots. That said, we encourage other researchers to conduct more similar studies that consider other NLUs and more SE tasks.

5.7 Conclusion & Future Work

Software chatbots are becoming popular in the SE community due to their benefits in saving development time and resources. An NLU lies at the heart of each chatbot to enable the understanding of the user's input. Selecting the best NLU to use for a chatbot that operates in the SE domain is a challenging task. In this chapter, we evaluate the performance of four widely-used NLUs, namely IBM Watson, Google Dialogflow, Rasa, and Microsoft LUIS. We assess the NLUs' performance in intents classification, confidence score, and entity extraction using two different tasks designed from a Repository and Stack Overflow contexts. When considering all three aspects (intents classification, confidence scores, and entity extraction), we find that IBM Watson is the best performing NLU for the studied SE tasks. For each individual aspect, in intents classification, IBM Watson outperforms other NLUs for both tasks. On the other hand, when it comes to confidence scores, all NLUs (except for Dialogflow) return high confidence scores for the correctly classified intents. Also, we find that Rasa is the most trustworthy NLU in terms of confidence score. Finally, LUIS and IBM Watson achieve the best results in extracting entities from the Repository and Stack Overflow tasks, respectively. Moreover, our results shed light on the characteristics that affect the NLUs' performance in intents classification (e.g., # Training Samples) and entity extraction (e.g., unique entities). Therefore, we encourage researchers to develop techniques and methods that enhance the NLUs' performance for tasks with different characteristics. We believe that our work guides chatbot practitioners to select the NLU that best fits the SE task performed by their chatbots.

Our study paves the way for future work in this area. First, our results show that NLUs tend to perform well when they are trained on more examples. Therefore, we plan to examine different dataset augmentation techniques to generate more training examples for the NLUs to enhance their

performance. Also, we believe that there is a need for more studies that compare different NLUs using more datasets to benchmark NLUs in the SE context. We contribute towards this effort by making our dataset publicly available Abdellatif et al. (2021d).

In this chapter, we find that NLUs perform better when it is trained on more queries. And the results of Chapters 3 & 4 show that training NLUs is a challenging task for chatbot developers because it requires dedicated time and resources to a create high-quality dataset to train the NLU. Therefore, in the following chapter, we evaluate an augmentation approach to facilitate the creation of a training dataset for chatbots in the SE domain.

Chapter 6

Improving the SE chatbot's accuracy

Software practitioners are increasingly adopting chatbots in their development tasks. This is driven by various chatbot usage-benefits in reducing the cost and speedingup the development process. Chatbots rely on the Natural Language Understanding (NLU) component to understand user's query. Prior to using the NLU, chatbot developers need to craft high-quality dataset for their training. NLUs yields to better performance when they are trained on more data. However, previous research shows that creating training dataset for the software engineering chatbot is expensive in terms of resources and time.

Therefore, in this chapter, we evaluate an approach that combines the synonyms replacement and paraphrasing augmentation techniques to augment datasets for software engineering chatbots. We evaluate the performance of the combined approach on three datasets from the software engineering domain. The results show that the combined approach does not improve the NLU's performance. Moreover, we find that using the approach or human augmented queries has a negligible to small effect on the NLU's confidence in its classification. Our results should alert the chatbot community on the limitations of current augmentation approaches when applied on software engineering domain.

6.1 Introduction

Chatbots have proven themselves to be a game changer in the customer service field and it is expanding to new areas Storey and Zagalsky (2016d). The wide-spread adoption of chatbots is due to their benefits in saving time, cost, and effort. The increased popularity and proven benefits of chatbots are driving the software engineering (SE) practitioners to develop chatbots for the SE domain. For example, Chun-Ting Lin and Huang (2020) developed the MSABot, a chatbot that assists developers in building and managing micro-services projects (e.g., set microservices project parameters). Abdellatif, Badran, and Shihab (2020b) developed the MSRBot to answer questions related to software projects (e.g., "Who fixed bug 5?").

Chatbots leverage a Natural Language Understanding (NLU) component to understand users' queries (i.e., messages). In essence, NLUs use AI and natural language processing techniques to extract structured information (the intent/purpose of the user's query and related entities) from unstructured input text. The performance of the NLU is directly related to the quality and diversity of the dataset used in its training Abdellatif, Badran, Costa, and Shihab (2021c). Indeed, many NLUs recommend that the training set includes semantically similar but syntactically varied queries to train the NLU on the different ways users can ask about the same information Docs (2020); Tmbo (2017). For example, the queries "List the developer who resolved issue 5", "Who fixed bug 5?", and "Which developer fixed issue 5?" have the same semantic (identify the developer who fixed a specific bug) but different syntax.

Nowadays, an extensive variety of NLUs are available online that developers can use in their chatbot implementation, such as Rasa (August) and Google (2020a). Nonetheless, despite the easy accessibility of NLUs, crafting a high-quality training dataset is a costly and time-consuming task. This is because chatbot developers need to brainstorm a variety of training queries in order to familiarize the NLU with new terms and diverse sentence structures of possible user queries. Moreover, chatbot developers will consistently augment their training set as they develop more training queries. Indeed, prior work shows that the lack of training datasets for SE-based chatbots is a challenge for SE practitioners Abdellatif, Badran, and Shihab (2020b); Dominic et al. (2020a). For example, Dominic et al. (2020a) reported that the absence of training queries limits their chatbot perfomance.

Likewise, Abdellatif, Badran, and Shihab (2020b) stated that the MSRBot failed to correctly classify some user queries because of the scarcity of training data. Consequently, this hinders the SE practitioners ability to develop more efficient SE chatbots as the training set would need to be crafted and augmented manually.

When an initial training dataset is created, *Augmentation* represents the process of creating new unobserved data points van Dyk and Meng (2001). This process is especially important for machine learning applications where adding more training data can improve the model's performance. In fact, a number of studies have focused on evaluating different augmentation approaches to improve sentiment analysis Marivate and Sefara (2020), hate-speech detection Rizos, Hemker, and Schuller (2019), and medical text prediction Amin-Nejad, Ive, and Velupillai (2020). Moreover, researchers have attempted to develop augmentation approaches to improve the NLU's performance Dopierre, Gravier, and Logerais (2021); Malandrakis et al. (2019); Wei and Zou (2019). For example, Dopierre et al. (2021) proposed an approach to improve the intents classification for several domains, such as the banking domain. Malandrakis et al. (2019) used the sequence-to-sequence model and variational autoencoders to enhance the NLU's performance in the intents classification task. However, none of these studies examined the combination of previously-used augmentation approaches and tailored them to augment SE chatbot datasets.

To address this gap, we explore an approach that combines synonym replacement and paraphrasing techniques to augment datasets for SE-based chatbots. For simplicity, we refer to this combined approach as ChatMent in the rest of this paper. ChatMent takes the queries in the original training set[1] as an input and uses them to augment more queries as an output. In particular, the approach contains four phases, a **Preprocessing phase**, meant to preprocess the queries and extract information that helps in the augmentation process; an **Augmentation phase**, that introduces new terms/keywords and changes the structure (paraphrases) the input queries to create candidate queries; a **Selection phase**, which selects the most useful candidate queries to keep; and a **Post-processing phase**, meant to merge the selected candidates with the original training set.

To assess the impact of using ChatMent on the NLU's performance, we perform an empirical study using three software engineering datasets, namely 1) Repository: contains questions asked

[1]Refers to the training dataset crafted by the chatbot developers to train the NLU.

by developers about their software projects (e.g., "Which are the most buggy files in the project?"), 2) Ask Ubuntu: composed of questions that developers ask about the Ubuntu OS (e.g., "Is it recommended to upgrade to Ubuntu 15.04"), and 3) Stack Overflow: contains software development related questions (e.g., "How to pass Ajax object data to python file?") from the online discussion forum Stack Overflow. These datasets include a total of 767 queries that belong to 19 different intents. Using these datasets, our study examines the following research questions:

RQ1: Can ChatMent improve the NLU's performance? While ChatMent was able to augment the dataset with new queries, we find that using the ChatMent does not improve the NLU's performance. However, we observe a clear improvement in the NLU's performance when using the human-augmented queries. Nevertheless, the ChatMent-augmented queries contain new terms and different sentence structure compared to the queries in the original training set.

RQ2: Does ChatMent increase the NLU's confidence in its classification? The NLU is more confident in its intents classification when it is trained on more queries. However, we find that using ChatMent has a negligible effect on the NLU's returned confidence scores whereas using the human-augmented queries yields a small effect on the NLU's confidence scores in some instances.

To this end, this work makes the following contributions:

- To the best of our knowledge, this is the first work that evaluates using an augmentation approach for SE-based chatbots.
- We explore the impact of using ChatMent on the NLU's performance using three datasets from the SE domain and different use-cases that vary in the number of initial training queries.
- We provide a replication package containing the implementation of ChatMent as a prototype tool and our results Abdellatif, Badran, Costa, and Shihab (2021a) to facilitate the replication and accelerate future research in the area.
- We report lessons learned to guide future research in augmenting datasets for SE chatbots.

6.1.1 Organization of the Chapter

The rest of the chapter is organized as follows. Section 6.2 provides a background on the chatbots and explains related terminologies used in the paper. We detail ChatMent and its components in Section 6.3. Section 6.4 describes the case study setup used to evaluate the efficiency of ChatMent. We report the case study results in Section 6.5. Section 6.6 discusses our findings. Section 6.7 discusses lessons learned. Section 6.8 discusses the threats to validity, and section 6.9 concludes the paper.

6.2 Background

Chatbots are software bots that interact with users through chat C. Lebeuf, Zagalsky, Foucault, and Storey (2019b). This simple method of interaction is what gives chatbots their appeal and makes them a suitable conduit between users and services C. Lebeuf, Storey, and Zagalsky (2018a) (e.g., Customer Service). To facilitate this chat-like interaction, modern chatbots leverage the critical natural language understanding (NLU) component which extracts structured information from unstructured text (user's query). Typically, NLU components extract two key pieces of information from the user's message; the intent and entities. The **intent** represents the intention/goal behind the user's message, while **entities** are important keywords in the message. For example, when a user asks "*What are the fixing commits for bug 5391?*", the intent is to know which commits fixed a specific bug in the project ('FixingCommit' intent), whereas the bug number ('5391') is the entity. When interacting with a chatbot, users are free to express the same intent in different ways. For example, the queries "Show the fixing commits for issue 5391" and "What changes solved 5391?" both have the same intent ('FixingCommit') but different syntax.

It is critical for chatbots to have a robust NLU that extracts the intent from the users' queries correctly as it gives the chatbot an accurate assessment of the users' intentions, leading it to take the right course of action (send a reply or perform an task). In contrast, a poorly performing NLU that misclassifies the users' intent will lead the chatbot to reply incorrectly and/or perform a wrong action. Thus, the incorrect intent extraction (or poor NLU's performance) has a direct and negative impact on the satisfaction of the chatbot users, which has also been shown in previous studies Ask

et al. (2016); Lastra (2016); C. Lebeuf, Storey, and Zagalsky (2018b).

When the NLU extracts an intent, it also returns a confidence score corresponding to that intent. The *confidence score* shows how confident the NLU is in its intent classification, and it has a value that ranges between 0 (i.e., not confident) to 1 (i.e., fully confident). Chatbot developers use the confidence score to determine whether the chatbot has understood the user's query well enough (high confidence score) in which case the chatbot should perform an action. Otherwise, if the user's query is not clear enough (low confidence score), the chatbot should ask the user to clarify or rephrase the query in order for the chatbot to better understand the intent Abdellatif et al. (2021c).

At its core, the NLU component is a machine learning model which requires training queries that represent the different ways a user could phrase a query for each intent. Usually, chatbot developers brainstorm to come-up with these training queries at the early stages of the chatbot development cycle Abdellatif, Badran, and Shihab (2020b). Once the chatbot is deployed and users come on board, the users' queries can by curated and used to augment the training set of the NLU and improve its performance Microsoft (2021a); Rasa (2021). Nevertheless, the early step of building the initial training dataset poses a real challenge to chatbot developers Abdellatif, Costa, et al. (2020) and may require the help of domain-experts (e.g., medical field) in order obtain a reasonable starting point for the dataset. Moreover, adding more training queries to the initial training dataset can improve the NLU's performance as shown in prior work Abdellatif et al. (2021c).

To overcome this challenge and help SE chatbot developers build their training datasets, we evaluate the combination of synonyms replacement and paraphrasing techniques to automatically augment the training set using small set of initial training queries as input.

6.3 Approach

The ChatMent is based on the key idea that, by using a small initial set of training queries (called original training set), we can augment (build) new queries that retain the same semantics while having different terms/keywords and brand new sentence structures. We design ChatMent to be a multi-phased approach shown in Figure 6.1. More specifically, the approach is composed of four phases:

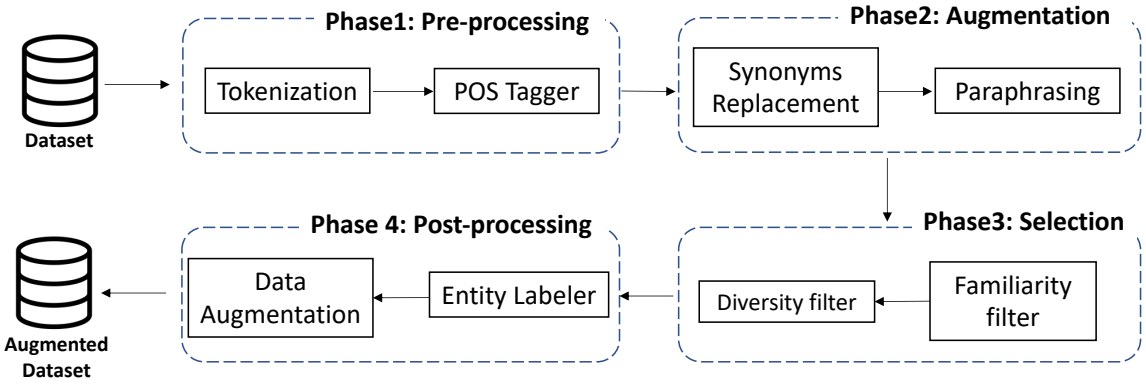


Figure 6.1: An overview of ChatMent framework.

(1) **Preprocessing phase.** In this phase, the original training set is preprocessed to tokenize the queries and identify the part-of-speech for each token. This information is then passed to the Augmentation phase.

(2) **Augmentation phase.** This is the core phase of ChatMent where all new queries (candidate queries) are generated. This phase introduces new keywords and rephrases the sentence structure of the queries from the original training set.

(3) **Selection phase.** Filters the candidate queries to keep only the queries with the highest potential of improving the NLU's performance.

(4) **Postprocessing phase.** This phase labels the entities in the candidate queries and merges them with the original training set to obtain the final (augmented) training set.

Next, we explain each phase and its components in details and we showcase a working example, presented in Figure 6.2, to demonstrate how each component works.

Preprocessing phase

In the first phase of ChatMent, we tokenize the queries in the original training set and extract the part-of-speech of each token which helps the Augmentation phase in generating candidate queries. Next we describe the steps composing this phase:

Tokenization Each training query is split into tokens using a pretrained model from the SpaCy library Spacy (2021). Spacy is a Python library for natural language processing and has been used

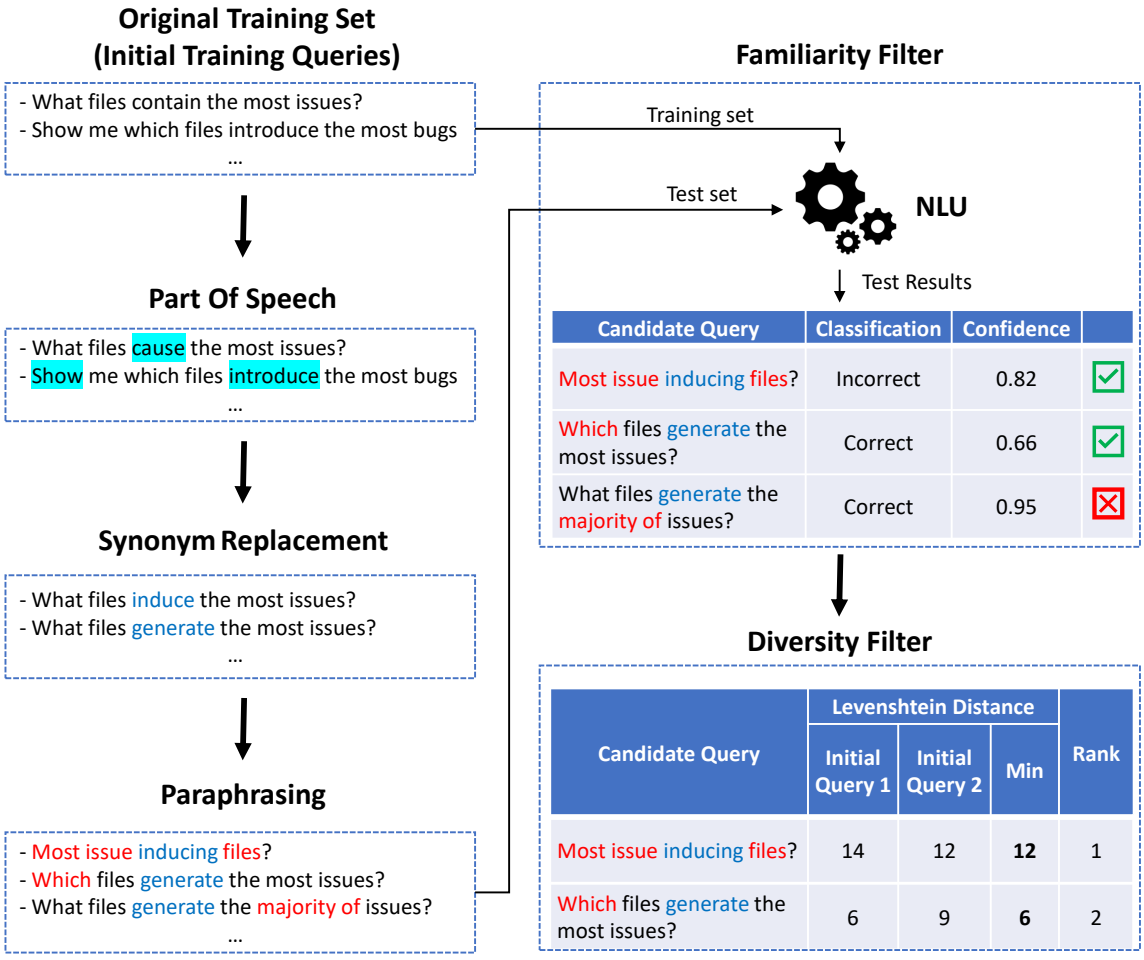


Figure 6.2: A working example of ChatMent.

in prior work Abdellatif, Costa, et al. (2020). Having the tokens helps the Augmentation phase to find and replace the synonyms for some tokens in the query.

Part-of-speech (POS) tagging. This component identifies the POS (e.g., verb, noun, adjective) for each token in the query. By extracting this information about the tokens, our Augmentation phase becomes resilient. For example, in case the original training set is already rich with noun synonyms, then the ChatMent user needs to diversify the dataset by having more synonyms of verbs. For this step, we leverage the POS tagger from the Spacy library. The working example in Figure 6.2 showcases the POS tagging component identifying and labelling the verb tokens (i.e., cause, show, and introduce) in the original training set.

Augmentation phase

The Augmentation phase is the heart of ChatMent, responsible for pumping new candidate queries that have different keywords and new sentence structures. This phase encompasses two steps 1) Synonyms replacement and 2) Paraphrasing. We detail each of these steps in this section.

Synonyms replacement. Given the output from the preprocessing phase, this component replaces certain tokens (e.g., verbs, nouns) with their synonyms to obtain new candidate queries. The goal here is to familiarize the NLU with more terms that might appear in the users' queries To obtain the list of synonyms for token, one could use any of the available thesauruses such as WordNet Miller (1995). However, since ChatMent targets SE chatbots, these thesauruses may lead to less-than-optimal results as they are not domain specific. For example, when looking for synonyms to the term 'bug', the WordNet thesauruses returns 'germ', 'microbe', and 'hemipteron'. This demonstrates the need for a more specific thesaurus that is well-suited for the specialized domain of SE. Therefore, we use a thesaurus specialized for the SE domain, which is a word2Vec model trained on Stack Overflow posts to capture the SE terms Efstathiou, Chatzilenas, and Spinellis (2018). With this SE-based thesaurus, we obtain the following synonyms for the the term 'bug': 'defect', 'bug/feature', and 'bugs'. Nevertheless, we encourage practitioners to use other thesauruses based on the context and domain of their chatbots.

In case a query has two or more tokens to be replaced, we create new candidate queries based on all the possible combinations of the replaceable tokens' synonyms. For example, if a query has two replaceable tokens and each token has three synonyms, we create six (2*3) new candidate queries. In the working example in Figure 6.2, the Synonym Replacement component replaces the verb token 'cause' from the query "What files cause the most issues?" with its synonyms 'induce' and 'generate', thus creating two new candidate queries (e.g., "What files induce the most issues").

Finally, in our study we configure the Synonym Replacement component to replace the verb tokens due to three main reasons. First, all the entities in our dataset are nouns, and replacing the entities will affect our ability to label them automatically later on. Moreover, the focus of our study is to improve intents classification task which requires that we change the query and not the entities in it. Second, most of the nouns in our datasets are proper nouns (e.g., 'Java', 'Ubuntu'), thus, they

do not actually have synonyms. Lastly, because of their abundance, replacing noun tokens requires a high computational cost as it generates a large number of candidate queries that will go through the rest of the components.

Paraphrasing. Introducing new terms in the candidate queries is not sufficient to improve the NLU's performance as chatbot users might use the same terms by phrase the question in a different way. For example, to identify the developer that fixed a certain bug, the user could ask "List the developers who fixed issue 5" or "Which developers worked on fixing issue 5?". In fact, this is why some NLUs recommend to include queries that have different sentence structure in the training set as it enhances the NLU's performance in intents classification Docs (2020); Tmbo (2017). Therefore, we diversify the sentence structure of candidate queries using a Paraphrasing component that restructures the queries while preserving their meaning (intent). This paraphrasing component takes as an input the candidate queries from the Synonym Replacement component.

To paraphrase queries, this component leverages the recent transformer based neural machine translation (seq2seq) model called BART Lewis et al. (2020). BART is a general language model proposed by Facebook AI and has been used by prior work for paraphrasing tasks Dopierre et al. (2021); Martin, Fan, de la Clergerie, Bordes, and Sagot (2021); West et al. (2021); S. Xu, Semnani, Campagna, and Lam (2020); J. Zhou, Gong, and Bhat (2020) as it achieves state-of-art performance in various NLP tasks (e.g., machine translation, summarization, and text generation) Lewis et al. (2020). BART is trained through corrupting the input example (e.g., delete one of its tokens) during the training stage and then predicting the correct form of the sentence corrupted sentence. We fine-tune BART to perform paraphrasing tasks (discussed in Section 6.4).

In the working example (figure 6.2), the Paraphrasing component takes the two candidate queries from the Synonym Replacement component as an input and outputs paraphrased queries (e.g., "Most issue inducing files?"). The final output of the Augmentation phase are new candidate queries that have both new terms and different sentence structures compared to the original training set.

Selection phase

Adding all candidate queries to the original training set might overfit the NLU, especially if the original training set is small. To mitigate this issue, we devise the Selection phase which filters the

candidate queries to keep only the queries that are useful for the NLU. We believe that augmenting candidate queries that are unfamiliar to the NLU (differ from the original training set) can improve the NLU's performance. Moreover, candidate queries that make the original training set more lexically diverse can potentially add more value to the NLU. Therefore, we devise the Familiarity filter and Diversity filter components to select those queries.

Familiarity filter. The main goal of this component is to filter out candidate queries that the NLU - trained on the original training set - is already familiar with. If the NLU correctly classifies the candidate query with a high confidence score, then the NLU is familiar with this candidate query. On the other hand, if a specific candidate query ends up being incorrectly classified or correctly classified but with a low confidence score, then the NLU is unfamiliar with that candidate query, which hints to its uniqueness compared to the original training set. Moreover, this query is likely to be a valuable addition to the training set of the NLU.

The best way to identify whether a candidate query is familiar to the NLU is to leverage the NLU itself to perform this task. In particular, we first train the NLU using the original training set and then test it using the candidate queries from the Augmentation phase. Then, we remove all candidate queries that are correctly classified with high confidence scores (higher than a set threshold). The remaining candidate queries are either incorrectly classified by the NLU or correctly classified but with a low confidence score (lower than the threshold).

The working example (Figure 6.2) shows the Familiarity filter where the NLU is trained on the original training set. To filter the candidate queries, the output of the Paraphrasing component is used as a test set for the NLU. In this example, the first query ("Most issue inducing files?") passes the filter as it was incorrectly classified. Although the second query ("Which files generate the most issues?") was classified correctly, it also passes the filter because the corresponding confidence score (0.66) is lower than the threshold (e.g., 0.7). The third query is discarded as it is correctly classified with a high confidence score. Finally, the first two queries that passed the filter are forwarded to the Diversity filter component.

Diversity filter. The goal of this filter is to select the candidate queries that are most syntactically different compared to the original training set. This helps to mitigate the issue of overfitting the NLU that may occur when the candidate queries do not increase the syntactical diversity of the original

training set.

As a mean to measure the diversity, this component computes the Levenshtein distance Levenshtein (1966) between the candidate queries and the original training set. More specifically, the Levenshtein distance calculates the number of edits (insertion or deletion of a character, or replacement of a character by another one) between two inputs (i.e., the candidate queries and original training set). The higher the Levenshtein distance, the more dissimilar the queries. This process is shown in the Diversity filter component shown in the working example (Figure 6.2). In the example, the Levenshtein distance between each of the candidate queries and original training set queries is computed. For instance, the Levenshtein distance between the candidate query "Most issue inducing files?" and initial query 1 from the original training set "What files contain the most issues?" is 14.

Then, we want to rank the candidate queries based on how diverse they are. So, we first obtain a single value for each candidate query by using the minimum Levenshtein distance between the candidate query and any initial query. In the working example, the minimum Levenshtein distance for the candidate query is shown in the 'Min' column. For example, the minimum Levenshtein distance for the candidate query "Most issue inducing files?" is 12. Next, we rank the candidate queries in a descending order using the minimum Levenshtein distance. This minimizes the chance of selecting a candidate query that is similar to any of the queries from the original training set. Finally, the top N ranking candidate queries in are kept by the Diversity filter while the rest is discarded (N is a number that can be configured by the user). In this study, we set $N = 1$ for the Diversity filter component. Therefore, the Diversity filter keeps only the top ranking candidate query, which is "Most issue inducing files?" in the working example.

Postprocessing phase

After obtaining the filtered candidate queries, its necessary to label the entities in these queries before merging them back with the original training set. Next we describe the components composing this phase:

Entity labeler. The candidate queries that are retained after the Selection phase do not contain any entity annotations. Such annotations are essential for some NLUs in the intents classification

step Abdellatif et al. (2021c). Hence, in this component, we label the entities in the candidate queries. To label the entities, we examined the Paraphrasing phase output (400 samples from different intents) and found that the entities remain the same or experience minor modifications only during that phase. For example, the FileName entity (e.g., 'ConsumerRecords') could be changed during our Augmentation phase (e.g., 'Consumer Records') due to using BART. The only exception here are the DateTime entities, where the a specific date (e.g., 21-09-2021) can be changed to 'last week'. Based on our observations, we define heuristics to label entities in the candidate queries. Therefore, the Entity labeler component reads all labeled entities in the training dataset and automatically labels the entities in the candidate queries using the defined heuristics.

Data merger. The output of any augmentation approach should be a training dataset that is ready for use. Therefore, this component is responsible for adding the candidate queries to their corresponding intents in the original training set. The chatbot practitioners can use this output to train the NLU used in their chatbots.

6.4 Case Study Setup

The goal of this study is to evaluate the impact of using ChatMent on the NLU's performance given a training dataset for SE-based chatbots. Thus, in this section, we detail our selection of the SE datasets used for the evaluation, NLU platform used for training and testing, and experiment design.

6.4.1 Datasets

To ensure that we evaluate ChatMent on a variety of SE datasets, we select three distinct SE-based datasets: Repository Abdellatif, Badran, and Shihab (2020b), Ask Ubuntu D. Braun, Hernandez Mendez, Matthes, and Langen (2017), and Stack Overflow Ye et al. (2016) datasets. Our dataset selection is based on three reasons. First, these datasets represent realistic questions that software practitioners ask about software projects and development. Second, they have adequate numbers of training and testing queries for each intent (ten or more queries per intent) to conduct proper evaluation. Finally, they are publicly available and have been used in previous studies Abdellatif et

Table 6.1: Intents distribution in the Repository task.

Dataset	Intent	Definition	Train	Test	Total
Repository	**BuggyCommitsByDate**	Present the buggy commit(s) which happened during a specific time period.	66	13	79
	BuggyCommit	Identify the bugs that are introduced because of certain commits.	52	9	61
	BuggyFiles	Determine the most buggy files in the repository to refactor them.	37	13	50
	FixCommit	Identify the commit(s) which fix a specific bug.	31	11	42
	BuggyFixCommits	Identify the fixing commits that introduce bugs at a particular time	32	7	39
	CountCommitsByDates	Identify the number of commits that were pushed during a specific time period.	11	21	32
	ExperiencedDevFixBugs	Identify the developer(s) who have experience in fixing bugs related to specific file.	15	14	29
	OverloadedDev	Determine the overloaded developer(s) with the highest number of unresolved bugs.	15	9	24
	FileCommits	View details about the changes that are occurred on on a file.	10	12	22
	CommitsByDate	Present the commit information (e.g., commit message) at a specific time.	8	12	20
Ask Ubuntu	**SoftwareRecommendation**	Looking for applications that perform specific task (e.g., extract images from PDF, video editing).	17	40	57
	MakeUpdate	Looking for information related to upgrading Ubuntu version to a newer version.	10	37	47
	ShutdownComputer	Related to shutdown the computer based on a condition (e.g., specified time) and fix shutdown issues in Ubuntu OS.	13	14	27
	SetupPrinter	Setup a printer and fix printer installation issues.	10	13	23
Stack Overflow	**LookingForCodeSample**	Looking for information related to implementation. This includes looking for code snippets, the functionality of a method, or information specific to the user's needs.	66	66	132
	UsingMethodImproperly	A method or a framework is being used improperly causing an unexpected or unwanted behaviour of the program in hand. This can be related to code bugs or to performance issues.	25	26	51

Dataset	Intent	Definition	Train	Test	Total
Stack Overflow	**LookingForBestPractice**	Looking for the recommended (best) practice, approach or solution for a problem.	6	6	12
	FacingError	Facing an error or a failure in a program, mostly in the form of an error message or a build failure.	5	5	10
	PassingData	Passing data between different frameworks or method calls.	5	5	10

al. (2021c); Larson et al. (2019); Shridhar et al. (2019); Shridhar, Jain, Agarwal, and Kleyko (2020). Next, we describe each dataset in details.

Repository: Contains questions asked by software practitioners about their software repositories to a chatbot, called MSRBot Abdellatif, Badran, and Shihab (2020b). Examples of queries in this dataset are "What are the commits that introduce bug HHH8492?" and "Which developers have the most bug assignments?". For this dataset, the MSRBot developers created the training set manually, and composed the test set from queries asked by the MSRBot users. Thus, the training and test sets in this dataset originate from a real-life use case of an SE-based chatbot in practice. Table 6.1 presents the intents in this dataset, their definitions, and the distribution of queries in the training and test sets corresponding to each intent. The Repository dataset contains 398 queries belonging to ten intents in total.

Ask Ubuntu: This dataset was constructed using the most popular posts from the Ubuntu Q&A community on Stack Exchange, one of the most popular online discussion forums D. Braun, Hernandez Mendez, et al. (2017). D. Braun, Hernandez Mendez, et al. (2017) selected the most popular questions with accepted answers from the Ubuntu community. These questions were then labeled by annotators recruited through Amazon Mechanical Turk (AMT). Examples from this dataset include "How to shutdown computer when users are logged on?" and "How to setup HP printer/scanner on Ubuntu?". This dataset contains 154 queries split into four intents as shown in Table 6.1. It is important to note that we discard the "Other" intent from the Ask Ubuntu dataset because it had only three queries, which is insufficient for our evaluation.

Stack Overflow: Contains software development questions posted on Stack Overflow Ye et al.

(2016). Stack Overflow is another popular Q&A website that developers use to help in their software development workflows. The questions in this dataset were initially collected by Ye et al. (2016) from the top six tags on Stack Overflow. Next, Abdellatif et al. (2021c) labeled the intents of these posts. In total, this dataset is composed of 215 queries and five different intents as shown in Table 6.1. Examples of questions in this dataset include "How to write an efficient hit counter for websites" and "How can I get Font X offset width in java2D?".

6.4.2 NLU

The goal of ChatMent is to augment a given training dataset to improve the NLU's performance. Hence, we need to select an NLU platform to train its model and perform our evaluation. For this study, we select Rasa, an open-source NLU platform developed by Rasa Technologies Rasa (August). Unlike third-party NLUs that operate on the cloud (e.g., Google Dialogflow), Rasa can be installed, configured, and run locally which consumes less resources. Moreover, Rasa has been used by prior work Abdellatif et al. (2021c); Abdellatif, Badran, and Shihab (2020b); Chun-Ting Lin and Huang (2020); Dominic et al. (2020b). In our implementation, we use Rasa version 2.5 as it was the latest stable version available at the time of our study.

6.4.3 BART tuning

As discussed in Section 6.3, the paraphrasing component uses BART to paraphrase the candidate queries outputted by the Synonyms replacement component. BART is trained on 160GB of documents (e.g. Wikipedia, news, stories) with a sentence reconstruction loss Lewis et al. (2020). To use BART, we need to fine-tune it to the specific task at hand which is to paraphrase queries in our case. Therefore, we use the following three datasets to fine-tune BART:

- **Quora Question Pairs:** Quora is a social Q&A website where users ask questions and other users answer them. The dataset consists of over 149,263 lines of potential duplicate pairs of questions obtained from Quora. For example, the "How to learn Java programming language?" question is duplicated with "How do I learn a computer language like java?".

- **Microsoft Research Paraphrase:** This dataset is composed of 3,749 pairs of sentences

extracted from the internet (e.g., news sources) and then annotated by humans to indicate whether each pair captures the same semantics.

- **Paraphrase Adversaries from Word Scrambling (PAWS-Labeled):** Y. Zhang, Baldridge, and He (2019) curated a dataset that is composed of pairs of sentences generated using word swapping and back-translation. Then, five annotators labeled these generated sentences to indicate whether they represent paraphrases of the original sentences. This dataset contains 25,368 pairs of paraphrased sentences.

These three datasets have been used in prior work Dopierre et al. (2021); West et al. (2021) which makes them a solid choice to fine-tune BART. For the specific BART implementation, we use the *BART-large* model which has 12 layers in the encoder and decoder, and more than 374 million of parameters. We train the BART model on a cloud with 6 core Intel E5-2683 v4 Broadwell, 64GB of RAM, and NVIDIA V100 Volta (32G HBM2 memory) GPU. We examined BART's output with different numbers of returned paraphrases (i.e., 3, 5, 7, and 10) and found that BART performs best in terms of having diverse sentence structure and preserving the semantics of the input when it returns 3 paraphrases at most.

6.4.4 Evaluation Settings

To evaluate the impact of using ChatMent on the NLU's performance, we need to train the NLU after augmenting the original training set using ChatMent and then evaluate its performance using the test set. Unlike the Repository dataset, all queries in the Ask Ubuntu and Stack Overflow datasets are collected from online forums (i.e., they are not used to implement chatbots). This means that for these datasets there is no predefined train-test splits. Therefore, we divide the Ask Ubuntu and Stack Overflow datasets into 50%-50% for training and test splits. More specifically, we apply random stratified sampling per intent to maintain a consistent distribution of the queries in the intents. This enables us to have enough training queries to apply ChatMent on, and enough test queries to evaluate the NLU's performance afterwards. Table 6.1 shows the distribution of queries for each intent in the training and test splits in the Repository, Ask Ubuntu, and Stack Overflow datasets.

Table 6.2: Performance comparison results for ChatMent against the baseline and human.

Experiment	Repository			Ask Ubuntu			Stack Overflow		
	T1	T3	T5	T1	T3	T5	T1	T3	T5
Baseline	44.73	62.05	67.02	72.93	83.68	86.99	33.04	35.81	-
ChatMent	44.29	62.69	68.17	71.76	84	87.51	31.17	36.52	-
Human	**55.87**	**64.3**	**69.92**	**77.05**	**85.31**	**90.18**	**37.36**	**40.2**	-

After obtaining the training and test splits for all datasets, we want to examine the impact of applying ChatMent on datasets with different training sizes. This helps us to explore the effect of using ChatMent at different use-cases (data is scarce). Therefore, we devise three evaluation scenarios: *T1*, *T3*, and *T5* where we randomly select one, three, and five queries per intent from the original training dataset for each scenario, respectively. We selected relatively small numbers of starting queries in our scenarios as we believe that ChatMent is most useful in cases where the chatbot developers are just starting to build their training set and want to add more queries to it.

6.4.5 Performance Evaluation

To assess the NLU's performance in classifying intents, we compute the widely used metrics of precision, recall, and F1-score. Precision is the percentage of the correctly classified queries to the total number of classified queries for that intent (i.e., Precision = $\frac{TP}{TP+FP}$). The recall is calculated as the percentage of the correctly classified queries to the total number of queries for that intent in the test set (i.e., Recall = $\frac{TP}{TP+FN}$). To have an overall score, we combine both the precision and recall using the F1-score weighted by class' support (weighted F1-score), which has been used in similar studies Abdellatif et al. (2021c); Barash et al. (2019); Ilmania et al. (2018). More specifically, we start by computing the F1-score (i.e., F1-score = $2 \times \frac{Precision \times Recall}{Precision + Recall}$) for all classes. Then, we aggregate all F1-scores using the weighted average, with the class' support as weights. It is important to note that, although we evaluate all three metrics, we only present the weighted F1-score in the paper. We added the precision, recall, and F1-score for each intent in all datasets to the Appendix.

6.5 Case Study Results

In this section, we present our evaluation results of ChatMent. For each RQ, we provide a motivation, detail the approach to answer the RQ, and present the results.

6.5.1 RQ1: Can ChatMent improve the NLU's performance?

Motivation: To improve the NLU's performance, chatbot developers typically add more queries to their training set. However, prior work shows that this process is a costly and time-consuming task Abdellatif, Badran, and Shihab (2020b); Dominic et al. (2020b). ChatMent is designed to automatically augment the original training set to achieve an improved performance of the NLU while saving the chatbot developers' time and effort. Therefore, in this RQ, we set out to assess the ChatMent's impact on the NLU's performance.

Approach: To answer this RQ, we perform three different experiments:

(1) **Baseline**: Establishes a baseline for the NLU's performance. In this experiment, we use the queries from each of the scenarios (*T1*, *T3* and *T5*) to train the NLU without augmenting new queries to the scenarios.

(2) **ChatMent**: Represents the case where ChatMent is used to augment the original training set. This experiment reflects the situation where a chatbot developer would start with a set of initial training queries and then apply the ChatMent to augment the original training set. In this experiment, we augment the scenarios *T1*, *T3*, and *T5* using ChatMent.

(3) **Human**: To put the ChatMent results into perspective, we evaluate the impact of using human-augmented queries on the NLU's performance. This experiment reflects another situation where a chatbot developer starts with a set of initial training queries and then manually augments the original training set. The human-augmented queries are high-quality and most likely can improve the NLU's performance compared to any augmentation approach. This experiment is performed by selecting one random query per intent from the training split that were not used in the scenarios *T1*, *T3*, and *T5*.

In the ChatMent and Human experiments, we use the selected candidate queries to augment the

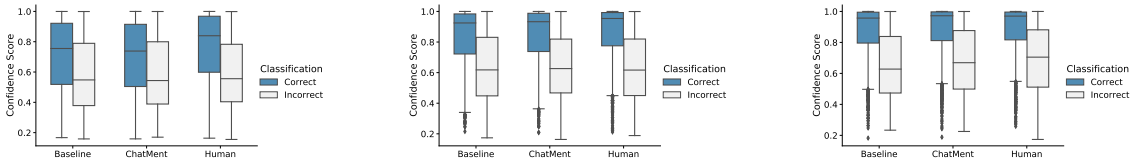
Table 6.3: Sample of ChatMent augmented queries.

Original Query	**Augmented Queries**	**Intent**
Show me the files which introduced more bugs	What are some examples of files that introduce bugs?	BuggyFiles
Show me the number of commits in last month	How many commits did you commit in the last month?	CommitsByDate
Shutdown problem in Ubuntu 16.04	What are the problems with Ubuntu 16.04 LTS?	ShutdownComputer
List the changes that resolved the defect ticket KAFKA 1482	List the commits that fixed the bug KAFKA-1988	FixCommit
What are the bug introducing change done last month	What bug introduce commits in 27/05/2018	BuggyCommit

queries from the scenario. To perform our evaluation, we follow the same steps for all experiments, datasets, and scenarios. First, we train the NLU using the scenario's queries resulting from the experiment. Next we use the test set to evaluate the NLU's performance for each experiment and record the results. It is important to note that in the Stack Overflow dataset, we only run our evaluation on scenarios *T1* and *T3*. This is because some intents (e.g., 'PassingData') in the *T5* scenario of the Stack Overflow dataset do not have enough training queries left to be used as input queries to randomly select from in the Human experiment.

Results: Table 6.2 presents the F1-scores for Baseline, ChatMent, and Human experiments on all scenarios and datasets. We find that using ChatMent does not improve the NLU's performance overall as shown in Table 6.2. For example, there is a 1% decrease in performance in scenario *T1* in the Ask Ubuntu dataset and a 1% performance increase in scenario *T5* in the Repository dataset by the ChatMent compared to the Baseline. Unsurprisingly, the Human experiment performs the best in all scenarios across the datasets in terms of F1-score.

Upon closer examination of the ChatMent's augmented queries, we have three main observations 1) ChatMent augments queries that have different sentence structures compared to the queries in the original training set. Table 6.3 presents a sample of the original training queries and their corresponding queries augmented through ChatMent. For instance, ChatMent augments the initial training query "Show me the files which introduced more bugs" to a new candidate query "What



(a) Scenario T1 (b) Scenario T3 (c) Scenario T5

Figure 6.3: The confidence score distributions for scenarios *T1*, *T3*, and *T5* in the Repository dataset.

are some examples of files that introduce bugs?". 2) Although ChatMent augments queries with different sentence structures, in most cases, it preserves the intent of the original queries as shown in Table 6.3. Among all augmented queries, we find only six candidate queries that had their intents changed compared to the initial training queries. For example, the initial training query "Which commits fixed KAFKA 2727?" asks about the fixing commit for a specific bug in a project ('Fix-Commit' intent), however, ChatMent augments it to "Which branches are connected to KAFKA 2727?" which changes the meaning of the query to ask about the branch associated with the bug ticket. 3) ChatMent fails to augment candidate queries for some intents (e.g., 'FacingError'). We further discuss this point in Section 6.6.

> *Augmenting the dataset using ChatMent does not improve the NLU's performance. Moreover, ChatMent fails to augment the queries of some intents. Nonetheless, we observe that the candidate queries generated by ChatMent have different sentence structures compared to the original training set while preserving their semantics (intents).*

6.5.2 RQ2: Does ChatMent increase the NLU's confidence in its classification?

Motivation: When using an NLU, chatbot developers rely on the confidence score returned with the classified intent to determine the chatbot's next action Abdellatif et al. (2021c). More specifically, developers tend to design their chatbots so that, if the NLU is not confident in its intent classification, the chatbot asks for further clarification from the user (i.e., "Sorry, I did not understand your question, could you please rephrase it?"). Otherwise, the chatbot answers user's

question/request. Developers also want the NLU to return higher confidence scores when it correctly classifies the intent and lower confidence scores for the misclassified intent. This limits the incorrect actions (actions based on a misunderstanding) taken by the chatbot. One way to increase the NLU's confidence in its classification is to train the NLU on more queries for each intent (i.e., augment the training set). Therefore, in this RQ, we study the effect of using ChatMent on the NLU's confidence, particularly with regards to the confidence scores returned with the correctly and incorrectly classified intents.

Approach: To answer this RQ, we use the output of the three experiments (Baseline, ChatMent, and Human) described in RQ1. More specifically, we examine the confidence scores returned by the NLU for the queries in the test set across all the experiments. By using the three experiments, we are able to establish a baseline (from the Baseline experiment) for how confident the NLU is in its intent classification. Then, we use that baseline to measure the impact of using ChatMent on the NLU's confidence (ChatMent experiment) compared to the impact of using human-augmented queries (Human experiment). We hypothesize that augmenting more queries increases the NLU's confidence scores for the correctly classified intents while decreasing its confidence for the misclassified intents. Finally, to compare the confidence scores for the correctly and incorrectly classified intents, we present the distributions of confidence scores for each case.

Results: In this RQ, we only discuss the results for the Repository dataset as our observations and findings in this RQ apply similarly to the Ask Ubuntu and Stack Overflow datasets. Nevertheless, we present the confidence score distributions for the Ask Ubuntu and Stack Overflow datasets in the Appendix.

Figure 6.3 shows the confidence score distributions for the correctly and incorrectly classified intents in the Repository dataset. From the figure, we observe that the median of confidence scores from correctly classified intents is higher than the ones from the misclassified intents in all experiments. In fact, these results are in-line with the ones in the prior work Abdellatif et al. (2021c).

On the other hand, training the NLU using the ChatMent-augmented queries does not improve the NLU's confidence in its classification compared to the baseline as shown Figure 6.3. One possible reason for this is that the NLU's performance is robust without any augmentation. For example, in scenario *T3* in the Baseline experiment of the Repository dataset where the NLU is

trained on one query per intent only, the NLU achieves confidence scores median of 0.92 for the correctly classified intents. Another interesting observation is that using the human-augmented queries (Human experiment) improves the NLU's confidence in its classification for scenarios *T1* and *T3* for the correctly classified intents only. As clearly shown in the Figure 6.3, the median of the confidence scores in the correct classifications in the *T1* scenario is 0.87 for the Human experiment, while the ChatMent and Baseline achieve 0.74 and 0.75 medians, respectively. For scenario *T5*, none of the augmentation experiments (i.e., Human and ChatMent) increase the NLU's confidence in its correct classifications or decrease its confidence in incorrect classifications. In other words, the human-augmented and ChatMent-augmented queries achieve comparable median of confidence scores to the baseline queries in the *T5* scenario.

To verify whether the difference in the confidence scores across the experiments' results (e.g., Human vs ChatMent) is statistically significant, we perform the non-parametric unpaired Mann-Whitney U test. We find that the difference between the Human and Baseline experiments is statistically significant (i.e., P-value < 0.05) in Scenarios *T1* and *T3* for the correctly classified confidence scores only. Also, we find that the difference in the correctly classified confidence scores distributions is significant between ChatMent and Human experiments in all scenarios *T1*, *T3*, and *T5*. For the confidence scores of the misclassified intents, we find that the differences between all experiments are insignificant.

Finally, to quantify the difference of the statistically significant changes, we use Cliff's Delta Cliff (1993) to compute the delta effect size between confidence scores of the experiments. Moreover, we use Romano, Kromrey, Coraggio, and Skowronek (2006) guide to interpret the delta effect size. We find that the difference in the confidence scores of correctly classified intents between ChatMent and both the Human and Baseline is small for scenario *T1* as shown in Table 6.4. In scenarios *T3* and *T5* however, the delta effect size is negligible for the correctly classified confidence scores across all experiments.

> *Using ChatMent does not increase the NLU's confidence in its classification. Nonetheless, the human augmented queries increase the NLU's confidence marginally when the original training set is smaller.*

Table 6.4: The Cliff's delta effect size for all experiment in the Repository dataset.

Classification	Scenario	Experiment 1	Experiment 2	Delta	Size
Correct	1	Baseline	ChatMent	0.021	Negligible
	3	Baseline	ChatMent	-0.039	Negligible
	5	Baseline	ChatMent	-0.068	Negligible
	1	Baseline	Human	-0.166	Small
	3	Baseline	Human	-0.123	Negligible
	5	Baseline	Human	-0.044	Negligible
	1	ChatMent	Human	-0.183	Small
	3	ChatMent	Human	-0.083	Negligible
	5	ChatMent	Human	0.026	Negligible
Incorrect	1	Baseline	ChatMent	-0.019	Negligible
	3	Baseline	ChatMent	-0.001	Negligible
	5	Baseline	ChatMent	-0.081	Negligible
	1	Baseline	Human	-0.027	Negligible
	3	Baseline	Human	0.017	Negligible
	5	Baseline	Human	-0.114	Negligible
	1	ChatMent	Human	-0.010	Negligible
	3	ChatMent	Human	0.019	Negligible
	5	ChatMent	Human	-0.032	Negligible

Table 6.5: The average number of ChatMent and human augmented queries.

Model	Repository			Ask Ubuntu			Stack Overflow		
	T1	T3	T5	T1	T3	T5	T1	T3	T5
ChatMent	0.7	1.8	2.2	0	0	0.1	0	0	-
Human	9.9	8.7	8	3	2.7	2.4	5.1	5.4	-

6.6 Discussion

As shown in the results of RQ1, the NLU's performance after applying ChatMent remains very close to the baseline. Prior work shows that NLUs tend to perform the best when they are trained on more queries Abdellatif et al. (2021c). Therefore, we calculate the average number of augmented queries (i.e., queries introduced by ChatMent and not in the original training set) per intent. We also perform the same analysis on the Human experiment results. Table 6.5 presents the average number of queries augmented per intent for the Repository, Ask Ubuntu, and Stack Overflow datasets in the ChatMent and Human experiments.

Table 6.6: Performance results as F1-score w/out using our Selection phase.

Models	Repository			Ask Ubuntu			Stack Overflow		
	T1	**T3**	**T5**	**T1**	**T3**	**T5**	**T1**	**T3**	**T5**
Random	55.87	64.3	69.92	77.05	85.31	90.18	37.36	40.2	-
Selection	**57.52**	**66.67**	**70.14**	**82.08**	**89.71**	**90.51**	**43.25**	**43.12**	-

From the results, we observe that ChatMent does not add new queries in the Ask Ubuntu (except for scenario *T5*) and Stack Overflow datasets, which could explain why ChatMent does not improve the NLU's performance in these two datasets. Thus, it is likely that the slight changes in performance between the baseline and ChatMent are due to some randomness in the NLU training process. On the other hand, we observe that ChatMent augments few queries to the Repository dataset as shown in Table 6.5. However, these augmented queries seem to have little effect on the NLU's performance as discussed in RQ1. The results also highlight that, in the Human experiment, more queries are augmented to all the datasets, allowing the NLU to train on more data. This better explains the increase in NLU's performance in the Human experiment. Finally, when looking at the results of each scenario, we find that ChatMent augments more queries in scenarios *T3* and *T5*, which is likely because the approach has more input queries to augment compared to the *T1* scenario. To better understand the reasons behind the small number of augmented queries by ChatMent, we examine the results in depth and notice that most of the candidate queries generated by the Augmentation phase are discarded by the Selection Phase. In other words, the candidate queries are either familiar to the NLU or not lexically diverse enough. Moreover, we also find that the candidate queries generated by the Augmentation phase are similar to each other.

Therefore, we further investigate the effectiveness of the Selection phase. To achieve this, we compare the performance of two trained NLU models. We train the first model, called Random model, by selecting one random query from the training split and adding it to the scenarios (e.g., *T1*). For the second model, we input the queries from the original training set into the Selection phase and add one output query from the Selection phase to the scenario dataset and train an NLU model (Selection model) using the augmented dataset. Finally, we evaluate the Random and Selection models using the test set for the Repository, Ask Ubuntu, and Stack Overflow. Table 6.6

presents the performance results of both the Random and Selection models in terms of F1-score. The Selection model achieves better performance compared to the Random model across all scenarios and datasets. The highest increase in F1-score is 5.89% in the Stack Overflow dataset (scenario *T1*), while the lowest performance improvement is 0.33% in the Ask Ubuntu dataset (scenario *T5*) over the Random model.

Our findings show that the Selection phase is effective in selecting queries that improve the NLU's performance. In light of this finding, we speculate that the majority of candidate queries generated by the Augmentation phase are not meaningful enough to pass the Selection phase. We reiterate that the Augmentation phase is a multi-step process that includes a Synonym Replacement component and Paraphrasing component that leverages state-of-the-art paraphrasing approach (BART). Nevertheless, the results show that more needs to be done in order to produce higher quality augmented queries, especially during the paraphrasing process. Moreover, our findings highlight the need for more investigations about the applicability of different augmentation techniques specialized for the SE-based chatbots.

6.7 Lessons Learned

Does relaxing the selection criteria help improve the NLU's performance? No. We configured ChatMent to augment more than one query per intent by loosening the selection criteria to select the top three queries rather than only the top one. We find that the NLU's performance decreases compared to augmenting one query. Upon closer examination of the results, we observe that the NLU misclassifies intents that share similar characteristics (i.e, queries with exclusive words and have distinct entity type) with other intents. This matches the findings in Abdellatif et al. (2021c) where the NLUs tend to correctly classify intents with exclusive words. Therefore, future augmentation approaches need to consider the characteristics of the augmented queries for each intent to increase the NLU's performance in intents classification.

There is a need for benchmarks of paraphrased queries for the SE domain. Prior work shows that fine-tuning the transformers using SE datasets is valuable compared to using general training data Mastropaolo et al. (2021); von der Mosel, Trautsch, and Herbold (2021). In our study,

we use three datasets (e.g., Quora question pairs) to fine-tune BART for the paraphrasing task. However, none of these datasets are related to the SE-domain. This might be one of the reasons behind the limited usefulness of the candidate queries augmented by the Augmentation phase as discussed in Section 6.6. Therefore, we explore using an SE dataset to fine-tune BART. In particular, we downloaded 44K Stack Overflow posts that marked as duplicates. We believe that the duplicated post titles are considered as paraphrased as they are asking about the same development problem Ahasanuzzaman, Asaduzzaman, Roy, and Schneider (2016). Then, we fine-tune BART using the Stack Overflow duplicate posts and the three corpora discussed in Section 6.4. Finally, we use the same evaluation process presented in Section 6.4 to asses ChatMent given the new fine-tuned BART model. The results show that the NLU's performance decreases when using this configuration. To better understand the reason for this decrease in performance, we examine the resulting queries and find that the augmented queries are paraphrased in a way to ask about achieving a task using a specific programming language. For example, the original query "What commits resolved the bug ticket KAFKA 1521" is augmented to the candidate query "What commits resolved the bug ticket KAFKA 1521 using PHP?". It is typical of Stack Overflow posts to mention the programming language in the title, therefore the dataset might bias BART to include a programming language in each query. To the best of our knowledge, there is no crafted dataset that contains pairs of paraphrased queries related to the SE domain. We believe that there is a need for benchmarks that contains paraphrased pairs of queries that represent different SE tasks.

NLUs are robust even when trained on few queries. In our study, we find that the NLU achieves a good performance even when trained on few queries per intent. In fact, with only one training query per intent (i.e., *T1* Scenario), the NLU performs considerably well, with an F1-Score ranging from 33% in the Stack Overflow dataset and up to 73% in the Ask Ubuntu dataset as shown in the baseline results of the *T1* scenario Table 6.2. This observation is further reinforced when looking at the baseline performance with three and five initial training queries (i.e., scenarios *T3* and *T5*), where the NLU achieves an F1-score ranging from 35% to 87% across the different datasets. Therefore, future augmentation approaches should focus on the quality of the generated queries more than the quantity as the NLU only needs a few queries to perform well.

Modifying the candidate queries using a rule-based component does not improve the results. We observe that the candidate queries generated by the Augmentation phase are similar to each other as discussed in Section 6.6. Hence, we attempted to diversify the candidate queries further by introducing another component to the Augmentation phase. More specifically, we develop a rule-based technique to transform a query into a question. For example, the Augmentation phase outputs the candidate query "Show me the number of commits in the last month", which is then input into the rule-based component, resulting in the query "What is the number of commits in the last month?". However, the results show that using the rule-based component does not improve the performance of the current version of ChatMent. This is because the Selection phase discards most of the candidate queries generated by the rule-based component as they are not meaningful enough to the NLU. Our finding confirms the robustness of the NLU and underlines the need of augmentation approaches that generate more diverse candidate queries that are useful to the NLU.

Some training queries improve the NLU's performance more than others. The results from Section 6.6 show that training the NLU on queries selected by Selection phase achieves better performance compared to the randomly selected queries, although all input queries were originally designed by humans. Therefore, the usefulness of the queries in the training set is an important aspect that chatbot developers need to consider. That said, it is difficult to decide the usefulness of the query as most of the NLU implementations are black box Abdellatif et al. (2021c). Therefore, we recommend chatbot developers to use the Selection phase to select the most useful queries to add to their training set.

Better augmentation techniques are needed for SE-based chatbots. We use the state-of-art augmentation techniques (i.e., synonyms replacement and BART) to augment the training datasets of SE chatbots. However, in our experiment, these approaches come-short when augmenting useful queries as discussed in Section 6.6. This raises a red flag for SE-chatbot community as the current state-of-art augmentation techniques (e.g., BART) might be limited in its ability to augment chatbot training datasets. This requires more investigation about the applicability of different augmentation techniques on the task of augmenting SE chatbot training datasets. Moreover, this highlights the need for specialized augmentation approaches for SE-based chatbots. We plan (and encourage

others) to explore other augmentation approaches (e.g., BERT, GPT2) and weak-supervision Z.-H. Zhou (2017) to augment SE-based chatbots training dataset.

6.8 Threats to Validity

In this section, we discuss the threats to internal, replicability, and external validity of our study.

Internal Validity: Concerns confounding factors that could have influenced our results. We configure BART to return three queries (sequences) which might impact the quality of the paraphrased queries. To alleviate this threat, the first author examined the output from BART using different numbers of returned queries (1, 3 ,5, 7, and 10) across the three datasets and found that configuring the returned queries to be three yields to the best results in terms of having different sentence structure and preserving semantics of the input queries. Another threat to internal validity is that we configure the Synonym Replacement component to replace only the verbs in a query. Thus, replacing words with different part-of-speech (e.g., nouns) might lead to different results. However, we find that replacing nouns is ineffective as most nouns in our datasets are proper nouns (e.g., 'Java', 'Ubuntu') which are words that do not have synonyms. We also find that words from with other part-of-speech (e.g., pronouns) are scarce in our datasets.

Replicability Validity: Concerns the possibility of replicating the study Epskamp (2019b). To enable the replication of this study we make our scripts and data publicly available at Abdellatif et al. (2021a). Since we used the open-source Rasa NLU in this study, we also provide details about the Rasa version we used to enable replication. However, we observed some randomness in the NLU's performance when trained and tested using the same data. This might affect our results and conclusion as the change in the NLU's performance across different experiments might have been caused by the model's training randomness. Nevertheless, we mitigate this by repeating the evaluation step three times for each experiment. Thus, the presented results in Section 6.5 are averaged from the three runs in order to reduce the randomness effect on the results.

External Validity: Concerns the generalization of our findings. In this study, we devise an approach to augment the training dataset for SE chatbots. And for the purpose of evaluating ChatMent, we use

three different datasets which might make our results not generalized for other SE datasets. However, these datasets have been used by previous studies to evaluate the NLUs and chatbots Abdellatif et al. (2021c); Larson et al. (2019); Shridhar et al. (2019, 2020) in the SE domain.

We use Rasa NLU to evaluate the impact of using ChatMent on the NLU's performance; hence our results might not be generalized to other NLUs. However, we select Rasa as it is an open-source NLUs which guarantee that its internal implementation stay the same during our entire study. Moreover, Rasa has been widely used by practitioners to develop SE chatbots Abdellatif et al. (2021c); Abdellatif, Badran, and Shihab (2020b); Chun-Ting Lin and Huang (2020); Dominic et al. (2020b).

6.9 Conclusion & Future Work

Software chatbots play important roles in the software engineering domain as they enable practitioners to perform tasks (e.g., run tests) through natural language. Chatbots rely on the NLU component to understand the user's input. Training the NLU on possible queries from users is important because it affects its ability to classify the user's intent behind the query. Creating datasets to train the NLU is a costly and time-consuming task. Therefore, in this chapter, we explore an augmentation approach (called ChatMent) to help chatbot developers create high-quality training dataset. Then, we evaluate the impact of using ChatMent on the NLU's performance using three SE datasets and the Rasa NLU platform. We find that training the NLU using ChatMent does not improve the NLU's performance and confidence in its intent classification. Nevertheless, we find that the Selection phase is effective in selecting candidate queries that are useful to the NLU.

The results in this paper outline some directions for future work. First, we are planning to evaluate ChatMent on more NLUs (e.g., Dialogflow, IBM Watson) to evaluate the generalizability of ChatMent. Second, we want to examine the performance of different Transformers (e.g., GPT2, BERT, RoBERTa) in the task of paraphrasing SE queries. As we discussed in Section 6.6, there is a lack of SE paraphrased queries to train the Transformers. Therefore, we want to investigate the use of Stack Overflow posts to design a benchmark containing paraphrased queries to help tune the transformers for the SE domain.

Chapter 7

Summary, Contributions and Future Work

Software chatbots draw the attention of practitioners in the SE domain. However, little is known about the challenges of developing SE-based chatbots and their benefits in assisting practitioners in their development tasks. We identified the challenges that face chatbot practitioners when developing chatbots. We highlighted the benefits of using chatbots in the SE domain. We presented empirical studies and proposed approaches to address some of the identified challenges. In this chapter, we summarize the thesis by presenting the main work and contribution in each chapter of the thesis. Additionally, we discuss future work related to chatbot development for SE domain.

7.1 Summary

The main focus of this thesis is to understand SE-based chatbot development challenges and values of using them in SE tasks, and improve the SE-based chatbots. First, we focus on studying the main challenges of developing chatbots. Second, we showcase the potential benefits of using chatbots in the software development process by developing an SE chatbot. The results from these two studies reveal that practitioners struggle to select an NLU platform for their chatbot implementation as it is critical for the chatbot's capability of understanding user's queries. Moreover, the results show that developing datasets to train the NLU is a costly and time-consuming task. Based on these

findings, we conduct an empirical study to find the best performing NLU in the SE context. Also, we propose an approach to augment the training dataset for SE chatbots. The presented research in this thesis provides the following contributions:

Chapter 3: Understanding the challenges of chatbot development

In Chapter 3, we provide the first attempt at understanding the challenges of chatbot development. We perform a qualitative study by analyzing posts on Stack Overflow to identify the major topics surrounding the discussions on chatbot development. Our results show that developers discuss 12 chatbot topics (e.g., intents & entities) that fall under five categories. Most of the posts belong to chatbot development, integration, and the natural language understanding (NLU) model. We also find that chatbot developers are highly interested in posts that are related to chatbot creation and integration into websites. Furthermore, we find that developers face challenges in the training of the chatbot's model.

Chapter 4: Determining the value of SE-context chatbots

This thesis presents the first study to use chatbots on software repositories. More specifically, we lay out a chatbot on top of software repositories to answer some of the most common software development/maintenance questions facing developers during their tasks. Furthermore, we conduct a user study to evaluate the proposed chatbot, and our findings show that most of the study participants (90%) find the chatbot to be either useful or very useful. Moreover, participants completed 90.8% of tasks correctly using the chatbot, compared to the 25.2% tasks completed without the chatbot. We find that the selection of NLU platform to use directly impacts the chatbot's accuracy in classifying user's queries. Finally, crafting a dataset to train the NLU is costly and time-consuming task for chatbot developers.

Chapter 5: Can we help developers to design more effective chatbots for the SE domain?

To help chatbot developers to design more effective chatbots, we perform the first work to evaluate widely used NLUs on two representative tasks from the SE domain. We find that NLU

tends to perform better when they are trained on more queries where IBM Watson achieves the best in terms of intents classification. Also, we find that most NLUs report high confidence scores for correctly classified intents. For the entity extraction task, the results show that LUIS performs the best in extracting entities from the Repository task (F1-measure 93.7%), while IBM Watson comes on top in the Stack Overflow task (F1-measure 68.5%).

Chapter 6: Improving the SE chatbot's accuracy

To assist chatbot developers to craft high-quality datasets to train the NLU, we evaluate the combination of the synonyms replacement and paraphrasing NLP techniques in an approach to augment the training datasets of SE-based chatbots. The approach augments queries that preserve similar semantics but have new terms/keywords and sentence structures compared to the queries in the chatbot training dataset. Our study shows that training the NLU using the combined approach does not improve the NLU's performance and confidence in its intent classification. Nevertheless, we find that the Selection phase is effective in selecting queries that improve the NLU's performance. Our results should alert the chatbot community on the limitations of current augmentation approaches when applied to the software engineering domain.

7.2 Future Work

We believe that this thesis makes significant contributions towards identifying the challenges of developing SE chatbots and understanding the values of using chatbots in the software development tasks. However, there are still more challenges that need to be addressed in order to ease the development and increase the adoption of chatbots in the software engineering domain. We discuss some avenues for future work.

7.2.1 Improving User-Chatbot Interaction

Our results in Chapter 3 show that chatbot developers have difficulties in designing conversation flow with the user and generating answers to user's queries. The results in Chapter 4 confirm these findings where the users of the MSRBot suggested enhancing the presentation of the chatbot's

answers, especially when the answer is too long and contains a lot of information. For future work, an additional investigation on the best way to present the chatbot's reply to the user is required to increase the adoption of chatbots in the software engineering domain.

7.2.2 Exploring the Use of Chatbots in Other Software Engineering Tasks

The results in Chapter 4 show that using chatbots layered on top of software repositories (i.e., code and issue tracking repositories) to answer software project related questions are promising in terms of usefulness, efficiency, and accuracy. Therefore, we are planning in the future to explore using chatbots on other software engineering tasks such as performing source code analysis (e.g., finding and fixing vulnerabilities in the source code) and answering code related questions (e.g., what technical debt exists in the code?).

7.2.3 Supporting more Users' Questions

Chapter 4 of this thesis shows that the users asked the MSRBot questions that are not yet supported. Therefore, the chatbot cannot answer those questions which negatively impacts the users' experience with the chatbot. This implies two future research directions 1) To study the types of questions that practitioners based on their roles (e.g., project managers) usually ask during their software development. Identifying the most frequent questions helps developing more task-specific chatbots (e.g., testing chatbots). 2) Develop approaches that answer user's questions dynamically (i.e., removing the need to have predefined questions). Both directions help increase the applicability of the chatbots in the software engineering domain by supporting more user's questions.

7.2.4 Enhancing the Unique Entities Extraction

Chatbots use the extracted intent and entities to perform the next action. The results of this thesis show that the NLUs have poor performance in identifying unique entities (i.e., entities that appear once in the chatbot's training dataset). Therefore, the chatbot performs wrong actions which lead to return incorrect answers to the user. Prior work shows that chatbots answers deeply affect the user's satisfaction. In the future, we plan to propose techniques to extract and augment the unique entities to increase the chatbot's accuracy in performing the intended tasks asked by the user.

7.2.5 Investigating the Impact of Using Chatbots on the Developers' Social Aspects

Although the results of this thesis present the potential benefits of using chatbots in the software development process. There is a lack of studies that focus on the social aspects of using chatbots. As a research community, we want to implement and integrate tools that increase the code quality and velocity of software development. However, we want to ensure that these tools have no negative impacts on the software practitioners (e.g., no social interactions between team members). Prior work shows that the social aspects are important towards team productivity and software sustainability. Future work should explore the impact of using chatbots on the social aspects of team members.

7.2.6 Evaluating the Performance of Transformers for Augmenting SE Dataset

In Chapter 6, we evaluate an approach that augments the training dataset for SE chatbots. A major problem with using the transformer (BART) to paraphrase queries is that all the paraphrased queries are not adding values to the NLU. In other words, the augmented queries are similar to the ones in the original training dataset for the chatbot. In the future, we plan to focus on evaluating the paraphrasing capabilities of different transformers using SE datasets and proposing more advanced augmentation approaches to help chatbot developers in crafting high-quality training datasets.

Appendix A

Appendix-A

Detailed Precision and Recall Values When Evaluating Intents Classifications.

In section 5.4.1, we presented the F1-measure values for each task when classifying intents. In this appendix, we add detailed precision, recall, and F1-measure values that are used to compute the F1-measure values for each task on each NLU.

Table A.1: Intents classification results for the Repository task.

Intent	IBM Watson			Dialogflow			Rasa			LUIS		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1
CountCommitsByDate	86.4	90.5	88.4	84.0	100.0	91.3	100.0	66.7	80.0	88.9	38.1	53.3
FileCommits	83.3	41.7	55.6	100.0	66.7	80.0	100.0	16.7	28.6	16.7	8.3	11.1
OverloadedDev	100.0	88.9	94.1	100.0	66.7	80.0	88.9	88.9	88.9	40.9	100.0	58.1
CommitsByDate	87.5	58.3	70.0	50.0	50.0	50.0	88.9	66.7	76.2	0.0	0.0	0.0
ExperiencedDevFixBugs	100.0	85.7	92.3	92.3	85.7	88.9	100.0	85.7	92.3	100.0	7.1	13.3
BuggyFiles	100.0	92.3	96.0	100.0	92.3	96.0	100.0	100.0	100.0	90.9	76.9	83.3
BuggyCommitsByDate	60.0	92.3	72.7	80.0	61.5	69.6	70.6	92.3	80.0	100.0	84.6	91.7
FixCommit	100.0	100.0	100.0	84.6	100.0	91.7	100.0	100.0	100.0	52.4	100.0	68.8
BuggyCommit	90.0	100.0	94.7	90.0	100.0	94.7	90.0	100.0	94.7	47.4	100.0	64.3
BuggyFixCommit	100.0	100.0	100.0	85.7	85.7	85.7	50.0	85.7	63.2	33.3	85.7	48.0

P: Precision, R: Recall, F1: F1-measure

Detailed Precision and Recall Values When Assessing Entity Extraction.

When discussing the entity extraction results in Section 5.4.3, we presented the F1-measure values for each task. Here, we present the detailed precision, recall, and F1-measure values that are used

Table A.2: Intents classification results for the Stack Overflow task.

Intent	IBM Watson			Dialogflow			Rasa			LUIS		
	P	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
UsingMethodImproperly	93.6	73.3	79.1	84.0	58.3	66.5	81.6	58.3	64.4	71.7	51.7	57.5
LookingForCodeSample	85.3	97.9	90.9	81.5	91.4	85.9	78.2	92.1	84.5	75.4	95.7	84.0
FacingError	80.0	80.0	80.0	60.0	60.0	60.0	30.0	40.0	33.3	10.0	10.0	10.0
PassingData	35.0	40.0	36.7	50.0	50.0	50.0	35.0	40.0	36.7	0.0	0.0	0.0
LookingForBestPractice	86.7	80.0	81.3	86.7	80.0	81.3	76.7	80.0	78.0	90.0	80.0	83.3

P: Precision, R: Recall, F1: F1-measure

to compute the F1-measure values presented earlier.

Table A.3: Entity extraction classification results for the Repository task.

Entity Type	IBM Watson			Dialogflow			Rasa			LUIS		
	P	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
FileName	8.6	100.0	15.9	86.4	73.1	79.2	93.8	57.7	71.4	83.3	76.9	80.0
JiraTicket	100.0	100.0	100.0	84.6	100.0	91.7	100.0	81.8	90.0	100.0	100.0	100.0
DateTime	78.1	96.2	86.2	56.9	55.8	56.3	100.0	100.0	100.0	98.1	98.1	98.1
CommitHash	100.0	100.0	100.0	100.0	100.0	100.0	100.0	80.0	88.9	100.0	100.0	100.0

P: Precision, R: Recall, F1: F1-measure

Table A.4: Entity extraction classification results for the Stack Overflow task.

Entity Type	IBM Watson			Dialogflow			Rasa			LUIS		
	P	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
ProgLanguage	88.4	96.2	92.0	90.7	97.2	93.7	92.1	91.5	91.4	95.9	80.2	86.8
Framework	70.7	63.2	65.4	85.3	43.2	56.0	89.1	42.1	56.1	88.6	41.1	54.4
Standards	81.0	42.9	54.1	85.7	47.6	56.9	17.5	14.3	14.9	23.8	14.3	17.5
API	50.9	39.2	43.3	84.0	32.4	42.8	62.3	20.3	29.9	48.6	16.2	23.5
Platform	76.9	61.5	67.4	64.1	53.8	55.9	84.6	69.2	75.1	53.8	38.5	43.6

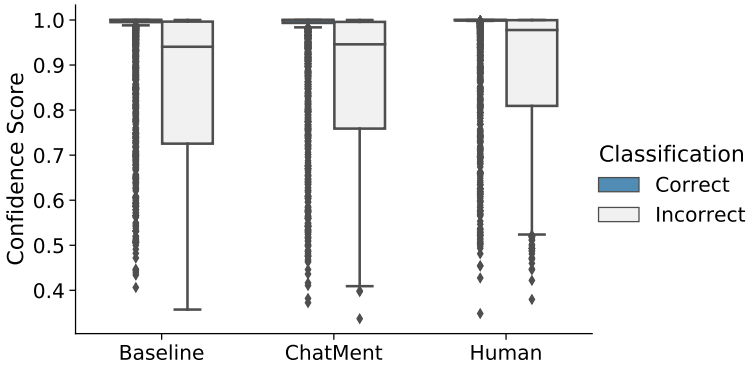
P: Precision, R: Recall, F1: F1-measure

Appendix B

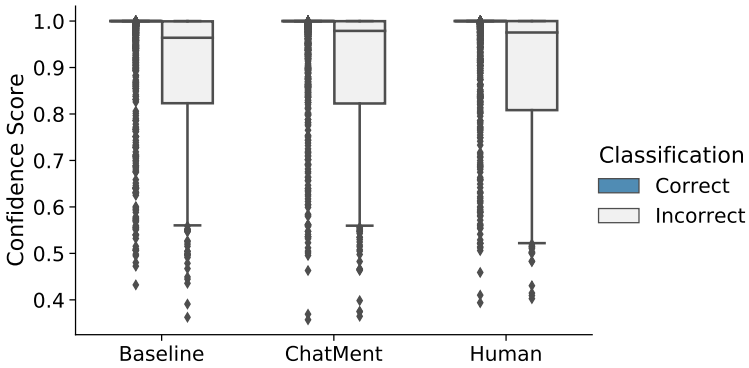
Appendix-B

Detailed Confidence Scores Distributions.

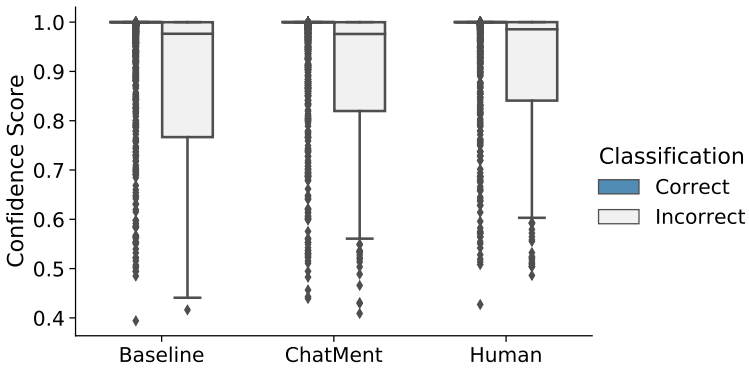
In Section 6.5, we presented the confidence scores distributions for the Repository dataset. In this appendix, we present the distribution of the confidence scores for the correctly and incorrectly classified intents in the Ask Ubuntu and Stack Overflow datasets.



(a) Scenario T1

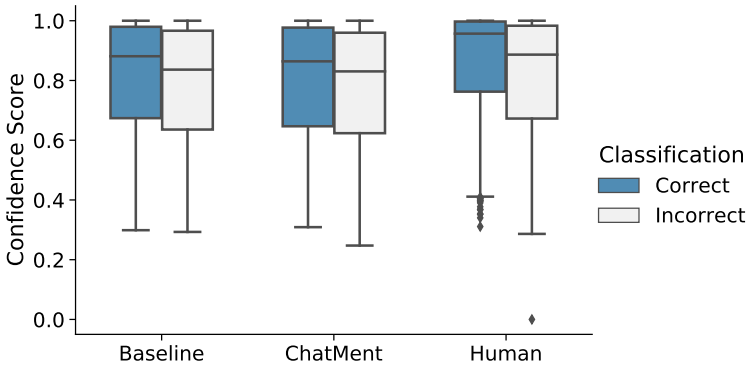


(b) Scenario T3

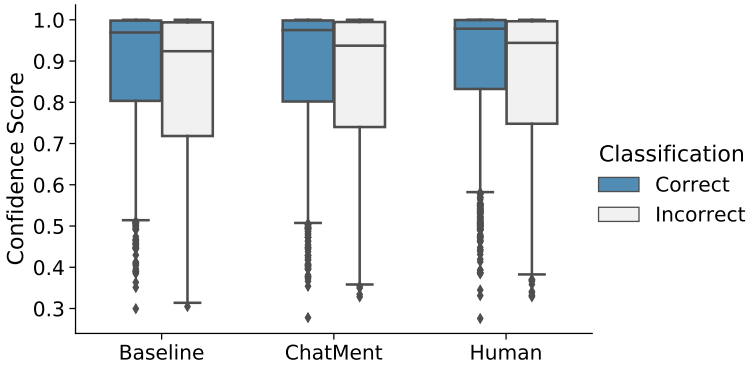


(c) Scenario T5

Figure B.1: The confidence score distributions for Scenarios T1, T3, and T5 in the Ask Ubuntu dataset.



(a) Scenario T1



(b) Scenario T3

Figure B.2: The confidence score distributions for Scenarios T1, T3, and T5 in the Stack Overflow dataset.

Detailed Delta Effect Size Between Confidence Scores of the Experiments.

When discussing the delta size effect in Section 6.5, we presented the size effect results for the Repository dataset. Here, we present the delta size effect between the experiments for the Ask Ubuntu and Stack Overflow datasets.

Table B.1: The Cliff's delta effect size for all experiment in the Ask Ubuntu dataset.

Classification	Split	Experiment 1	Experiment 2	Delta	Size
	1	Baseline	ChatMent	0.009	Negligible
Correct	3	Baseline	ChatMent	0.028	Negligible
	5	Baseline	ChatMent	-0.020	Negligible
	1	Baseline	Human	-0.213	Small
	3	Baseline	Human	-0.035	Negligible
	5	Baseline	Human	-0.073	Negligible
	1	ChatMent	Human	-0.231	Small
	3	ChatMent	Human	-0.062	Negligible
	5	ChatMent	Human	-0.051	Negligible
Incorrect	1	Baseline	ChatMent	-0.021	Negligible
	3	Baseline	ChatMent	-0.026	Negligible
	5	Baseline	ChatMent	-0.037	Negligible
	1	Baseline	Human	-0.167	Small
	3	Baseline	Human	-0.030	Negligible
	5	Baseline	Human	-0.084	Negligible
	1	ChatMent	Human	-0.149	Small
	3	ChatMent	Human	-0.007	Negligible
	5	ChatMent	Human	-0.047	Negligible

Table B.2: The Cliff's delta effect size for all experiment in the Stack Overflow dataset.

Classification	Split	Experiment 1	Experiment 2	Delta	Size
Correct	1	Baseline	ChatMent	0.021	Negligible
	3	Baseline	ChatMent	-0.039	Negligible
	1	Baseline	Human	-0.166	Small
	3	Baseline	Human	-0.123	Negligible
	1	ChatMent	Human	-0.183	Small
	3	ChatMent	Human	-0.083	Negligible
Incorrect	1	Baseline	ChatMent	-0.019	Negligible
	3	Baseline	ChatMent	-0.001	Negligible
	1	Baseline	Human	-0.027	Negligible
	3	Baseline	Human	0.017	Negligible
	1	ChatMent	Human	-0.010	Negligible
	3	ChatMent	Human	0.019	Negligible

References

Abdalkareem, R., Shihab, E., & Rilling, J. (2017, March). What do developers use the crowd for? a study using stack overflow. *IEEE Softw.*, *34*(2), 53–60. doi: 10.1109/MS.2017.31

Abdalkareem, R., Shihab, E., & Rilling, J. (2017, Mar). What do developers use the crowd for? a study using stack overflow. *IEEE Software*, *34*(2), 53-60. doi: 10.1109/MS.2017.31

Abdellatif, A., Badran, K., Costa, D., & Shihab, E. (2021a). *Chatment: A data augmentation approach for software engineering chatbots.* https://github.com/ahmad-abdellatif/Augmentation. ((Accessed on 10/16/2021))

Abdellatif, A., Badran, K., Costa, D., & Shihab, E. (2021b). A comparison of natural language understanding platforms for chatbots in software engineering. *IEEE Transactions on Software Engineering (TSE)*, 1-1. doi: 10.1109/TSE.2021.3078384

Abdellatif, A., Badran, K., Costa, D., & Shihab, E. (2021c). A comparison of natural language understanding platforms for chatbots in software engineering. *IEEE Transactions on Software Engineering (TSE)*, 1-1. doi: 10.1109/TSE.2021.3078384

Abdellatif, A., Badran, K., Costa, D., & Shihab, E. (2021d). *A comparison of natural language understanding platforms for chatbots in software engineering — zenodo.* https://zenodo.org/record/4734080. ((Accessed on 02/10/2021))

Abdellatif, A., Badran, K., & Shihab, E. (2019a). *ahmad-abdellatif/msrbot: Msrbot framework.* https://github.com/ahmad-abdellatif/MSRBot. ((Accessed on 10/10/2019))

Abdellatif, A., Badran, K., & Shihab, E. (2019b). MSRBot: Using bots to answer questions from software repositories. *Empirical Software Engineering (EMSE)*, To Appear.

Abdellatif, A., Badran, K., & Shihab, E. (2019c, July). MSRBot: Using Bots to Answer Questions

from Software Repositories. *Empirical Software Engineering*. Retrieved from https://doi.org/10.5281/zenodo.3382071 doi: 10.5281/zenodo.3382071

Abdellatif, A., Badran, K., & Shihab, E. (2020a). Msrbot: Using bots to answer questions from software repositories. *Empirical Software Engineering (EMSE)*. doi: 10.1007/s10664-019-09788-5

Abdellatif, A., Badran, K., & Shihab, E. (2020b). Msrbot: Using bots to answer questions from software repositories. *Empirical Software Engineering (EMSE)*, *25*, 1834-1863.

Abdellatif, A., Costa, D. E., Badran, K., Abdelkareem, R., & Shihab, E. (2020). Challenges in chatbot development: A study of stack overflow posts. In *Proceedings of the 17th international conference on mining software repositories (msr'20)* (p. To Appear).

Acharya, M. P., Parnin, C., Kraft, N. A., Dagnino, A., & Qu, X. (2016). Code drones. In *Proceedings of the 38th international conference on software engineering companion* (p. 785788). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/2889160.2889211 doi: 10.1145/2889160.2889211

Agus Santoso, H., Anisa Sri Winarsih, N., Mulyanto, E., Wilujeng saraswati, G., Enggar Sukmana, S., Rustad, S., ... Firdausillah, F. (2018). Dinus intelligent assistance (dina) chatbot for university admission services. In *2018 international seminar on application for technology of information and communication* (p. 417-423).

Ahasanuzzaman, M., Asaduzzaman, M., Roy, C. K., & Schneider, K. A. (2016). Mining duplicate questions in stack overflow. In *Proceedings of the 13th international conference on mining software repositories* (p. 402412). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/2901739.2901770 doi: 10.1145/2901739.2901770

Ahmed, S., & Bagherzadeh, M. (2018). What do concurrency developers ask about? a large-scale study using stack overflow. In *Proceedings of the 12th acm/ieee international symposium on empirical software engineering and measurement* (pp. 30:1–30:10). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/3239235.3239524 doi: 10.1145/3239235.3239524

Ahmed, T. M., Bezemer, C.-P., Chen, T.-H., Hassan, A. E., & Shang, W. (2016). Studying

the effectiveness of application performance management (apm) tools for detecting performance regressions for web applications: An experience report. In *Proceedings of the 13th international conference on mining software repositories* (pp. 1–12). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/2901739.2901774 doi: 10.1145/2901739.2901774

Ali, N., Guhneuc, Y. G., & Antoniol, G. (2013, May). Trustrace: Mining software repositories to improve the accuracy of requirement traceability links. *IEEE Transactions on Software Engineering*, *39*(5), 725-741. doi: 10.1109/TSE.2012.71

Amazon. (2019, Dec). *Amazon lex - build conversation bots.* https://aws.amazon.com/lex/. ((Accessed on 12/12/2019))

Amin-Nejad, A., Ive, J., & Velupillai, S. (2020, May). Exploring transformer text generation for medical dataset augmentation. In *Proceedings of the 12th language resources and evaluation conference* (pp. 4699–4708). Marseille, France: European Language Resources Association. Retrieved from https://aclanthology.org/2020.lrec-1.578

Apple. (2020). *Siri - apple.* https://www.apple.com/ca/siri/. ((Accessed on 01/08/2020))

Ask, J. A., Facemire, M., Hogan, A., & Conversations, H. N. B. (2016). The state of chatbots. *Forrester. com report*, *20*.

AWS, A. (2019). *Document history for amazon lex - amazon lex.* https://docs.aws.amazon.com/lex/latest/dg/doc-history.html. ((Accessed on 12/12/2019))

Baby, C. J., Khan, F. A., & Swathi, J. N. (2017, April). Home automation using iot and a chatbot using natural language processing. In *2017 innovations in power and advanced computing technologies (i-pact)* (p. 1-6). IEEE Press. doi: 10.1109/IPACT.2017.8245185

Bagherzadeh, M., & Khatchadourian, R. (2019). Going big: A large-scale study on what big data developers ask. In *Proceedings of the 2019 27th acm joint meeting on european software engineering conference and symposium on the foundations of software engineering* (p. 432-442). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/3338906.3338939 doi: 10.1145/3338906.3338939

Bajaj, K., Pattabiraman, K., & Mesbah, A. (2014). Mining questions asked by web developers.

In *Proceedings of the 11th working conference on mining software repositories* (pp. 112–121). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/2597073.2597083 doi: 10.1145/2597073.2597083

Banerjee, S., & Cukic, B. (2015). On the cost of mining very large open source repositories. In *Proceedings of the first international workshop on big data software engineering* (pp. 37–43). Piscataway, NJ, USA: IEEE Press. Retrieved from http://dl.acm.org/citation.cfm?id=2819289.2819301

Bankier, J. G., & Gleason, K. (2014). *Institutional repository software comparison.* Unesco.

Barash, G., Farchi, E., Jayaraman, I., Raz, O., Tzoref-Brill, R., & Zalmanovici, M. (2019). Bridging the gap between ml solutions and their business requirements using feature interactions. In *Proceedings of the 2019 27th acm joint meeting on european software engineering conference and symposium on the foundations of software engineering* (p. 10481058). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/3338906.3340442 doi: 10.1145/3338906.3340442

Barua, A., Thomas, S. W., & Hassan, A. E. (2014, Jun 01). What are developers talking about? an analysis of topics and trends in stack overflow. *Empirical Software Engineering*, *19*(3), 619–654.

Basu, A., & Banerjee, K. (2021, jun). Designing a bot for efficient distribution of service requests. In *2021 ieee/acm third international workshop on bots in software engineering (botse)* (p. 16-20). Los Alamitos, CA, USA: IEEE Computer Society. Retrieved from https://doi.ieeecomputersociety.org/10.1109/BotSE52550.2021.00011 doi: 10.1109/BotSE52550.2021.00011

Begel, A., Khoo, Y. P., & Zimmermann, T. (2010, May). Codebook: discovering and exploiting relationships in software repositories. In *2010 acm/ieee 32nd international conference on software engineering* (Vol. 1, p. 125-134). doi: 10.1145/1806799.1806821

Begel, A., & Zimmermann, T. (2014a). Analyze this! 145 questions for data scientists in software engineering. In *Proceedings of the 36th international conference on software engineering* (pp. 12–23). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/2568225.2568233 doi: 10.1145/2568225.2568233

Begel, A., & Zimmermann, T. (2014b). *Appendix to analyze this! 145 questions for data scientists in software engineering - microsoft research.* https://www.microsoft.com/en-us/research/publication/appendix-to-analyze-this-145-questions-for-data-scientists-in-software-engineering. ((Accessed on 12/20/2018))

Beschastnikh, I., Lungu, M. F., & Zhuang, Y. (2017). Accelerating software engineering research adoption with analysis bots. In *Proceedings of the 39th international conference on software engineering: New ideas and emerging results track* (p. 3538). IEEE Press. Retrieved from https://doi.org/10.1109/ICSE-NIER.2017.17 doi: 10.1109/ICSE-NIER.2017.17

Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003, March). Latent dirichlet allocation. *The Journal of Machine Learning Research*, *3*, 993–1022. Retrieved from http://dl.acm.org/citation.cfm?id=944919.944937

Boiteux, M. (2018). *Messenger at f8 2018 - messenger developer blog.* https://blog.messengerdevelopers.com/messenger-at-f8-2018-44010dc9d2ea. ((Accessed on 12/21/2019))

BotSE. (2019). *1st international workshop on bots in software engineering.* http://botse.org/. ((Accessed on 11/19/2019))

Bradley, N. C., Fritz, T., & Holmes, R. (2018). Context-aware conversational developer assistants. In *Proceedings of the 40th international conference on software engineering* (p. 9931003). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/3180155.3180238 doi: 10.1145/3180155.3180238

Braun, D., Hernandez Mendez, A., Matthes, F., & Langen, M. (2017, August). Evaluating natural language understanding services for conversational question answering systems. In *Proceedings of the 18th annual SIGdial meeting on discourse and dialogue* (pp. 174–185). Saarbrücken, Germany: Association for Computational Linguistics. Retrieved from https://aclanthology.org/W17-5522 doi: 10.18653/v1/W17-5522

Braun, D., Hernandez-Mendez, A., Matthes, F., & Langen, M. (2017, August). Evaluating natural

language understanding services for conversational question answering systems. In *Proceedings of the 18th annual sigdial meeting on discourse and dialogue* (pp. 174–185). Saarbrcken, Germany: Association for Computational Linguistics. Retrieved from http://www.aclweb.org/anthology/W17-3622

Braun, V., & Clarke, V. (2006). Using thematic analysis in psychology. *Qualitative research in psychology*, *3*(2), 77–101.

Brundage, M., Avin, S., Wang, J., Belfield, H., Krueger, G., Hadfield, G., ... Anderljung, M. (2020). *Toward trustworthy ai development: Mechanisms for supporting verifiable claims.*

Cameron, G., Cameron, D., Megaw, G., Bond, R., Mulvenna, M., O'Neill, S., ... McTear, M. (2018). Best practices for designing chatbots in mental healthcare: A case study on ihelpr. In *Proceedings of the 32nd international bcs human computer interaction conference.* Swindon, GBR: BCS Learning & Development Ltd. Retrieved from https://doi.org/10.14236/ewic/HCI2018.129 doi: 10.14236/ewic/HCI2018.129

Canonico, M., & De Russis, L. (2018). A comparison and critique of natural language understanding tools. *CLOUD COMPUTING 2018*, 120.

Care, P. (2020). *Florence - your health assistant.* https://www.florence.chat/. ((Accessed on 01/08/2020))

Carvalho, A., Luz, W., Marcilio, D., Bonifácio, R., Pinto, G., & Canedo, E. D. (2020). C-3pr: A bot for fixing static analysis violations via pull requests.

Casagrande, M. (2019). *node.js - how to store and retrieve the chat history of the dialogflow? - stack overflow.* https://stackoverflow.com/questions/49665510/how-to-store-and-retrieve-the-chat-history-of-the-dialogflow. ((Accessed on 12/21/2019))

Cerf, V. (1973). *Rfc0439: Parry encounters the doctor.* USA: RFC Editor.

Chen, G., Chen, C., Xing, Z., & Xu, B. (2016, Sep.). Learning a dual-language vector space for domain-specific cross-lingual question retrieval. In *2016 31st ieee/acm international conference on automated software engineering (ase)* (p. 744-755).

Chun-Ting Lin, S.-P. M., & Huang, Y.-W. (2020). Msabot: A chatbot framework for assisting in the development and operation of microservice-based systems. In *Proceedings of the 2nd*

international workshop on bots in software engineering. IEEE Press.

Cliff, N. (1993). Dominance statistics: Ordinal analyses to answer ordinal questions. *Psychological Bulletin*, *114*, 494-509.

Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, *20*(1), 37-46. doi: 10.1177/001316446002000104

Conference, T. C. (2019). *The chatbot conference.* https://www.chatbotconference.com/. ((Accessed on 11/09/2019))

Cornu, B., Durieux, T., Seinturier, L., & Monperrus, M. (2015). Npefix: Automatic runtime repair of null pointer exceptions in java. *CoRR*, *abs/1512.07423*. Retrieved from http://arxiv.org/abs/1512.07423

Cruzes, D. S., & Dyba, T. (2011, Sep.). Recommended steps for thematic synthesis in software engineering. In *2011 international symposium on empirical software engineering and measurement* (p. 275-284). doi: 10.1109/ESEM.2011.36

Damir. (2020, 03). *Natural language processing.* https://stackoverflow.com/questions/4115526/natural-language-processing. ((Accessed on 03/09/2020))

Daniel, G., & Cabot, J. (2021). The software challenges of building smart chatbots. In *2021 ieee/acm 43rd international conference on software engineering: Companion proceedings (icse-companion)* (p. 324-325). doi: 10.1109/ICSE-Companion52605.2021.00138

Daniel, G., Cabot, J., Deruelle, L., & Derras, M. (2020). Xatkit: A multimodal low-code chatbot development framework. *IEEE Access*, *8*, 15332-15346.

Dependabot. (2020). *Dependabot.* https://dependabot.com/. ((Accessed on 08/26/2020))

DeployBot. (2020). *Code deployment tools.* https://deploybot.com/. ((Accessed on 08/27/2020))

Dialogflow. (2020a). *Developer entities.* https://dialogflow.com/docs/entities/developer-entities. ((Accessed on 03/10/2020))

Dialogflow. (2020b). *History — dialogflow documentation.* https://cloud.google.com/dialogflow/docs/history. ((Accessed on 02/07/2020))

Dialogflow. (2020c). *Training phrases.* https://dialogflow.com/docs/intents/

training-phrases. ((Accessed on 02/24/2020))

Dialogflow, G. (2020). *Build an agent.* https://cloud.google.com/dialogflow/docs/quick/build-agent. ((Accessed on 01/16/2020))

Digkas, G., Lungu, M., Chatzigeorgiou, A., & Avgeriou, P. (2017). The evolution of technical debt in the apache ecosystem. In *Software architecture* (pp. 51–66). Cham: Springer International Publishing.

Docs, M. (2020). *Good example utterances.* https://docs.microsoft.com/en-us/azure/cognitive-services/luis/luis-concept-utterance. ((Accessed on 09/30/2021))

Dominic, J., Houser, J., Steinmacher, I., Ritter, C., & Rodeghero, P. (2020a). Conversational bot for newcomers onboarding to open source projects. In *Proceedings of the ieee/acm 42nd international conference on software engineering workshops* (p. 4650). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/3387940.3391534 doi: 10.1145/3387940.3391534

Dominic, J., Houser, J., Steinmacher, I., Ritter, C., & Rodeghero, P. (2020b). Conversational bot for newcomers onboarding to open source projects. In *Proceedings of the 2nd international workshop on bots in software engineering.* IEEE Press.

Dopierre, T., Gravier, C., & Logerais, W. (2021). Protaugment: Unsupervised diverse short-texts paraphrasing for intent detection meta-learning. In *The 59th annual meeting of the association for computational linguistics and the 11th international joint conference on natural language processing* (Vol. abs/2105.12995).

"Dutta, S., Joyce, G., & Brewer, J. ("2018"). "utilizing chatbots to increase the efficacy of information security practitioners". In D. "Nicholson (Ed.), *"advances in human factors in cybersecurity"* (pp. "237–243"). "Springer International Publishing".

Efstathiou, V., Chatzilenas, C., & Spinellis, D. (2018). Word embeddings for the software engineering domain. In *Proceedings of the 15th international conference on mining software repositories* (p. 3841). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/3196398.3196448 doi: 10.1145/3196398.3196448

Epskamp, S. (2019a). Reproducibility and replicability in a fast-paced methodological world.

Advances in Methods and Practices in Psychological Science, *2*(2), 145-155. Retrieved from https://doi.org/10.1177/2515245919847421 doi: 10.1177/2515245919847421

Epskamp, S. (2019b). Reproducibility and replicability in a fast-paced methodological world. *Advances in Methods and Practices in Psychological Science*, *2*(2), 145-155. doi: 10.1177/2515245919847421

Erlenhov, L., de Oliveira Neto, F. G., Scandariato, R., & Leitner, P. (2019). Current and future bots in software development. In *Proceedings of the 1st international workshop on bots in software engineering* (p. 711). IEEE Press. Retrieved from https://doi.org/10.1109/BotSE.2019.00009 doi: 10.1109/BotSE.2019.00009

Erlenhov, L., Neto, F. G. d. O., & Leitner, P. (2020). An empirical study of bots in software development–characteristics and challenges from a practitioner's perspective. In *Proceedings of the 2020 27th acm sigsoft international symposium on foundations of software engineering*.

Exchange, S. (2019). *Stack exchange data dump.* https://archive.org/details/stackexchange. ((Sept. 2019))

Facebook. (2019). *Wit.ai.* https://wit.ai/. ((Accessed on 12/12/2019))

Feng, D., Shaw, E., Kim, J., & Hovy, E. (2006). An intelligent discussion-bot for answering student queries in threaded discussions. In *Proceedings of the 11th international conference on intelligent user interfaces* (pp. 171–177). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/1111449.1111488 doi: 10.1145/1111449.1111488

Fleiss, J. L., & Cohen, J. (1973). The equivalence of weighted kappa and the intraclass correlation coefficient as measures of reliability. *Educational and Psychological Measurement*, *33*(3), 613-619. doi: 10.1177/001316447303300309

Fritz, T., & Murphy, G. C. (2010). Using information fragments to answer the questions developers ask. In *Proceedings of the 32nd acm/ieee international conference on software engineering - volume 1* (pp. 175–184). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/1806799.1806828 doi: 10.1145/1806799.1806828

G, C. N. (2019). *botframework - how to add custom choices displayed through prompt*

options inside cards & trigger actions on choice click in bot v4 using c#? - stack overflow. (https://stackoverflow.com/questions/56280689/how-to-add-custom-choices-displayed-through-prompt-options-inside-cards-trigge(Accessed on 01/16/2020))

G, T. (2016). *Artificial intelligence - comparison between luis.ai vs api.ai vs wit.ai?* https://stackoverflow.com/questions/37215188/comparison-between-luis-ai-vs-api-ai-vs-wit-ai. ((Accessed on 04/11/2020))

Gao, J., Chen, J., Zhang, S., He, X., & Lin, S. (2019, March). Recognizing biomedical named entities by integrating domain contextual relevance measurement and active learning. In *2019 ieee 3rd information technology, networking, electronic and automation control conference (itnec)* (p. 1495-1499). doi: 10.1109/ITNEC.2019.8728991

Gensim. (2019). *gensim: Topic modelling for humans.* https://radimrehurek.com/gensim/. ((Accessed on 12/03/2019))

Git client - glo boards — gitkraken. (2019). https://www.gitkraken.com/. ((Accessed on 03/04/2019))

Google. (2019a). *Dialogflow.* https://dialogflow.com/. ((Accessed on 01/09/2019))

Google. (2019b). *query - dialogflow.* https://dialogflow.com/docs/reference/agent/query. ((Accessed on 10/10/2019))

Google. (2019c). *Training — dialogflow.* https://dialogflow.com/docs/training. ((Accessed on 02/16/2019))

Google. (2020a). *Dialogflow.* https://dialogflow.com/. ((Accessed on 01/16/2020))

Google. (2020b). *Dialogflow.* https://dialogflow.com/. ((Accessed on 02/05/2020))

Google. (2020). *Dialogflow agent validation.* https://cloud.google.com/dialogflow/es/docs/agents-validation. ((Accessed on 09/02/2020))

Google. (2020a). *Google assistant, your own personal google.* https://assistant.google.com/. ((Accessed on 01/08/2020))

Google. (2020b). *Integrations-dialogflow documentation.* https://cloud.google.com/dialogflow/docs/integrations/. ((Accessed on 01/16/2020))

Greenkeeper. (2019). *Automate your npm dependency management.* https://greenkeeper.io/. ((Accessed on 03/10/2020))

Gregori, E. (2017). Evaluation of modern tools for an omscs advisor chatbot.

Gupta, M., Sureka, A., & Padmanabhuni, S. (2014). Process mining multiple repositories for software defect resolution from control and organizational perspective. In *Proceedings of the 11th working conference on mining software repositories* (pp. 122–131). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/2597073.2597081 doi: 10.1145/2597073.2597081

Han, J., Pei, J., & Kamber, M. (2011). *Data mining: Concepts and techniques*. Elsevier Science. Retrieved from https://books.google.ca/books?id=pQws07tdpjoC

Han, J., Shihab, E., Wan, Z., Den, S., & Xia, X. (2019). What do programmers discuss about deep learning frameworks. *Empirical Software Engineering (EMSE)*, To Appear.

Hassan, A. E. (2008, Sept). The road ahead for mining software repositories. In *2008 frontiers of software maintenance* (p. 48-57). doi: 10.1109/FOSM.2008.4659248

Höst, M., Regnell, B., & Wohlin, C. (2000, Nov 01). Using students as subjects—a comparative study of students and professionals in lead-time impact assessment. *Empirical Software Engineering*, *5*(3), 201–214. Retrieved from https://doi.org/10.1023/A:1026586415054 doi: 10.1023/A:1026586415054

Hovel, M. (2017). *node.js - how to start a conversation from nodejs client to microsoft bot - stack overflow.* https://stackoverflow.com/questions/46183295/how-to-start-a-conversation-from-nodejs-client-to-microsoft-bot. ((Accessed on 12/20/2019))

IBM. (2019a). *IBM Watson.* https://www.ibm.com/watson. ((Accessed on 11/20/2019))

IBM. (2019b). *Watson assistant v1.* https://cloud.ibm.com/apidocs/assistant-v1. ((Accessed on 11/18/2019))

IBM. (2019c). *Watson conversation.* https://www.ibm.com/watson/services/conversation/. ((Accessed on 01/09/2019))

IBM. (2020a). *Creating entities.* https://cloud.ibm.com/docs/services/assistant?topic=assistant-entities#entities. ((Accessed on

03/11/2020))

IBM. (2020b). *Defining intents.* https://cloud.ibm.com/docs/services/assistant?topic=assistant-intents#intents-entity-references. ((Accessed on 02/24/2020))

IBM. (2020c). *Entities.* https://cloud.ibm.com/docs/services/assistant-icp?topic=assistant-private-entities#entity-described. ((Accessed on 01/09/2020))

Ilmania, A., Abdurrahman, Cahyawijaya, S., & Purwarianti, A. (2018). Aspect detection and sentiment classification using deep neural network for indonesian aspect-based sentiment analysis. In *2018 international conference on asian language processing (ialp)* (p. 62-67). doi: 10.1109/IALP.2018.8629181

The impact of conversational bots in the customer experience - good rebels. (2020). https://www.goodrebels.com/theimpactofconversationalbotsinthecustomerexperience. ((Accessed on 08/05/2020))

Jha, S. (2019, June). *Chatbot application life cycle - data driven investor - medium.* https://medium.com/datadriveninvestor/chatbot-application-life-cycle-8b2d083650a8. ((Accessed on 12/16/2019))

Jira client — atlassian marketplace. (2019). https://marketplace.atlassian.com/apps/7070/jira-client?hosting=server&tab=overview. ((Accessed on 03/04/2019))

Jurafsky, D., & Martin, J. H. (2009). *Speech and language processing (2nd edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc.

Kabinna, S., Bezemer, C.-P., Shang, W., & Hassan, A. E. (2016). Logging library migrations: A case study for the apache software foundation projects. In *Proceedings of the 13th international conference on mining software repositories* (pp. 154–164). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/2901739.2901769 doi: 10.1145/2901739.2901769

Khomh, F., Adams, B., Dhaliwal, T., & Zou, Y. (2015, Apr 01). Understanding the impact of rapid releases on software quality. *Empirical Software Engineering*, *20*(2), 336–373. Retrieved from https://doi.org/10.1007/s10664-014-9308-x doi: 10.1007/s10664-014-9308-x

Koetter, F., Blohm, M., Kochanowski, M., Goetzer, J., Graziotin, D., & Wagner, S. (2018). Motivations, classification and model trial of conversational agents for insurance companies. In *11th international conference on agents and artificial intelligence.*

Kubernetes. (2016). *k8s-ci-robot (kubernetes prow robot).* https://github.com/k8s-ci-robot. ((Accessed on 03/10/2020))

Kumar, R., Bansal, C., Maddila, C., Sharma, N., Martelock, S., & Bhargava, R. (2019). Building sankie: An ai platform for devops. In *Proceedings of the 1st international workshop on bots in software engineering* (pp. 48–53). Piscataway, NJ, USA: IEEE Press. Retrieved from https://doi.org/10.1109/BotSE.2019.00020 doi: 10.1109/BotSE.2019.00020

Larson, S., Mahendran, A., Peper, J. J., Clarke, C., Lee, A., Hill, P., ... Mars, J. (2019, November). An evaluation dataset for intent classification and out-of-scope prediction. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (emnlp-ijcnlp)* (pp. 1311–1316). Hong Kong, China: Association for Computational Linguistics. Retrieved from https://aclanthology.org/D19-1131 doi: 10.18653/v1/D19-1131

Lastra, D. (2016). *The impact of conversational bots in the customer experience.* https://www.goodrebels.com/the-impact-of-conversational-bots-in-the. ((Accessed on 04/21/2020))

Lebeuf, C., Storey, M., & Zagalsky, A. (2018c, January). Software bots. In (Vol. 35, p. 18-23). doi: 10.1109/MS.2017.4541027

Lebeuf, C., Storey, M., & Zagalsky, A. (2018d, January/February). Software bots. *IEEE Software*, *35*(1), 18-23. Retrieved from doi.ieeecomputersociety.org/10.1109/MS.2017.4541027 doi: 10.1109/MS.2017.4541027

Lebeuf, C., Storey, M.-A., & Zagalsky, A. (2018a). Software bots. *IEEE Software*, *35*(1), 18-23.

doi: 10.1109/MS.2017.4541027

Lebeuf, C., Storey, M.-A., & Zagalsky, A. (2018b). Software bots. *IEEE Software*, *35*(1), 18-23. doi: 10.1109/MS.2017.4541027

Lebeuf, C., Zagalsky, A., Foucault, M., & Storey, M.-A. (2019a). Defining and classifying software bots: A faceted taxonomy. In *Proceedings of the 1st international workshop on bots in software engineering* (p. 16). IEEE Press. doi: 10.1109/BotSE.2019.00008

Lebeuf, C., Zagalsky, A., Foucault, M., & Storey, M.-A. (2019b). Defining and classifying software bots: A faceted taxonomy. In *Proceedings of the 1st international workshop on bots in software engineering* (p. 16). IEEE Press. Retrieved from https://doi.org/10.1109/BotSE.2019.00008 doi: 10.1109/BotSE.2019.00008

Lebeuf, C. R. (2018). *A taxonomy of software bots: towards a deeper understanding of software bot characteristics* (Unpublished doctoral dissertation).

Lenberg, P., Feldt, R., & Wallgren, L. G. (2015). Human factors related challenges in software engineering: An industrial perspective. In *Proceedings of the eighth international workshop on cooperative and human aspects of software engineering* (pp. 43–49). Piscataway, NJ, USA: IEEE Press.

Levenshtein, V. I. (1966, February). Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, *10*, 707.

Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., ... Zettlemoyer, L. (2020, July). BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th annual meeting of the association for computational linguistics* (pp. 7871–7880). Online: Association for Computational Linguistics. Retrieved from https://aclanthology.org/2020.acl-main.703 doi: 10.18653/v1/2020.acl-main.703

Liu, X., Zhang, S., Wei, F., & Zhou, M. (2011). Recognizing named entities in tweets. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies - volume 1* (pp. 359–367). Stroudsburg, PA, USA: Association for Computational Linguistics. Retrieved from http://dl.acm.org/citation.cfm?id=2002472.2002519

Luis (language understanding) cognitive services microsoft azure. (2019). https://www.luis.ai/home. ((Accessed on 02/20/2019))

Malandrakis, N., Shen, M., Goyal, A., Gao, S., Sethi, A., & Metallinou, A. (2019, November). Controlled text generation for data augmentation in intelligent artificial agents. In *Proceedings of the 3rd workshop on neural generation and translation* (pp. 90–98). Hong Kong: Association for Computational Linguistics. Retrieved from https://aclanthology.org/D19-5609 doi: 10.18653/v1/D19-5609

Mamykina, L., Manoim, B., Mittal, M., Hripcsak, G., & Hartmann, B. (2011). Design lessons from the fastest QA site in the west. In *Proceedings of the sigchi conference on human factors in computing systems* (pp. 2857–2866). New York, NY, USA: ACM. doi: 10.1145/1978942.1979366

Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S. J., & McClosky, D. (2014). The Stanford CoreNLP natural language processing toolkit. In *Association for computational linguistics (acl) system demonstrations* (pp. 55–60). Retrieved from http://www.aclweb.org/anthology/P/P14/P14-5010

Marbot. (2019). *marbot - incident management for aws.* https://marbot.io/. ((Accessed on 12/18/2019))

Marbot. (2020). *Chatbot for aws monitoring.* https://marbot.io/. ((Accessed on 08/20/2020))

Marivate, V., & Sefara, T. (2020). Improving short text classification through global augmentation methods. In A. Holzinger, P. Kieseberg, A. M. Tjoa, & E. Weippl (Eds.), *Machine learning and knowledge extraction* (pp. 385–399). Cham: Springer International Publishing.

Martin, L., Fan, A., de la Clergerie, É., Bordes, A., & Sagot, B. (2021). Muss: Multilingual unsupervised sentence simplification by mining paraphrases. *arXiv preprint arXiv:2005.00352*.

Martinez, M., & Monperrus, M. (2016). Astor: A program repair library for java (demo). In *Proceedings of the 25th international symposium on software testing and analysis* (p. 441444). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/2931037.2948705 doi: 10.1145/2931037.2948705

Mastropaolo, A., Scalabrino, S., Cooper, N., Nader Palacio, D., Poshyvanyk, D., Oliveto, R., &

Bavota, G. (2021). Studying the usage of text-to-text transfer transformer to support code-related tasks. In *2021 ieee/acm 43rd international conference on software engineering (icse)* (p. 336-347). doi: 10.1109/ICSE43902.2021.00041

Matthies, C., Dobrigkeit, F., & Hesse, G. (2019). An additional set of (automated) eyes: Chatbots for agile retrospectives. In *Proceedings of the 1st international workshop on bots in software engineering* (p. 3437). IEEE Press. Retrieved from https://doi.org/10.1109/BotSE.2019.00017 doi: 10.1109/BotSE.2019.00017

McCallum, A. K. (2002). *Mallet: A machine learning for language toolkit.* http://mallet.cs.umass.edu/. ((Accessed on 12/03/2019))

McHugh, M. (2012, 10). Interrater reliability: The kappa statistic. *Biochemia medica*, *22*, 276-82. doi: 10.11613/BM.2012.031

Microsoft. (2019a). *Good example utterances - language understanding - azure cognitive services.* https://docs.microsoft.com/en-us/azure/cognitive-services/luis/luis-concept-utterance. ((Accessed on 11/19/2019))

Microsoft. (2019b). *Luis: Language understanding intelligent service.* https://www.luis.ai/home. ((Accessed on 01/09/2019))

Microsoft. (2019c). *Prediction scores - language understanding - azure cognitive services.* https://docs.microsoft.com/en-us/azure/cognitive-services/luis/luis-concept-prediction-score. ((Accessed on 11/10/2019))

Microsoft. (2020a). *Entity types - language understanding - azure cognitive services.* https://docs.microsoft.com/en-us/azure/cognitive-services/luis/luis-concept-entity-types. ((Accessed on 03/10/2020))

Microsoft. (2020b). *Microsoft bot framework.* https://dev.botframework.com/. ((Accessed on 01/16/2020))

Microsoft. (2020c). *Quickstart: Create a new app in the luis portal - azure cognitive services — microsoft docs.* https://docs.microsoft.com/en-us/azure/cognitive-services/luis/get-started-portal-build-app. ((Accessed on 01/16/2020))

Microsoft. (2021a, 05). *Language understanding - bot service.* https://docs

.microsoft.com/en-us/azure/bot-service/bot-builder-concept-luis?view=azure-bot-service-4.0#best-practices-for-language-understanding. ((Accessed on 07/09/2021))

Microsoft. (2021b, 09). *Luis (language understanding) - cognitive services.* https://www.luis.ai/. ((Accessed on 07/09/2021))

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th international conference on neural information processing systems - volume 2* (pp. 3111–3119). USA: Curran Associates Inc. Retrieved from http://dl.acm.org/citation.cfm?id=2999792.2999959

Milenkovic, M. (2019, Oct.). *The future is now - 37 fascinating chatbot statistics.* https://www.smallbizgenius.net/by-the-numbers/chatbot-statistics/. ((Accessed on 12/18/2019))

Miller, G. A. (1995, November). Wordnet: A lexical database for english. *Commun. ACM*, *38*(11), 3941. Retrieved from https://doi.org/10.1145/219717.219748 doi: 10.1145/219717.219748

Mohit, B. (2014). Named entity recognition. In I. Zitouni (Ed.), *Natural language processing of semitic languages.* Springer, USA.

Monperrus, M. (2019). Explainable software bot contributions: Case study of automated bug fixes. In *Proceedings of the 1st international workshop on bots in software engineering* (p. 1215). IEEE Press. Retrieved from https://doi.org/10.1109/BotSE.2019.00010 doi: 10.1109/BotSE.2019.00010

Mordinyi, R., & Biffl, S. (2017, Sept). Exploring traceability links via issues for detailed requirements coverage reports. In *2017 ieee 25th international requirements engineering conference workshops (rew)* (p. 359-366). doi: 10.1109/REW.2017.69

Munir, H., Wnuk, K., & Runeson, P. (2016, Apr 01). Open innovation in software engineering: a systematic mapping study. *Empirical Software Engineering*, *21*(2), 684–723. Retrieved from https://doi.org/10.1007/s10664-015-9380-x doi: 10.1007/s10664-015-9380-x

Munoz, S., Araque, O., Llamas, A. F., & Iglesias, C. A. (2018, Aug). A cognitive agent for mining bugs reports, feature suggestions and sentiment in a mobile application store. In *2018 4th international conference on big data innovations and applications (innovate-data)* (p. 17-24). doi: 10.1109/Innovate-Data.2018.00010

Murgia, A., Janssens, D., Demeyer, S., & Vasilescu, B. (2016a). Among the machines: Human-bot interaction on social q&a websites. In *Proceedings of the 2016 chi conference extended abstracts on human factors in computing systems* (pp. 1272–1279). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/2851581.2892311 doi: 10.1145/2851581.2892311

Murgia, A., Janssens, D., Demeyer, S., & Vasilescu, B. (2016b). Among the machines: Human-bot interaction on social q&a websites. In *Proceedings of the 2016 chi conference extended abstracts on human factors in computing systems* (pp. 1272–1279). New York, NY, USA: ACM. doi: 10.1145/2851581.2892311

Nadi, S., Krüger, S., Mezini, M., & Bodden, E. (2016). Jumping through hoops: Why do java developers struggle with cryptography apis? In *Proceedings of the 38th international conference on software engineering* (pp. 935–946). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/2884781.2884790 doi: 10.1145/2884781.2884790

Ni, L., Lu, C., Liu, N., & Liu, J. (2017). Mandy: Towards a smart primary care chatbot application. In *Knowledge and systems sciences* (pp. 38–52). Singapore: Springer Singapore.

Nick. (2018). *NLP - build chatbot for education purporse.* https://stackoverflow.com/questions/52206324/build-chatbot-for-education-purporse. ((Accessed on 11/25/2019))

(NLTK), N. L. T. (2019a). *Natural language toolkit - nltk 3.4.5 documentation.* https://www.nltk.org/. ((Accessed on 12/12/2019))

(NLTK), N. L. T. (2019b). *Nltk's list of english stopwords.* https://gist.github.com/sebleier/554280. ((Accessed on 12/23/2019))

Overflow, S. (2017). *python - dataset to train mitie ner model - stack overflow.* https://stackoverflow.com/questions/46602495/dataset-to

-train-mitie-ner-model. ((Accessed on 01/13/2020))

Overflow, S. (2019a). *nlp - is there a dataset that provides shopping conversations? - stack overflow.* https://stackoverflow.com/questions/55324833/is-there-a-dataset-that-provides-shopping-conversations. ((Accessed on 01/13/2020))

Overflow, S. (2019b). *Stack overflow developer survey 2019.* https://insights.stackoverflow.com/survey/2019. ((Accessed on 01/09/2020))

Paikari, E., Choi, J., Kim, S., Baek, S., Kim, M., Lee, S., ... et al. (2019). A chatbot for conflict detection and resolution. In *Proceedings of the 1st international workshop on bots in software engineering* (p. 2933). IEEE Press. doi: 10.1109/BotSE.2019.00016

Paikari, E., & van der Hoek, A. (2018). A framework for understanding chatbots and their future. In *2018 ieee/acm 11th international workshop on cooperative and human aspects of software engineering (chase)* (p. 13-16).

Phaithoon, S., Wongnil, S., Pussawong, P., Choetkiertikul, M., Ragkhitwetsagul, C., Sunetnanta, T., ... Matsumoto, K. (2021). Fixme: A github bot for detecting and monitoring on-hold self-admitted technical debt. In *Proceedings of the 36th ieee/acm international conference on automated software engineering (ase'21).*

Qasse, I. A., Mishra, S., & Hamdaqa, M. (2021). icontractbot: A chatbot for smart contracts' specification and code generation. In *2021 ieee/acm third international workshop on bots in software engineering (botse).*

Rasa. (2019a). *Confidence and fallback intents.* https://rasa.com/docs/rasa/api/http-api/. ((Accessed on 11/18/2019))

Rasa. (2019b). *Duckling.* https://duckling.wit.ai/. ((Accessed on 12/07/2019))

Rasa. (2019c). *multiple entity recognition issue #427.* https://github.com/RasaHQ/rasa_nlu/issues/427. ((Accessed on 11/19/2019))

Rasa. (2019d). *rasa/crf_entity_extractor.py at master rasahq/rasa.* https://github.com/RasaHQ/rasa/blob/master/rasa/nlu/extractors/crf_entity_extractor.py. ((Accessed on 11/25/2019))

Rasa. (2020a). *Frequently asked questions.* https://rasa.com/docs/nlu/faq/. ((Accessed on 02/18/2020))

Rasa. (2020b). *Rasa: Open source conversational ai.* https://rasa.com/. ((Accessed on 02/20/2020))

Rasa. (2020c). *Training data format.* https://rasa.com/docs/nlu/0.13.8/dataformat/. ((Accessed on 03/10/2020))

Rasa. (2021, 06). *Introduction to rasa x.* https://rasa.com/docs/rasa-x/. ((Accessed on 07/09/2021))

Rasa. (August, 2021). *Open source conversational ai — rasa.* https://rasa.com/. ((Accessed on 08/22/2021))

Ratan. (2017). *Rasa nlu parse request giving wrong intent result - stack overflow.* https://stackoverflow.com/questions/46466222/rasa-nlu-parse-request-giving-wrong-intent-result. ((Accessed on 12/20/2019))

Ratinov, L., & Roth, D. (2009). Design challenges and misconceptions in named entity recognition. In *Proceedings of the thirteenth conference on computational natural language learning* (pp. 147–155). Stroudsburg, PA, USA: Association for Computational Linguistics. Retrieved from http://dl.acm.org/citation.cfm?id=1596374.1596399

Rizos, G., Hemker, K., & Schuller, B. (2019). Augment to prevent: Short-text data augmentation in deep learning for hate-speech classification. In *Proceedings of the 28th acm international conference on information and knowledge management* (p. 9911000). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/3357384.3358040 doi: 10.1145/3357384.3358040

Robillard, M. P., Marcus, A., Treude, C., Bavota, G., Chaparro, O., Ernst, N., ... Wong, E. (2017, Sept). On-demand developer documentation. In *2017 ieee international conference on software maintenance and evolution (icsme)* (p. 479-483). doi: 10.1109/ICSME.2017.17

Röder, M., Both, A., & Hinneburg, A. (2015). Exploring the space of topic coherence measures. In *Proceedings of the eighth acm international conference on web search and data mining* (pp. 399–408). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/2684822.2685324 doi: 10.1145/2684822.2685324

Romano, J., Kromrey, J. D., Coraggio, J., & Skowronek, J. (2006). Appropriate statistics for ordinal level data: Should we really be using t-test and cohen'sd for evaluating group differences on the nsse and other surveys. In *annual meeting of the florida association of institutional research* (pp. 1–33).

Romero, R., Parra, E., & Haiduc, S. (2020a). Experiences building an answer bot for gitter. In *Proceedings of the 2nd international workshop on bots in software engineering*. IEEE Press.

Romero, R., Parra, E., & Haiduc, S. (2020b). Experiences building an answer bot for gitter. In *Proceedings of the ieee/acm 42nd international conference on software engineering workshops* (p. 6670). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/3387940.3391505 doi: 10.1145/3387940.3391505

Rosen, C., & Shihab, E. (2016, June). What are mobile developers asking about? a large scale study using stack overflow. *Empirical Software Engineering*, *21*(3), 1192–1223. Retrieved from http://dx.doi.org/10.1007/s10664-015-9379-3 doi: 10.1007/s10664-015-9379-3

Rychalska, B., Glabska, H., & Wroblewska, A. (2018a, Oct). Multi-intent hierarchical natural language understanding for chatbots. In *2018 fifth international conference on social networks analysis, management and security (snams)* (p. 256-259). doi: 10.1109/SNAMS.2018.8554770

Rychalska, B., Glabska, H., & Wroblewska, A. (2018b, Oct). Multi-intent hierarchical natural language understanding for chatbots. In *2018 fifth international conference on social networks analysis, management and security (snams)* (p. 256-259). doi: 10.1109/SNAMS.2018.8554770

Salman, I., Misirli, A. T., & Juristo, N. (2015, May). Are students representatives of professionals in software engineering experiments? In *2015 ieee/acm 37th ieee international conference on software engineering* (Vol. 1, p. 666-676). doi: 10.1109/ICSE.2015.82

Sankar, G. R., Greyling, J., Vogts, D., & du Plessis, M. C. (2008). Models towards a hybrid conversational agent for contact centres. In *Proceedings of the 2008 annual research conference of the south african institute of computer scientists and information technologists on it research in developing countries: Riding the wave of technology* (pp. 200–209). New York, NY, USA:

ACM. Retrieved from http://doi.acm.org/10.1145/1456659.1456683 doi: 10.1145/1456659.1456683

Sawant, A. A., & Bacchelli, A. (2017, Jun 01). fine-grape: fine-grained api usage extractor – an approach and dataset to investigate api usage. *Empirical Software Engineering*, *22*(3), 1348–1371. Retrieved from https://doi.org/10.1007/s10664-016-9444-6 doi: 10.1007/s10664-016-9444-6

Şerban, D., Golsteijn, B., Holdorp, R., & Serebrenik, A. (2021, June 4). Saw-bot: Proposing fixes for static analysis warnings with github suggestions. In *Proceedings - 2021 ieee/acm 3rd international workshop on bots in software engineering, botse 2021* (pp. 26–30). United States: IEEE Computer Society. doi: DOI10.1109/BotSE52550.2021.00013

Sharma, V. S., Mehra, R., & Kaulgud, V. (2017). What do developers want?: An advisor approach for developer priorities. In *Proceedings of the 10th international workshop on cooperative and human aspects of software engineering* (pp. 78–81). Piscataway, NJ, USA: IEEE Press. Retrieved from https://doi.org/10.1109/CHASE.2017.14 doi: 10.1109/CHASE.2017.14

Shihab, A. A. D. C. K. B. R. A. E. (2020). *Dataset.* https://zenodo.org/record/3610714. ((Accessed on 01/16/2020))

Shihab, E., Jiang, Z. M., Adams, B., Hassan, A. E., & Bowerman, R. (2011). Prioritizing the creation of unit tests in legacy software systems. *Software: Practice and Experience*, *41*(10), 1027-1048. Retrieved from https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.1053 doi: https://doi.org/10.1002/spe.1053

Shridhar, K., Dash, A., Sahu, A., Pihlgren, G. G., Alonso, P., Pondenkandath, V., ... Liwicki, M. (2019, Jul). Subword semantic hashing for intent classification on small datasets. *2019 International Joint Conference on Neural Networks (IJCNN)*. Retrieved from http://dx.doi.org/10.1109/IJCNN.2019.8852420 doi: 10.1109/ijcnn.2019.8852420

Shridhar, K., Jain, H., Agarwal, A., & Kleyko, D. (2020, November). End to end binarized neural networks for text classification. In *Proceedings of sustainlp: Workshop on simple and efficient natural language processing* (pp. 29–34). Online: Association for Computational Linguistics. Retrieved from https://aclanthology.org/2020.sustainlp-1.4

doi: 10.18653/v1/2020.sustainlp-1.4

Shuvro, M. (2019). *python - how to resume or restart paused conversation in rasa - stack overflow.* https://stackoverflow.com/questions/57365685/how-to-resume-or-restart-paused-conversation-in-rasa. ((Accessed on 12/20/2019))

Siddiqui, T., & Ahmad, A. (2018). Data mining tools and techniques for mining software repositories: A systematic review. In V. B. Aggarwal, V. Bhatnagar, & D. K. Mishra (Eds.), *Big data analytics* (pp. 717–726). Singapore: Springer Singapore.

Sillito, J., Murphy, G. C., & Volder, K. D. (2008, July). Asking and answering questions during a programming change task. *IEEE Transactions on Software Engineering*, *34*(4), 434-451. doi: 10.1109/TSE.2008.26

Śliwerski, J., Zimmermann, T., & Zeller, A. (2005). When do changes induce fixes? In *Proceedings of the 2005 international workshop on mining software repositories* (pp. 1–5). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/1082983.1083147 doi: 10.1145/1082983.1083147

Snyk. (2019). *Snyk bot.* https://github.com/snyk-bot?tab=repositories. ((Accessed on 03/10/2020))

Spacy. (2021, July). *Industrial-strength natural language processing in python.* https://spacy.io/. ((Accessed on 07/03/2021))

Spearman. (2008). Spearman rank correlation coefficient. In *The concise encyclopedia of statistics* (pp. 502–505). New York, NY: Springer New York. Retrieved from https://doi.org/10.1007/978-0-387-32833-1_379 doi: 10.1007/978-0-387-32833-1_379

StackOverflow. (2019). *PHP mysqli query returns empty error message.* https://stackoverflow.com/questions/25941078/php-mysqli-query-returns-empty-error-message. ((Accessed on 07/20/2020))

StackOverflow. (2020). *Chatbot dialogflow returning answers with confidence score below ml classification threshold.* https://stackoverflow.com/questions/54218274/dialogflow-returning-answers-with-confidence-score-below-ml-classification-thres. ((Accessed on 08/19/2020))

StackOverflow. (2020). *Dialogflow matches irrelevant phrases to existing intents.*

https://stackoverflow.com/questions/49560851/dialogflow-matches-irrelevant-phrases-to-existing-intents. ((Accessed on 08/27/2020))

Storey, M.-A., & Zagalsky, A. (2016a). Disrupting developer productivity one bot at a time. In *Proceedings of the 2016 24th acm sigsoft international symposium on foundations of software engineering* (pp. 928–931). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/2950290.2983989 doi: 10.1145/2950290.2983989

Storey, M.-A., & Zagalsky, A. (2016b). Disrupting developer productivity one bot at a time. In *Proceedings of the 2016 24th acm sigsoft international symposium on foundations of software engineering* (pp. 928–931). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/2950290.2983989 doi: 10.1145/2950290.2983989

Storey, M.-A., & Zagalsky, A. (2016c). Disrupting developer productivity one bot at a time. In *Proceedings of the 2016 24th acm sigsoft international symposium on foundations of software engineering* (p. 928931). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/2950290.2983989 doi: 10.1145/2950290.2983989

Storey, M.-A., & Zagalsky, A. (2016d). Disrupting developer productivity one bot at a time. In *Proceedings of the 2016 24th acm sigsoft international symposium on foundations of software engineering* (p. 928931). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/2950290.2983989 doi: 10.1145/2950290.2983989

Sumo. (2020). *5 ecommerce chatbots (plus how to build your own in 15 minutes).* https://sumo.com/stories/ecommerce-chatbot-marketing. ((Accessed on 01/08/2020))

TechCrunch. (2017). *Wit.ai is shutting down bot engine as facebook rolls nlp into its updated messenger platform.* (https://techcrunch.com/2017/07/27/wit-ai-is-shutting-down-bot-engine-as-facebook-rolls-nlp-into-its-updated-messenger-platform (Accessed on 12/12/2019))

Tian, Y., Thung, F., Sharma, A., & Lo, D. (2017a). APIBot: Question answering bot for api

documentation. In *Proceedings of the 32nd ieee/acm international conference on automated software engineering* (pp. 153–158). IEEE Press.

Tian, Y., Thung, F., Sharma, A., & Lo, D. (2017b). Apibot: Question answering bot for api documentation. In *Proceedings of the 32nd ieee/acm international conference on automated software engineering* (pp. 153–158). Piscataway, NJ, USA: IEEE Press. Retrieved from http://dl.acm.org/citation.cfm?id=3155562.3155585

Tjong Kim Sang, E. F., & De Meulder, F. (2003). Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Proceedings of the seventh conference on natural language learning at hlt-naacl 2003 - volume 4* (pp. 142–147). Stroudsburg, PA, USA: Association for Computational Linguistics. Retrieved from https://doi.org/10.3115/1119176.1119195 doi: 10.3115/1119176.1119195

Tmbo. (2017). *multiple entity recognition issue #427 rasahq/rasa.* https://github.com/RasaHQ/rasa/issues/427. ((Accessed on 09/30/2021))

Toxtli, C., Monroy-Hernández, A., & Cranshaw, J. (2018). Understanding chatbot-mediated task management. In *Proceedings of the 2018 chi conference on human factors in computing systems* (pp. 58:1–58:6). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/3173574.3173632 doi: 10.1145/3173574.3173632

Treude, C., Barzilay, O., & Storey, M.-A. (2011). How do programmers ask and answer questions on the web? (nier track). In *Proceedings of the 33rd international conference on software engineering* (pp. 804–807). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/1985793.1985907 doi: 10.1145/1985793.1985907

Urli, S., Yu, Z., Seinturier, L., & Monperrus, M. (2018a). How to design a program repair bot?: Insights from the repairnator project. In *Proceedings of the 40th international conference on software engineering: Software engineering in practice* (pp. 95–104). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/3183519.3183540 doi: 10.1145/3183519.3183540

Urli, S., Yu, Z., Seinturier, L., & Monperrus, M. (2018b). How to design a program repair bot?: Insights from the repairnator project. In *Proceedings of the 40th international conference on software engineering: Software engineering in practice* (pp. 95–104). New York, NY, USA:

ACM. doi: 10.1145/3183519.3183540

Vale, L. d. N., & Maia, M. d. A. (2021). Towards a question answering assistant for software development using a transformer-based language model. In *2021 ieee/acm third international workshop on bots in software engineering (botse).*

Valtolina, S., Barricelli, B. R., & Gaetano, S. D. (2020). Communicability of traditional interfaces vs chatbots in healthcare and smart home domains. *Behaviour & Information Technology*, *39*(1), 108-132. Retrieved from https://doi.org/10.1080/0144929X.2019.1637025 doi: 10.1080/0144929X.2019.1637025

van Dyk, D. A., & Meng, X.-L. (2001). The art of data augmentation. *Journal of Computational and Graphical Statistics*, *10*(1), 1–50. Retrieved from http://www.jstor.org/stable/1391021

van Tonder, R., & Goues, C. L. (2019). Towards s/engineer/bot: Principles for program repair bots. In *Proceedings of the 1st international workshop on bots in software engineering* (p. 4347). IEEE Press. Retrieved from https://doi.org/10.1109/BotSE.2019.00019 doi: 10.1109/BotSE.2019.00019

Vasconcelos, M., Candello, H., Pinhanez, C., & dos Santos, T. (2017, 10). Boa: A language and infrastructure for analyzing ultra-large-scale software repositories. In *Brazilian symposium on human factors in computing systems.*

von der Mosel, J., Trautsch, A., & Herbold, S. (2021). *On the validity of pre-trained transformers for natural language processing in the software engineering domain.*

Wallace, R. (1995). Artificial linguistic internet computer entity (alice). *City*.

Wan, Z., Xia, X., & Hassan, A. E. (2019). What is discussed about blockchain? a case study on the use of balanced lda and the reference architecture of a domain to capture online discussions about blockchain platforms across the stack exchange communities. *IEEE Transactions on Software Engineering*, 1-1. doi: 10.1109/TSE.2019.2921343

Wei, J., & Zou, K. (2019, November). EDA: Easy data augmentation techniques for boosting performance on text classification tasks. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (emnlp-ijcnlp)* (pp. 6382–6388). Hong Kong, China: Association for

Computational Linguistics. Retrieved from https://aclanthology.org/D19-1670 doi: 10.18653/v1/D19-1670

Weizenbaum, J. (1966, January). Eliza-a computer program for the study of natural language communication between man and machine. *Commun. ACM*, *9*(1), 36-45. Retrieved from https://doi.org/10.1145/365153.365168 doi: 10.1145/365153.365168

Wessel, M., de Souza, B. M., Steinmacher, I., Wiese, I. S., Polato, I., Chaves, A. P., & Gerosa, M. A. (2018a, November). The power of bots: Characterizing and understanding bots in oss projects. *Proc. ACM Hum.-Comput. Interact.*, *2*(CSCW). Retrieved from https://doi.org/10.1145/3274451 doi: 10.1145/3274451

Wessel, M., de Souza, B. M., Steinmacher, I., Wiese, I. S., Polato, I., Chaves, A. P., & Gerosa, M. A. (2018b, November). The power of bots: Characterizing and understanding bots in oss projects. *Proc. ACM Hum.-Comput. Interact.*, *2*(CSCW), 182:1–182:19. Retrieved from http://doi.acm.org/10.1145/3274451 doi: 10.1145/3274451

Wessel, M., & Steinmacher, I. (2020). The inconvenient side of software bots on pull requests. In *Proceedings of the 2nd international workshop on bots in software engineering*. IEEE Press.

West, P., Lu, X., Holtzman, A., Bhagavatula, C., Hwang, J. D., & Choi, Y. (2021, August). Reflective decoding: Beyond unidirectional generation with off-the-shelf language models. In *Proceedings of the 59th annual meeting of the association for computational linguistics and the 11th international joint conference on natural language processing (volume 1: Long papers)* (pp. 1435–1450). Online: Association for Computational Linguistics. Retrieved from https://aclanthology.org/2021.acl-long.114 doi: 10.18653/v1/2021.acl-long.114

Woodman, L. (2018). *Facebook chat bot (php webhook) sending multiple replies - stack overflow*. https://stackoverflow.com/questions/36609549/facebook-chat-bot-php-webhook-sending-multiple-replies. ((Accessed on 12/20/2019))

Wyrich, M., & Bogner, J. (2019). Towards an autonomous bot for automatic source code refactoring. In *Proceedings of the 1st international workshop on bots in software engineering* (p. 2428). IEEE Press. doi: 10.1109/BotSE.2019.00015

Xu, A., Liu, Z., Guo, Y., Sinha, V., & Akkiraju, R. (2017). A new chatbot for customer service on social media. In *Proceedings of the 2017 chi conference on human factors in computing systems* (p. 35063510). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/3025453.3025496 doi: 10.1145/3025453.3025496

Xu, B., Xing, Z., Xia, X., & Lo, D. (2017a). AnswerBot: Automated generation of answer summary to developers technical questions. In *Proceedings of the 32nd ieee/acm international conference on automated software engineering* (pp. 706–716). Piscataway, NJ, USA: IEEE Press.

Xu, B., Xing, Z., Xia, X., & Lo, D. (2017b). Answerbot: Automated generation of answer summary to developersź technical questions. In *Proceedings of the 32nd ieee/acm international conference on automated software engineering* (pp. 706–716). Piscataway, NJ, USA: IEEE Press. Retrieved from http://dl.acm.org/citation.cfm?id=3155562.3155650

Xu, S., Semnani, S. J., Campagna, G., & Lam, M. S. (2020). Autoqa: From databases to qa semantic parsers with only synthetic training data. In *In proceedings of the 2020 conference on empirical methods in natural language processing.*

Xuan, J., Martinez, M., DeMarco, F., Clment, M., Marcote, S. L., Durieux, T., ... Monperrus, M. (2017, Jan). Nopol: Automatic repair of conditional statement bugs in java programs. *IEEE Transactions on Software Engineering*, *43*(1), 34-55. doi: 10.1109/TSE.2016.2560811

Yang, X.-L., Lo, D., Xia, X., Wan, Z.-Y., & Sun, J.-L. (2016, Sep 01). What security questions do developers ask? a large-scale study of stack overflow posts. *Journal of Computer Science and Technology*, *31*(5), 910–924.

Ye, D., Xing, Z., Foo, C. Y., Ang, Z. Q., Li, J., & Kapre, N. (2016, March). Software-specific named entity recognition in software engineering social content. In *2016 ieee 23rd international conference on software analysis, evolution, and reengineering (saner)* (Vol. 1, p. 90-101). doi: 10.1109/SANER.2016.10

Zamanirad, S., Benatallah, B., Chai Barukh, M., Casati, F., & Rodriguez, C. (2017). Programming bots by synthesizing natural language expressions into api invocations. In *Proceedings of the 32nd ieee/acm international conference on automated software engineering* (p. 832-837). IEEE Press.

Zamora, J. (2017). I'm sorry, dave, i'm afraid i can't do that: Chatbot perception and expectations. In *Proceedings of the 5th international conference on human agent interaction* (pp. 253–260). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/3125739.3125766 doi: 10.1145/3125739.3125766

Zhang, Q., Fu, J., Liu, X., & Huang, X. (2018). Adaptive co-attention network for named entity recognition in tweets.. Retrieved from https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16432

Zhang, Y., Baldridge, J., & He, L. (2019). PAWS: Paraphrase Adversaries from Word Scrambling. In *Proc. of naacl.*

Zhou, J., Gong, H., & Bhat, S. (2020). Pie: A parallel idiomatic expression corpus for idiomatic sentence generation and paraphrasing. In *The joint conference of the 59th annual meeting of the association for computational linguistics and the 11th international joint conference on natural language processing (acl-ijcnlp 2021), mwe workshop, 2021.*

Zhou, Z.-H. (2017, 08). A brief introduction to weakly supervised learning. *National Science Review*, *5*(1), 44-53. Retrieved from https://doi.org/10.1093/nsr/nwx106 doi: 10.1093/nsr/nwx106

Zhu, X., & Goldberg, A. (2009).